Wrangle OpenStreetMap Data

September 21, 2017

0.1 Udacity Data Analyst Nanodegree

0.1.1 P4 - Wrangle OpenStreetMap Data

Author: Jack Bae

0.2 Introduction

This project involves auditing data of an area of the world in OpenStreetMap then performing an exploratory data analysis in MongoDB.

I chose Atlanta, Georgia area from OpenStreetMap OSM XML data from MapZen as this is where I currently reside. I am hoping to find new areas of interest through analyzing this data.

0.2.1 Data Audit

As part of the data audit, I first parsed first parsed through the OSM file to count the top-level tags using iterative parsing process.

```
In [ ]: def count_tags(filename):
                tags = {}
                for event, elem in ET.iterparse(filename):
                    if elem.tag in tags:
                        tags[elem.tag] += 1
                    else:
                        tags[elem.tag] = 1
                return tags
        counted_tags = count_tags(filename)
        pprint.pprint(counted_tags)
```

```
{'bounds': 1,
 'member': 39544,
 'nd': 13391239,
 'node': 11865021,
 'osm': 1,
 'relation': 4078,
 'tag': 6277197,
 'way': 864467}
```

I then checked the "k" value for each tag to see if there are any potential problems. * "lower" indicates tags contain only valid lowercase letters * "lower_colon" indicates indicates otherwise valid tags with a colon in their names * "problemchars" indicates tags with problematic characters * "other" indicates other tags that do not fall into the other three categories

```
In [ ]: lower = re.compile(r'^([a-z]|_)*$')
        lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
        problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')

        def key_type(element, keys):
            if element.tag == "tag":
                if lower.search(element.attrib['k']):
                    keys["lower"] += 1
                elif lower_colon.search(element.attrib['k']):
                    keys["lower_colon"] += 1
                elif problemchars.search(element.attrib['k']):
                    keys["problemchars"] += 1
                else:
                    keys["other"] += 1
            return keys

        def process_map(filename):
            keys = {"lower": 0, "lower_colon": 0, "problemchars": 0, "other": 0}
            for _, element in ET.iterparse(filename):
                keys = key_type(element, keys)

            return keys

        keys = process_map(filename)
        pprint.pprint(keys)
```

```
{'lower': 2934952, 'lower_colon': 2448002, 'other': 894238, 'problemchars': 5}
```

I also counted number of contributed users as part of data audit.

```
In [ ]: def get_user(element):
            return element.get("uid")

        def process_map(filename):
            users = []
            for _, element in ET.iterparse(filename):
                if element.tag == "node" or element.tag == "relation" or element.tag == "way":
                    e = get_user(element)
                    if e in users:
                        continue
                    else:
                        users.append(e)
            return users
```

```
contributed_users = process_map(filename)
len(contributed_users)
```

```
790
```

0.3 Problems Encountered in the Map

0.3.1 Inconsistent Street Name Abbreviation

As the data in OpenStreetMap are generated by users, the main problem in the data is street name abbreviation inconsistencies. An audit and modification of incorrect abbreviation in street names are needed to have consistent OpenStreetMap data. In this process, I have fixed the inconsistent abbreviations in street address to have a more consistent full street address.

- Ave, Ave. -> Avenue
- Blvd, Blvd. -> Boulevard
- Ct, Ct. -> Court
- Dr, Dr. -> Drive
- Ln, Ln. -> Lane
- Pkwy, Pkwy. -> Parkway
- Rd, Rd. -> Road
- St, St. -> Street

```
In [ ]: from collections import defaultdict

        street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)

        expected = ["Avenue", "Boulevard", "Commons", "Court", "Drive", "Lane", "Parkway",
                                "Place", "Road", "Street"]

        mapping = {'Ave'  : 'Avenue',
                   'Ave.'  : 'Avenue',
                   'Blvd' : 'Boulevard',
                   'Blvd.' : 'Boulevard',
                   'Ct'   : 'Court',
                   'Ct.'   : 'Court',
                   'Dr'   : 'Drive',
                   'Dr.'   : 'Drive',
                   'Ln'   : 'Lane',
                   'Ln.'   : 'Lane',
                   'Pkwy' : 'Parkway',
                   'Pkwy.' : 'Parkway',
                   'Rd'   : 'Road',
                   'Rd.'   : 'Road',
                   'St'   : 'Street',
                   'St.'   : 'Street'}

        def audit_street_type(street_types, street_name):
            m = street_type_re.search(street_name)
```

```
        if m:
            street_type = m.group()
            if street_type not in expected:
                street_types[street_type].add(street_name)

def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")

def audit(osmfile):
    osm_file = open(osmfile, "r")
    street_types = defaultdict(set)
    for event, elem in ET.iterparse(osm_file, events=("start",)):

        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_street_name(tag):
                    audit_street_type(street_types, tag.attrib['v'])

    return street_types


def update_name(name, mapping):
    name = name.split(' ')
    type = name[-1]
    if type in mapping:
        name[-1] = mapping[type]
    name = ' '.join(name)
    return name


st_types = audit(filename)

pprint.pprint(dict(st_types))

for st_type, ways in st_types.items():
    for name in ways:
        better_name = update_name(name, mapping)
        print (name, "=>", better_name)
```

0.3.2 Inconsistent Postal Codes

Another common user-generated inconsistency in the data is the postal code. One of the main problem contributing to the inconsistency in postal code is the division between 5 digit postal code and 9 digit postal code. Consolidating all zipcodes to 5 digits would bring consistency in postal code data.

0.3.3 Data Overview

After auditing data using `audit.py`, I converted `atlanta_georgia.osm` into a json file format to import into MongoDB. The `atlanta_georgia.osm.json` file was created from `data.py` then imported into MongoDB using `mongoimport -d atlanta -c atlanta --file atlanta_georgia.osm.json`.

File Size

- `atlanta_georgia.osm`: 2.55 GB
- `atlanta_georgia.osm.json`: 2.89 GB

Number of Unique Users `> db.atlanta.distinct("created.user").length`

```
2492
```

Number of Nodes `> db.atlanta.find({"type":"node"}).count()`

```
11864930
```

Number of Ways `> db.atlanta.find({"type":"way"}).count()`

```
864427
```

Number of Cafe `> db.atlanta.find({amenity:"cafe",type:"node"}).count()`

```
174
```

Number of Fast Food Restaurant `> db.atlanta.find({amenity:"fast_food",type:"node"}).count()`

```
544
```

Top 10 Amenities `> db.atlanta.aggregate([ {$match:{"amenity":{"$exists":1},"type":"node"}}, {"$group":{"_id":"$amenity","count":{"$sum":1}}}, {$sort:{"count":-1}}, {"$limit":10} ])`

```
{ "_id" : "place_of_worship", "count" : 3993 } { "_id" : "grave_yard", "count"
: 2047 } { "_id" : "school", "count" : 2045 } { "_id" : "restaurant", "count" :
1030 } { "_id" : "fast_food", "count" : 544 } { "_id" : "bench", "count" : 356
} { "_id" : "fuel", "count" : 339 } { "_id" : "fire_station", "count" : 237 } {
"_id" : "atm", "count" : 217 } { "_id" : "post_office", "count" : 212 }
```

Top 10 Contributing Users `> db.atlanta.aggregate([ {"$match":{"type":"node"}}, {"$group":{"_id":"$created.user","count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":10} ])`

```
{ "_id" : "Liber", "count" : 5172224 } { "_id" : "Saikrishna_FultonCountyImport",
"count" : 2167668 } { "_id" : "woodpeck_fixbot", "count" : 1444867 } { "_id"
: "Jack the Ripper", "count" : 372135 } { "_id" : "afonit", "count" : 300265 }
{ "_id" : "rjhale1971", "count" : 253450 } { "_id" : "Jack Kittle Buildings",
"count" : 227517 } { "_id" : "maven149", "count" : 218973 } { "_id" : "Chris
Lawrence", "count" : 119056 } { "_id" : "macon_cfa", "count" : 86889 }
```

Top 10 Cuisines `> db.atlanta.aggregate([ {"$match": {"cuisine": {"$exists": 1}}}, {"$group": { "_id": "$cuisine","count": {"$sum": 1}}}, {"$sort": {"count": -1}}, {"$limit": 10} ])`

`{ "_id" : "burger", "count" : 281 } { "_id" : "pizza", "count" : 144 } { "_id" : "mexican", "count" : 134 } { "_id" : "american", "count" : 134 } { "_id" : "sandwich", "count" : 133 } { "_id" : "chicken", "count" : 118 } { "_id" : "coffee_shop", "count" : 84 } { "_id" : "chinese", "count" : 59 } { "_id" : "diner", "count" : 51 } { "_id" : "italian", "count" : 42 }`

0.3.4 Additional Ideas About the Dataset

Looking back at top 10 contributed user query,

`> db.atlanta.aggregate([ {"$match":{"type":"node"}}, {"$group":{"_id":"$created.user","count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":10} ])`

the second result of `{ "_id" : "Saikrishna_FultonCountyImport", "count" : 2167668 }` indicated that this user might be importing data from another source. The OpenStreetMap Wiki confirmed this, indicating that Fulton County GIS data has been imported into the OpenStreetMap. Similar process could be implemented to improve other metro Atlanta counties such as DeKalb, Gwinnett, Cobb, and Clayton Counties. Gwinnett County provides GIS infomation that could help improving address information for OpenStreetMap.

While importing GIS data could bring in a rich set of data, it could also cause a problem in deciding duplicate entries between GIS and user-inputted data. OpenStreetMaps could also lose its strengh of having up-to-date user input data by implementing County GIS data.

0.3.5 Conclusion

As seen in the data audit phase, the OpenStreetMap data for Atlanta, GA region contains inconsistent data. The number of users contributed to this dataset may be a big reason why there are inconsistencies in the data. One effort to alleviate this issue would be importing data from reputable sources such as the GIS information that was imported from Fulton County database.

0.3.6 Files

- `sample.py` - Creates `sample.osm` data
- `sample.osm` - A sample of the `atlanta_georgia.osm` data
- `mapparser.py` - Finds number of unique tags
- `tags.py` - Counts key types from dictionary
- `users.py` - Counts number of contributed users
- `audit.py` - Audits and updates street name
- `data.py` - Converts osm data into json file format