OpenStreetMap Project
Data Wrangling with MongoDB

Cameron McLeod – April '15 Cohort

Map Area: Vancouver, BC, Canada

Source: https://mapzen.com/data/metro-extracts
https://s3.amazonaws.com/metro-extracts.mapzen.com/vancouver_canada.osm.bz2

1. Problems Encountered in the Map

To start analyzing the dataset, and get a scope of the dataset size, I ran modified mapparser.py and tags.py files. Results below:

Mapparser:
```
Counter({ 'nd': 412540, 'node': 332303, 'tag': 126024, 'way': 53983, 'member': 6313,
          'relation': 686, 'bounds': 1, 'osm': 1})
```

Tags:
```
{'lower': 115490, 'lower_colon': 7827, 'other': 2691,
 'problemchars': 16}
```

This showed me I have a relatively large dataset with ~850,000 tags. Considering the size of the dataset I had few problem characters. The 'lower_colon' value of 7827 shows I could have a large number of addresses with address elements – each potentially with their own set of issues.

I ran the audit-project.py file to explore the dataset and found the following problems:

- Inconsistent street naming conventions and inaccurate naming
- Inaccurate city name data with misspellings
- Invalid postal codes

Inconsistent Street Naming Conventions

Some streets had an unnecessary "Vancouver" at the end of the street name, or inconsistent street type abbreviations like in the case study. I also checked for any orientation abbreviations (N,S,E,W for North, South, East, West) which occur in the street names. I used audit-project.py to test the cleaning of the mentioned values then implemented these procedures in the data-project.py file on import.

Inaccurate City Name Data

The city element of address needed some cleaning as well.

```
#result of a city name counter script
Counter({'Vancouver': 843, 'North Vancouver': 21, 'vancouver': 14,
         'Vancouver, BC': 9, 'Vancovuer': 5, 'North Vancouver City': 3,
         'north vancouver': 1})
```

I resolved the above issues by providing proper capitalizations and removing erroneous values (BC, City).

Invalid Postal Codes

Canadian postal codes are 6 digit alternating alphanumeric codes of the type "A1A 1A1", where the space is optional. Vancouver specifically requires a V as the first digit, where DFIOQU are not allowed throughout.

```
#regular expression to filter for incorrect postal codes
re.compile(r'^(?!.*[DFIOQU])[V][0-9][A-Z] ?[0-9][A-Z][0-9]$')
```

The postal code element of address needed some cleaning. I developed some procedures to remove unnecessary "BC" at the beginning of the postal code, change "0" values mistyped as "O" and change lower case values to upper case.

2. Data Overview

This section contains basic statistics about the dataset and the MongoDB queries used to gather them.

File sizes

```
vancouver_canada.osm ......... 70.5 MB
vancouver_canada.osm.json .... 87.6 MB
```

```
# Number of documents
> db.vancouver.find().count()
      386286

# Number of nodes
> db.vancouver.find({"type":"node"}).count()
      332300

# Number of ways
> db.vancouver.find({"type":"way"}).count()
      53963

# Number of unique users
> len(db.vancouver.distinct("created.user"))
      575

# top 10 Sorted list of unique documents tagged with a “sport”
> db.vancouver.aggregate([ { "$match" : {"sport" : {"$ne" : None}}},
                           { "$group" : { "_id" : "$sport", "count" : { "$sum" : 1 } } },
                           { "$sort" : { "count" : -1 } }, { "$limit" : 10 }])
```

Result – names of the top 10 “sport” tagged items:

```
{u'_id': u'tennis', u'count': 74}
{u'_id': u'baseball', u'count': 65}
{u'_id': u'soccer', u'count': 47}
{u'_id': u'swimming', u'count': 45}
{u'_id': u'basketball', u'count': 29}
{u'_id': u'skateboard', u'count': 7}
{u'_id': u'multi', u'count': 7}
{u'_id': u'bowls', u'count': 4}
{u'_id': u'equestrian', u'count': 4}
{u'_id': u'canadian_football', u'count': 3}
```

```
# List document names that are skateparks
> db.vancouver.aggregate( [ { "$match" : { "sport" : "skateboard" } },
                            { "$project" : { "_id" : 0 , "sport" : 1 , "name" : 1 } }])
```

Result:

```
{u'sport': u'skateboard'}
{u'sport': u'skateboard'}
{u'sport': u'skateboard'}
{u'name': u'Leeside Tunnel skateboard park', u'sport': u'skateboard'}
{u'name': u'UBC skatepark', u'sport': u'skateboard'}
{u'name': u'China Creek Skatepark', u'sport': u'skateboard'}
{u'name': u'Hastings skateboard park', u'sport': u'skateboard'}
```

There are 7 documents tagged with skateboard, 4 of which are named skate parks in Vancouver. From further investigation, the other three are also tagged with “leisure” : “pitch”, which appears to be a redundancy in the database in this case. Future investigation of these unnamed documents could be to find the locations of their nodes, then name them or delete them as duplicates.

3. Additional Ideas

There is a lot of construction in Vancouver, as such, building names are changing and businesses are moving. There is a “building” : “construction” key value pair in the system already, which will allow for a placeholder for a future building. However there could be an additional “renovation” and “demolition” key pair with elements “entry date” and the projected dates for “start” and “completion” of the renovations or demolitions (other than the “landuse” : “brownfield” pair which exists already). Users could then query for business demolition start dates, and either remove the businesses from the system or replace values with the new business values. This additional feature would aid in database upkeep.

Another opportunity to have these values updated would be to access the development application information page at http://former.vancouver.ca/devapps/, check for new applications and place them at addresses in the database. This would allow people to see what buildings are being proposed on the map, as well as what current businesses may be displaced or will go through renovations. It may also be helpful for people deciding what neighborhoods to move to long term, as they will see what possible amenities and neighbors they will have in the future. Issues I see with this proposal would be address uniqueness, failed building proposals and inaccurate construction timelines – leaving us with a bloated database. Additionally, to match these development applications to addresses, the address and latitude/longitude should be assessed with matches or close matches. This would be done using either the latitude/longitude of the current building at the address, or nearby addresses.

Additional data exploration using MongoDB queries

Top 5 Most popular cuisines

```
> db.char.aggregate [ { "$match" : { "cuisine" : {"$ne" : None} ,
                                     "amenity" : { "$exists" : 1 },
                                     "amenity" : "restaurant" } },
                      { "$group" : { "_id" : "$cuisine" , "count" : { "$sum" : 1 } } },
                      { "$sort" : { "count" : -1 } }, { "$limit" : 5}])
```

Result:

```
{u'_id': u'japanese', u'count': 39}
{u'_id': u'chinese', u'count': 36}
{u'_id': u'sushi', u'count': 23}
{u'_id': u'vietnamese', u'count': 14}
{u'_id': u'pizza', u'count': 12}
```

We love Asian restaurants in Vancouver!

Conclusion

There is definitely more work to be done to make this database complete, however there appear to be a large group of users in the process of updating. With the correct tools, this database could provide some very unique insights.

In terms of the process of cleaning data – coding out every error that I found would be very time consuming. When doing some of the postal code corrections, I considered that doing a find/replace in a text document would be faster than implementing the code for the few occurrences (<5) of each error. However I can see how in very larger datasets this would not be feasible. It is important to distinguish when data is "clean enough" for the purpose of the project.