Data Wrangle Project

by Carvus Andrew Byrd IV

Location: Raleigh, NC

I choose Raleigh because I grew up 30 miles outside of Raleigh and spent 5 years at NCSU. I know the area pretty well and found pleasure in helping update the map.

Sources:

https://www.openstreetmap.org/relation/179052
https://mapzen.com/data/metro-extracts/#raleigh-north-carolina
https://docs.mongodb.org/manual/

Problems Encountered in the Map:

Upon downloading the data of Raleigh, NC, I ran some of the same code we used for Lesson 6 in regards to auditing basic tag types (mapparser.py) and errors within the tags (tags.py). I moved to further exploring the Street Addresses and Postcodes for obvious issues (audit.py). After I felt I have identified the larger issues among these two location tags, I cleaned the data (data.py) and uploaded the data to MongDB for further analysis within their framework.

Auditing:

mapparser.py:

```
{'bounds': 1,
 'member': 7964,
 'nd': 2486815,
 'node': 2207144,
 'osm': 1,
 'relation': 795,
 'tag': 828063,
 'way': 225021}
```

The map parser query was used to determine the count of tags. This exercise is mostly concerned with the "node" and "way" tags but it is still helpful to run.

tags.py:

```
{'lower': 511442, 'lower_colon': 273000, 'other': 43621, 'problemchars': 0}
```

The tags query allows the user determine if the "k" value of each tag has problematic characters (i.e. !@#$%^&*) within the field. For example, the "lower" description represents a count of "k" value tags that do not have problematic characters and are all lower case. Now that we know there aren't any major character issues, we can look at some of the abbreviations and such.

audit.py:

Street Address (Excerpt):

```
{'100': set(['100']),
 '1000': set(['Six Forks Road #1000']),
 '17': set(['US Highway 17']),
 '206': set(['Barrett Dr Suite 206']),
 '501': set(['US 15;US 501']),
 '54': set(['Highway 54',
            'State Highway 54',
            'West Highway 54',
            'West NC Highway 54']),
 '55': set(['Highway 55', 'NC Highway 55', 'US 55']),
 '70': set(['US 70']),
 '751': set(['NC Highway 751']),
 'Ave': set(['Atlantic Ave',
             'E. Winmore Ave',
             'East Winmore Ave',
             'Fernway Ave',
             'Glenwood Ave',
             'Mountford Ave',
             'S Boylan Ave',
             'S. Boylan Ave']),
 'Blvd': set(['Airport Blvd',
              'Capital Blvd',
              'Martin Luther King Jr Blvd',
              'Martin Luther King Jr. Blvd',
              'Martin Luther King Junior Blvd',
              'Southpoint Autopark Blvd',
              'Witherspoon Blvd']),
 'Blvd.': set(['Durham-Chapel Hill Blvd.',
               'N Fordham Blvd.',
               'Southpoint Auto Park Blvd.']),
 'CIrcle': set(['Meadowmont Village CIrcle']),
 'Chatham': set(['East Chatham']),
 'Cir': set(['Braddock Cir', 'Daybrook Cir']),
 'Ct': set(['Ashley Springs Ct', 'Bevel Ct', 'Braidwood Ct', 'Gulf Ct']),
 'Dr': set(['Allister Dr',
            'Calabria Dr',
            'Crab Orchard Dr',
            'Detrick Dr',
            'Fig Vine Dr',
            'Gold Star Dr',
            'Sablewood Dr',
            'Stinson Dr',
            'Triangle Plantation Dr',
            'University Dr',
            'Waterford Lake Dr',
            'Westgate Dr']),
  ....}
```

Postcodes (Excerpt):

```
{....
'27612-3451': 2,
 '27612-3452': 3,
 '27612-3453': 4,
 '27612-3454': 3,
 '27612-3455': 3,
 '27612-3456': 5,
 '27612-3461': 3,
 '27612-3462': 4,
 '27612-3463': 2,
 '27612-3464': 2,
 '27612-3466': 4,
 '27612-3467': 7,
 '27612-3468': 9,
 '27612-3800': 11,
 '27612-3901': 4,
 '27612-3902': 5,
 '27612-3903': 5,
 '27612-3904': 2,
 '27612-3905': 3,
 '27612-3906': 7,
 '27612-3907': 7,
 '27612-3910': 6,
 ....}
```

This output from the “audit.py” displays street addresses that are not expected (as opposed to expected variety like “Street”, “Court”, “Road”, etc.). Most of these descriptions are quite fixable abbreviations that can be expanded to their proper form. Some of the first few with numbers on the end must be apartment numbers or highway numbers (very common in that area) that are not formatted correctly.

To determine issues within postcodes, I had the audit query give me all zip codes that weren’t exactly 5 characters long. This allowed me to see there are postcodes that are incorrect in terms of showing multiple codes for one given area (i.e. 27XXX:27XXX), postcodes with the “-XXXX” extension at the end, or even 2 letters (“NC”) at the beginning.

Cleaning:

Now that we have identified street types/addresses and postcode issues, we can use the “data.py” query to both clean said issues and convert the XML file to JSON (so that we can upload to MongoDB). To clean the street types, I replaced the abbreviations with the matching expected non-abbreviated street type (like below).

```
# Street values I have found that should be edited programmatically
mapping = { "St": "Street",
            "St.": "Street",
```

```
            "ST": "Street",
            "St,": "Street",
            "Ave": "Avenue",
            "Ave.": "Avenue",
            "Rd": "Road",
            "Rd.": "Road",
            "Blvd": "Boulevard",
            "Blvd.": "Boulevard",
            "blvd": "Boulevard",
            "Courth": "Court",
            "Ct": "Court",
            "DR": "Drive",
            "Dr": "Drive",
            "Dr.": "Drive",
            "Ln": "Lane",
            "Ter": "Terrace",
            "drive": "Drive",
            "broadway": "Broadway",
            "CIrcle": "Circle",
            "Cir": "Circle",
            "Pl": "Place",
            "Pkwy": "Parkway",
            "Pky": "Parkway",
            "Pl": "Place",
            }

#function to update the street name using the mapping above
def update_name(name, mapping):
    for key in mapping.keys():
        if name.find(key) != -1:
            name = name[:name.find(key)]+mapping[key]
    return name
```

To clean the postcodes, I used the logic below to help sort out the various issues that arose in the data.

```
#function to update zipcodes based on length logic
def update_code(postal_code):
    #turn postcode into a list split among dash
    split_code = re.split('-',postal_code)
    if len(split_code) > 1:
        #if code is a list, does it have any letters in?
        m = postcode_letter_re.search(split_code[0])
        if m:
            #if the first element has letters, code equals first 5 characters
of 2nd element
            postal_code = split_code[1]
        if len(split_code[-1]) < 5:
            if len(split_code) > 2:
                #if postcode has both prefix letters and dash extension
```

```
                postal_code = split_code[1]
            elif len(split_code) < 3:
                #if it just has an dash extension
                postal_code = split_code[0]
    else:
        if postcode_letter_re.search(split_code[0]):
            #if anything else, return none
            postal_code = None
    return postal_code
```

Within the python shell, I used the mongoimport request below:

```
mongoimport --file /data/raleigh_north-carolina.osm.json --db cities
--collection raleigh
```

This allowed me to run a basic query to help determine everything ran successfully.

```
> db.raleigh.count()
2432165
```

Data Overview:

File Size:

The original XML file is 457.6 MB, while the converted JSON file is a little larger at 531.5 MB.

Number of Unique Users:

```
>pipeline = [{"$group" : {"_id" : "$created.user"} },
            {"$group" : {"_id" : 1, "count" : {"$sum" : 1 }}}]

[{u'_id': 1, u'count': 725}]
```

Top Contributing User:

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

[{u'_id': u'jumbanho', u'count': 1581257}]
```

Number of Nodes and Ways:

```
> db.raleigh.find({"type":"node"}).count()
2207140

> db.raleigh.find({"type":"way"}).count()
225017
```

Top 10 Amenities in Raleigh

```
>pipeline = [{"$match" : {"amenity" : {"$exists" : "true"}}},
                { "$group" : { "_id" : "$amenity",
                "count" : {"$sum" :1}}},
                { "$sort" : {"count" : -1}},
                {"$limit":10}]

[{u'_id': u'parking', u'count': 1929},
 {u'_id': u'bicycle_parking', u'count': 552},
 {u'_id': u'place_of_worship', u'count': 544},
 {u'_id': u'restaurant', u'count': 511},
 {u'_id': u'fast_food', u'count': 253},
 {u'_id': u'school', u'count': 226},
 {u'_id': u'fuel', u'count': 204},
 {u'_id': u'bench', u'count': 131},
 {u'_id': u'bank', u'count': 113},
 {u'_id': u'swimming_pool', u'count': 106}]
```

What kinds of Restaurants (Top 15)?

```
>pipeline = [{"$match" : {"amenity" : "restaurant"}},
                { "$group" :
                 { "_id" : "$cuisine","count" : {"$sum" : 1}}},
                { "$sort" : {"count" : -1}},
                {"$limit":15}]

[{u'_id': None, u'count': 186},
 {u'_id': u'mexican', u'count': 39},
 {u'_id': u'american', u'count': 37},
 {u'_id': u'pizza', u'count': 32},
 {u'_id': u'italian', u'count': 28},
 {u'_id': u'burger', u'count': 25},
 {u'_id': u'chinese', u'count': 22},
 {u'_id': u'sandwich', u'count': 15},
 {u'_id': u'asian', u'count': 15},
 {u'_id': u'japanese', u'count': 15},
 {u'_id': u'regional', u'count': 14},
 {u'_id': u'seafood', u'count': 8},
 {u'_id': u'steak_house', u'count': 6},
 {u'_id': u'greek', u'count': 6},
 {u'_id': u'thai', u'count': 5}]
```

Other Ideas for Cleaning Data:

One issue that caught my eye towards the end of the project was the disparity among tag titles, such as zip codes and postcodes. It appears that postcodes are more commonly used by there are addresses using zip codes instead. One could consolidate that field along with others that I am quite sure are very similar. This makes aggregating data in MongoDB afterwards so much easier. One would have to be careful consolidating too many fields as they may serve other purposes or disrupt other users' queries.

Also, I found that a lot of address fields did not have either a street name/type or postcode (even though the address field is present in the document) as seen below. It's tough to replace a value that isn't there (with a function). It likely that this information would need to be entered individually.

Most Occuring Street Addreses:

```
[{u'_id': None, u'count': 1499},
 {u'_id': u'Street', u'count': 195},
 {u'_id': u'Glenwood Avenue', u'count': 117},
 {u'_id': u'Yadkin Drive', u'count': 111},
 {u'_id': u'Place', u'count': 103},
 {u'_id': u'Preston Grove Avenue', u'count': 102},...]
```

Most Occurring Postcodes:

```
[{u'_id': None, u'count': 7528},
 {u'_id': u'27612', u'count': 1732},
 {u'_id': u'27560', u'count': 1555},
 {u'_id': u'27609', u'count': 1289},
 {u'_id': u'27519', u'count': 901},
 {u'_id': u'27701', u'count': 682},...]
```

Conclusion:

I could see this being a small help to the users of this map. I know I didn't fix all the problems but I think I made a small dent. It seems that there have been quite a few users helping so I would think this map should continue to get better. I can also see where this can go bad if you are a novice data wrangling (I had to go through the whole process 3 or so times before I felt good about result) and you upload a bunch of junk to the map. I'm sure there could even be some folks doing it on purpose. Either way, open source mapping is very fascinating and I hope I can contribute more in the future.