Data Wrangling with OpenStreetMaps and SQL

OpenStreetMap is a volunteered geographic information project that runs on open source collaboration to provide free and editable maps to everyone on this planet (assuming they have internet access).

As expected with user generated content, inconsistencies and errors (misspellings, innacurate info, etc) in the data are present. The goal of this project is to:

- Download map data of a selected city
- Import the data into an sqlite database
- Correct the problems and fix errors

Map Area

The island of O'ahu, HI, United States (Open Street Map Relation: Honolulu, HI)

I chose O'ahu for several reasons, among which are my intense affinity for the area from having spend considerable time there.

Process

This project will include three steps:

1. **Data Review:** In which the OpenStreetMap .xml file is audited and the data types cleaned in preparation for analysis
2. **SQL Preparation:** Where the cleaned .xml file is parsed into .csv files suitable for SQL query
3. **Query**: In which SQL queries are applied to answer general questions about the area, such as number of retaurants

Contents

I have included all of the individual code files used in this project as part of the submission package:

1. **oahu_audit.py** Gather list of users by uid
2. **oahu_street_clean.py** Standardize street names
3. **oahu_zip_clean.py** Standardize zip codes
4. **oahu_phone_clean.py** Standardize phone numbers
5. **oahu_tag_clean.py** Cleans any troublesome tags in the data set
6. **oahu_csv_convert.py** Convert the .xml file into a series of .csv files
7. **oahu_schema.py** Create the proper shcemas
8. **oahu_sql_prep.py** Prepares the files for SQL queries
9. **oahu_sql_queries** A series of SQL queries run on the converted .csv files

All of the actual data wrangling, cleaning, queries, and results were obtained by running the included **P3_O'ahu_Code.ipynb** file.

Overview of the Data

Before we begin, let's get a summary of the file sizes for all of the data used in this project

File Sizes:

In [1]:
```
import os
print 'The file downloaded for the map data of the city of Honolulu, HI is
 {} MB'.format(os.path.getsize("c:\honolulu_hawaii.osm")/1.0e6)
print 'The csv file for nodes is {} MB'.format(os.path.getsize("nodes.csv"
)/1.0e6)
print 'The csv file for nodes_tags is {} MB'.format(os.path.getsize("nodes
_tags.csv")/1.0e6)
print 'The csv file for ways is {} MB'.format(os.path.getsize("ways.csv")/
1.0e6)
print 'The csv file for ways_nodes is {} MB'.format(os.path.getsize("ways_
nodes.csv")/1.0e6)
print 'The csv file for ways_tags is {} MB'.format(os.path.getsize("ways_t
ags.csv")/1.0e6)
print 'The db file for Oahu OpenStreetMap  is {} MB'.format(os.path.getsiz
e("OpenStreetMap_Oahu.db")/1.0e6)
```

```
The file downloaded for the map data of the city of Honolulu, HI is 61.975
581 MB
The csv file for nodes is 24.392537 MB
The csv file for nodes_tags is 0.631892 MB
The csv file for ways is 1.947941 MB
The csv file for ways_nodes is 8.52166 MB
The csv file for ways_tags is 3.866938 MB
The db file for Oahu OpenStreetMap  is 35.2 MB
```

Data Review

My initial data review involved checking

- Street Names (inlcuding abbreviations)
- Postal Codes
- Phone Numbers

The goal was to assess the consistency and quality for these three elements, and to standardize the information.

Problem with the Data - Standardization

Among the expected problems with the data were inconsistent street name abbreviations, such as Bl, Bl. Blvd, and Blvd. for "Boulevard". I also expect similar inconsistencies in the formatting and presentation of zip codes and phone numbers.

To mitigate this, I will standardize all three by first identifying the errors, then correcting them by:

1. Create a list of "standardized" street names/zip code/phone number
2. Audit the data to identify non-standardized abbreviations, and to replace them with the standardized forms

Using street names as an example, the process will involve creating a function "audit_street_type" which collects the last word in the "street_name" string. Any words not present in the original list of standardized names will be added to the "street_types" dictionary, and will eventually be corrected by the "update_name_function.

The code to perform these tasks are contained in the "oahu_street_clean.py", "oahu_clean_zip.py", and "oahu_clean_phone.py" files

Street Standardization Code sample:

```
# Identify unique cases not handled by the "update_name" function

for street_type, ways in street_types.iteritems():
        for name in ways:
            if "Suite"  in name:
                name = name.split(", Suite")[0].strip()
            if "#" in name:
                name = name.split(" #")[0].strip()
            if "," in name:
                name = name.split(", ")[0].strip()
            if "Suite" in name:
                name = name.split(" Suite")[0].strip()
            if "Building" in name:
                name = name.split(" Building")[0].strip()
            if "Ste" in name:
                name = name.split(" Ste")[0].strip()
            if "St" in name:
                name = name.split(" St")[0].strip()
            name_improv_first = update_name(name, mapping1, street_type_re)
            name_improv_sec = update_name(name_improv_first, mapping2, street_type_pre)

            print name, "=>", name_improv_first, "=>", name_improv_sec
```

```
Ala Ike => Ala Ike => Ala Ike
Lusitania => Lusitania => Lusitania
Pensacola => Pensacola => Pensacola
Ena Rd => Ena Road => Ena Road
Ala Napunani => Ala Napunani => Ala Napunani
Marchant street => Marchant Street => Marchant Street
State Highway 83 => State Highway 83 => State Highway 83
kinalau Pl => kinalau Place => kinalau Place
Moanalua => Moanalua => Moanalua
```

This updated all substrings, such that: “Ena Rd” becomes “Ena Road”. It is important to note that this does not mean, all street adresses on the island of O'ahu have been cleaned. Our expected list in the regex and anticipiated alternate street type endings are very likely incomplete. However, this is a good start.

Zip Code Standardization Code sample:

<img src="https://github.com/steodorovich/dand_osm/blob/master/zip_correction.png?raw=true" "Zip Code Correctione">

As we see, all the zip codes now conform to our five-digit standard, and any +4 extensions or instances in which "HI" was present have been removed.

Phone Number Standardization Code sample:

<img src="https://github.com/steodorovich/dand_osm/blob/master/phone_correction.png?raw=true" "Phone Correction">

Once again the results are that any oddly formatted phone numbers, such as having "1" at the front, or the area code in parentheses such as (808), or using dashes or dots to separate the digits (i.e. 808.555.1212) are removed.

SQL Queries

We're now ready to use the created .csv files for sql queries: nodes_path, node_tags, ways, ways_nodes and ways_tags

In [2]:

```
import sqlite3
import csv
from pprint import pprint

sqlite_file = "OpenStreetMap_Oahu.db"
conn = sqlite3.connect(sqlite_file)
cur = conn.cursor()

cur.execute('DROP TABLE IF EXISTS nodes')
conn.commit()

cur.execute('''
    Create Table nodes(id INTEGER, lat REAL, lon REAL, user TEXT, uid INTE
GER, version TEXT, changeset INTEGER, timestamp TEXT)
''')

conn.commit()
```

Number of Nodes

```
In [25]: # Counting number of nodes
         conn = sqlite3.connect(sqlite_file)
         cur = conn.cursor()
         cur.execute('''
             SELECT COUNT(*) FROM nodes;
         ''')
         all_rows = cur.fetchall()
         print('Number of nodes are:{}').format(all_rows)
         conn.commit()
```

```
Number of nodes are:[(284402,)]
```

Number of Ways

```
In [26]: # Counting number of ways
         conn = sqlite3.connect(sqlite_file)
         cur = conn.cursor()
         cur.execute('''
             SELECT COUNT(*) FROM ways;
         ''')
         all_rows = cur.fetchall()
         print('Number of ways are:{}').format(all_rows)
         conn.commit()
```

```
Number of ways are:[(31698,)]
```

Number of Unique Users

```
In [27]: # Counting number of unique users
         conn = sqlite3.connect(sqlite_file)
         cur = conn.cursor()
         cur.execute('''
         SELECT COUNT(DISTINCT(e.uid))
         FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e;
         ''')
         all_rows = cur.fetchall()
         print('Number of unique users are:{}').format(all_rows)
         conn.commit()
```

```
Number of unique users are:[(560,)]
```

Top 10 Contributors

In [28]:
```
# TOP 10 contributing users
conn = sqlite3.connect(sqlite_file)
cur = conn.cursor()
cur.execute('''
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;
''')
all_rows = cur.fetchall()
print('Number of unique users are:')
pprint(all_rows)
conn.commit()
```

```
Number of unique users are:
[(u'Tom_Holland', 99721),
 (u'cbbaze', 32648),
 (u'julesreid', 13633),
 (u'ikiya', 12715),
 (u'abishek_magna', 11601),
 (u'kr4z33', 10521),
 (u'Chris Lawrence', 9180),
 (u'pdunn', 8936),
 (u'woodpeck_fixbot', 8199),
 (u'aaront', 7789)]
```

Number of users with fewer than 10 updates

In [30]:
```
# Number of users less than 10
conn = sqlite3.connect(sqlite_file)
cur = conn.cursor()
cur.execute('''
SELECT COUNT(*)
FROM
    (SELECT e.user, COUNT(*) as num
     FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
     GROUP BY e.user
     HAVING num<10)  u;
''')
all_rows = cur.fetchall()
print('Number of unique users appearing less than 10 times are:')
pprint(all_rows)
conn.commit()
```

```
Number of unique users appearing less than 10 times are:
[(259,)]
```

Number of Users with only one update

In [29]:
```
# Number users appearing once
conn = sqlite3.connect(sqlite_file)
cur = conn.cursor()
cur.execute('''
SELECT COUNT(*)
FROM
    (SELECT e.user, COUNT(*) as num
     FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
     GROUP BY e.user
     HAVING num=1)  u;
''')
all_rows = cur.fetchall()
print('Number of unique users only appearing once are:')
pprint(all_rows)
conn.commit()
```

```
Number of unique users only appearing once are:
[(123,)]
```

Metropolitan areas of O'ahu

In [31]:
```
# List metro areas of O'ahu
conn = sqlite3.connect(sqlite_file)
cur = conn.cursor()
cur.execute('''
SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags UNION ALL
      SELECT * FROM ways_tags) tags
WHERE tags.key LIKE '%city'
GROUP BY tags.value
ORDER BY count DESC;
''')
all_rows = cur.fetchall()
print('1):')
pprint(all_rows)
conn.commit()
```

```
1):
[(u'Honolulu', 1189),
 (u"Hale'iwa", 19),
 (u'Aiea', 14),
 (u'Kailua', 14),
 (u'Ewa Beach', 13),
 (u'Kapolei', 11),
 (u'Haleiwa', 8),
 (u'honolulu', 7),
 (u'2', 6),
 (u'Honlulu', 6),
 (u'Mililani', 6),
 (u'Pearl City', 6),
 (u'28', 5),
 (u'Honolulu, HI', 4),
 (u'Wahiawa', 4),
 (u'100', 3),
 (u'Hale\u2018iwa', 3),
 (u'Kaneohe', 3),
 (u'10', 2),
 (u'20', 2),
 (u'4', 2),
 (u'48', 2),
 (u'50', 2),
 (u'6', 2),
 (u'Hauula', 2),
 (u'Waimanalo', 2),
```

Top 10 Tourist ameneties

In [32]:
```
# Top 10 tourist ameneties
import pprint
cur.execute ("SELECT tags.value, COUNT(*) as count FROM (SELECT * FROM nodes_tags UNION ALL \
             SELECT * FROM ways_tags) tags \
             WHERE tags.key LIKE '%tourism'\
             GROUP BY tags.value \
             ORDER BY count DESC LIMIT 10;")
pprint.pprint(cur.fetchall())
```

```
[(u'attraction', 96),
 (u'hotel', 52),
 (u'viewpoint', 34),
 (u'museum', 24),
 (u'picnic_site', 15),
 (u'artwork', 10),
 (u'guest_house', 6),
 (u'information', 6),
 (u'hostel', 5),
 (u'camp_site', 4)]
```

Number of restaurants by metropolitan area

In [33]:
```
# Number of restaurants by metro area
import pprint
cur.execute("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 = 'city'\
            GROUP BY nodes_tags.value ORDER BY num DESC;")
pprint.pprint(cur.fetchall())
```

```
[(u'Honolulu', 48),
 (u"Hale'iwa", 2),
 (u'Kapolei', 2),
 (u'Haleiwa', 1),
 (u'Honlulu', 1),
 (u'Honolulu, HI', 1),
 (u'Kailua', 1),
 (u'Waipahu', 1)]
```

Top 10 types of food

In [34]:
```
# Top 10 types of food
conn = sqlite3.connect(sqlite_file)
cur = conn.cursor()
cur.execute('''
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;
''')
all_rows = cur.fetchall()
print('1):')
pprint.pprint(all_rows)
conn.commit()
```

```
1):
[(u'japanese', 14),
 (u'pizza', 10),
 (u'american', 9),
 (u'chinese', 9),
 (u'regional', 8),
 (u'sushi', 7),
 (u'indian', 6),
 (u'asian', 5),
 (u'italian', 5),
 (u'international', 4)]
```

Summary

Although this process went a long way to cleaning and standardizing the data, there are clearly several more steps that are required to ensure the data is completely accurate and internally consistent.

Further, there seems to be several instances in which data is either missing or incorrectly added. For example, the data review showed that there are 48 restaurants in the Honolulu area, however, even a cursory web search would indicate at many times that number. Other examples of this sort of inaccuracy are also present.

This exercise also exposed the limits of the sort of labels used in OSM datasets. Tourist ameneties did not include Beaches, Parks, surfing supplies, snorkeling equipment rentals, or other attractions closely associated with Hawai'i.

Lastly, the OSM project is a wonderful example of both the benefits and shortcomings of depending on independent users contributing information. This sort of "crowdsourcing" strategy provides the opportunity for anyone to contribute information that may either be missing or needs to be updated. Unfortunately, the lack of standards or uniform requirements does result in data that is often quite messy and inconsistent.

In [ ]: