DAND P3 (Data Wrangling and OpenStreetMap)

Map Area

Greater Philadelphia, PA, USA

https://www.openstreetmap.org/export#map=11/40.0026/-75.1181

The area I studied was actually the Greater Philadelphia Metro Area rather than just Philadelphia proper–-the XML I worked with was not bound to the city itself, but was a wider region as linked to above.

Overview of the Data

Overall size of the data[1]:

```
import os
folder = 'C:\Users\Andrew\Documents\Jupyter\OSM Python 2\New folder'
folder_size = 0
for (path, dirs, files) in os.walk(folder):
    for file in files:
        filename = os.path.join(path, file)
        folder_size = os.path.getsize(filename)
        print filename " = %0.1f MB" % (folder_size/(1024*1024.0))
        print file, " = %0.1f MB" % (folder_size/(1024*1024.0))

nodes.csv  = 242.9 MB
nodes_tags.csv  = 14.3 MB
osm.db  = 470.1 MB
ways.csv  = 17.6 MB
ways_nodes.csv  = 86.6 MB
ways_tags.csv  = 50.6 MB
```

Number of nodes:

```
SELECT COUNT(distinct id) from nodes

[(3023787,)]
```

[1] All queries were executed in a Jupyter notebook, but I have removed the Python code needed to execute queries (e.g., “QUERY = ... , c.execute(QUERY), c.fetchall()”) for the sake of readability and have retained just the queries and their results.

Number of ways:

```
SELECT COUNT(distinct id) from ways

[(304928,)]
```

Number of users:

```
SELECT COUNT(distinct uid) from nodes

[(1941,)]
```

Top ten users by number of contributions:

```
SELECT user, count(*) from nodes group by user order by count(user) desc limit 10

[(u'dchiles', 766876),
 (u'woodpeck_fixbot', 554609),
 (u'NJDataUploads', 286251),
 (u'kylegiusti', 90134),
 (u'WesWeaver', 87588),
 (u'choess', 82097),
 (u'Matt1993', 78922),
 (u'crystalwalrein', 70340),
 (u'dankpoet', 61087),
 (u'Louise Belcher', 59064)]
```

List of top 20 amenities in the map area:

```
SELECT value, count(value) from nodes_tags where key = 'amenity' group by value order
by count(value) desc limit 20

[(u'parking', 11108),
 (u'school', 992),
 (u'place_of_worship', 874),
 (u'restaurant', 754),
 (u'bank', 368),
 (u'fast_food', 328),
 (u'fuel', 292),
 (u'university', 232),
 (u'pharmacy', 152),
 (u'hospital', 146),
 (u'fire_station', 140),
 (u'cafe', 132),
 (u'college', 122),
 (u'library', 122),
 (u'post_office', 114),
 (u'social_facility', 98),
 (u'swimming_pool', 94),
 (u'bar', 88),
 (u'grave_yard', 84),
 (u'shelter', 70)]
```

A quick look at how wheelchair accessible the region, according to the fallible data:

```
SELECT key, value, count(*) from nodes_tags where key = 'wheelchair' group by value

[(u'wheelchair', u'limited', 19),
 (u'wheelchair', u'no', 55),
 (u'wheelchair', u'yes', 169)]
```

Problems Encountered in the Map

The data of the Greater Philly area suffers from what appear to be the same problems in all other datasets studied by Udacity students in this project, at least in the United States. These include:

- Incorrectly entered street names. These include simple errors like "ave" for "Avenue" and more challenging problems like full addresses and apartment/suite numbers where an address suffix should be. These, for example, are some of the problematic street names[2]:

```
SELECT *, count(value) from nodes_tags where type = 'addr' and key = 'street' and VALUE
not LIKE '%Street%' and VALUE NOT LIKE '%Road%' and VALUE NOT LIKE '%Avenue%' \
and VALUE NOT LIKE '%Pike%' and VALUE NOT LIKE '%Way%' and VALUE NOT LIKE '%Drive%' \
and VALUE NOT LIKE '%Boulevard%' group by value order by count(value) asc limit 10

[(1483624883,
  u'street',
  u'1 Brookline BlvdHavertown, PA 19083(610) 446-1234',
  u'addr',
  1),
 (1696895249, u'street', u'12th and Vine', u'addr', 1),
 (2003088513, u'street', u'15th and Master', u'addr', 1),
 (2003096911, u'street', u'16th and Stiles', u'addr', 1),
 (1687412493, u'street', u'1st and Seneca Sts.', u'addr', 1),
 (1695470222, u'street', u'200 Manor Ave. Langhorne, PA 19047', u'addr', 1),
 (1698272302,
  u'street',
```

[2] The following query does not account for all of the proper street suffixes, but initial querying resulted in proper street names appearing in the results; the restrictions in this query are just enough to reveal some problematic addresses.

```
  u'2245 E. Lincoln Hwy, Langhorne, PA 19047',
  u'addr',
  1),
 (1698269474,
  u'street',
  u'2275 E Lincoln Hwy, Langhorne, PA 19047',
  u'addr',
  1),
 (4544354970L, u'street', u'2515 Metropolitan Dr', u'addr', 1),
 (367962924, u'street', u'Alfa Terrace', u'addr', 1)]
```

Some of these are not fixable without in-depth programmatic corrections, and some of them are almost not fixable programatically at all–-”16th and Stiles”, for instance, is not an address, but an informal way of giving directions.

- Inconsistent and incorrect postcodes. Cleaning was able to remove a sizable number of incorrect codes, such as those with state abbreviastions (e.g., “PA 19145”) or those with four extraneous digits (e.g, '19010-3224'). However, truncated codes with only four digits remain in the dataset (e.g., '1713', as seen below) and are more difficult to clean.

```
SELECT value, count(value) from nodes_tags where key = 'postcode' group by value order
by count(value) asc limit 20

[(u'08003', 1),
 (u'08026', 1),
 (u'08031', 1),
 (u'08033', 1),
 (u'08036', 1),
 (u'08046', 1),
 (u'08060', 1),
 (u'08065', 1),
 (u'08066', 1),
 (u'08070', 1),
 (u'08077', 1),
 (u'08091', 1),
 (u'08103', 1),
 (u'08107', 1),
 (u'08110', 1),
 (u'08343', 1),
 (u'08515', 1),
 (u'08611', 1),
 (u'08850', 1),
 (u'1713', 1)]
```

Zip codes with extra strings can can be easily clean programmatically with regex, but zip codes that only contain four digits present a special challenge. One would have to use somewhat sophisticated techniques of finding the latitude and longitude of neighboring nodes and amending the zip code to that of its neighbors.

- Illogically named amenities. The OSM project lacks standards for how to label nodes, complicating the analysis process. In the list of amenities above, we can already see that "restaurant," "cafe" and "fast food" are separately named tags for what is arguably a selfsimilar attribute. But the problem becomes more acute the lower down the list of amenities we get:

  ```
  SELECT value, count(value) from nodes_tags where key = 'amenity' group by value order
  by count(value) asc limit 20

  [(u'Bank Branch', 1),
   (u'animal_boarding', 1),
   (u'animal_training', 1),
   (u'bar;pub', 1),
   (u'blood_donation', 1),
   (u'boat_rental', 1),
   (u'charging_station', 1),
   (u'clock', 1),
   (u'coworking_space', 1),
   (u'crematorium', 1),
   (u'education', 1),
   (u'fish_hatchery', 1),
   (u'monastery', 1),
   (u'money_transfer', 1),
   (u'music_school', 1),
   (u'optician', 1),
   (u'parking garage', 1),
   (u'parking_exit', 1),
   (u'public_bookcase', 1),
   (u'vehicle_inspection', 1)]
  ```

  The chosen names become almost comical; what is a ‘public bookcase’? What ages are taught at ‘education’? Is ‘crematorium’ separate from a funeral home?

Additional Ideas

One of the most helpful changes to how OSM data is managed would be to restrict what kinds of street names are uploaded. We can see the proportion of streets that were missed in our cleaning function[3]:

```
SELECT (x.number * 1.0) / (y.number) from \
(SELECT count(*) as number from nodes_tags where type = 'addr' and key = 'street' \
```

[3] In this case, we’ll retain all the expected street name values in our query

```
and VALUE not LIKE '%Street%' and VALUE NOT LIKE '%Road' and VALUE NOT LIKE '%Avenue'
and VALUE NOT LIKE '%Pike' and VALUE NOT LIKE '%Way' and VALUE NOT LIKE '%Drive' and
VALUE NOT LIKE '%Boulevard' and VALUE NOT LIKE '%Trail' \
and VALUE NOT LIKE '%Court' and VALUE NOT LIKE '%Lane' and VALUE NOT LIKE '%Trail' \
and VALUE NOT LIKE '%Commons' and VALUE NOT LIKE '%Parkway' and VALUE NOT LIKE
'%Place') \
x join  \
(SELECT count(*) as number from nodes_tags where type = 'addr' and key = 'street') \
y on 1=1

[(0.06718783663168262,)]
```

We can see that between user-input and our cleaning function, we were able to account for a good bit of the street names. But we still have almost 7 percent of street names as erroneous. Restricting what is allowed to be entered as a street name (for instance, rejecting all submissions that don't end in one of the values listed above) could get this figure to approach 0. The challenges is both how to code the implementation of those restrictions, as well as having to account for every possible street ending: What of the odd-named "Bend"?

We see a similar problem with special characters:

```
SELECT * from nodes_tags where value like '%;%' and key not like 'census:population'
limit 10

[(169021824, u'destination:ref', u'PA 23;I 76 East', u'regular'),
 (357337215, u'gnis:feature_id', u'1200187;1200188', u'regular'),
 (357355528, u'gnis:feature_id', u'1209024;1194501', u'regular'),
 (357372102, u'gnis:feature_id', u'2347815; 2347816', u'regular'),
 (357748506, u'gnis:created', u'01/22/2008;08/23/2007', u'regular'),
 (357748506, u'gnis:feature_id', u'2340253;2373644', u'regular'),
 (816238961,
  u'traffic_sign:ref',
  u'R2-1-25;
http://deldot.gov/information/pubs_forms/manuals/de_mutcd/pdf/standard_signs_010810.pdf
',
  u'regular'),
 (816255920,
  u'traffic_sign:ref',
  u'R1-1;
http://deldot.gov/information/pubs_forms/manuals/de_mutcd/pdf/standard_signs_010810.pdf
',
  u'regular'),
 (1893589238, u'addr:suite', u'Hillcrest I;Suite 150', u'regular'),
 (2002961958,
  u'wheelchair:description',
  u'boxes are always on the floor; narrow aisles',
  u'regular')]
```

The semicolon both makes the data unusable as well as indicates where unusable data would be: “boxes are always on the floor; narrow aisles” requires either human eyes or AI to interpret. Even that key, “wheelchair: description”, is an error, as a “wheelchair” key already exists allowing for yes/no/limited/unknown values:

```
SELECT key, value, count(*) from nodes_tags where key = 'wheelchair' group by value

[(u'wheelchair', u'limited', 24),
 (u'wheelchair', u'no', 62),
 (u'wheelchair', u'unknown', 4),
 (u'wheelchair', u'yes', 288)]
```

And bafflingly, even high-volume contributors seem to not even bother to spell out the full names of cities on which they’re scribing data:

```
QUERY = "SELECT nodes.user, nodes_tags.value, count(*) from nodes_tags join nodes on
nodes.id = nodes_tags.id where key = 'addr:city' group by nodes.user order by
count(nodes_tags.value) desc limit 5"
c.execute(QUERY)
c.fetchall()

[(u'James Michael DuPont', u'Lawrence twp', 2547),
 (u'Matt1993', u'Ogden', 1158),
 (u'Luisi Mouse', u'Pemberton', 374),
 (u'chrmur', u'Philadelphia', 247),
 (u'eugenebata', u'Philadelphia', 211)]
```

If a mandatory tutorial so that users know what values already exist would be too prohibitive, perhaps someone should at least send James an email.