Data Wrangling OpenStreetMap Data of Singapore

By Erwin, February 2017

The purpose of this project is to audit, analyze and clean Singapore OpenStreetMap (OSM) XML. After the data is cleaned, it will be transformed into a JSON format to be uploaded into a MongoDB Database.

OpenStreetMap view of Singapore: https://www.openstreetmap.org/relation/536780

About Singapore: https://en.wikipedia.org/wiki/Singapore

Reason to analyse Singapore OSM: I have been staying here for study and work for more than 8 years ever since I graduated from my high school in Indonesia. The multicultural characteristics in Singapore also creates a very interesting content and challenges for this project.

1. Problems Encountered in the Data

Some problems were observed after analysing a smaller subset of the data:

- Data points belonging to **other countries** (from Malaysia and Indonesia)
- Unstandardized **street name** format, such as Jl, Jln
- Unstandardized **house numbers**
- Unstandardized **phone numbers**
- Invalid **postal codes** (Singapore postal codes should be 6 digits)
- Names in **other languages**, such as in Chinese, Malay, and Indian.

Data points belonging to other countries

Singapore is the world's only island city-state surrounded by much larger neighbouring nations. As such, it is not surprising that in the OSM data, there will be data points from neighbouring countries, particularly cities from Malaysia (e.g. Johor Bahru) and Indonesia (e.g.Batam).

In this project, we are only interested to analyse Singapore OSM data. Therefore, we exclude data points that do not belong to Singapore. In **inSingapore()** function, elements that do not belong to Singapore, will be ignored.

Unstandardized street name format

By analysing some of the street elements, I found out that there is some unstandardized format. This is due to:

- Abbreviation of some street elements, such as **Block → Blk**, **Avenue → Ave**.
- Influence of Malay language (one of Singapore's main mother tongues) and its abbreviation in the street elements, such as **Bukit → Bkt (means Hill)**, **Jalan → Jl, Jln (means street)**.

Unstandardized house numbers

Singaporeans like short forms of everything, including the house number format as well. As such, it creates an inconsistent house number formatting.

An example of full form of Singapore address: **Jalan Kukoh 10, #05-69, Singapore 162010**. This format includes:

- Street name (i.e. Jalan Kukoh)
- Block number (i.e. 10)
- Unit numbers (i.e. #05-69)
- Postal Code (i.e. 162010)

The nightmare is that, it may be written in various ways:

- **10 Jalan Kukoh** may be written as **Jalan Kukoh Blk 10** or even **Jalan Kukoh 10**.
- **Singapore 162010** may be written as **S162010**.

To resolve this issue, I will keep the value of the tag if it contains the block number or unit number or both.

Unstandardized phone numbers

Inconsistency that is observed in the phone number format:

- **+65** country code may or may not appear in the phone number.
- **Hyphen ('-')** that splits the phone number. E.g. 8402-5704.

As such, I standardize it to be '+6584025704' (Country Code + Phone Number without hyphen)

Invalid postal codes

Singapore postal codes should be 6 characters long. As such, postal codes that do not fit in to this formatting will be discarded. I also observed that there is an arbitrary value containing **'<different>'** in the postal code tag. As such, this will also be removed in the data cleaning process.

Names in other languages

Because there are four official languages used in Singapore, we can observe there are instances where the name of the elements is in Chinese, Malay, or Indian. As such, those alternate names will be kept in an array.

2. Overview of the Dataset

In this section, I will explore some basic data summary about the dataset and also the MongoDB queries used.

File Sizes

Original Dataset: Singapore.osm → 304 MB

Cleaned and Transformed Dataset: Singapore.osm.json → 431MB

MongoDB Queries

```
client = MongoClient()

db = client.final_project

collection = db.singaporeOSM
```

#Number of documents

```
collection.find().count()

>  1603885
```

#Number of nodes

```
collection.find({"type":"node"}).count()

> 1403235
```

#Number of ways

```
collection.find({"type":"way"}).count()

> 200605
```

#Number of unique users

```
len(collection.distinct("created.user"))

> 1749
```

#Top religions ranked by the count of its 'place_of_worship'

```
pipeline = [{"$match":{"amenity":{"$exists":1},
        "amenity":"place_of_worship"}},
        {"$group":{"_id":"$religion", "count":{"$sum":1}}},
        {"$sort":{"count":-1}}]
result = collection.aggregate(pipeline)
pprint.pprint(list(result))


> [{u'_id': u'muslim', u'count': 495},
 {u'_id': u'christian', u'count': 191},
 {u'_id': None, u'count': 97},
 {u'_id': u'buddhist', u'count': 71},
 {u'_id': u'hindu', u'count': 17},
 {u'_id': u'taoist', u'count': 8},
 {u'_id': u'jewish', u'count': 4},
 {u'_id': u'sikh', u'count': 3}]
```

Surprisingly, Muslim has the most number of 'place_of_worship' when the country most populous religion is Buddhism.

#top 10 popular cuisine types

```
pipeline = [{"$match":{"amenity":{"$exists":1}, "amenity":"restaurant"}},
        {"$group":{"_id":"$cuisine", "count":{"$sum":1}}},
        {"$sort":{"count":-1}}, {"$limit":10}]
result = collection.aggregate(pipeline)
pprint.pprint(list(result))


> [{u'_id': None, u'count': 907},
{u'_id': u'chinese', u'count': 116},
{u'_id': u'japanese', u'count': 66},
{u'_id': u'indian', u'count': 39},
{u'_id': u'korean', u'count': 39},
{u'_id': u'italian', u'count': 38},
{u'_id': u'asian', u'count': 30},
{u'_id': u'pizza', u'count': 27},
{u'_id': u'regional', u'count': 26},
{u'_id': u'seafood', u'count': 17}]
```

Interestingly, Japanese cuisines seems to be popular in Singapore.

3. Further Exploration

Information appearing in arbitrary field that is not expected

There are instances, such as, phone number which appears in random field that is not expected to appear at. To resolve this situation, it is extremely complex because we may need to iterate through 'v' attribute for all tags. We may be able to use advanced regex to handle this issue, but it may not be fool proof as well. As such, it will be good to show the comparison of false positive or false negative for this issue.

Alternative ways to verify data accuracy

The other way to improve the accuracy of the data is to verify latitude, longitude, street name, phone number, or post code with credible database such data.gov.sg or Google Map. The challenge here is that there may be conflicts when different data sources using different terminology but referring to the same entity. As such, we may apply prioritizing for data sources, in which we can choose to use which data source first in case of conflicts.

Utilizing geospatial features in OSM

The power of Open Street Map is that there are latitude and longitudes tagged to various entities. As such, we can use it for geospatial query, such as:

- How many restaurants within 500 meter of a location?
- If someone wants to open a restaurant business, he/she can explore the number of competitors nearby.
- Property pricing exploration: which factors (distance to train station, schools, shopping malls) correlate to property prices? Of course, we will need to merge with additional data sources to get property prices.

All this can be implemented with Turf.js and Leaflet, and it can potentially become a very serious and long-term project.

4. References

- https://github.com/ryancheunggit/Udacity/blob/master/P3/code.py
- https://www.openstreetmap.org/relation/536780
- https://en.wikipedia.org/wiki/Singapore
- https://en.wikipedia.org/wiki/Batam
- https://en.wikipedia.org/wiki/Johor_Bahru