Data Wrangle Open Street Map

Data Wrangling with MongoDB

Map Area: San Francisco, CA, United States

Location: http://www.openstreetmap.org/relation/111968

https://s3.amazonaws.com/metro-extracts.mapzen.com/san-francisco_california.osm.bz2

1. Problems Encountered in the Map

After downloading the San Francisco City area data, I first used the provisional code (`takesample.py`) to create a smaller sample of the `osm` file `sample.osm`. The purpose of taking the sample is to reduce the data exploring time because the original San Francisco City data set is relatively large. Therefore, in this section, all the analysis with regard to exploring the data problem is based on the `sample.osm`.

By running it against a provisional `data.py` file, `mapparser.py`, `audittagtype.py` and `auditstreettype.py` and I noticed several issues with the data, which will be discussed in the following sections:

Non-uniformed address formats in tag field

Similar as the class excise, I first run the `mapparser.py` on the `sample.osm` to count occurrences of each tag and the result is show as following

```
'member': 3514,
'nd': 370824,
'node': 301055,
'osm': 1,
'relation': 328,
'tag': 121079,
'way': 32389
```

To investigate the tag properties, I add additional functionality to `mapparser.py`. The top 30 most common key values (the `k` attribute for each tag element) is showing as following:

```
('building', 22906),
 ('highway', 8884),
 ('name', 6617),
 ('source', 5980),
 ('addr:housenumber', 4696),
 ('addr:city', 4323),
 ('tiger:county', 4079),
 ('addr:street', 4007),
 ('tiger:cfcc', 3989),
 ('tiger:name_base', 3792),
('tiger:name_type', 3563),
('tiger:zip_left', 3223),
 ('tiger:zip_right', 3080),
 ('tiger:reviewed', 2666),
 ('created_by', 2243),
 ('tiger:tlid', 2039),
 ('tiger:source', 2016),
 ('tiger:separated', 1950),
 ('redwood_city_ca:bld_gid', 1865),
 ('redwood_city_ca:addr_id', 1714)),
('amenity', 1690),
 ('oneway', 1433),
 ('service', 994),
 ('addr:postcode', 889),
 ('foot', 604),
 ('bicycle', 603),
 ('leisure', 580),
 ('maxspeed', 553),
 ('massgis:cat', 453),
 ('ref', 425)
```

Surprisingly, the top key values are building and highway instead of addresses. Also, it seems there are multiple different formats to mark addresses. For example, address with the format "`tiger:xxx` (Topologically Integrated Geographic Encoding and Referencing system (TIGER) data format), address with the format "`addr:xxx`", address with USGS Geographic Names Information System (GNIS) data format (i.e., `amenity=library`,) and address with "`address`" as the key value. Also, some of the address formats may be human generated, i.e., '`redwood_city_ca:bld_gid`.

Street address may not be complete

By further checking the key value pairs of some address format, we found that some of the addresses are not complete. For example, following addresses may appear which indicates the first one miss the house number. For consistency, an empty value for the house number will be assigned to uniform the address format.

```
<tag k="address" v="North Lincoln Avenue"/>
<tag k="address" v="135 North Lincoln Avenue"/>
```

Inconsistency with street name abbreviations

First, I ran the `auditstreettype.py` provided from the lesson 3 and the results are very similar to the Chicago data in the class: the street names are not very consistent with different abbreviations. Also, several typos are observed from the full street name, i.e., `Boulavard(Boulevard), steet(street)`.

```
#A: 1
730: 1
Abenue: 1
Alameda: 39
Alley: 2
Ave: 8
avenue: 1
Avenue: 1309
Blvd: 3
Blvd.: 1
Boulavard: 1
Boulevard: 86
Ct: 1
Ctr: 1
D: 1
Drive: 201
Embarcadero: 1
Gardens: 4
Geary: 1
H: 2
Highway: 4
I-580: 1
Lane: 78
Las: 1
Plaza: 4
Plz: 5
Polk: 1
Pulgas: 1
Rd: 1
Real: 48
Road: 290
Square: 1
St: 15
St.: 1
Steet: 1
Street: 1520
```

Bridgeway: 1	Loop: 1	Terrace: 9
Broadway: 35	Mason: 2	Via: 1
Circle: 16	North: 1	Walk: 1
Clement: 1	Ora: 2	Way: 163
Clemente: 1	Park: 1	West: 1
Court: 84	Parkway: 3	Wharf: 1
	Path: 2	
	Place: 44	

For consistency, I translated all abbreviations into the full forms, e.g. `Blvd to Boulavard` and all "." characters were removed. The gold standard document that I used here is the USPS Street Suffixes file (USPS Street Suffixes.xls) which is downed from(http://mydatamaster.com/wp-content/files/streetsuffix.zip), imported and parsed from an excel file. Additional misspellings were added to the mapping table in `readaddresssuffixes.py`.

Inconsistency with city name format and unexpected city counts

By setting `attrib['k']` values to `"addr:city"` and made some corresponding changes on the `audittag.py`, below are the values the city names and they show following issues:

Alameda: 10	Mill Valley: 1	SAN CARLOS: 1
Albany: 9	Moss Beach: 1	San Carlos: 3
Belmont: 1	Newark: 1	San Francicsco: 41
Berkeley: 539	oakland: 1	San Francisco: 453
Burlingame: 18	Oakland: 107	san Francisco: 1
Castro Valley: 2	Oakland, Ca: 1	San Leandro: 6
Colma: 3	Oakland, CA: 1	San Mateo: 8
Daly City: 5	Orinda: 1	San Pablo: 1
East Palo Alto: 2	Pacifica: 12	Sausalito: 4
El Cerrito: 9	Palo Alto: 165	Union City: 26
Emeryville: 2	Piedmont: 363	Walnut Creek: 22
Fremont: 1	Point Richmond: 1	Woodside: 1
Half Moon Bay: 3	Redwood City: 2352	
Hayward: 4	Richmond: 140	
Menlo Park: 1		

1. Name with some redundant info, i.e., Oakland, Ca, Oakland, CA. Here the state information is also included in the city name field.

2. Upper case and lower case are mixed up for the city name, i.e., oakland and Oakland.

3. Mis-spell the city name, i.e., San Francicsco

4. It includes nearby cities, i.e., Half Moon Bay, Fremont, and surprisingly, the top city in the list is Redwood City instead of San Francisco. Basically it means the metro data not only includes the San Francisco city area but also the surrounding cities in the San Francisco Bay area. When I download the file, I noticed that in the OpenStreet map data set, there is another dataset called "San Francisco Bay Area", it may be an interesting topic to do a comparison these two data set and see the difference. As it

will be out of scope of this project, I will not discuss this topic here. I think it may be a good idea to rename the data to “San Francisco Metropolitan Area”.

For consistency, I uniform the city name by removing the redundant info, uniform the upper case and lower case issue with only capitalizing the first letter and correcting the misspelling issues. The detail is in `uniformcityname.py`.

Please note, same results can also be obtained by running the following command on mongodb with the imported jason file to the database in which `sample` is the collection.

```
> db.sample.aggregate([{"$match":{"address.city":{"$exists":1}}}, {"$group":{"_id":"$address.city", "count":{"$sum":1}}}])
```

Multiple Zip codes and inconsistent formats

After reviewing the jason file pulled from the original data.py file, the zipcode is normally exist with the following tags: ‘`addr:postcode', 'tiger:zip_left', 'tiger:zip_left_*, 'tiger:zip_right', and 'tiger:zip_right_*'`, where * is a number.

Considering multiple Zip codes are presented in the data under various permutations of `tiger:zip_left` and `tiger:zip_right`, I uniform all zipcodes from all sources into a single set, and populate this into `zipcodes`, which is under the `address` field.

By setting `attrib['k']` values appropriate values on the `audittag.py`, I found there are multiple formats we found in the `zipcode` field.

1. Main 5-digit Zip codes, i.e., 94002.
2. Main 5-digit Zip codes with 4-digit extension, i.e., 94025-1246.
3. Multiple Zip codes with/without redundancy may be included in the attribute value, i.e., 94112; 94127; 94127.

For consistency, a valid 5-digit Zip codes are checked and formatted before assigning to the zipcodes field. Specifically, I striped all leading and trailing characters before and after the main 5-digit zip code, which dropped all leading state characters (as in “CA 94541”) and 4-digit zip code extensions following a hyphen and the detail is in uniformzipcode.py.

2. Data Overview

This section contains basic statistics about the dataset and the MongoDB queries used to gather them. Please note, the data set used in this section is based on the data preprocessing mentioned in the previous section with the full data set (`san-francisco_california.osm`).

# File sizes

```
san-francisco_california.osm ......... 648 MB
san-francisco_california.osm.json .... 938 MB
```

# Number of documents

```
> db.sf.find().count()
          3334437
```

# Number of nodes

```
> db.sf.find({"type":"node"}).count()
       3010456
```

# Number of ways

```
> db.sf.find({"type":"way"}).count()
      323870
```

# Number of unique users

```
> db.sf.distinct("created.user").length
       1902
```

# Top 1 contributing user

```
> db.sf.aggregate([{"$group":{"_id":"$created.user", "count":{"$sum":1}}},
{"$sort":{"count":-1}}, {"$limit": 1}])
{ "_id" : "ediyes", "count" : 729862}
```

# Number of users appearing only once (having 1 post)

```
> db.sf.aggregate([{"$group":{"_id":"$created.user",
"count":{"$sum":1}}},{"$group":{"_id":"$count",
"num_users":{"$sum":1}}},{"$sort":{"_id":1}}, {"$limit":1}])
{ "_id" : 1, "num_users" : 453}
```

# most popular Zip codes in San Francisco:

```
> db.sf.aggregate([{"$match": {"address.zipcodes": {"$exists" :1}}},
...                      {"$unwind": "$address.zipcodes"},
...                      {"$group": {"_id": "$address.zipcodes", "count": {"$sum":1}}},
...                      {"$sort": {"count": -1}},
...                      {"$limit": 1}
...                      ])
{ "_id" : "94549", "count" : 722}
```

\#Most common building types/entries:

```
>db.sf.aggregate([{"$match": {"building": {"$exists" :1}}},
...                      {"$group": {"_id": "$building", "count": {"$sum":1}}},
...                      {"$sort": {"count": -1}},
...                      {"$limit": 5}
...                      ])
{ "_id" : "yes", "count" : 212816 }
{ "_id" : "residential", "count" : 6194 }
{ "_id" : "house", "count" : 4341 }
{ "_id" : "apartments", "count" : 836 }
{ "_id" : "retail", "count" : 732 }
```

Apparently, the majority of the buildings here are only marked as building (yes, or no) without the building type.

# Find the 10 restaurants with the phone number (if exists) nearby Union Square (within the circle centered on 37.7574306,-122.5083055 and with a radius of 10)

```
> db.sf.find({"pos":{"$geoWithin": {"$center": [[37.7574306,-122.5083055], 10]}},
...                "amenity":"restaurant"},
...               {"_id":0, "name":1, "phone" :1}
...               ).limit(10).pretty()
{ "name" : "Park Chalet Garden Restaurant" }
{ "name" : "Beach Chalet" }
{ "name" : "Thanh Long" }
{ "name" : "Cajun Pacific" }
{ "name" : "Celia's Mexican" }
{ "name" : "Thai Cottage Restaurant" }
{ "name" : "Outerlands", "phone" : "415-661-6140" }
{ "name" : "Wo's Restaurant" }
{ "name" : "Golden Gate Pizza & Indian" }
{ "name" : "Judahlicious" }{ "name" : "Pacific Cafe", "phone" : "415 387-7091" }
{ "name" : "Pagan", "phone" : "415 751 2598" }
```

# Most common street

```
db.sf.aggregate([{"$match": {"address.street": {"$exists": 1}}},
...                     {"$group": {"_id": "$address.street", "count" : {"$sum":1}}},
...                     {"$sort": {"count": -1}},
...                     {"$limit": 1}
...                     ])
{ "_id" : "Church street", "count" : 334 }
```

# Top religions

```
> db.sf.aggregate([{"$match":{"amenity":{"$exists":1}, "amenity":"place_of_worship"}},
...                    {"$group":{"_id":"$religion", "count":{"$sum":1}}},
...                    {"$sort":{"count":-1}},{"$limit":3}])
{ "_id" : "christian", "count" : 992 }
{ "_id" : null, "count" : 34 }
{ "_id" : "buddhist", "count" : 25 }
```

# Most popular cuisines

```
> db.sf.aggregate([{"$match":{"amenity":{"$exists":1}, "amenity":"restaurant",
"cuisine":{"$exists":1}}},
...                     {"$group":{"_id":"$cuisine", "count":{"$sum":1}}},
...                     {"$sort":{"count":-1}}, {"$limit":2}])
{ "_id" : "mexican", "count" : 184 }
{ "_id" : "pizza", "count" : 129 }
```

3. Other ideas about the datasets and Conclusion

#Total number of nodes without/with address info:

```
>db.sf.aggregate([{"$match" : {"type": "node", "address": {"$exists" :0}}},
...                         {"$group": {"_id": "totalnodewithoutaddress", "count":
{"$sum":1}}}
...                         ])
{ "_id" : "totalnodewithoutaddress", "count" : 2995738 }
```

```
db.sf.aggregate([{"$match" : {"type": "node", "address": {"$exists" :1}}},
...                         {"$group": {"_id": "totalnodewithaddress", "count":
{"$sum":1}}}
...                         ])
{ "_id" : "totalnodewithaddress", "count" : 14718 }
```

By conducting the above query, it clearly shows that we are running serious data completeness issue as the majority nodes (99.5%) do not include addresses. An interesting exercise would be to compress this data by removing these nodes which could potentially reduce the database size significantly. One implementation could be simply drop these fields from the database. However, it is really application based decision as we may lose a lot of useful information. Another thought would be correlate with other data sources to complete the dataset. For example, one simple way is to apply the same framework on the San Francisco bay area data set and try to correlate the incomplete nodes for missing info. We can also correlate specific node, i.e., restaurant, hospital and any point of interest, with Google Maps API or Yahoo Map API to obtain the address info. Furthermore, we can also crow this information on online websites like Google, Yelp or even social networking websites. The new information can be maintained in a separated database and further analysis will be done by correlating these two datasets with more advanced analysis tools or platforms. It essentially will become a big data problem. The same methodology can also be used to verify the existing dataset.

# Top 5 popular cities in node places:

```
> db.sf.aggregate([{"$match": {"address.city": {"$exists" :1}, "type": "node",}},
... ...                        {"$group": {"_id": "$address.city", "count": {"$sum":1}}},
... ...                        {"$sort": {"count": -1}},
... ...                        {"$limit": 5}
... ...                        ])
{ "_id" : "San francisco", "count" : 3533 }
{ "_id" : "Berkeley", "count" : 3471 }
{ "_id" : "Redwood city", "count" : 3092 }
{ "_id" : "Richmond", "count" : 1325 }
{ "_id" : "Oakland", "count" : 274 }
```

# Top 5 popular cities in way field:

```
> db.sf.aggregate([{"$match": {"address.city": {"$exists" :1}, "type": "way",}},
... ...                        {"$group": {"_id": "$address.city", "count": {"$sum":1}}},
... ...                        {"$sort": {"count": -1}},
... ...                        {"$limit": 5}
... ...                        ])
{ "_id" : "Redwood city", "count" : 20468 }
{ "_id" : "Piedmont", "count" : 3806 }
{ "_id" : "Berkeley", "count" : 2022 }
{ "_id" : "Palo alto", "count" : 1558 }
{ "_id" : "San francisco", "count" : 1014 }
```

By running the above queries, surprisingly, the top city with node places field is San Francisco while the top city in field is Redwood City and San Francisco is only ranked on top 5 in the list. An interesting study will be to get the unique list of the road list from the `way` field and perform a comparison with the one from the address field of the `node` field. For this purpose, I would like to revise my code in data.py and using the similar methodology to deal with the `way` field. Then I can extract the unique street name list from both `node` and `way` field and a run the comparison. During this period, I expect there will be more data cleaning and validation to explore and many loops of the cleaning, data loading and verification need to be performed.

In general, after this review of the data it's obvious that the San Francisco area data is incomplete. An interesting topic would be to do some cross-validation work between the San Francisco data set and the San Francisco bay area dataset from the Open Street Map. In addition, cross validation of Open Street Map data with Google Maps API or other Map API data will be another interesting topic.