Data Wrangling on OpenStreetMap with MongoDB

Ariel Ma

Map Area: Melbourne, VIC, Australia

1. Problems Encountered in the Map

After reviewing a sample file (about 40MB) of Melbourne OSM data generated using the provided code, I noticed there are 3 main problems:

1. Inconsistent street types or wrongly-spelt street types. For example, street type Drive is sometimes written as *Dr*. or *Dr*; street type Road is sometimes spelt *roah* or written as *road.*
2. In Australia, city name is not usually included in the address. The *City* is usually referring to the CBD of metropolitan areas in each state. Areas other than the CBD are suburbs. Having checked the sample file, I noticed there are node or way elements that have both k='addr:city' and k='addr:suburb' tags and the values are the same. There are also elements with only k='addr:city' tag but the value of it is actually the suburb name. For elements that have both city and suburb tags but different values, the values of city tags are not correct. Therefore, the value of addr:city attribute will be considered as suburb names and will be cleaned together with the suburb names.
3. Wrong suburb names. By comparing the suburb names in the OSM file with a standard list of Victoria suburbs, I noticed that there are many misspelt suburb names and also capitalisation of the suburb names are inconsistent.

2. Methodology Overview

There are 3 steps I have adopted to finally convert the OSM file into cleaned JSON format:

1. Auditing
2. Cleaning
3. Converting

Street types, suburb names and values of addr:city attribute are 3 fields I am going to clean. The raw data will be read element by element into memory and the cleaning is happening on the fly based on the standard list created in the audit step. The cleaned element is then converted into JSON format and written into a JSON file to be ready for MongoDB import. In this section I will explain in details from the process of auditing to cleaning for each field. Also I will explain how I converted the data and The below chart shows briefly the process.

Auditing

1. Street types
   `auditing_street_type.py` is used to audit the street types. An initial list `expected_street_types` was manually created with all frequently used street types. The python code will then check element and compare the street type of the value of `addr:street` attribute. If the street type is not in `expected_street_types`, it will be added into a map `street_types` as the key, and the street name containing the unexpected street type will be added into the value list.
   After one scan of the whole file, the expected_street_type list will be manually updated by looking at the output of `street_types` as there are correct street types that were first missing in the initial expected street type list.
   After all correct street types are added to `expected_street_types`, a map street_type_mapping will be manually created with the wrong street type as the key and the corrected street type as the value. When later updating the street types, this map will be used as a standard for the python code to update unexpected street types.

2. Values of addr:city attribute
   I created `auditing_cities.py` to check if the value of `k='addr:city'` are correctly reflecting the city names. The full OSM will be scanned and the number of elements that containing both `k='addr:city'` and `k='addr:suburb'` will be counted and the value of these attributes are analysed.
   There are 4452778 elements that are either tagged node or way. 36710 elements out of 4452778 have both `k='addr:city'` and `k='addr:suburb'` attributes. For all 36710 elements that have both attributes, 32253 are have the same value for city and suburb and the values for city are actually the suburb names. For the remaining 4457 elements that have different values for the two attributes, the value of city attributes are all suburb names. For 40132 elements that only have city attribute, the values are all suburb names.
   Based on the above findings, I have decided to consider the values of `k='addr:city'` attribute as suburb names and will be updated together with suburb names.

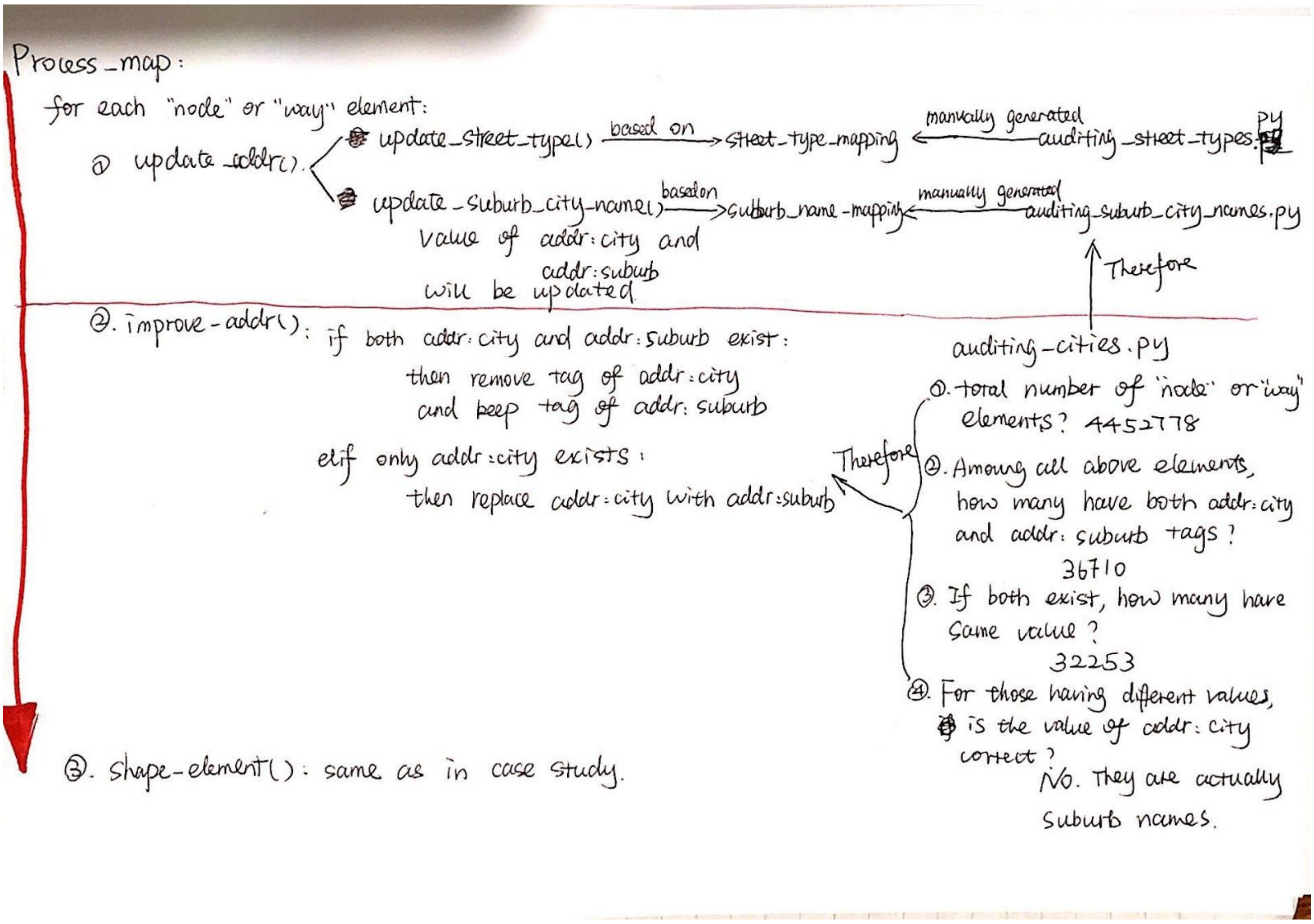
3. Suburb names
   `auditing_suburb_city_names.py` is the code where attribute `k='addr:city'` and `k='addr:suburb'` are scanned and audited. A CSV file Australian_Post_Codes_Lat_Lon.csv which containing a list of Australian suburb names is downloaded. This list is then filtered down only Melbourne suburbs and imported as a standard list. The suburb names in the OSM file will be compared against the standard list. For any suburb names not exists in the standard list, a set `unexpected_suburbs` will be created. Most unexpected names are due to capitalisation inconsistency or misspelling, e.g. St Kilda is wrongly presented as StKilda; Mount Waverley is misspelt as Mount Waverly. There are cases where the suburb name has been changed but in the OSM it is still using the old name, E.g. a suburb used to be called Rosebud West but now is named Capel Sound; however Rosebud West is still used in the OSM file.
   A map `suburb_name_mapping` is then manually created with the unexpected suburb name as the key and the correct suburb name as the value. This map will be used as a standard list to update the unexpected suburb names.

Cleaning

convert_xml_to_json.py is written to clean the address in the OSM file.

1. Updating
   This step is to update the wrong street types in the value of `k=’addr:street’` attribute and the wrong suburb names of values of `k=’addr:city’` and `k=’addr:suburb’` attributes.
   The OSM file will be read into memory and parsed iteratively. For each element, the `update_addr` function will replace the street type or the suburb/city name if they are unexpected based on two maps `street_type_mapping` and `suburb_name_mapping` which were manually created in the auditing step.
2. Improving
   This step is to remove tag that has `k=’addr:city’` attribute if both `k=’addr:city’` and `k=’addr:suburb’` attributes exist in one node; or to update the attribute `k=’addr:city’` to `k=’addr:suburb’` if only `k=’addr:city’` exists in one node.
3. Converting
   This step is to convert the OSM xml element into JSON based on the predefined JSON format as in the case study and generate the JSON file:

```
{
        "id": "2406124091",
        "type: "node",
        "Visible":"true",
        "created": {
                "version":"2",
                "changeset":"17206049",
                "timestamp":"2013-08-03T16:43:42Z",
                "user":"linuxUser16",
                "uid":"1219059"
        },
        "pos": [41.9757030, -87.6921867],
        "address": {
                "housenumber": "5157",
                "postcode": "60625",
                "street": "North Lincoln Ave"
        },
        "amenity": "restaurant",
        "cuisine": "mexican",
        "name": "La Cabana De Don Luis",
        "phone": "1 (773)-271-5176"
}
```



Process_map:

for each "node" or "way" element:

① update_addr():

- update_street_type() based on → street_type_mapping ← manually generated auditing_street_types.py
- update_suburb_city_name() basedon → suburb_name-mapping ← manually generated auditing_suburb_city_names.py

value of addr:city and addr:suburb will be updated

② improve-addr(): if both addr:city and addr:suburb exist:

then remove tag of addr:city and keep tag of addr:suburb

elif only addr:city exists:

then replace addr:city with addr:suburb

③. shape-element(): same as in case study.

auditing-cities.py

↑ Therefore

①. total number of "node" or "way" elements? 4452778

②. Among all above elements, how many have both addr:city and addr:suburb tags? 36710

③. If both exist, how many have same value? 32253

④. For those having different values, is the value of addr:city correct? No. They are actually suburb names.

Therefore

3. Data Overview

The JSON file was created with cleaned data in last section and imported into MongoDB. This section will show statistics about the dataset and the MongoDB queries used to gather them.
File Size:
melbourne_australia.osm ---------------- 877MB
Melbourne_australia.osm.json --------- 1.18GB

# Import JSON file to MongoDB

```
      mongoimport -d osm -c melbourne --file
~/data-analysis/data_wrangling_project/melbourne_australia.osm.json
```

# Number of documents:

```
      > db.melbourne.find().count()
      4452778
```

# Number of nodes

```
      > db.melbourne.find({"type":"node"}).count()
      3915311
```

# Number of ways

```
      > db.melbourne.find({"type":"way"}).count()
      537467
```

# Number of unique users

```
      > db.melbourne.distinct("created.user").length
      2605
```

# Top 10 contributing user

```
      > db.melbourne.aggregate([{'$group': {'_id':'$created.user',
'count':{'$sum':1}}}, {'$sort': {'count':-1}}, {'$limit':10}])
      { "_id" : "CloCkWeRX", "count" : 1184622 }
      { "_id" : "Leon K", "count" : 450589 }
      { "_id" : "melb_guy", "count" : 285643 }
      { "_id" : "Glen", "count" : 182608 }
      { "_id" : "AlexOnTheBus", "count" : 125533 }
      { "_id" : "dssis1", "count" : 95849 }
      { "_id" : "stevage", "count" : 92002 }
      { "_id" : "Canley", "count" : 87960 }
      { "_id" : "Supt_of_Printing", "count" : 78367 }
      { "_id" : "heyitsstevo", "count" : 75643 }
```

# Top 10 appearing amenities

```
      > db.melbourne.aggregate([{'$match':
{'amenity':{'$exists':1}}},{'$group': {'_id':'$amenity', 'count':{'$sum':1}}},
{'$sort': {'count':-1}}, {'$limit':10}])
      { "_id" : "parking", "count" : 9136 }
      { "_id" : "restaurant", "count" : 1974 }
      { "_id" : "school", "count" : 1730 }
      { "_id" : "bench", "count" : 1597 }
      { "_id" : "cafe", "count" : 1580 }
      { "_id" : "fast_food", "count" : 1487 }
      { "_id" : "toilets", "count" : 1311 }
      { "_id" : "post_box", "count" : 1019 }
      { "_id" : "bicycle_parking", "count" : 845 }
      { "_id" : "fuel", "count" : 836 }
```

# Top 10 appearing cuisines

```
      > db.melbourne.aggregate([{'$match': {'amenity':{'$exists':1}, 'cuisine':
{'$exists':1}, 'amenity':"restaurant"}},{'$group': {'_id':'$cuisine',
'count':{'$sum':1}}}, {'$sort': {'count':-1}}, {'$limit':10}])
      { "_id" : "chinese", "count" : 153 }
      { "_id" : "italian", "count" : 143 }
      { "_id" : "japanese", "count" : 108 }
      { "_id" : "indian", "count" : 105 }
      { "_id" : "thai", "count" : 92 }
      { "_id" : "pizza", "count" : 91 }
      { "_id" : "vietnamese", "count" : 71 }
      { "_id" : "asian", "count" : 52 }
      { "_id" : "korean", "count" : 28 }
      { "_id" : "greek", "count" : 28 }
```

4. Additional Ideas

When running the query to get number of nodes with address:

```
> db.melbourne.find({address: {'$exists':true}}).count()
```

It is returning 117267.

Then running the query to get number of nodes with address that do not have suburb and postcode:

```
      db.melbourne.find({"address": {'$exists':true},
"address.suburb":{'$exists': false}, "postcode": {'$exists':false}}).count()
```

It is returning only 38720.

This means that for ⅓ of nodes that have address field, there is no suburb information. This can be confusing as in Melbourne, there are lots of streets with the same name but in different suburbs. So further cleaning of the data may be conducted to add suburb information for addresses.