P3- Data Wrangling with Mongo DB

(Open Street Map)

Map Area: New York City, United States of America
http://www.openstreetmap.org/relation/175905#map=10/40.6978/-73.9792

1. Problems encountered in the map

After downloading a custom extract of the NY City area and running a sample data set against my process_data.py file, I noticed the following problems with the data:

1.1 Problem in the extract

While obtaining the smaller sample, there was some problem in the dataset, due to which I could not parse the extract to obtain a smaller data set. On investigating the source of error, I found invalid characters like below were present in the custom extract:

```
<tag k="&#2;" v="Blunauer"/>                    line no 1515617
```

Since it was creating issues in parsing the file, I had to manually remove such lines from the dataset.

1.2 Presence of Non ASCII characters in the data set:

While processing the data set, I found non ASCII characters/other language symbols found in the dataset like :

```
<tag k="name" v="慈濟大學"/>
<tag k="name:zh" v="慈濟大學"/>
<tag k="addr:city" v="花蓮縣"/>
<tag k="addr:full" v="97004花蓮縣花蓮市中央路三段701號"/>
<tag k="wikipedia" v="zh:慈濟大學"/>
<tag k="addr:street" v="中央路三段"/>
<tag k="addr:district" v="花蓮市"/>
</way>
```

Since it was not possible to have Chinese/any other language/non ASCII values for any of the key/value pair in the New York city dataset, I skipped the processing of such characters in my python function is_ascii in my process_data.py file:

```
def is_ascii(str):
        return all(ord(chars) < 128 for chars in str)
```

1.3 Over abbreviated street names

Analyzing data revealed street and city name abbreviations usage. I updated all substrings in problematic address strings, such that “W 56th St #1H” becomes “West 56th Street #1H”.

1.4 Incorrect postal codes

```
> db.nyc.aggregate([{"$match":{"address.postcode":{"$exists":1}}},
{"$group":{"_id":"$address.postcode", "count":{"$sum":1}}}, {"$sort":{"count":-1}}])
```

Here are the top 3 results with count:

```
{ "_id" : "11101", "count" : 1739 }
{ "_id" : "11211", "count" : 1723 }
{ "_id" : "11206", "count" : 1575 }
```

….

On analyzing the data I found that many of the postal codes contained alphabets and were more than 5 characters long which was against the normal zip code format of 5 digits from 0 to 9. So I cleaned the data for these invalid pin codes in my data set

1.5 Sort cities by count

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

Results for the corresponding cities

```
{ "_id" : "New York", "count" : 1997 }
{ "_id" : "?????????", "count" : 835 }
{ "_id" : "Imus", "count" : 792 }
{ "_id" : "Brooklyn", "count" : 707 }
……
```

On analyzing the data, I found certain cities were not in United States itself like Imus in in Philippines, Anderlecht in Belgium and Gundersheim in Germany

Second level tags whose first level tags were already encountered in the extract

```
<node changeset="24427052" id="2986320002" lat="40.6783625" lon="-73.9481562"
timestamp="2014-07-29T17:41:45Z" uid="1851008" user="YamaOfParadise" version="1">
        <tag k="name" v="Nostrand Avenue" />
        <tag k="train" v="yes" />
        <tag k="network" v="Long Island Rail Road" />
        <tag k="railway" v="stop" />
        <tag k="operator" v="Long Island Rail Road" />
        <tag k="public_transport" v="stop_position" />
        <tag k="railway:position" v="1.6" />
</node>
```

Here k="railway" was already encountered in the extract and then k="railway:position" was again found. I created a dictionary for railway in this case and put the corresponding key in it like this in the resulting json :

{"name": "Nostrand Avenue", "created": {"changeset": "24427052", "user": "YamaOfParadise", "version": "1", "uid": "1851008", "timestamp": "2014-07-29T17:41:45Z"}, "pos": [40.6783625, -73.9481562], "train": "yes", "public_transport": "stop_position", "operator": "Long Island Rail Road", "railway": {"position": "1.6", "railway": "stop"}, "type": "node", "id": "2986320002", "network": "Long Island Rail Road"}

Other second level tags

Street names in second level "k" tags pulled from Tiger GPS data and divided into segments, in the following format:

```
<tag k="tiger:cfcc" v="A63" />
<tag k="tiger:county" v="New York, NY" />
<tag k="tiger:reviewed" v="no" />
```

These have also been split in a second level dictionary like the address field.

2. Overview of the Data

This section contains basic statistics about the dataset, the MongoDB queries used to gather them, and some additional ideas about the data in context.

File sizes:

new_york_city.osm	258.84 MB
new_york_city.osm.json	287.37 MB

Number of records:

```
> db.nyc.find().count()
1316799
```

Size of database

```
> db.nyc.dataSize()
322945653
```

Number of node :

```
> db.nyc.find({"created.user"}).count()
1096579
```

Number of ways :

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

219988

Number of distinct users :

```
> db.nyc.distinct("created.user").length
11640
```

Top contributing user:

```
> db.nyc.aggregate([{"$group":{"_id":"$created.user","count":{"$sum":1}}},
        {"$sort":{"count":-1}},{"$limit":1}])
{ "_id" : "Rub21_nycbuildings", "count" : 411984 }
```

User having only one post

```
> db.nyc.aggregate([{"$group":{"_id":"$created.user","count":{"$sum":1}}},
{"$group":{"_id":"$count","user_count":{"$sum":1}}},{"$sort":{"_id":1}},{"$limit":1}])
{ "_id" : 1, "user_count" : 3716 }
```

Here "_id" is the number of posts

3. Additional ideas about the datasets

User contribution statistics

The contributions of users seems incredibly skewed, possibly due to automated versus manual map editing (*the word "bot" appears in some usernames*). Here are some user percentage statistics:

- Top user contribution percentage("Rub21_nycbuildings") : 31.28%
- Top 2 user contribution("Rub21_nycbuildings" and "smlevine") : 43.2%

On considering these contribution percentages, I feel if more "real users" are motivated to contribute to the Open Street Map data it might spur the creation of more efficient bots to improve the accuracy of the data set.

Additional Data exploration using Mongo DB

Top amenities

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

        { "_id" : "bicycle_parking", "count" : 3528 }
        { "_id" : "restaurant", "count" : 1562 }
        { "_id" : "parking", "count" : 1146 }
```

…

As observed in the result, the top amenities were bicycle parking, restaurants and parkings

Most popular capacity of bicycle parkings

```
> db.nyc.aggregate([{"$match":{"capacity":{"$exists": true,"$ne": null}, "amenity":
"bicycle_parking"}},{"$group": {"_id":"$capacity", "count":{"$sum":1}}},   {"$sort":{"count":-1}},
{"$limit": 3}])
```

```
{ "_id" : "2", "count" : 2131 }
{ "_id" : "5", "count" : 545 }
{ "_id" : "capacity", "count" : 275 }
```

Here the maximum capacity had been for 2.

Most popular cuisine in restaurants

```
> db.nyc.aggregate([{"$match":{"cuisine":{"$exists":true,"$ne": null},"amenity":  "restaurant"}},
{"$group":{"_id":"$cuisine", "count":{"$sum":1}}}, {"$sort":{"count":-1}},{"$limit":10}])
```

```
{ "_id" : "italian", "count" : 103 }
{ "_id" : "american", "count" : 76 }
{ "_id" : "mexican", "count" : 75 }
{ "_id" : "pizza", "count" : 75 }
….
```

These results show that “Italian” cuisine is the most popular cuisine in NYC.

Types of parking

```
> db.nyc.aggregate([{"$match":{"parking":{"$exists": true,"$ne": null},"amenity": "parking"}},
{"$group": {"_id":"$parking", "count":{"$sum":1}}}, {"$sort":{"count":-1}}])
```

```
{ "_id" : "surface", "count" : 389 }
{ "_id" : "underground", "count" : 40 }
{ "_id" : "multi-storey", "count" : 19 }
{ "_id" : "garage", "count" : 5 }
{ "_id" : "lane", "count" : 1 }
```

These results show that surface level parking is most common in NYC, followed by underground, multi storey, garage and lane

4. Conclusion

After this review of the data it’s obvious that the New York city area contained invalid data(due to presence of non – interpretable characters and other non ASCII characters), is incomplete and inconsistent at certain places (due to presence of other countries like China, Germany data present in New York City data), though I believe it has been well cleaned for the purposes of this exercise. It interests me to notice a fair amount of GPS data makes it into OpenStreetMap.org on account of users’ efforts, whether by scripting a map editing bot or otherwise. With a rough GPS data processor in place and working together with a more robust data processor similar to process_data.py. I think it would be possible to input a great amount of cleaned data to OpenStreetMap.org.