Project4

July 27, 2017

1 OpenStreetMap Data Case Study

1.1 Map Area

Rhode Island, United States - http://download.geofabrik.de/north-america/us/rhode-island-latest.osm.bz2 I picked this data because last time I visited, the state was beautiful so I'd like to know more about Rhode Island.

1.2 Problems Encountered in the Map

When I looked up the list of tags (I had no idea what tags mean), I could see that lots of data are very dirty (which means not consistent/no uniformity), so I decide to clean it up so that be able to get more accurate statistics.

```
In [3]: street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)
        street_types = defaultdict(int)

        def audit_street_type(street_types, street_name):
            """
            audit street type, and save all the types of data in street_types
            """
            m = street_type_re.search(street_name)
            if m:
                street_type = m.group()

                street_types[street_type] += 1

        def print_sorted_dict(d):
            """
            print sorted dictionary
            """
            keys = d.keys()
            keys = sorted(keys, key=lambda s: s.lower())
            for k in keys:
                v = d[k]
                print ("%s: %d" % (k, v))
```

```
def is_street_name(elem):
    """
    it is a name of street only when the tag of the element is "tag" and its value of
    """
    return (elem.tag == "tag") and (elem.attrib['k'] == "addr:street")

def audit():
    """
    audit it
    """
    for event, elem in ET.iterparse(osm_file):
        if is_street_name(elem):
            audit_street_type(street_types, elem.attrib['v'])
    print_sorted_dict(street_types)

if __name__ == '__main__':
    audit()
```

```
106: 1
146: 1
201: 1
4: 1
A: 1
Alley: 2
Ave: 68
ave: 1
Ave.: 4
Avenue: 296
Blvd: 2
Boulevard: 18
BowenStreet: 1
Broadway: 4
Circle: 10
Court: 14
Ct.: 1
Dr: 12
Dr.: 21
Drive: 159
Highway: 26
HIGHWAY: 1
Hill: 2
Hwy: 1
Island: 1
Lane: 24
Ln: 1
Parkway: 5
Pike: 22
PIKE: 1
```

```
Place: 4
Plaza: 2
Raod: 1
Rd: 79
Rd.: 1
Road: 242
road: 1
Sq.: 1
Square: 3
St: 57
St.: 9
Street: 373
Trail: 4
Way: 50
wht: 1
Wy: 2
```

I can see a lot! Now it's time to make auditing more specifically.

```
In [4]: street_types = defaultdict(set)

        #the list of expected name of streeets
        expected_street_types = ["Broadway", "Lane", "Hill", "Street", "Plaza", "Highway", "Wa
        #extended list of expected name of streets (description below)
        expected_street_types.extend(['A', '4', 'Alley'])
        def audit_street_type(street_types, street_name):
            """
            same as above audit function, but at this time, save only those that is not in the
            """
            m = street_type_re.search(street_name)
            if m:
                street_type = m.group()
                if street_type not in expected_street_types:
                    street_types[street_type].add(street_name)

        def audit():
            for event, elem in ET.iterparse(osm_file, events=("start",)):
                if 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
```

```
In [5]: osm_file.seek(0)

        #List of dictionary for mapping (swapping the word)
        mapping = {"Ave": "Avenue",
                   "Ave.": "Avenue",
                   "Blvd": "Boulevard",
                   "Dr": "Drive",
                   "Dr.": "Drive",
                   "Raod": "Road",
                   "Rd.": "Road",
                   "Sq.": "Square",
                   "Wy": "Way",
                   "St": "Street",
                   "St.": "Street",
                   "Rd": "Road"
                    }

        mapping.update({"BowenStreet": "Bowen Street"})

        def update_name(name, mapping):
            """
            swap the name by mapping
            (ex. if the value is "Rd", swap it to the definition of "Rd" which is "Road")
            """
            temp = ""
            m = street_type_re.search(name)
            if m:
                for value in mapping:
                    if name.find(value) != -1:
                        temp = name[:name.index(value)] + mapping[value]
            name = temp
            return name

        def replace_word():
            """
            replace the word and update it
            """
            st_types = audit()
            for st_type, ways in st_types.items():
                for name in ways:
                    update_name(name, mapping)
        #             better_name = update_name(name, mapping)
        #             print (name, "=>", better_name)


        if __name__ == '__main__':
            replace_word()
```

```
In [6]: #Additional corrections: Avenue A, Lane 4, wht, Fones Alley, BowenStreet.
        #Avenue A, Lane 4, Fones Alley actually exist, so add on the list 'expected'
        #Add "BowenStreet": "Bowen Street" to dictionary
        #I couldn't find the definition of 'wht'
```

Done with the street! Next is the state name

```
In [7]: #similar to the street, but state at this time
        state = []
        osm_file.seek(0)

        def audit_state():
            for event, elem in ET.iterparse(osm_file, events=("start",)):
                if elem.tag == "way":
                    for tag in elem.iter("tag"):
                        if tag.attrib['k'] == "addr:state":
                            if tag.attrib['v'] != 'RI':
                                state.extend([tag.attrib['v']])
            return state

        mapping_state = {"RO": "RI",
                   "ri": "RI",
                   "Rhode Island": "RI",
                    }

        def update_state_name():
            for i in range(0, len(state)):
                state[i] = "RI"

        def replace_state_name():
            state = audit_state()
            for ways in state.items():
                for name in ways:
                    update_state_name(name, mapping)

        if __name__ == '__main__':
            update_state_name()
```

Done with state! Done with everything I've planned! Next one is to convert the cleaned data into csv, then start working on data!

2 OpenStreetMap Data Case Study

First thing to see is the file size. ## File Size

```
rhode-island-latest.osm .. 182 MB
Project4 ................. 279 MB
nodes.csv ................ 70 MB
nodes_tags.csv ........... 2.4 MB
ways.csv ................. 5.2 MB
ways_tags.csv ............ 12 MB
ways_nodes.cv ............ 22 MB
```

2.1 Did my Python code really work?

To test if my python code really changes the data, I will try the following code. (my code cleanse the name of road, such that "Blvd"=>"Boulevard")

```
SELECT *
FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) tags
WHERE value
LIKE '%Blvd';
```

```
19343479|name_type|Blvd|tiger
19345366|name_type|Blvd|tiger
19345373|name_type|Blvd|tiger
19346665|name_type|Blvd|tiger
19347232|name_type|Blvd|tiger
19347234|name_type|Blvd|tiger
19351170|name_type|Blvd|tiger
19351284|name_type|Blvd|tiger
19351289|name_type|Blvd|tiger
19351916|name_type|Blvd|tiger
19351928|name_type|Blvd|tiger
19351936|name_type|Blvd|tiger
19351942|name_type|Blvd|tiger
```

Only dump rows! I skim through the rows but couldn't find any value named "Blvd", which is dirty data before I cleaned it, so my previous python code fixed things correctly.

2.2 What data tags contain

First thing I would like to test was the tags of each: nodes tags and ways tags, as I could not understand how they are organized. In order to see it, I decide to see the most frequently used keys and values, and possibly types, in nodes_tags table. In this case, in order to see how much nodes_tags table and ways_tags are different, I did not merge two tables for query.

```
SELECT count(id), key, value, type
FROM nodes_tags
GROUP BY key
ORDER BY count(id) DESC;
```

```
count|key|value|type
6813|source|wind|generator
6160|attribution|Office of Geographic and Environmental information (MassGIS)|regular
4638|name|Shell|regular
4466|power|generator|regular
```

I could see that tags in nodes are mostly used to describe what kind of building (facility) nodes are. Interestingly, most of them are used for power generator and MassGIS. On next, I will do the same procedure, but from ways_tags table, so that how much they are different.

```
count|key        |value|type
41776|building   |yes|regular
39782|highway    |service|regular
30872|county     |Kent, RI|tiger
30728|cfcc       |A41|tiger
30069|name       |Our Lady of Mercy Catholic Parish|regular
30002|reviewed   |no|tiger
25914|name_base  |Frenchtown|tiger
24494|name_type  |Rd|tiger
18506|zip_left   |02818|tiger
17282|zip_right  |02818|tiger
5991 |source     |massgis_import_v0.1_20071009101303|regular
4947 |upload_uuid|bulk_upload.pl-dd183b84-dae0-48c2-b387-c35f2e313537|tiger
4939 |tlid       |58923536|tiger
4235 |oneway     |yes|regular
```

Now I can see the difference more easily! However, most of data are hard to understand, suggesting need more data cleaning.

2.3 Who contributed the most

Next thing is, to see who contributed to the OSM the most! I got this idea from the SQL example. However, I wanted to approach from both nodes and ways.

```
SELECT count(user), user
FROM nodes
GROUP BY user
ORDER BY count(user) DESC;

COUNT|NAME OF USER
337837|woodpeck_fixbot
212313|greggerm
26848|Zirnch
17328|maxerickson
16771|John Wrenn
16333|ZeLonewolf
9693|morganwahl
9432|Roman Guy
9419|GeoStudent
8439|TIGERcnl
7229|jerryam
7036|jremillard-massgis
6787|42429
6094|Alex KG Ellis
5966|OMMB
5620|MassGIS Import
```

I can see that woodpeck_fixbot (looks like a bot) contributed the most in the state of Rhode Island. greggerm, the second most contributor, however, looks like a human. I was interested in, so I personally looked up the history of his edit, and I could find out that he is actually a human. It is amazing how much he has contributed for osm, compared to other users.

```
COUNT|NAME OF USER
37093|greggerm
18472|bot-mode
3328|DaveHansenTiger
2601|Zirnch
1759|maxerickson
1645|GeoStudent
1162|John Wrenn
1058|Roman Guy
1032|Alex KG Ellis
1004|jremillard-massgis
```

In nodes_tables, greggerm won the bot! He had contributed more than double of bot.

2.4 Postal Code and the City

Then I will compare the postal code and cities by counting how many of specific city/zip code are used, and matching those numbers.

```
SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags
UNION ALL
SELECT * FROM ways_tags) tags
WHERE tags.key like '%postcode%'
group by tags.value
order by count desc;
```

```
ZIP CODE|COUNT
02818|245
02912|113
02806|109
02920|51
02910|48
02906|47
02907|44
02919|31
```

```
SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags
UNION ALL
SELECT * FROM ways_tags) tags
WHERE tags.key like '%city%'
group by tags.value
order by count desc;
```

```
CITY|COUNT
East Greenwich|236
Providence|201
Barrington|108
Portsmouth|95
Cranston|72
Warwick|48
Pawtucket|33
Johnston|31
North Kingstown|30
Newport|28
```

According to the Google Maps, 02818 = East Greenwich, 02912 = Providence, 02806 = Barrington, 02920 = Cranston. Assuming one city may have more than one postal code (i.e. Warwick has 3 zip codes: 02818, 02886, 02887, 02888, 02889) I could know that they are mostly correct. To prove it, I will run the following code:

```
SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags
UNION ALL
SELECT * FROM ways_tags) tags
WHERE tags.key like '%postcode%' AND (tags.value LIKE '%02886%' OR tags.value LIKE '%02888%' OI
GROUP BY tags.value
ORDER BY count DESC;
```

```
ZIP CODE|COUNT
02818|245
02889|20
02886|10
02888|4
```

Assuming that Zip Code 02818 is a county mixed with the city of Warwick and East Greenwich, I can calculate the sum of counts of two citys (236+48=284) and the sume of counts of zip codes (245+20+10+4=279) I can see that they are mostly correct, with only 1.8% difference.

2.5 Timestamp

Next, I will see if there is any relationship between the timestamp and the number of contributes.

```
SELECT count(timestamp), timestamp
FROM nodes
GROUP BY timestamp
HAVING count(timestamp) > 90
ORDER BY count(timestamp) desc;
```

```
COUNT|TIME
93|2016-07-21T03:44:54Z
92|2016-07-22T20:04:24Z
92|2016-09-02T18:35:24Z
```

```
91|2016-07-13T19:48:23Z
91|2016-12-23T13:31:17Z
91|2016-12-24T17:28:43Z
91|2017-01-25T15:25:23Z
```

93 contributions in one second, or 92, 91. It doesn't make any sense that a human made it (unless it was really popular website like Facebook or Instagram), I can assume that a bot made those modification at that specific time.

2.6 Number of amenity

Next thing is to get the number of amenities that have the most in the state.

```
SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key like '%amenity%'
GROUP BY value
ORDER BY num DESC
LIMIT 20;
```

```
AMENITY|COUNT
school|602
place_of_worship|506
grave_yard|407
restaurant|177
library|115
fire_station|112
parking|88
fast_food|78
bench|72
kindergarten|63
cafe|58
waste_basket|55
fuel|53
post_office|45
police|36
townhall|32
bank|22
pharmacy|17
bicycle_parking|16
social_facility|16
```

Lots of schools, place of worship, and grave yards are not really special, but the number of restaurants and fast food looks relevantly less than other states. This is only a guess, I need to look up same data from other state (or city as some cities are larger than Rhode Island) to prove it. But when I try only looking up the list of restaurants,

```
SELECT COUNT(*)
FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) tags
WHERE value like '%restaurant%';
```

347.

It suggests me there needs more data cleaning so that I could get more reliable statistics.

3 Suggestion

I could be able to improve by auditing more values, such as city names, zip code, type of amenity, and many others. By doing so, I will be able to get much more accurate and reliable statistics. I would suggest that, if OSM has such algorithm (not necessarily python, any language is fine) so that they filtered the data before a user or a bot inserts any, the data in OSM would be much better. However, I can see some anticipated issues as well. Every data in OSM is made by human, and human always makes mistake. If they make a mistake on the name of street (not the type of street, but the name of street itself!) no one will be able to catch it unless living in that area, so does my python or any other's code for data cleaning. One other thing that concerns is, there are so many exceptions on the actual street name. Avenue A, for example, is a legit name, but it gave me an error. There will be more than more exceptions, and it will require a lot more effort to clean them up.

4 Conclusion

Although I did some of data cleaning, the data overview and statistics above show that there still needs more data and data cleaning, especially in the key, value pair of tags where the contributor has no limit to write anything. However, I still can say that my data cleaning has been helpful for getting more accurate statistics.