# DAND Project 3: Wrangle OpenStreetMap Data

Author: CHOWY

Udacity Login: [chowy1026@gmail.com](mailto:chowy1026@gmail.com)

## Choose a Map Area

I have decided to work with Singapore. I live here and it will be fun to work with data that is relatable. Singapore OSM data is readily available at MapZen Metro Extracts. Being 302MB uncompressed, it also fulfills the OSM XML file size requirement (at least 50MB uncompressed).

## Processing DataSet

I used the python script provided to extract about 10 MB of singapore.osm data into sample\_singapore.osm file. Having a quick look at the sample\_singapore file, I soon realized that the key-value pairs of tags under node, way and relation elements vary a lot.

I decided to write a couple of python scripts: 1) get\_unique\_tag\_keys.py – to see what are the keys under node, way and relation; and 2) get\_unique\_kv.py, to build a dictionary of all k keys and its corresponding unique v values.

I saved the print output of those scripts into log/txt files. Attached are the last couple versions of them.

### Problems within Raw DataSet and Solutions

P1: Malaysian and Indonesian data mixed in dataset. It makes the dataset inevitably larger, and hard to format later.

S1: It makes sense to remove those non-SG data. The initial and obvious filters were first country and cities. Then it was the different formats of phone numbers, zipcode, and/or addresses etc. Some of the entities also had source by local officials or specific projects. These information, when available, were used to filter out the non-SG data. This is in remove\_non\_sg.py. This however isn’t enough to filter out all the non-SG data, I had to dig deeper to look into street names and sometimes entity names. This is leads to problem 2, P2.

P2: There is no obvious pattern to differentiate street names and entity names. Although the official languages are now different, Malaysia, Singapore and Indonesia shared similar historic background. Some names also appear in both in Singapore and other neighboring countries.

S2: Was looking for Google map API so I could verify the location of street names and business names. But couldn’t find one. I ended up checking ambiguous names manually with Google map on a browser. The list of names that don’t belong to SG are put in remove\_non\_sg.py.

P3: Key inconsistency (ie postcode vs postal\_code vs addr:postcode)

S3: Wrote script to merge those keys. The code is in fix\_osm.py.

P4: keys-values mismatched. Such as address unit entered as address street, address housenumber entered as address unit.

S4: Manually examine and develop a dictionary to correct them. RegEx can’t be used here too because the format of the values entered in aren’t consistent. The code is in fix\_osm.py

P5: There are a lot of typos and spelling errors, in addition to inconsistent abbreviation and capitalization.

S5: Again, I generated a list of errors and abbreviations that need to be corrected. The code is in fix\_osm.py

P6: Data entered in very inconsistent format. For example: some phone/fax numbers are entered without country code, some without.

S6: Programmatically write code to clean up the inconsistency. The code is in fix\_osm.py

P7: Too many keys and values to clean, manually. As I work on cleaning the data, I realize it is taking a long time (without sophisticated machine learning application).

S7: I therefore only focus on the data that I could. The tag keys that are relevant and generic to all nodes, ways and relations are narrowed down to:

* name (or alt\_name/name:en/alt\_name:en) whichever available
* user
* addr:housenumber, addr:housename, addr:unit, addr:street, addr:postcode, addr:city, addr:country
* contact:email, contact:fax, contact:phone, contact:website
* social:facebook, social:twitter, social:google\_plus, social:instagram
* amenity, building, building:use, landuse

Keys that are specific to <node>s

* 'religion', 'denomination','cuisine', 'highway', 'exit\_to'

Keys that are specific to <way>s

* 'highway', 'oneway', 'service', 'railway', 'bridge', 'place', 'natural', 'landuse', 'leisure', 'tourism', 'school'

Keys that are specific to <relation>s

* 'type', 'route', 'from', 'to'

#### Python Code and Files for Fixing / Cleaning Data and Convert Data to JSON

Written Python codes/files for this section are:

* get\_unique\_tag\_keys.py
* get\_unique\_kv.py
* remove\_non\_sg.py
* regex\_utility.py
* fix\_osm.py
* osm\_to\_json.py

I went through about 10 iterations of cleaning and fixing data. The final version of OSM before converted to JSON was singapore\_fixed\_10.osm.

As I reviewed again the final cleaned OSM file, I found that there are lots of nodes and ways that have don’t have a “name” field, nor any other human-readable identifier property. For curiosity and analysis sake, I generated two versions of JSON files:

* singapore\_fixed\_10.osm.json – one with only nodes and ways that have names/identifier (namely name:en, alt\_name or alt\_name:en)
* singapore\_fixed\_10.osm.\_all.json – one with all the nodes and ways, regardless they have a name field or not.

The singapore\_fixed\_10.osm file is slightly less than 290 MB, where as the JSON files are 433 MB (singapore\_fixed\_10.osm.\_all.json) and 19MB (singapore\_fixed\_10.osm.json) respectively.

## Import JSON to MongoDB

After starting the MongoDB, I imported the JSON files into ‘singapore’ db, but two separate collections, ‘sg’ and ‘sg\_all’.

Data with names

dand\_p3 $ mongoimport --db singapore --collection sg --file singapore\_fixed\_10.osm.json

All data

dand\_p3 $ mongoimport --db singapore --collection sg\_all --file singapore\_fixed\_10.\_all.json

## Querying Data

Data Analysis in MongoDB Shell Command:

> show dbs

examples   0.026GB

local      0.000GB

**singapore  0.125GB**

udacity    0.036GB

> use singapore

switched to db singapore

> show collections

sg

sg\_all

File Counts in each Collection:

> db.sg.find().count()

41231

> db.sg\_all.find().count()

1532810

Document Count of Node Type in each Collection:

> db.sg.find({'type':'node'}).count()

11330

> db.sg\_all.find({'type':'node'}).count()

1361676

Document Count of Way Type in each Collection:

> db.sg.find({'type':'way'}).count()

29894

> db.sg\_all.find({'type':'way'}).count()

171089

Number of Distinct Users in each Collection:

> db.sg.distinct('created.user').length

1163

> db.sg\_all.distinct('created.user').length

1593

User who contributed the most in each collection:

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

{ "\_id" : "JaLooNz", "count" : 15170 }

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

{ "\_id" : "JaLooNz", "count" : 368557 }

However, when comparing top 20 contributors from each collection, the results is rather contradicting and confusing. Other than ‘JaLooNz’, ‘cboothrovd’, ‘rene78’, ‘zomgvivian’ and ‘kingrollo’, the rest of the top contributors are different.

In the dataset that includes nodes/ways without `name`, `sg\_all` collection, I see user `berjaya` contributed sizeable amount of records/documents. `berjaya` is a Malay word. I suspect there are still a good amount of non-SG data in the `sg\_all` collection. Those might be un-identifiable due to the lack of `name` and `address` properties.

> db.sg.aggregate([{"$group":{"\_id":"$created.user", "count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":20}])

{ "\_id" : "JaLooNz", "count" : 15170 }

{ "\_id" : "zomgvivian", "count" : 2657 }

{ "\_id" : "cboothroyd", "count" : 1777 }

{ "\_id" : "Cort Wee", "count" : 1361 }

{ "\_id" : "rene78", "count" : 792 }

{ "\_id" : "CitymapperHQ", "count" : 742 }

{ "\_id" : "yourealwaysbe", "count" : 666 }

{ "\_id" : "danielkok", "count" : 642 }

{ "\_id" : "ck77", "count" : 625 }

{ "\_id" : "Paul McCormack", "count" : 601 }

{ "\_id" : "klcaaron", "count" : 527 }

{ "\_id" : "kingrollo", "count" : 500 }

{ "\_id" : "tpsmartcp", "count" : 478 }

{ "\_id" : "testerx1", "count" : 372 }

{ "\_id" : "Siewjy", "count" : 352 }

{ "\_id" : "hiddenAce", "count" : 334 }

{ "\_id" : "drewroud", "count" : 324 }

{ "\_id" : "Singeo", "count" : 323 }

{ "\_id" : "Duane Bong", "count" : 288 }

{ "\_id" : "jaredc", "count" : 279 }

> db.sg\_all.aggregate([{"$group":{"\_id":"$created.user", "count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":20}])

{ "\_id" : "JaLooNz", "count" : 368557 }

{ "\_id" : "berjaya", "count" : 103776 }

{ "\_id" : "rene78", "count" : 81406 }

{ "\_id" : "cboothroyd", "count" : 78127 }

{ "\_id" : "Luis36995", "count" : 41976 }

{ "\_id" : "ridixcr", "count" : 40616 }

{ "\_id" : "kingrollo", "count" : 40328 }

{ "\_id" : "calfarome", "count" : 34815 }

{ "\_id" : "lmum", "count" : 33847 }

{ "\_id" : "Sihabul Milah", "count" : 29144 }

{ "\_id" : "jaredc", "count" : 27536 }

{ "\_id" : "nikhilprabhakar", "count" : 26085 }

{ "\_id" : "Jothirnadh", "count" : 25061 }

{ "\_id" : "manings", "count" : 23527 }

{ "\_id" : "yurasi", "count" : 22507 }

{ "\_id" : "matx17", "count" : 21741 }

{ "\_id" : "zomgvivian", "count" : 20402 }

{ "\_id" : "poornibadrinath", "count" : 19470 }

{ "\_id" : "fusionstream", "count" : 19075 }

{ "\_id" : "singastreet", "count" : 18615 }

>

In `sg` collection, the top 10 contributors contributed 60% of the data; whereas in `sg\_all` collection, the top 10 contributor contributed 55.6% of the data. Both dataset were rather skewed.

The number of users who contributed only one post in each collection:

> db.sg.aggregate([{"$group":{"\_id":"$created.user", "count":{"$sum":1}}}, {"$group":{"\_id":"$count", "num\_users":{"$sum":1}}}, {"$sort":{"\_id":1}}, {"$limit":1}])

{ "\_id" : 1, "num\_users" : 446 }

> db.sg\_all.aggregate([{"$group":{"\_id":"$created.user", "count":{"$sum":1}}}, {"$group":{"\_id":"$count", "num\_users":{"$sum":1}}}, {"$sort":{"\_id":1}}, {"$limit":1}])

{ "\_id" : 1, "num\_users" : 419 }

Investigation on `amenity` property. It turns out the highest count amenity for `sg` collection is `restaurant`; while for `sg\_all` collection, it is `parking`. It explains why so many nodes doesn’t have `name` property. We don’t generally assign `name` to parkings.

My initial idea was that, only the nodes/ways with `name` is meaningful data. Apparently that would be a very wrong assumption. My approach would have excluded those parking data, or any meaningful structures that don’t have `name` property.

> db.sg.aggregate([{"$match":{"amenity":{"$exists":1}}}, {"$group":{"\_id":"$amenity", "count":{"$sum":1}}}, {"$sort":{"count": -1}}, {"$limit":20}])

{ "\_id" : "restaurant", "count" : 1180 }

{ "\_id" : "parking", "count" : 760 }

{ "\_id" : "school", "count" : 461 }

{ "\_id" : "cafe", "count" : 349 }

{ "\_id" : "fast\_food", "count" : 318 }

{ "\_id" : "place\_of\_worship", "count" : 314 }

{ "\_id" : "taxi", "count" : 259 }

{ "\_id" : "fuel", "count" : 192 }

{ "\_id" : "bank", "count" : 135 }

{ "\_id" : "police", "count" : 126 }

{ "\_id" : "food\_court", "count" : 108 }

{ "\_id" : "community\_centre", "count" : 106 }

{ "\_id" : "bar", "count" : 96 }

{ "\_id" : "pub", "count" : 55 }

{ "\_id" : "bus\_station", "count" : 53 }

{ "\_id" : "college", "count" : 47 }

{ "\_id" : "hospital", "count" : 46 }

{ "\_id" : "atm", "count" : 43 }

{ "\_id" : "library", "count" : 39 }

{ "\_id" : "shelter", "count" : 37 }

> db.sg\_all.aggregate([{"$match":{"amenity":{"$exists":1}}}, {"$group":{"\_id":"$amenity", "count":{"$sum":1}}}, {"$sort":{"count": -1}}, {"$limit":20}])

{ "\_id" : "parking", "count" : 1999 }

{ "\_id" : "restaurant", "count" : 1320 }

{ "\_id" : "school", "count" : 506 }

{ "\_id" : "cafe", "count" : 364 }

{ "\_id" : "place\_of\_worship", "count" : 358 }

{ "\_id" : "taxi", "count" : 334 }

{ "\_id" : "fast\_food", "count" : 333 }

{ "\_id" : "fuel", "count" : 267 }

{ "\_id" : "swimming\_pool", "count" : 264 }

{ "\_id" : "toilets", "count" : 226 }

{ "\_id" : "shelter", "count" : 202 }

{ "\_id" : "atm", "count" : 181 }

{ "\_id" : "bank", "count" : 150 }

{ "\_id" : "police", "count" : 138 }

{ "\_id" : "food\_court", "count" : 129 }

{ "\_id" : "bar", "count" : 107 }

{ "\_id" : "community\_centre", "count" : 107 }

{ "\_id" : "parking\_entrance", "count" : 76 }

{ "\_id" : "bench", "count" : 75 }

{ "\_id" : "bus\_station", "count" : 64 }

Food has always been a big part of Singapore culture. It is not surprising lots of the nodes are restaurants. I decided to query more on the restaurant data to see what’s the most popular cuisine.

Turns out, there is a great amount of restaurant nodes, 53.9% of `sg` collection and 58.2% of the `sg\_all` collection, don’t have cuisine stated. With those restaurant nodes that has cuisine property specified, we see the top 10 cuisines are the same for both `sg` and `sg\_all` collections.

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

{ "\_id" : null, "count" : 636 }

{ "\_id" : "Chinese", "count" : 112 }

{ "\_id" : "Japanese", "count" : 65 }

{ "\_id" : "Korean", "count" : 38 }

{ "\_id" : "Indian", "count" : 36 }

{ "\_id" : "Italian", "count" : 33 }

{ "\_id" : "Pizza", "count" : 27 }

{ "\_id" : "Asian", "count" : 26 }

{ "\_id" : "Regional", "count" : 20 }

{ "\_id" : "Thai", "count" : 17 }

>

> db.sg\_all.aggregate([{"$match": {"amenity": {"$exists":1}, "amenity":"restaurant"}}, {"$group": {"\_id": "$cuisine", "count":{"$sum":1}}}, {"$sort": {"count": -1}}, {"$limit": 10}])

{ "\_id" : null, "count" : 768 }

{ "\_id" : "Chinese", "count" : 112 }

{ "\_id" : "Japanese", "count" : 65 }

{ "\_id" : "Korean", "count" : 38 }

{ "\_id" : "Indian", "count" : 37 }

{ "\_id" : "Italian", "count" : 34 }

{ "\_id" : "Pizza", "count" : 27 }

{ "\_id" : "Asian", "count" : 26 }

{ "\_id" : "Regional", "count" : 25 }

{ "\_id" : "Thai", "count" : 17 }

## Suggestions on Improving DataSet

After going through this exercise, the biggest aspect that needs improvement is the quality of data. Due to the very similar cultural background with neighboring countries, there are a lot of invalid Singapore data mixed in the dataset. With data that’s relevant to Singapore, they are also not necessarily all consistent, complete nor in uniform format.

This is not uncommon for crowd-sourced data. To improve the quality, we could:

1. consult government websites of countries to determine the valid data format
2. add validation to the interface where users enter data.
3. build better forms to show previously defined options, instead of textboxes (which could be more error prone).

Undesired of the above suggestions could be:

* it costs more to create/maintain the forms/channels for data collection
* users might be discouraged by the rigidity of process, and in turn lead to lower contributions.