OpenStreetMap Project – Peter Heimann

Map Area:

The area around my home town of New Brunswick, NJ, USA:
http://overpass-api.de/query_form.html
(node(40.4,-74.6,40.6,-74.2);<;);out;
File size: 56.8MB

I also downloaded a subset (40.49,-74.41,40.51,-74.39), for initial examination. The initial problems described below were found and fixed using this subset.

Problems Encountered:

Most of the nodes did not have any information on "created," so the original code from Lesson 6 yielded an empty dictionary. For these nodes, I decided to explicitly show the "created" as "None" rather than leaving it blank.

Under the child element "tag", I noticed keys that begin with "gnis:", which is the USGS Geographic Names Information System, a database that contains millions of names for geographic features in the United States and Antarctica. I decided to translate the gnis keys into the "addr:" keys from the Lesson 6 scripts, and to set the value of the "created:user" key to "GNIS." This will overwrite any existing user values, but I did not notice any in the osm files.

I also noticed keys that begin with "tiger:". According to the OSM Wiki, "The Topologically Integrated Geographic Encoding and Referencing system (TIGER) data, produced by the US Census Bureau, is a public domain data source which has many geographic features. The TIGER/Line files are extracts of selected geographic information, including roads, boundaries, and hydrography features. All of the roads were imported into OSM in 2007 and 2008, populating the nearly empty map of the United States." Again, I translated the tiger keys into the existing "addr:" keys, and I set the value of the "created:user" key to "TIGER".

Finally, I noticed that there are many different keys for the places (nodes), such as "amenity - post_office," "amenity - place_of_worship," "shop – supermarket," "gnis:Class – Populated Area," etc. I translated all of these keys to a single key, "place_type."

In summary, here is the translation, which in some cases had to be done separately for nodes and ways:

Node/Way	OSM Key	Output Key
Node	addr	sub-keys: city, state, street, housenumber, zipcode
Either	created_by	created:user
Either	Source	created:user
Node	amenity	place_type
Node	gnis:Class	place_type
Node	gnis:county_name	addr:county
Node	gnis:County	addr:county
Node	gnis:ST_alpha	addr:state
Node	name	name
Way	name	street
Node	addr:postcode	addr:zipcode
Node	shop	place_type
Way	tiger:county	county
Way	tiger:zip_left	zipcode

I added all of these translations to the Lesson 6 script. Although it would have been more elegant to create a separate dictionary with these translations, I decided to write explicit code for this small number of translations, as shown in the cleanup.py file used for this project.

I ran the python code in the cleanup.py file, and then loaded the resulting output, new_brunswick.osm.json, into MongoDB, with an explicit collection name "nb":

```
Owners-MacBook-Air:Project Owner$ ./bin/mongoimport --file
new_brunswick.osm.json --collection nb --db osm
```

Overview of the Data:

Number of documents in the database:

```
from pymongo import MongoClient
import pprint
client = MongoClient('localhost:27017')
db = client.osm
result = db.nb.find().count()
pprint.pprint(result)

447637
```

I then ran various queries on this database, not only to get statistics, but also to find data issues that I did not notice when perusing the json files. These data issues can be either in the key, as described above, or in the value, as shown later

As an example of errors still present in the keys, here is a count of first-level types:

```
from pymongo import MongoClient
import pprint
client = MongoClient('localhost:27017')
db = client.osm
pipeline = [
    { "$group" : {"_id" : "$type", "count" : {"$sum":1}}},
    { "$sort" : {"count":-1}}
    ]
result = db.nb.aggregate(pipeline)
pprint.pprint(result)

{u'ok': 1.0,
 u'result': [{u'_id': u'node', u'count': 397934},
             {u'_id': u'way', u'count': 49613},
             {u'_id': u'multipolygon', u'count': 86},
             {u'_id': u'gas', u'count': 3},
             {u'_id': u'Public', u'count': 1}]}
```

I saw that 89% of the documents are nodes, and nearly all of the remaining 11% are way, with no relations. But I saw that there is still more cleanup to be done. I looked at the contents of a few of these non-node, non-way documents, and I saw things like:

```
u'type': u'multipolygon', u'natural': u'wood'
u'type': u'multipolygon', u'landuse': u'recreation_ground'
u'type': u'gas', u'location': u'underground', u'man_made': u'pipeline'
u'type': u'Public', u'aeroway': u'aerodrome'
```

I decided to translate "type:multipolygon" to "type:relation," and "type:gas" and "type:Public" to "type:way." Note that the rest of the information (natural:wood, location:underground, aeroway:aerodrome, etc.) would still be captured by the cleanup.py script.

As an example of errors still present in the values, I looked at counts of zip code values. I saw three problems:

1. Some zip codes have zip+4, others just the 5-digit code. Note that the extract.osm file did not have any zip+4 values. They appear to happen mostly with the key "addr:postcode".
2. Zip codes preceded by the text string "NJ" (2 unique values)
3. Multiple zip codes separated by a colon. These appear to happen only on a node_type of "way," and they are probably roads that go through several zip codes. I decide to keep only the first zip code in each pair.

At this point, I added code to the cleanup.py script to fix each of these issues, and I re-created the database. I confirmed that I did successfully fix all of the node_type and zip code errors, with one exception, a zip code of "c":

```
{u'_id': u'c', u'count': 1}
```

I assumed that the users of this data-wrangling output can live with this single error, and I moved on.

As another example of the search for errors in the values, I looked at place_types, to see the result of the translations of keys shown in the table above:

```
pipeline = [
    { "$group" : {"_id" : "$place_type", "count" : {"$sum":1}}},
    { "$sort" : {"count":-1}}
    ]

{u'ok': 1.0,
 u'result': [{u'_id': None, u'count': 444908},
             {u'_id': u'parking', u'count': 972},
             {u'_id': u'school', u'count': 346},
             {u'_id': u'place_of_worship', u'count': 320},
             {u'_id': u'Populated Place', u'count': 155},
             {u'_id': u'restaurant', u'count': 115},
             {u'_id': u'fast_food', u'count': 62},
             {u'_id': u'grave_yard', u'count': 59},
             {u'_id': u'bank', u'count': 50},
             {u'_id': u'supermarket', u'count': 47},
             {u'_id': u'fire_station', u'count': 45},
             etc.
```

I saw many different values for this place_type key, but nothing out of the ordinary, other than that these map data contain more graveyards than banks or supermarkets!

Additional Ideas:

I decided to take a closer look at the creators, because, during the cleanup phase, I saw that there were several sources that had contributed to the OSM data. I wanted to explore a bit more as to which source provided which information. There were 51 unique creators:

```
pipeline = [
    { "$group" : {"_id" : "$created", "count" : {"$sum":1}}},
    { "$group" : {"_id" : "unique users", "count" : {"$sum":1}}}
    ]

{u'ok': 1.0, u'result': [{u'_id': u'unique users', u'count': 51}]}
```

Next, I looked at the distribution of these creators, because I had noticed so many nodes/ways with no creator when I was doing the original cleanup:

```
pipeline = [
    { "$group" : {"_id" : "$created", "count" : {"$sum":1}}},
    { "$sort" : {"count":-1}}
    ]

u'ok': 1.0,
u'result': [{u'_id': u'None', u'count': 383422},
             {u'_id': {u'user': u'ArcGIS Exporter'}, u'count': 41266},
             {u'_id': {u'user': u'TIGER'}, u'count': 16108},
             {u'_id': {u'user': u'NJ2002LULC'}, u'count': 2500},
```

```
             {u'_id': {u'user': u'Bing'}, u'count': 2122},
             {u'_id': {u'user': u'GNIS'}, u'count': 968},
             {u'_id': {u'user': u'Rutgers'}, u'count': 616},
             {u'_id': {u'user': u'TIGER/Line\xae 2008 Place Shapefiles
(http://www.census.gov/geo/www/tiger/)'},
              u'count': 139},
             {u'_id': {u'user': u'bing'}, u'count': 130},
             {u'_id': {u'user': u'USGS Geonames'}, u'count': 107},
             {u'_id': {u'user': u'Merkaartor 0.12'}, u'count': 74},
             {u'_id': {u'user': u'NJ 2002 LULC'}, u'count': 23},
             {u'_id': {u'user': u'local_knowledge'}, u'count': 23},
etc.
```

I saw that 86% of the documents had no creator identified. Of the remainder, 64% were created by "ArcGIS Exporter." According to their web site, "ArcGIS Online is a collaborative, cloud-based platform that allows members of an organization to use, create, and share maps, scenes, apps, and data, and access authoritative basemaps and ready-to-use apps."

Another major creator was NJ2002LULC, which is the NJ Landuse Import into OSM.

I saw that TIGER was the creator for 25% of these creator-identified documents, and GNIS was the creator for 1.5%.

I also saw that there are values "Bing" and "bing" for created:user. I cannot assume that these are the same user, so I will leave this as is until I have had a chance to investigate further.

Next, I looked at creators only for the documents that had a non-null value for place_type, because these are likely to be real places, not just lat-lon coordinates:

```
pipeline = [
    { "$match" : { "place_type": {"$ne":None} } },
    { "$group" : {"_id" : "$created", "count" : {"$sum":1}}},
    { "$sort" : {"count":-1}}
    ]

{u'ok': 1.0,
 u'result': [{u'_id': u'None', u'count': 1705},
             {u'_id': {u'user': u'GNIS'}, u'count': 798},
             {u'_id': {u'user': u'Bing'}, u'count': 133},
             {u'_id': {u'user': u'USGS Geonames'}, u'count': 53},
             {u'_id': {u'user': u'local_knowledge'}, u'count': 18},
             {u'_id': {u'user': u'local knowledge'}, u'count': 6},
             {u'_id': {u'user': u'bing, personal location survey'},
              u'count': 4},
             {u'_id': {u'user': u'Rutgers'}, u'count': 3},
             {u'_id': {u'user': u'osmsync:dero'}, u'count': 3},
             {u'_id': {u'user': u'Mapbox Satellite'}, u'count': 2},
             {u'_id': {u'user': u'NJ2002LULC'}, u'count': 2},
             {u'_id': {u'user': u'Local Knowledge'}, u'count': 1},
             {u'_id': {u'user': u'osmsync:dero; anonymous reader
comment'},
              u'count': 1}]}
```

This list was quite different, in that ArcGIS and TIGER were absent, and NJ2002LULC was much further down on the list. It is not surprising that the identified place names come from different sources than the simple nodes.

This is about as far as I could go based on just the statistics of the documents in this map file. The next step is to determine how we wish to use these data (i.e. are we more interested in simple nodes, or more interested in actual places (e.g. shops, restaurants, schools, etc.). Once we have decided this, we can investigate these various sources of data, and determine which ones are most likely to be trustworthy.

Conclusion:

This project showed how messy the OSM data can be, and it showed two different methods for cleaning the data: visual examination of the json file (which is more readable than the initial xml file), and queries of the data after they are loaded into MongoDB. This should be an iterative process, until the data are clean enough for their intended use.

One final comment: my cleanup.py code explicitly treated each type of error that I found. For instance, the zip+4 issue was only fixed for addr:postcode, while the multiple colon-delimited zipcodes were only fixed for tiger:zip_left. It would have been better if I had used a more generic method, such as first classifying everything into zipcodes and then doing the cleanup. Also, it would have been better if I had used a dictionary to translate the OSM keys to output keys, instead of writing a separate line for each one. The structure of the present code did not lend itself to that approach, but in the future I will structure my code to allow this general approach.