Project 3: Data Wrangle OpenStreetMaps Data

Ritu Venkatesh

Map Area: Portland, OR, United States

https://www.openstreetmap.org/export#map=13/45.5118/-122.7928

Section 1: Problems Encountered in the Map:

I picked the data for NW Portland and some part of SW Beaverton – an area where I have lived and worked in the past. I found that the dataset was mostly clean with a few inconsistencies which I found by running the audit_street_map.py program with the expected variable set to []:

- The Street Types were at times abbreviated and at other times spelt out fully - Ave, Hwy, Rd, St were some of the abbreviations used.
- I also noticed that some addresses did not end in 'Street' like they should – for example SW Malcolm GLN should have been Southwest Malcolm Glen Street, SW Blanton should have been Southwest Blanton Street.
- The street direction were also inconsistent – sometimes SW and NW were used, whereas at other times they were fully spelt out as Southwest and Northwest

I used the update_name function to update the abbreviated street names and directions before they were written out to the JSON file. (see the data.py file attached with this submission for the full code). A query on the DB showed that all the names had been updated as expected.

I then imported the JSON file into MongoDB on my local machine using the following command:

```
C:\Udacity\Nanodegree\Data Wrangling in MongoDB\Project>mongoimport /db streetma
pdb /c pdx /file "NWPortland.osm.json"

2015-08-25T15:09:57.523-0700    connected to: localhost
2015-08-25T15:10:00.518-0700    [###.....................] streetmapdb.pdx
21.0 MB/143.2 MB (14.7%)
2015-08-25T15:10:03.518-0700    [########................] streetmapdb.pdx
50.2 MB/143.2 MB (35.1%)
2015-08-25T15:10:06.532-0700    [##############..........] streetmapdb.pdx
81.8 MB/143.2 MB (57.2%)
2015-08-25T15:10:09.520-0700    [###################.....] streetmapdb.pdx
112.8 MB/143.2 MB (78.8%)
2015-08-25T15:10:12.453-0700    imported 474227 documents
```

Section 2: Data Overview

File sizes:

NWPortland.osm……101, 382 KB (99 MB)

NWPortland.osm.json….146, 587 KB (143 MB)

Number of documents:

```
> db.pdx.find().count()
```

474227

Number of Nodes:

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

424146

Number of Ways:

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

50081

Number of unique users

```
> len(db.pdx.distinct("created.user"))
```

189

Top contributing user

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

{_id: Peter Dobratz_pdxbuildings, count: 245010}

Number of users appearing only once (having 1 post)

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

{ "_id" : 1, "count_users" : 48 }

Section 3: Additional Ideas

Contributor Statistics

While it might appear that the top contributor (Peter Dobratz_pdxbuildings) made a little more than 51% of the contribution (51.67% actually), on looking at the list of contributors, I found that Peter Dobratz had in fact contributed under 2 ids – 'Peter Dobratz_pdxbuildings' and 'Peter Dobratz'. The sum of his contributions is hence much higher:

```
> db.pdx.aggregate([{"$match": {"created.user": {"$regex": "Peter Dobratz"}}}, {
                                "$group": {"_id": "$created.user", "count": {"$sum":1}}},
                                {"$sort": {"count": -1}}])
{ "_id" : "Peter Dobratz_pdxbuildings", "count" : 245010 }
{ "_id" : "Peter Dobratz", "count" : 30528 }


> db.pdx.find({"created.user": {"$regex": "Peter Dobratz"}}).count()
275538
```

Which is 58.10% of the total contribution. In addition, I noted that less than 10% of Peter Dobratz's entries had a postal code:

```
> db.pdx.find({"created.user": {"$regex": "Peter Dobratz"}, "address.postcode": {"$exists": 1}}).count()
22653
> db.pdx.find({"created.user": {"$regex": "Peter Dobratz"}, "addr:postcode": {"$exists": 1}}).count()
0
> db.pdx.find({"created.user": {"$regex": "Peter Dobratz"}, "addr:postcode:left" : {"$exists": 1}}).count()
0
> db.pdx.find({"created.user": {"$regex": "Peter Dobratz"}, "addr:postcode:right": {"$exists": 1}}).count()
0
```

The number of entries created by Peter Dobratz by postal code are below:

```
>  db.pdx.aggregate([{"$match": {"created.user": {"$regex": "Peter Dobratz"},
                                "address.postcode":   {"$exists": 1}}},
                                {"$group": {"_id": "$address.postcode",  "count": {"$sum":1}}}])
{ "_id" : "97298", "count" : 1 }
{ "_id" : "97201", "count" : 17 }
{ "_id" : "97005-2992", "count" : 1 }
{ "_id" : "97003", "count" : 1788 }
{ "_id" : "97078", "count" : 171 }
{ "_id" : "97007", "count" : 3720 }
{ "_id" : "97006", "count" : 3778 }
{ "_id" : "97229", "count" : 1101 }
{ "_id" : "97225", "count" : 5025 }
{ "_id" : "97221", "count" : 1187 }
{ "_id" : "97005", "count" : 4920 }
{ "_id" : "97239", "count" : 890 }
{ "_id" : "97124", "count" : 54 }
```

Location Based Queries

I created a 2d Geospatial index on the “pos” variable in the collection as follows:

```
>db.pdx.ensureIndex({pos: ‘2d’})
{
      "createdCollectionAutomatically" : false,
      "numIndexesBefore" : 1,
      "numIndexesAfter" : 2,
      "ok" : 1
}
```

Then I queried the number of places to eat near the location [45.5342611,-122.8476254] which is an exit ramp to Bethany Boulevard and Cornell Road:

```
> db.pdx.find({"pos":{"$near": [45.5342611,-122.8476254] },
                              "amenity": {"$regex": "fast_food|pub|restaurant"}},
                    {"_id":0, "amenity": 1, "name": 1})
{ "amenity" : "restaurant", "name" : "Taste of Sichuan" }
{ "amenity" : "fast_food", "name" : "Burger King" }
{ "amenity" : "pub", "name" : "Golden Valley Brewery" }
{ "amenity" : "restaurant", "name" : "Papa's Pizza" }
{ "amenity" : "restaurant", "name" : "Sushi Town" }
{ "amenity" : "restaurant", "name" : "Jimmy John's" }
{ "amenity" : "restaurant", "name" : "Lentil Garden" }
{ "amenity" : "fast_food", "name" : "Pizza Schmizza" }
{ "amenity" : "fast_food", "name" : "Baskin Robbins" }
{ "amenity" : "fast_food", "name" : "Subway" }
{ "amenity" : "pub", "name" : "Oak Hills Brewpub" }
{ "amenity" : "fast_food", "name" : "Burger King" }
{ "amenity" : "fast_food", "name" : "Jack in the Box" }
{ "amenity" : "fast_food", "name" : "Carl's Junior" }
{ "amenity" : "restaurant", "name" : "Applebee's" }
{ "amenity" : "restaurant", "name" : "Newport Bay Seafood Broiler" }
{ "amenity" : "fast_food", "name" : "Panda Express" }
{ "amenity" : "fast_food", "name" : "Taco Bell" }
….
```

There were a total of 90 locations found near the coordinates provided:

```
> db.pdx.find({"pos":{"$near": [45.5342611,-122.8476254] },
                              "amenity": {"$regex: "fast_food|pub|restaurant"}}) .count()
90
```

Here is the breakdown of places by amenities (see the query.py file in the list of submitted files for the code).

```
drinking_water: 26, public_building: 1, fitness_center: 1, recycling: 7, car_wash: 1, telephone: 6,
 library: 5, clinic: 1, college: 2, parking: 30, post_office: 7, cafe: 28, toilets: 10, police: 1, clock: 2,
hospital: 4, veterinary: 4, bench: 41, kindergarten: 3, shelter: 6, fountain: 2, fuel: 22, pharmacy: 2,
 bicycle_parking: 22, fast_food: 30, fire_station: 6, post_box: 34, pub: 7, waste_basket: 6, dentist: 11,
doctors: 6, bank: 10, place_of_worship: 47, school: 14, bar: 2, restaurant: 52, parking_entrance: 1,
atm: 2, swimming_pool: 1, vending_machine: 2
```

I also tried to limit the maxDistance for the location search. The result wasn't too different:

```
> db.pdx.find({"pos":{"$near": [45.5342611,-122.8476254], "$maxDistance": .10 },
          "amenity": {"$regex": "fast_food|pub|restaurant"}},
          {"_id":0, "amenity": 1, "name": 1}).count()
85
```

Other Queries

Number of houses mapped on NW Caitlin Terrace:

```
> db.pdx.find({"address.street": {"$regex": "Caitlin Terrace"},
                                    "address.housenumber": {"$exists": 1}}).count()
27
```

Number of Libraries in the map area chosen:

```
> db.pdx.find({"amenity": "library", "name": {"$regex": "[Ll]ibrary"}}).count()

5
```

Top 10 amenities:

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

{ "_id" : "parking", "count" : 236 }
{ "_id" : "restaurant", "count" : 118 }
{ "_id" : "fast_food", "count" : 81 }
{ "_id" : "school", "count" : 69 }
{ "_id" : "place_of_worship", "count" : 68 }
{ "_id" : "cafe", "count" : 49 }
{ "_id" : "bench", "count" : 41 }
{ "_id" : "fuel", "count" : 39 }
{ "_id" : "bank", "count" : 37 }
{ "_id" : "post_box", "count" : 34 }
```

Most popular cuisine:

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

{ "_id" : null, "count" : 25 }
```

OOPs it looks like not all restaurants have the cuisine listed. Try again:

```
> db.pdx.aggregate([{"$match": {"amenity": {"$exists": 1}, "amenity": "restaurant",
                                "cuisine": {"$exists":1}}},
                    {"$group": {"_id": "$cuisine", "count": {"$sum": 1}}},
                    {"$sort": {"count": -1}}, {"$limit":5}])

{ "_id" : "pizza", "count" : 13 }
{ "_id" : "italian", "count" : 10 }
{ "_id" : "mexican", "count" : 9 }
{ "_id" : "chinese", "count" : 7 }
{ "_id" : "asian", "count" : 6 }
```

Conclusion

After reviewing the data, it would appear that the dataset is quite clean albeit not complete. There are several nodes that have location coordinates but no name or any other data to identify the location. Also, some of the restaurants don't have a cuisine listed.

One option to get this dataset to be complete is to automate the creation of missing information by cross-referencing using the Yelp API. Yelp has a vast DB of businesses and restaurants which could be tapped. Yelp's search API offers the ability to search by location parameters which we have in the OpenStreetMap dataset as well. So for the data that is missing a name or a cuisine for example, we can use Yelp's API to search by location and then use Yelp's Business API to fill in the missing information. Yelp's Business API contains the name of the business as well as a categories property, which, in case of a restaurant, could provide information about cuisine.

There are other ways of filling the cuisine for a restaurant as well. For someone who is familiar with the area, they could adopt a similar approach as we did for the street name update and create a list of restaurants and cuisines and then run through the dataset to either validate or insert the value for cuisine as needed. This is approach however, is more hands-on and requires someone who is intimately knowledgeable about the area.