OpenStreetMap Data Wrangling (SQL)

Map Area

Geneva, Switzerland

References

http://overpass-api.de/api/map?bbox=6.1324,46.1756,6.2122,46.2140
https://github.com/wwu247/Udacity-DAND-P3_Wrangle_OpenStreetMap
https://gist.github.com/carlward/54ec1c91b62a5f911c42#file-sample_project-md

Introduction

Geneva, Switzerland is my father's hometown, and this project sets to explore open street map data of a neighborhood in the area. This project explores the formatting and conversion of open street map download files to csvs, and subsequently that of CSV files to SQL tables to be queried to see what database querying reveals. A script was run in Python to create tables from the xmls, the code was tested against a smaller sample file and then run against the original file once debugged.

A few metadata are provided below, with the relevant code to obtain them.

```
import sqlite3
from pprint import pprint
import os
statinfo=os.stat('Geneve_street.osm')

print 'Geneva street map file size--', statinfo.st_size, "bytes"
```

Geneva street map file size-- 52220914 (52.2 MB) bytes

```
QUERY = 'PRAGMA PAGE_SIZE;'
c.execute(QUERY)
all_rows = c.fetchall()

print('1) PRAGMA PAGE_SIZE:')
pprint(all_rows)
```

1) PRAGMA PAGE_SIZE:
[(1024,)]

Pragma page count and size are descriptors for the size of the database file, measured in bytes. The page size in SQLite was defaulted to 1024 bytes until recently.

2) PRAGMA PAGE_COUNT:
[(27330,)]

The following is a query for the number of nodes retrieved from the downloads.

```
QUERY3='SELECT COUNT(*) FROM nodes;'
c.execute(QUERY3)
all_rows = c.fetchall()
print('3) Number of Nodes:')
pprint(all_rows)
```

3) Number of Nodes:
[(204284,)]

Same goes for the number of ways--

4) Number of Ways:
[(36738,)]

Problems Encountered

Some place (ie proper nouns) were capitalized and others were not. In addition to the presence of logograms used in the translation of place names, French place names were used to describe most of the map area. Characters with diacritical marks, whose format is included in unicode or hexidecimal notation, but not in ASCII, were therefore included. Accounting for these characters was important for the purposes of analysis and dealt with by formatting available in the regular expression flags.

Street types
Here were some of the auditing functions with sample results of their execution on the file provided.

```
def audit_street_type(street_name, mapping):
    street_name = street_name.replace('  ',' ')
    m = street_type_prefix.search(street_name)
    if m:
        street_type = m.group()
        if street_type in mapping_prefix:
            street_name = re.sub(street_type, mapping[street_type], street_name)
    return street_name
```

rue Saint-Victor => Rue Saint-Victor

rue Rousseau => Rue Rousseau
Blvd. du pont d'arve => Boulevard Du Pont D'arve

The first lines of code were to replace double spaces with a single space. The few lines following were used to replace the abbreviated or misspelled street names with names from the mapping list. In addition, following are examples of street name encoding changes--

Rue de l'Ev\xeach\xe9 => Rue de l'Evêché
Rue du Rh\xf4ne => Rue du Rhône
Rue de la Tour-Ma\xeetresse => Rue de la Tour-Maîtresse
Rue de Neuch\xe2tel => Rue de Neuchâtel

Postcodes

A few of the postcodes had spaces or extraneous characters. The following function was used to replace white spaces and limit the characters to the first five spaces.

```
def update_postcode(zipcode):
    zipcode = re.sub(" ", "", zipcode)
    zipcode = re.sub(r"\s+", "", zipcode)
    zipcode = zipcode[:5]
    return zipcode
```

E.g. 122 4=>1224

Phone Numbers

A few of the phone numbers had hyphens or white spaces--the following function was used to clean the fields containing phone numbers.

```
def update_phone(phone):
    better_phone = phone.replace(" ", "")
    better_phone = re.sub("-", "", better_phone)
    better_phone = re.sub(r"\s+", "", better_phone)
    return better_phone
```

E.g. +41 22-4324378 => +41224324378

Atypical Postcodes

One of the postcodes was ‘74240’, where the rest were four digits and began with a ‘12’.

QUERY9 = "SELECT * FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) WHERE key = 'postcode' AND value NOT LIKE '%12%';"

9) Atypical Postcode Entries:

[(26695858, u'postcode', u'74240', u'addr'),
(2140010189, u'postcode', u'74240', u'addr'),
(2140012373, u'postcode', u'74240', u'addr'),
(2140012378, u'postcode', u'74240', u'addr'),
(2140015809, u'postcode', u'74240', u'addr'),
(2140045444, u'postcode', u'74240', u'addr'),
(2900961505, u'postcode', u'74240', u'contact'),
(172162545, u'postcode', u'74240', u'addr'),
(172312934, u'postcode', u'74240', u'addr'),
(199420967, u'postcode', u'74240', u'addr'),
(199422778, u'postcode', u'74240', u'addr'),
(199798290, u'postcode', u'74240', u'addr'),
(199798292, u'postcode', u'74240', u'addr'),
(200400323, u'postcode', u'74240', u'addr'),
(200400324, u'postcode', u'74240', u'addr'),
(200783164, u'postcode', u'74240', u'addr')]

A SQL query revealed the following below about that postal code--

"SELECT * FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) WHERE id LIKE '%21400%' LIMIT 5;"

10) Weird zip code search

[(2140010189, u'city', u'Gaillard', u'addr'),
(2140010189, u'housenumber', u'2', u'addr'),
(2140010189, u'postcode', u'74240', u'addr'),
(2140010189, u'street', u'Rue de Moellesulaz', u'addr'),
(2140010189, u'amenity', u'bank', u'regular')]

It turns out that the zip code was in a city in France named Gaillard. The map area encompassed a small area across the Swiss-French border.

Data Overview and Additional Ideas

Number of Unique Users

"SELECT COUNT(DISTINCT(e.uid)) FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e;"

6) Number of Unique Users:
[(380,)]

The top 10 contributing users

"SELECT e.user, COUNT(*) as num FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e GROUP BY e.user ORDER BY num DESC LIMIT 10;"

7) TOP 10 contributing users:
[(u'x-pab', 141707),
 (u'Marc Mongenet', 62958),
 (u'floyd70', 13989),
 (u'b_l', 4937),
 (u'Alex-7', 3158),
 (u'MFlamm', 2947),
 (u'mdk', 1020),
 (u'SwisstopoImports', 674),
 (u'Erdling1957', 645),
 (u'lhapaipai', 538)]

The following were the top ten postcodes in the map area.
"SELECT tags.value, COUNT(*) as count FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) tags WHERE tags.key='postcode' GROUP BY tags.value ORDER BY count DESC LIMIT 10;"

8) Postcodes by Count:
[(u'1204', 1001),
 (u'1201', 964),
 (u'1205', 194),
 (u'1206', 116),
 (u'1227', 92),
 (u'1225', 74),
 (u'1207', 59),
 (u'1202', 35),
 (u'1223', 30),
 (u'1234', 24)]

Normally postal codes began with the numbers '12'. However, one was 74240

"SELECT * FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) WHERE key = 'postcode' AND value NOT LIKE '%12%';"

9) Atypical Postcode Entries:

[(26695858, u'postcode', u'74240', u'addr'),
 (2140010189, u'postcode', u'74240', u'addr'),
 (2140012373, u'postcode', u'74240', u'addr'),
 (2140012378, u'postcode', u'74240', u'addr'),
 (2140015809, u'postcode', u'74240', u'addr'),
 (2140045444, u'postcode', u'74240', u'addr'),
 (2900961505, u'postcode', u'74240', u'contact'),
 (172162545, u'postcode', u'74240', u'addr'),
 (172312934, u'postcode', u'74240', u'addr'),
 (199420967, u'postcode', u'74240', u'addr'),
 (199422778, u'postcode', u'74240', u'addr'),
 (199798290, u'postcode', u'74240', u'addr'),
 (199798292, u'postcode', u'74240', u'addr'),
 (200400323, u'postcode', u'74240', u'addr'),
 (200400324, u'postcode', u'74240', u'addr'),
 (200783164, u'postcode', u'74240', u'addr')]

It turns out that the zip code was in a city in France named Gaillard. The map area encompassed a small area across the Swiss-French border.

A country delimiter may suit the map data well. A reference to the country under focus, i.e. where an entry is generated, may help eliminate error.

Additionally, linking an image classification from images on Google street view or another photo service to classify the tag names presents an interesting opportunity.

Benefits--it may allow for faster and more precise tagging of places. Second, it could present opportunities for researchers, such as sociologists, to understand how public space is organized.

Problems--the technical expertise needed to implement this may be very high, requiring thorough image testing and algorithm testing. In addition, the level of detail in image rendering would be arbitrary and may not its users needs.

380 unique users contributed to the map area. Initiatives to further crowd source the gathering of data could serve to refine entries, reduce error, and increase area detail. As suggested by the projects referenced above, an incentive structure, perhaps influenced by findings in the behavioral sciences, may inform this development, in the hopes of making the data more tractable for analysis.

Additional Data Exploration

"SELECT value, COUNT(*) as num FROM nodes_tags WHERE key='amenity' GROUP BY value ORDER BY num DESC LIMIT 10;"

11) Most Common Amenities:

```
[(u'restaurant', 259),
 (u'bench', 161),
 (u'recycling', 95),
 (u'drinking_water', 86),
 (u'pharmacy', 69),
 (u'waste_basket', 65),
 (u'cafe', 62),
 (u'parking', 62),
 (u'fountain', 54),
 (u'ferry_terminal', 45)]
```

"SELECT nodes_tags.value, COUNT(*) as num FROM nodes_tags JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='restaurant') i ON nodes_tags.id=i.id WHERE nodes_tags.key='cuisine' GROUP BY nodes_tags.value ORDER BY num DESC LIMIT 10;"

12) Most Frequently Occurring Restaurant Cuisines:

```
[(u'italian', 31),
 (u'regional', 21),
 (u'french', 13),
 (u'pizza', 12),
 (u'chinese', 10),
 (u'japanese', 8),
 (u'asian', 6),
 (u'kebab', 5),
 (u'thai', 5),
 (u'burger', 4)]
```

Conclusion

While the data in the area is somewhat limited, the contributors' contributions is impressive and begs the question as to the data inputting methods and the incentives, if any, in place for such work to happen. Other attempts to automate the process of tagging may further serve to speed up data collection and add to the total. Perhaps there is room for 3D rendering of public spaces in the future.