Data Wrangling with MongoDB Final Project with OpenStreetMap

By Todd Trautman

I chose to use a custom data set exported with the Open Street Map Export Tool centered around Apex, NC and Cary, NC in the USA. The uncompressed file size of the file "map-4" is 51,481,318 bytes.

Problems Encountered in the Map

Two iPython notebooks were used, and are included for additional reference. The first, "Map Analysis" was used to audit data sets and prepare techniques for cleaning. The second, "Prepare for Database" was used to clean the XML Open Street Map file, and rewrite it as a JSON file.

Phone Numbers

One aspect I cleaned up with the data was the phone number format. I chose to standardize with the format +<country code> <area code> <local number> per the wiki.openstreetmap.org documentation.

The following regex statements were compiled:

```
#
# Phone number regex
#
#Gold Standard: +1 919 555 1212
phoneformat = re.compile(r'\+1 \d\d\d \d\d\d \d\d\d\d')
#Ex: 919 5551212 or 919-5551212
badphoneformat = re.compile(r'\d\d\d\W\d\d\d\d\d\d\d')
#Ex: (919) 555-1212 or (919) 555 1212
badphoneformat2 = re.compile(r'\(\d\d\d\)\W\d\d\d\W\d\d\d\d')
#Ex: 919-555-1212
badphoneformat3 = re.compile(r'\d\d\d\W\d\d\d\W\d\d\d\d')
#Ex: 9195551212
badphoneformat4 = re.compile(r'\d\d\d\d\d\d\d\d\d\d')
```

When creating the json file to import into MongoDB, the values for key's phone, fax, contact:phone, and contact:fax were passed through the following scrubbing process to standardize the format:

```
def Phone_Processing(node,key,value):
#
# Any tags that have the phrase 'phone' or 'fax' in the key are scrubbed to
# match a gold standardof a phone number format for US.
#
    if re.search(badphoneformat,value):
        node[key] = "+1 "+value[-11:-8]+" "+value[-7:-4]+" "+value[-4:]
    elif re.search(badphoneformat2,value):
        node[key] = "+1 "+value[-13:-10]+" "+value[-8:-5]+" "+value[-4:]
    elif re.search(badphoneformat3,value):
        node[key] = "+1 "+value[-12:-9]+" "+value[-8:-5]+" "+value[-4:]
    elif re.search(badphoneformat4,value):
        node[key] = "+1 "+value[-10:-7]+" "+value[-7:-4]+" "+value[-4:]
    else: node[key] = value

    return(node)
```

Each value was inspected to see if it met the standard. If it did not, it was passed through the Phone_Processing function for standardization. If a phone number not from the US, (Example: +65 6323 1232), it not modified. After modification, all phone numbers were re-audited to confirm they met the standard to ensure a greedy regular expression did not compromise the result.

City Names

For the map region, there are four possible city names: Apex, Morrisville, Raleigh, and Cary. Of the 1986 occurrences, there were four situations where the data needed to be cleaned. The following function was used to update to the proper values:

```
valid_cities = ['Apex','Morrisville','Raleigh','Cary']
def scrub_city(i):

    if i not in valid_cities:
        for valid in valid_cities:
            if re.search(valid,i,re.IGNORECASE): i = valid

    return (i)
```

Street Names

The last word in street names was harmonized. For example, "Pl" was converted to "Place" and "Dr" was converted to "Drive". This used the standard techniques learned in class.

Overview of the Data

The iPython notebook "Using MongoDB for Openstreet Data" was used for these exercises and is included for additional reference.

Basic Details

After performing the initial audit, cleaning, and exporting the data to json format, it was imported into MongoDB using the command line tool mongoimport. 261,261 documents were imported.

```
MongoDB shell version: 3.0.1
connecting to: test
> use openstreet
switched to db openstreet
> show collections
caryapex
system.indexes
> db.caryapex.count()
261261
```

The number of nodes imported is 238,674 and the number of ways imported is 22,585. There are 193 unique user ID's.

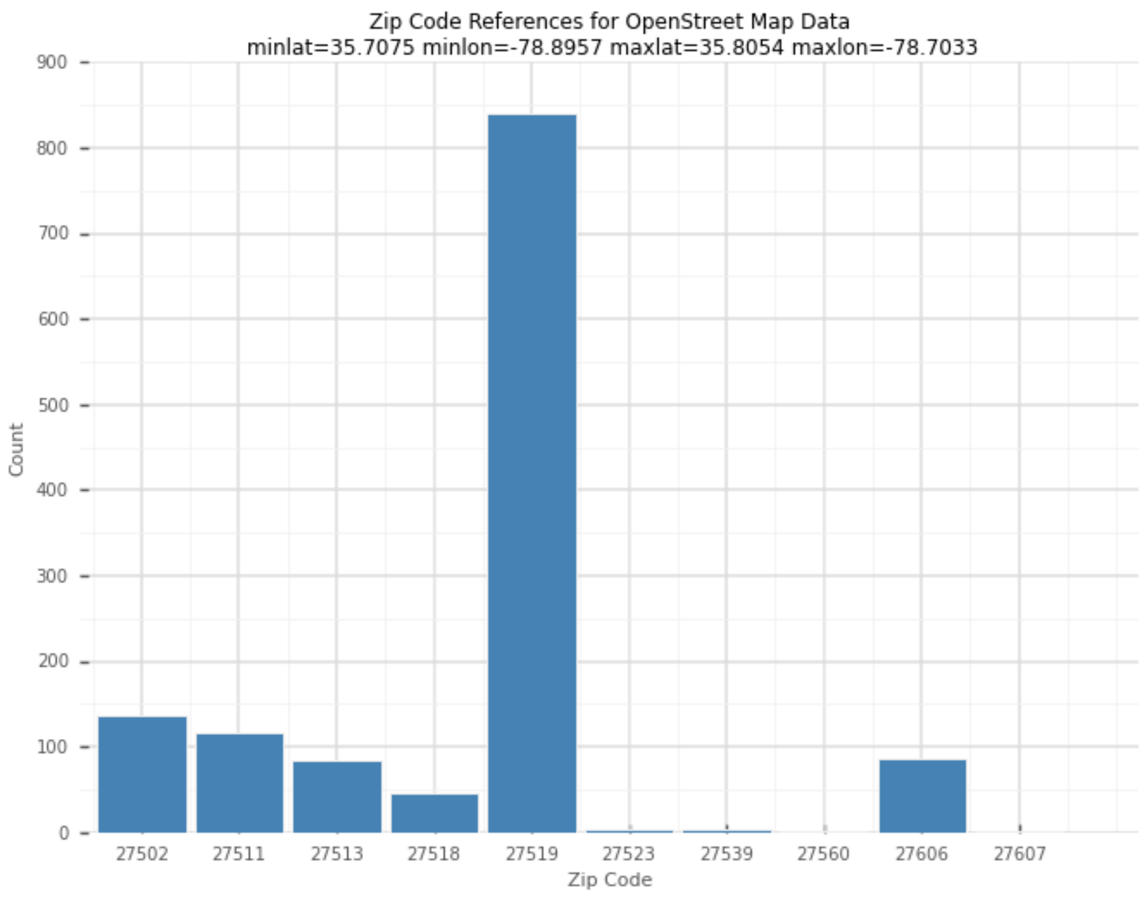
```
nodesq = {"type":"node"}
print db.caryapex.find(nodesq).count()

238674

waysq = {"type":"way"}
print db.caryapex.find(waysq).count()

22585

len(db.caryapex.distinct("created.user"))

193
```

Zip Code References

Using the data set, which zip codes have the most entries? I took advantage of the aggregate capabilities of MongoDB. Given that some zip codes were in 5 digit format and others in the longer format, I used the $substr operator to extract all zip codes in 5 digit format.

```
def make_pipeline():
    pipeline = [
      {"$match" : {"address.postcode" : {"$exists" :1}}},
      {"$project" : {"shortzip" : {"$substr":[ "$address.postcode", 0, 5
]}}},
      {"$group" : {"_id" : "$shortzip","count":{"$sum" : 1}}},
      {"$sort" : {"count" : -1}},
                ]
    return(pipeline)
pipeline = make_pipeline()
result = db.caryapex.aggregate(pipeline)
```

Results were converted to a pandas dataframe and plotted using ggplot.

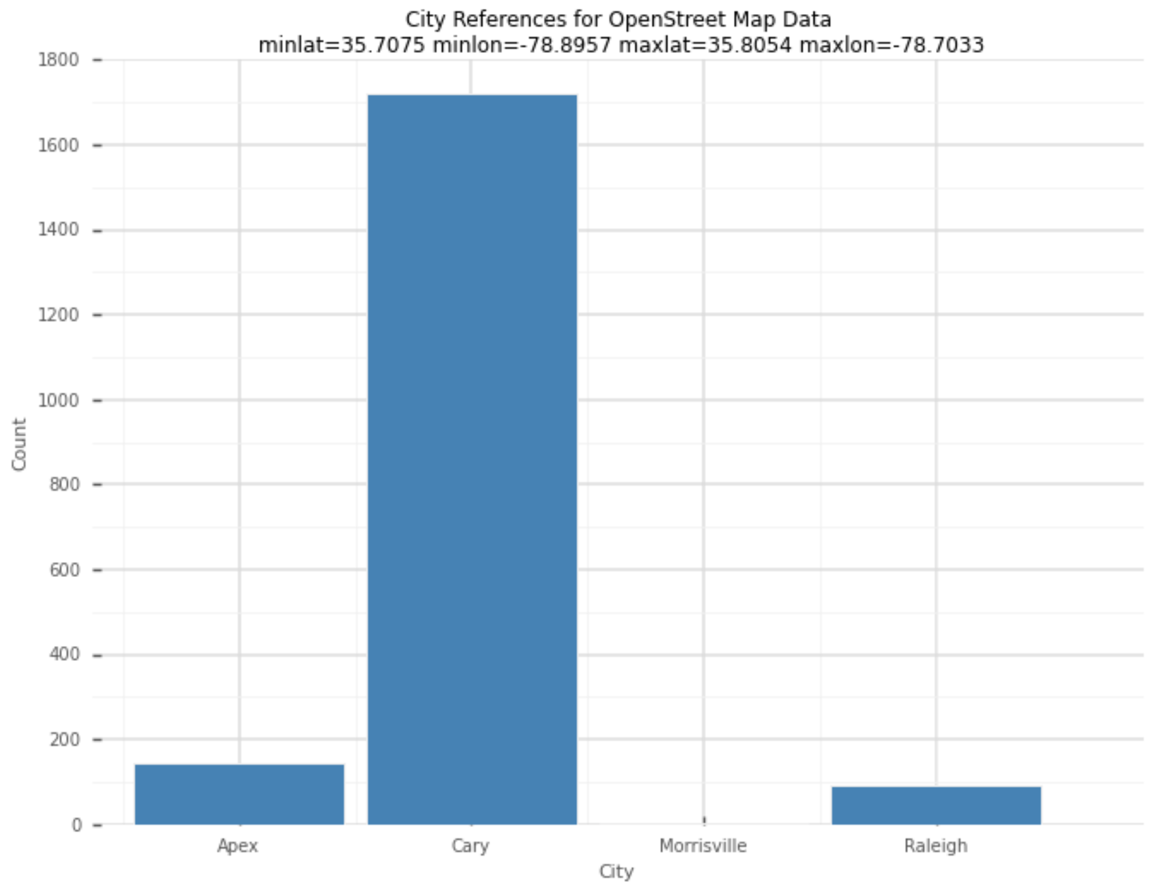


City References

Similar to the zip code analysis, the number of city references was also explored.

```
def city_pipeline():
    pipeline = [
                {"$match" : {"address.city" : {"$exists" :1}}},
                {"$group" : {"_id" : "$address.city","count":{"$sum" : 1}}},
                {"$sort" : {"count" : -1}},
                ]
    return(pipeline)
pipeline = city_pipeline()
result = db.caryapex.aggregate(pipeline)
pprint.pprint(result["result"])

[{u'_id': u'Cary', u'count': 1720},
 {u'_id': u'Apex', u'count': 145},
 {u'_id': u'Raleigh', u'count': 92},
 {u'_id': u'Morrisville', u'count': 2}]
```

Results were converted to a pandas dataframe and plotted using ggplot.



Amenity & Cuisine

For the key of “amenity” the top 10 amenities in the area are:

```
def amenity_pipeline():
    pipeline = [
                {"$match" : {"amenity" : {"$exists" :1}}},

                {"$group" : {"_id" : "$amenity","count":{"$sum" : 1}}},
                {"$sort" : {"count" : -1}},
                {"$limit" : 10}
               ]
    return(pipeline)
pipeline =amenity_pipeline()
result = db.caryapex.aggregate(pipeline)
for i in result["result"]:
    print "%s: %s" % (i["_id"],i["count"])

parking: 202
restaurant: 93
fast_food: 80
bench: 67
bicycle_parking: 61
school: 59
place_of_worship: 55
vending_machine: 48
waste_basket: 43
dentist: 35
```

For the key of “cuisine” the top 10 types of cuisine are:

```
def cuisine_pipeline():
    pipeline = [
                {"$match" : {"cuisine" : {"$exists" :1}}},

                {"$group" : {"_id" : "$cuisine","count":{"$sum" : 1}}},
                {"$sort" : {"count" : -1}},
                {"$limit" : 10}
               ]
    return(pipeline)
pipeline =cuisine_pipeline()
result = db.caryapex.aggregate(pipeline)
for i in result["result"]:
    print "%s: %s" % (i["_id"],i["count"])
```

```
pizza: 19
sandwich: 19
mexican: 12
chinese: 11
burger: 11
ice_cream: 8
chicken: 7
american: 7
asian: 6
japanese: 6
```

Way with the Most Referenced Nodes

Each way has a list of referenced nodes stored in an array with the key value of "node_refs". To determine which node has the most referenced nodes, I used the $project stage operator to create a field labeled "numnodes". The value of this field was determined by the length of the "node_refs" array using the $size operator. Unfortunately the node did not have a name, but the number of nodes was 626.

```
def noderef_pipeline():
    pipeline = [
                {"$match" : {"type" : "way","node_refs":{"$exists": 1}}},
                {"$project" :{ "_id" : 1, "id":1,
                              "name" : 1,"timestamp": "$created.timestamp",
                              "numnodes": {"$size" : "$node_refs"}  }},
                {"$sort" : {"numnodes" : -1}},
                {"$limit" : 1}
               ]
    return(pipeline)
pipeline =noderef_pipeline()
result = db.caryapex.aggregate(pipeline)
print result["result"][0]["numnodes"]
626
```

By updating the pipeline $match to include the key "name," I learned the way with the most references that has a name is Lake Johnson with 323 referenced nodes.

Additional Ideas for the Dataset

With more time, I would like to utilize MongoDB's geospatial capabilities to explore the proximity of emergency first responders to locations such as elementary schools. One approach would be to extract a collection of documents detailing schools, police stations, fire stations, emergency rooms, and ambulance depots. Using the $near operator for each school, a listing of nearby first responders could be produced and each school ranked in terms of access to nearby facilities.

Another possible use for geospatial capabilities would be to do initial market research on where to open a new type of business.   For example, by extracting all the restaurants with a cuisine of pizza, and combining with reviews from Yelp might identify the best location for an underserved market in desperate need of better pizza.