Final Project for Data Wrangling with MongoDB

By Sunila Nagal in fulfillment of Udacity's Data Analyst Nanodegree Project 3

1: Problems Encountered in the Map

1.1 Unexpected Tags

mapparser.py was used to count occurances of each tag, with the result:

bounds:1

member: 1457

nd: 274607

node: 231242

osm: 1

relation: 337

tag: 114246

way: 24544

Functionality was added to examine the keys stored in each tag element in the 'k' attribute. The 20 most common key values were:

('highway', 1788),
('name', 1384),
('tiger:county', 653),
('tiger:cfcc', 636),
('tiger:name_base', 580),
('tiger:name_type', 552),
('tiger:zip_left', 512),
('tiger:zip_right', 492),
('building', 472),
('tiger:reviewed', 459),
('source', 296),
('service', 242),
('oneway', 216),
('amenity', 143),
('lanes', 128),
('natural', 124),
('tiger:tlid', 103),
('tiger:source', 103),
('tiger:upload_uuid', 101),
('ref', 100),

Using tags.py, I parsed through the tag data and sampled a maximum of 20 values for each key. This gave insight on some inconsistencies with data like: Street names abbreviation is some places and full names in other.

1.2 Extra Encoded Data and Systems:

By examining what 'tag' keys were present in the first several million lines, it was clear that at least two separate subsystems were encoded via 'tag' elements: Topologically Integrated Geographic Encoding and Referencing system (TIGER)] and USGS Geographic Names Information System(GNIS) data.

1.3 Specialized Tag Elements:

There are hundreds of tag keys that appeared only once in the data.

1.4 Multiple Zip Codes:

Zip codes were presented in the data under various names like 'tiger:zip_left' and 'tiger:zip_right', defined as semicolon delimited lists or colon delimited ranges.

1.5 Abbreviated Street Names:

There were inconsistencies with street name abbreviations. For consistency, I translated abbreviations into the long forms e.g. Kalakaua Ave was updated to Kalakaua Avenue, Kipapa Dr was updated to Kipapa Drive

1.6 Phone Numder format:

Phone numbers in the OpenStreetMap data for Honolulu, Hawaii are in all different formats. I formatted all of them to be in standard phone number format of (XXX) XXX-XXXX

1.7 Relation and Member Elements:

The dataset has 'relation' and 'member' tags that are used to define logical or geographical groups. For the purposes of populating a document database of 'node' and 'way' tags, I concluded that the best option was to parse these tags out.

2: Data Overview

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

The queries are included in 'query.py'

honolulu_hawaii.osm: 50.4M

Number of documents:

>db.honolulu.count()

51158

Number of nodes and ways:

>db.honolulu.find({‘type’:’node’}).count()

46244

>db.honolulu.find ({‘type’:’way’}).count()

4908

Top 5 contributing users:

```
pipeline = [
        {'$match': {'created.user':{'$exists':1}}},
        {'$group': {'_id':'$created.user',
                'count':{'$sum':1}}},
        {'$sort': {'count':-1}},
        {'$limit' : 5}
]
cursor = db.honolulu.aggregate(pipeline)
```

Result:

```
[{u'_id': u'Tom_Holland', u'count': 18778},
 {u'_id': u'cbbaze', u'count': 3064},
 {u'_id': u'ikiya', u'count': 2548},
 {u'_id': u'pdunn', u'count': 1992},
 {u'_id': u'Chris Lawrence', u'count': 1842}]
```

Restaurants in Honolulu, Hawaii:

```
pipeline = [
        {'$match': {'amenity':'restaurant',
                'name':{'$exists':1}}},
        {'$project':{'_id':'$name',
                 'cuisine':'$cuisine',
                 'contact':'$phone'}},
        {'$limit': 10}
]
```

```
cursor = db.honolulu.aggregate(pipeline)
```

Result:

```
{u'_id': u'Chiang Mai'},
 {u'_id': u"Zippy's Kaimuki"},
 {u'_id': u"Anna Miller's"},
 {u'_id': u'BC Burritto'},
 {u'_id': u'Lanikai Juice'},
 {u'_id': u"Eggs'n Things Restaurant"},
 {u'_id': u'teddys burgers'},
 {u'_id': u'Makahiki \u2014 The Bounty of the Islands',
  u'cuisine': u'international'},
 {u'_id': u'Treetops Restaurant'},
 {u'_id': u"Kaka'ako Kitchen"}
```

User with most contribution:

```
pipeline = [{'$group':
            {'_id':'$created_user',
                'count':{
                   '$sum':1
                   }
            }
        },{
           '$sort':{
              'count':-1
           }
        },{
           '$limit': 1
        }]
cursor = db.honolulu.aggregate(pipeline)
```

Result:

```
[{u'_id': None, u'count': 51158}]
```

Most common building type:

```
pipeline = [{
```

```
        '$match': {
            'building': {
                '$exists': 1}}}
            , {
        '$group': {
            '_id': '$building',
           'count': {
                '$sum': 1}}}
          , {
         '$sort': {
            'count': -1}}
        , {
      '$limit': 10}]
cursor = db.honolulu.aggregate(pipeline)
```

Result:

```
[{u'_id': u'yes', u'count': 656},
 {u'_id': u'apartments', u'count': 72},
 {u'_id': u'house', u'count': 70},
 {u'_id': u'commercial', u'count': 48},
 {u'_id': u'school', u'count': 20},
 {u'_id': u'hotel', u'count': 10},
 {u'_id': u'hangar', u'count': 8},
 {u'_id': u'roof', u'count': 8},
 {u'_id': u'college', u'count': 6},
 {u'_id': u'cabin', u'count': 4}]
```

No. of Nodes without addresses:

```
pipeline = [{
        '$match': {
            'type': 'node',
             'address': {
      '$exists': }}
      }, { '$group': {
```

```
        '_id': 'Nodes without addresses',
        'count': {
           '$sum': 1}}}]
cursor = db.honolulu.aggregate(pipeline)
```

Result:

```
[{u'_id': u'Nodes without addresses', u'count': 46152}]
```

3: Conclusion:

Additional Ideas

With the queries conducted, there was more of a focus on the study of node 'places' rather than ways. Over 90% of nodes do not include addresses and the large number of 'nd' reference tags.

There are several opportunities for cleaning and validating data that is left unexplored. Zip codes are provided in various different tags repeatedly for eg. the repetitive information can be deleted, missing information can be pulled and populated from different sources like Google Maps API (e.g. every node can be populated with latitude-longitude coordinate information) can be used to fill any missing latitude-longitude coordinate information.

Google Maps API for Geocoding is the process of converting addresses (like "1946, Chiba-ken, Ala Moana Boulevard, Honolulu, Hawaii") into geographic coordinates (like "lat" : 21.2919452, "lng" : -157.8506689). Reverse geocoding is the process of converting geographic coordinates into a human-readable address. The Google Maps Geocoding API's reverse geocoding service also lets one find the address for a given place ID. So, this can be used to fill missing informations like, latitide, longitude from addresses, or missing addresses, from available latitude, longitude information.

The anticipated problem with using Google Maps API would be that Google Maps might not have most updated information on addresses of businesses as well. If same latitude, longitude gives 2 different addresses via 2 different systems (say OpenStreetMap, and Google Maps) then we will not know which one is the correct one.

Comments

This project was focused to understand obtaining data in xml format, analyzing it, cleaning and formatting it and saved in JSON format to be fed to MongoDB database for running database queries. Database queries were formulated for extracting relevant and important information. I believe this purpose is served in this exercise.