OpenStreetMap Project

Data Wrangling with MongoDB

James Gallagher

Map Area: Greater Vancouver Regional District (GVRD), BC, Canada

Map of City of Vancouver: https://www.openstreetmap.org/relation/1852574#map=12/49.2573/-123.1241

INTRODUCTION

I had chosen this particular area as it was a place in which one of my friends had been born in, with one of his personal books based there. I needed a larger size file so some of my original plans were not applicable so that I could conduct a better analysis.

The Greater Vancouver area of around 2.4 million inhabitants is the third most populous metropolitan area in the country and the most populous in Western Canada. Vancouver is one of the most ethnically and linguistically diverse cities in Canada; 52% of its residents have a first language other than English.

PROBLEMS ENCOUNTERED

Street name typos

The domain knowledge of the city really helped me here to easily spot typos. For example, ('ing George Hwy') might not sound odd to a foreigner for a street name, but I realized right away it was supposed to be "King" George and also the highway had actually recently been changed to a Boulevard. So I renamed this correctly to "King George Boulevard". Another typo I fixed was an extra leading space in the street name ('Beatty St'). Here are all the manual changes I made:

```
changes = { 'ing George Hwy.': 'King George Boulevard', 'W15th
            st': 'W 15th Street',
            'Howe St. Vancouver': 'Howe Street',
            'W. Hastings St. Vancouver': 'West Hastings Street', 'Expo
            Blvd, #3305': 'Expo Boulevard',
            ' Beatty St': 'Beatty Street'}
```

Skipped street names

For cases where it wasn't clear on what the correct street name should be, I decided to add these to a list and ignore them if I found a matching value. For example, Park is too vague and I wouldn't know what to rename it to.

```
skip = ["10","32500","99","Tsawwassen","Park","Terminal","8500"]
```

Street type abbreviations standardized

Other than correcting typos and skipping/ignoring a few street names, I also changed many similar street types to a non-abbreviated form after discovering these issues in the audit (before importing into MongoDB). For example, below are the 5 types of street which I standardized by adding them to the mapping dictionary in the audit file (4. audit.py).

```
'St': 'Street', 'St.': 'Street', 'Street3': 'Street', 'st': 'Street', 'street': 'Street'
```

Pre-existing type field

In one of my earlier MongoDB imports, I saw that the total number of ways and number of nodes did not equal the total number of documents. I investigated and found that there were actually a few other types. How could this be? I searched the original dataset and found that there was alreay a 'type' field in a very few number of elements and it was overriding my type setting code which I had near the start of my function. I adjusted my python cleaning code (5. data.py)) to move the type setting (to node or way) into later on in the code so that it woudln't be overriden.

Custom id instead of ObjectID

When I first imported the data into MongoDB, I realized the ObjectID field was redundant as each node or way already had a unique field in id. So I renamed id to "_id" and when I reimported the data MongoDB allowed me to override it.

Field names with colon (:)

Field names with double colons were caught by a regular expression I created and were not included in the final JSON file. However, the lower_colon regex did not match all fields with colons because a few fields had uppercase letters as well. Instead of writing a separate regular expression to catch this, I just handled it by exception and removed the first part of the field before the colon.

```
"geobase:acquisitionTechnique" : "GPS", --> CHANGED TO
"acquisitionTechnique" : "GPS",
```

OVERVIEW OF THE DATA

Retrieval Date and Time

Jan 30th, 2015 12:54 AM (Pacific)

Coordinate Box Bounds (the square area that I limited my data to)

Minimum Latitude: 49.0072 Maximum Latitude: 49.4431
Minimum Longitude: -123.3517 Maximum Longitude: -122.2037

Overpass API Query

```
(node(49.0072,-123.3517,49.4431,-122.2037);<;);out;
```

File Sizes

Initial XML File - 169 MB - GVRD - Vancouver - OSM XML Raw.osm

Final JSON File - 160 MB - GVRD - Vancouver - OSM.json

Bash

```
$ ls -lh          GVRD\ -\ Vancouver\ -\ OSM\ XML\ Raw.osm
-rw-r--r    --1 user                   Administ          169M Feb  1 03:08 GVRD - Vancouver - OSM XML Raw.osm
-----------------------------------------------------------------------------------------
$ ls -lh GVRD\ -\ Vancouver\ -\ OSM\ XML\ Raw.osm.json
-rw-r--r--            1 user          Administ          160M Feb  2 00:09 GVRD - Vancouver - OSM.json
```

MongoDB Queries

Number of Total Documents (nodes and ways)

```
> db.van.find().count()
1538273
```

Number of nodes

```
> db.van.find( { "type": "node" } ).count()
1365649
```

Number of ways

```
> db.van.find( { "type": "way" } ).count() 172624
```

Number of unique users/contributors (no actual user names provided)

```
> db.van.distinct( "created_by" ).length 21
```

Top Contributer

```
> db.van.aggregate( [{"$match" : {"created_by": {"$exists": 1}}},
                             {"$group": {"_id": "$created_by", "count": {"$sum":1}}},
                             {"$sort": {"count" : -1}},{"$limit":1}] ) { "_id" :
"JOSM", "count" : 4423 }
```

JOSM is actual an open source editor for OSM written in Java and not one single contributer per se.

Number of unique sources of data

```
> > db.van.distinct( "source" ).length 261
```

Top 5 sources of data

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

{ "_id" : "City of Surrey 2010 GIS Data", "count" : 80084 } { "_id" :
"Geobase_Import_2009", "count" : 27306 }
{ "_id" : "GeobaseNHN_Import_2009", "count" : 8850 } { "_id"
: "Bing", "count" : 7241 }
{ "_id" : "PGS", "count" : 5560 }
```

Additional MongoDB Queries

Top 10 Amenity Types

```
> db.van.aggregate( [{"$match": {"amenity": {"$exists": 1} } }, {"$group": {"_id ":"$amenity",
"count": {"$sum":1} } }, {"$sort": {"count":-1} },{"$limit": 5}]
)
{ "_id" : "parking", "count" : 2773 } { "_id" :
"restaurant", "count" : 937 } { "_id" : "school",
"count" : 632 }
{ "_id" : "fast_food", "count" : 487 } { "_id" :
"bench", "count" : 454 }
```

Top 5 Fast Food Restaurants

```
> db.van.aggregate( [{"$match": {"amenity": "fast_food" } }, {"$group": {"_id":"$name", "count": {"$sum

{ "_id" : "McDonald's", "count" : 59 } { "_id" :
"Subway", "count" : 58 }
{ "_id" : "Tim Hortons", "count" : 31 } { "_id" :
"Wendy's", "count" : 21 }
{ "_id" : "A&W", "count" : 18 }
```

Top 3 Coff ee Shops

```
> db.van.aggregate( [{"$match": {"amenity": {"$exists": 1}, "amenity" : "cafe"} }, {"$group":
{"_id":"$name", "count": {"$sum":1} } }, {"$sort": {"count":-1} } ,{"$limit": 3}] )

{ "_id" : "Starbucks", "count" : 94 }
{ "_id" : "Starbucks Coffee", "count" : 35 } { "_id" :
"Tim Hortons", "count" : 23 }
```

Total Number of Coff ee Shops/ Places in which serve coffee

```
> db.van.find({"amenity": "cafe"}).count() 426
```

About in 1 out every 3 coff ee shops is a Starbucks. No surprise there!

As I drive around town I notice some streets are named after one of the 10 Canadian Provinces.

```
> db.van.distinct("address.street", {"address.street" : {"$in":[/^Ontario/,/^Quebec/, /^British
Columbia/,/^Alberta/,/^Sasketchewan/,/^Manitoba/,/^Newfoundland and Labrador/, /^Nova
scotia/,/^New Brunswick/,/^Prince Edward Island/]}})
[ "Manitoba Street", "Quebec Street", "Alberta Road", "Ontario Street" ]
```

I also notice that a lot of streets, buildings, and landmarks are named after royalty. This should be expected as Canada was and is part of the British Commonwealth with the Queen as the official head of state.

```
> db.van.distinct("name", {"name": {"$in" : [ /^King\s/,/^Queen\s/,/^Prince\s/, /^Princess\s/]
} } )
[
            "King George", "King George Inn & Suites", "King Edward", "Queen
            Elizabeth Theatre", "King Sushi", "Queen Bee Flowers", "Queen
            Charlotte Channel", "Prince Charles Elementary", "Prince Chinese
            Seafood Restaurant", "King George Boulevard", "Prince Edward
            Street", "Queen Elizabeth Park",
            "Princess Street", "King Edward Street",
            "Princess Avenue", "Prince Albert Street",
            ...
```

```
> db.van.distinct("address.street", {"address.street": {"$in" : [ /^King\s/,
/^Queen\s/,/^Prince\s/,/^Princess\s/,/^Royal\s/ ] } } )
[
            "King George Boulevard", "Princess Drive", "Royal Crescent", "King
            Road", "Queen Mary Boulevard",
            "Prince Charles Boulevard", "Royal Oak Avenue",
            "Royal Avenue East", "Royal Avenue", "Princess Crescent", "King
            Street"
]
```

Looks like the correction I made from “ing George Highway” to “King George Boulevard” worked fine!

OTHER IDEAS ABOUT THE DATASET

A lot of these stats have to be taken with a grain of salt, because it seems a very small percentage of the data are populated with much data other than location info, id, and type which can be seen from visual inspection of the data. However, there is no built in function in MongoDB to easily see how much exactly this percentage is. So I decided to add a field in the document itself that stores the number of fields in the document originally with the field name “orig_numFields” (I did not include this field itself when doing the calculation). I realized that there could be a diff erence between nodes and ways so I ran seperate queries.

Nodes

Here is the result of the query to see the number of node documents with 3 fields:

```
> db.van.find({"type":"node","orig_numFields":3}).count()
1256905
```

The total number of node documents (from the earlier queries) is 1365649. So we can see that 92% of the node documents look like this:

```
> db.van.findOne({"type":"node","orig_numFields":3})
{
            "_id" : "24657643",
            "type" : "node", "pos" :
            [
                        49.0352951, -
                        122.2293237
            ],
            "orig_numFields" : 3
}
```

Nonetheless, it is still interesting to see some of insights from the limited node documents data which can be further investigated with more complete datasets

Ways

And for ways (each node has at least 3 fields, 2 is lowest number of fields - so no need to explicitly query ‘way’ type)

```
> db.van.find({"orig_numFields":2}).count() 2717
```

Compared with 172624 total ways, only 1.5% of ways look like this:

```
> db.van.findOne({"type":"way","orig_numFields":2})
{ "_id" : "23198532", "type" : "way", "orig_numFields" : 2 }
```

Additional Information

The data has been cleaned and organized well enough for the intents and purposes of this project. However, there are some improvements that I would suggest going forward with more in-depth analysis of the OSM data. Firstly, this project would have been optimized with additional general database implementations, in which are listed below.

An improvement I suggest is to streamline the processing of raw OSM data. Currently, the processing of the OSM data is static and requires many stages to be done by hand. It would be great if the structure was dynamic and the database generation could automatically be updated when updates were made to the OSM data. This would be beneficial for quickly responding to changes in the OSM data due to new developments or addressing changes.

The last suggestion I have is to improve the sanity checks within the cleaning code as well as the database structure. I am sure there are many spelling mistakes that still exist in the OSM data. A more thorough spell checker could be implemented to correct these issues. There could be some additional sanity checks on the addresses in the data as well. Each component of the addresses could be checked to make sure the value is within an expected bound. The benefit of adding these checks are that the data is cleaned more thoroughly, but the downside is the code would have to hold more verified data to implement the checks. The database structure could also catch redundancies and inconsistencies with the addresses. Each full address should be a unique location. With each of these checks, the code will have to contain more expected information about the particular area and therefore makes the code very location specific. The code wouldn't be scalable either, the checks would be invalid if the area data were expanded beyond the current bounds.

I think the analysis conducted worked very well for the scope of work of this project. It was very rewarding to go through the process of analyzing the OSM data and creating this report.

An issue in which I can forsee from the proposed implementations is that despite the fact it will increase ease of use, it may make it more difficult for external parties to interpret the data – and it may take quite a while to successfully implement the features in a way which will be optimal and allow for quick and easy data analysis. The use of the Google Maps API would help with the issue I first experienced when using another location, as it would allow for larger data samples to be taken, meaning a more accurate and detailed analysis can be conducted and would be a very simple implementation, requiring the addition of a bit of code and a Google API key.

Lastly, in the project I encountered an issue with data interpretation in relation to postcodes in which was solved simply after gathering a sample from a different region (the one used in this thesis).