Data Wrangle OpenStreetMap Data

D. Chris Young

OpenStreetMap Area Phoenix, AZ United States
Download source: https://mapzen.com/data/metro-extracts

Problems encountered in the map

The first step was to explore the structure of the OSM file. The tags.py script iterates through the file and counts the items for each start tag, enumerates the keys for the element attributes and iterates the “k” attribute keys for the tag nodes within node/way. In addition, it prints all of the values with problem characters. The only key with problem characters within the file is “p.o.box:en”.

One observation from this data exploration step is there are 458 items that have been tagged with “fixme” as the key:

'node-FIXME': 123
'node-fixme': 6
'way-FIXME': 205
'way-FIXME:access': 1
'way-FIXME:name': 6
'way-fixme': 117

These records will be ignored in addition to the key with the problem character.

The next step was to explore the address attribute keys for the node/way tags with the addr_check.py script. The output isolates address fields between main keys with one colon after “addr” and sub fields separated by multiple colons (i.e. addr:street:housenumber). The OSM for Phoenix does not have any sub fields for any of the address fields but will be ignored just in case the structure of the OSM changes in the future.

The next step was to explore the street field for potential issues related to inconsistent usage of common fields in streets such as pre-direction and street types. Street suffixes from the USPS was used to construct a list of common first letters to narrow down the results. A street mapping dictionary was created for each instance of an abbreviation and the full word. The scrub_street function within scrub_addr.py module was used to clean the addr:street field.

The next step was to explore the values for the city, state, postcode and country addr fields. The city field had the following observations: 1 street address instead of a city, case inconsistency, spelling issues and city names with the state/postcode.

In order to scrub the city name, city data was downloaded from the United States Zip Codes organization website for the state of Arizona. BeautifulSoup was used to parse the HTML file in the city_data.py script to create lists for zip codes and common cities. The cities list was used as a method to resolve spelling by using SequenceMatcher for a fuzzy comparison of the input city and match in the list. The highest match target with a minimum acceptable match of .75 was used as the city name. The case and extra fields were handled with capwords and splitting the string and taking the first value. The street address was handled manually since there was only 1 record. The scrub_city function within the scrub_addr.py module was used to clean the addr:city field.

The postcode field had 1 record that was just AZ, several zip codes were prefixed with “AZ” and 1 record had visible Unicode characters. The scrub_zip function within the scrub_addr.py module was used to clean the addr:postcode field.

The state field had spelling issues, AZ was spelled out and US was used for 2 records. Since the OSM for Phoenix should all be AZ this was used for all values. The country field only had 1 record that was AZ which was updated to US to match all other values. Both state and country were cleaned directly in the prep_data.py script.

The prep_data.py script performed all of the data cleaning steps and created the output json file to load in MongoDB. The processing time of the OSM was 3.483435 minutes for 2,580,718 records.

Overview of data

The size of the uncompressed phx.osm file is 528.6 MB. The output phx.json file is 593 MB. The processing time for MongoDB load was 15.989068 minutes for 2,580,718 documents.

```
node_count = db.phx.find({"type" : "node"}).count()
print "Node count: %i" % (node_count)
>>>
Node count: 2293618

way_count = db.phx.find({"type" : "way"}).count()
print "Way count: %i" % (way_count)

>>>
Way count: 286976

users = db.phx.distinct("created.user")
print "Distinct user count: %i" % (len(users))

>>>
Distinct user count: 951
```

A MongoDB aggregate query shows the top 10 amenities are:

```
amenity = db.phx.aggregate([{"$match" : {"amenity" : {"$exists" : 1}}},
                    {"$group" : {"_id" : "$amenity" , "count" : {"$sum" : 1}}},
                    {"$sort" : {"count" : -1}}, {"$limit" : 10}])
print "Top 10 amenities:"
for a in amenity:
   item = "{} count = {}".format(a['_id'],a['count'])
   pprint(item)

>>>
Top 10 amenities:
'parking count = 2853'
'school count = 1224'
'place_of_worship count = 939'
'fast_food count = 821'
'restaurant count = 755'
'fuel count = 704'
'swimming_pool count = 480'
'shelter count = 397'
'bench count = 335'
'bank count = 249'
```

One way the dataset could be used in another application is mapping locations by the type of parking available in the vicinity of other nearby amenities.

```
parking = db.phx.aggregate([{"$match" : {"amenity" : {"$exists" : 1},"amenity" : "parking"}},
                    {"$match" : {"parking" : {"$exists" : 1}}},
                    {"$group" : {"_id" : "$parking" , "count" : {"$sum" : 1}}},
                    {"$sort" : {"count" : -1}}])
print "Parking types:"
for p in parking:
   item = "{} count = {}".format(p['_id'],p['count'])
   pprint(item)

>>>
Parking types:
'surface count = 384'
'multi-storey count = 111'
'garage count = 12'
'underground count = 9'
'park_and_ride count = 8'
'carports count = 4'
'garage_boxes count = 1'
```

The phx osm amenities file contains additional data for the place_of_worship, top 10 fast-food, top 10 restaurants and top 10 cuisine (fast_food and restaurant combined) amenities.

```
religion = db.phx.aggregate([{"$match" : {"amenity" : {"$exists" : 1},"amenity" :
"place_of_worship"}},
                    {"$match" : {"religion" : {"$exists" : 1}}},
                    {"$group" : {"_id" : "$religion" , "count" : {"$sum" : 1}}},
                    {"$sort" : {"count" : -1}}])

fast_food = db.phx.aggregate([{"$match" : {"amenity" : {"$exists" : 1},"amenity" : 'fast_food'}},
                    {"$match" : {"name" : {"$exists" : 1}}},
                    {"$group" : {"_id" : "$name" , "count" : {"$sum" : 1}}},
                    {"$sort" : {"count" : -1}}, {"$limit" : 10}])

restaurant = db.phx.aggregate([{"$match" : {"amenity" : {"$exists" : 1},"amenity" :
'restaurant'}},
                    {"$match" : {"name" : {"$exists" : 1}}},
                    {"$group" : {"_id" : "$name" , "count" : {"$sum" : 1}}},
                    {"$sort" : {"count" : -1}}, {"$limit" : 10}])

cuisine = db.phx.aggregate([{"$match" : {"amenity" : {"$exists" : 1},"amenity" : {"$in" :
['fast_food', 'restaurant']}}},
                    {"$match" : {"cuisine" : {"$exists" : 1}}},
                    {"$group" : {"_id" : "$cuisine" , "count" : {"$sum" : 1}}},
                    {"$sort" : {"count" : -1}}, {"$limit" : 10}])
```

For a quick summary, below is the number 1 item for each:

- Religion: Christian
- Fast-Food: McDonald’s
- Restaurant: Denny’s
- Cuisine: Burger

Other ideas

Bicycle data is very prevalent in the OSM for Phoenix and could be used in bicycle GPS apps to identify accessibility routes. The data showed inconsistent case and usage of the designated terminology so these were cleaned in the scrub_bicycle function within the scrub_addr.py module.

```
bicycle = db.phx.aggregate([{"$match" : {"bicycle" : {"$exists" : 1}}},
                    {"$group" : {"_id" : "$bicycle" , "count" : {"$sum" : 1}}},
```

```
                    {"$sort" : {"count" : -1}}])
print "Bicycle access:"
for b in bicycle:
   item = "{} count = {}".format(b['_id'],b['count'])
   pprint(item)

>>>
Bicycle access:
'yes count = 11066'
'no count = 2789'
'designated count = 276'
'traffic_signals count = 3'
'permissive count = 1'
'destination count = 1'
```

An initial observation of the OSM tags revealed a fuel type field following the similar format as address fields (fuel:"type"). The fuel_check.py script explores all possible key-value combinations for keys prefixed fuel. The output confirms the fields are primarily yes/no indicators. The fuel field with no colon only has "charcoal" values so these were ignored. The create_fuel function within the create_custom module creates a fuel_type dictionary for each fuel type.

The data could be used in a mapping application to search fuel amenity locations that have a specific type available such as cng or electricity. With the increase of alternative fuel vehicles, this kind of data could be very useful in maps for customers in unknown locations (most people cannot afford a Tesla!). However, the data is only populated in 19 of the 704 fuel amenity locations. This suggests the information is not captured for all locations so usage would be limited without increased user contribution for this data in the Phoenix area.

One way to really improve the usefulness of the address information in the data would be to verify and update. During the postcode cleaning process steps were taken to ensure the postcode was in the correct format. However, in order to confirm the postcode is valid additional steps would need to be taken. The USPS has an Address Standardization API and companies such as Smarty Streets offer enhanced services based on the USPS API. The address for the node could be passed through the Smart Streets API and the results returned are verified and in the correct USPS format. These steps would also improve the accuracy of the open source project.

A free account was created for Smart Streets and the smartystreets_example.py is an example script passing an actual school address from the dataset with the Smarty Streets API results at the bottom of the script for illustration purposes. The API also includes additional metadata such as latitude and longitude which could be added to the dataset.

References:

https://www.openstreetmap.org/relation/111257#map=9/33.6055/-112.1237
http://www.unitedstateszipcodes.org/az/#zips
http://pe.usps.gov/text/pub28/28apc_002.htm
https://www.usps.com/business/web-tools-apis/address-information-api.htm#_Toc410982981
https://smartystreets.com/
https://github.com/smartystreets/LiveAddressSamples/blob/master/python/street-address.py