OSM Data Wrangle Project

Denver Metro Data

denverMetro.osm data size - 654 MB denverMetro.osm.json data size - 628 MB

To extract the data the following web site was used: Overpass API Data Extraction (http://overpass-api.de/query_form.html). When attempting to get a smaller sample, the size ended up being less the the required 50 MB. The majority of Denver was included to make sure the end size was large enough for the requirement.

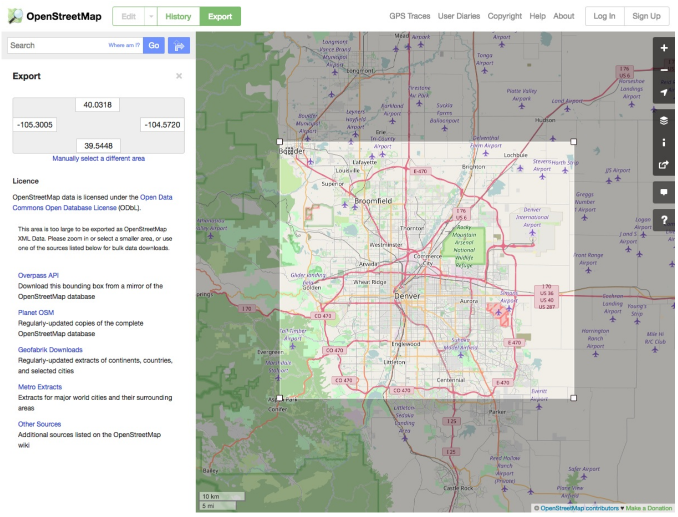
The next line is the query used to acquire the data set.

(node(39.5448,-105.3005,40.0318,-104.5720);<;);out meta;

Mapzen (https://mapzen.com/data/metro-extracts/) was looked into. An account can be created and new custom extracts can be created and downloaded. This is takes a little longer and is not recommended for this project because there is a certain size of file we were looking for and it could be a couple of hours wasted.

Sceenshot of the denver metro extract from OpenStreetMaps

The map area not darkened is the area downloaded from the osm data.



Problems Encountered in the Map Data

Some of the issues encoutered happened in the creation of the json file before even investigating the mongodb. It was useful to standardize the phone numbers and much of the other data before the json file was created. The mongo database collection that was imported showed additional issues that were not addressed in the initial json file creation. Mainly bugs of the regular expressions implementation or unaccounted for entry types.

- Non-Standardization of Street Names
- Postal Code format inconsistent
    - Addtional problems: Incorrectly entered zipcodes (i.e. 'CO' or a house number instead of a zip code) Zip codes with the following format XXXXX-XXXX
- Phone Numbers inconsistent
    - Additioanal problems Multiple phone numbers entered in same field Erroneous data entries

Non-Standardize Street Names

The standardizing of street names was performed specifically to remove the abbreviations and have 'mostly' standard names (i.e. 'Str' was converted to 'Street'). There are many other substitutions that could be performed ('Hwy' and the identified number of a Highway), but this was just the standard substitutions.

Postal Code Formats

For this project the postal codes were transformed so all were in the 5-digit format. Some postal codes were incorrectly entered as the state or a street number. A missing format of '99999' was entered in their place.

Phone Numbers

The python library 'phonenumbers' was used to create consitent formats. The library did check for a 'US' country code of '+1'. There were additional entry errors, such as 2 phone numbers in the field separated by a semi-colon. There were also some businesses that used a mix of alphanumeric numbers which needed to be converted to numbers for the 'phonenumber' library to create a consistent phone number.

Other issues

I looked into the city names and found some inconsistencies. This could be addressed in future work

Number of documents

```
> db.osmDenver.find().count()
```

3330300

Number of nodes

```
> db.osmDenver.find({"type":"node"}).count()
```
2960676

Number of ways

```
> db.osmDenver.find({"type":"way"}).count()
```
369567

Number of unique users

```
> print(len(db.osmDenver.distinct("created.user")))
```
1786

Top user by contributions

```
> cursor = db.osmDenver.aggregate([{"$group":{"_id":"$created.user", "count":{"$sum":1}}},
                   {"$sort":{"count":-1}}, {"$limit":1}])
for cur in cursor:
    print(cur)
```
{u'count': 689747, u'_id': u'chachafish'}

```
>cities = db.osmDenver.aggregate([{"$match":{"address.city":{"$exists":1}}},
                   {"$group":{"_id":"$address.city", "count":{"$sum":1}}},
                   {"$sort":{"count":-1}},{"$limit":5}])
```

Top 5 ZipCodes Listed

```
>zipmatch = db.osmDenver.aggregate([{"$match":{"address.zipcode":{"$exists":1}}},
                        {"$group":{"_id":"$address.zipcode", "count":{"$sum":1}}},
                        {"$sort":{"count":-1}},{"$limit":5}])

for z in zipmatch:
    print(z)
```
{u'count': 9606, u'_id': u'80211'} {u'count': 4323, u'_id': u'80212'} {u'count': 3686, u'_id': u'80026'} {u'count': 2776, u'_id': u'80205'} {u'count': 2549, u'_id': u'80204'}

Top Five Amenities Listed

```
>amenity = db.osmDenver.aggregate([{"$match":{"amenity":{"$exists":1}}},
                   {"$group":{"_id":"$amenity", "count":{"$sum":1}}},
                   {"$sort":{"count":-1}},{"$limit":5}])
for amenityType in amenity:
    print(amenityType)
```
{u'count': 12685, u'_id': u'parking'} {u'count': 1938, u'_id': u'restaurant'} {u'count': 1238, u'_id': u'school'} {u'count': 939, u'_id': u'fast_food'} {u'count': 845, u'_id': u'bicycle_parking'}

Additional Work

Additional digging discovered more inconsistent data. The 'city' field sometimes had states included and there were some entered in all caps with others having all lower case even through the cities were the same.

A few bugs in my coding were discovered after the import into the mongo database. One was an error in the length of the zip codes which allowed for some 6 digit zip codes. There was also an error on the street addresses. A lower case 'ct' was uncaught.

Landuse would be an interesting field to look into. One 'landuse' type that caught my eye was 'brownfield'. I am curious if this is related to an EPA assessment or something else. There are 166 entries with 'brownfield' 'landuse' type. It would also be interesting to determine what kind of other sights are a close distance to said 'brownfield'.

1) City field Audit Solution

```
Auditing the city names to create a standard set of names in the search area would be performed in a
 similar manner as the street name audits.  A python set would be created to identify the city names
 in the xml osm data.  If a comma is in the city names it could automaticlally be split using the fi
rst item in the split list.  The names would be put to all lower and then the first letter of each w
ord could be capitalized.  This is not all inclusive and may not cover all issues with the data.
```

Benefits

```
1. Having standard cities allows for consistent groupings
2. Search Criteria would lead to more accurate searches of amenities associated with cities.  (Would
 be kinda fun to create a 'competition' between neighboring cities with osm data.  Who has the most
restaurants?  Who has the most parking lots?...)
3. A jQuery drop menu option linked to a mongo db query would
   provide a consistent group of cities and it would look professional.
```

Anticipated Issues

```
1.  City fields that include the state might not have the state field entered.  A second validation
check might be necessary
2.  The city may be spelled incorrectly.  A list of city names in the state might be necessary to va
lidate.  The latitude and longitude could be used to try to validate a city name.
```

In [ ]:

```
```