OpenStreetMap Data Case Study

WGU C750

Kathryn Haynes

Map Area

Houston, TX, United States

http://www.openstreetmap.org/relation/2688911

I am assessing the area of OSM data of Houston, Texas for two reasons, one it is the town that I got my first job in after receiving my Associates Degree, and two because it is the closest metropolitan area to my home town.

Problems Encountered in the Map

After reviewing a small subset of the Houston data, I noticed the following main problems in the data set:

Problem 1: Several of the files had inconsistent names for street type (e.g., Ave, Ave. for Avenue).

Problem 2: Many zip codes contained nine digits vs. the standard five-digit format.

Specifically, I sought to correct problems 1 & 2.

Problem 1: Inconsistent street types

To audit the data, first I did four things: (1) used a helper function to analyze every 5 rows of data, (2) built a list of expected street types, (3) used regular expression to scan the last word in the street name, and (4) created and printed a dictionary of all street types that were not in the expected dictionary.

```
def update_street_name(street_name, mapping):
    street_name = street_name.replace(' ', ' ')
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        street_type2 = ' '.join(street_name.split()[0:])
```

```
        if street_type in mapping.keys():
            #print 'Before: ' , name
            street_name = re.sub(street_type, mapping[street_type],street_name)
            #print 'After: ', name

    return street_name
```

Printing the dictionary that resulted from this auditing, I quickly realized that there were 3 major types of issues: (1) inconsistent capitalization, (2) streets with numbers at the end (e.g., 18th floor), and (3) streets with no street type. To resolve (1), I built a mapping dictionary with the wrong types as keys and the corrected types as values:

```
mapping = { "St": "Street",
            "St.": "Street",
            "Ave" : "Avenue",
            "Ave.": "Avenue",
            "Rd.": "Road",
            "Rd": "Road",
            "E" : "East",
            "W" : "West",
            "S" : "South",
            "N" : "North"
            }
```

Regarding issue (2), numbers at the end of the street name, I ultimately elected to leave most of these alone as I felt values like "Suite 303" and "18th floor" could be important parts of the address for some businesses. In one case, though ("Kendall Square - 3"), the number didn't make much sense based on my intuition, and I opted to truncate to remove the number entirely.

Problem 2: Inconsistent postal codes (zip codes)

The issues in the postal code field were much simpler to resolve, overall. I iterated over all of the zip codes and made sure they were 5 digits.

```
# Here we iterate over the postal codes making them a uniform 5 digit number

def update_postal_code(postal_code):
    postal_code = postal_code.upper()
    if ' ' not in postal_code:
        if len(postal_code) != 5:
            postal_code = postal_code[0:5]
    return postal_code
```

Data Overview:

Audit.ipynb	5.79kB
create_DB	32.1kB
OSM_To_CSV.ipynb	17.4kB
data.py	13.9kB
schema.py	2.56kB
nodes.csv	98MB
Nodes_tags.csv	2.11MB
Ways.csv	9.49MB
Ways_tags.csv	17.4MB
Houston_Tx.db	184MB
HoustonTex.xml	1.32GB
Houston_Texas_Sample.osm	267MB

Number of nodes and ways

In [7]:
```
# number of nodes

cur.execute('SELECT COUNT(*) FROM nodes')
all_rows = cur.fetchall()
print(all_rows)
# number of ways
cur.execute('SELECT COUNT(*) FROM ways')
all_rows = cur.fetchall()
print(all_rows)
```

```
[(1167464,)]
[(158927,)]
```

Top 3 users

In [8]:

```
#Top 3 users
QUERY = '''
SELECT DISTINCT nodes.user, COUNT(*)
FROM nodes
GROUP BY nodes.uid
ORDER BY COUNT(*) DESC
LIMIT 3;
'''

cur.execute(QUERY)
all_rows = cur.fetchall()
print(all_rows)
```

```
[('afdreher', 91474), ('scottyc', 81680), ('skquinn', 78720)]
```

Percent of Nodes users

In [17]:

```
# % of nodes users
QUERY = '''
SELECT DISTINCT nodes.user, COUNT(*) * 100.0 / (SELECT COUNT(*) FROM nodes)
FROM nodes
GROUP BY nodes.uid
ORDER BY (COUNT(*) * 100.0 / (SELECT COUNT(*) FROM nodes)) DESC
LIMIT 10;
'''

cur.execute(QUERY)
all_rows = cur.fetchall()
print(all_rows)
```

```
[('afdreher', 7.835273721502333), ('scottyc', 6.996361343904394), ('skquinn', 6.742820335359378), ('woodpeck_fixbot', 5.737821466015226), ('TexasNHD', 3.3022859805527194), ('cammace', 2.985531031363708), ('m_with_z', 2.2261071861744774), ('clay_c', 2.177026443641945), ('antiviro', 1.880229283301241), ('Memoire', 1.7700759937779666)]
```

Most repeated street names

In [9]:

```
#Street names
QUERY = '''
SELECT ways_tags.value, COUNT(*)
FROM ways_tags
WHERE ways_tags.key = 'name'
AND ways_tags.type = 'regular'
GROUP BY ways_tags.value
ORDER BY COUNT(*) DESC
LIMIT 10;
'''

cur.execute(QUERY)
all_rows = cur.fetchall()
print(all_rows)
```

```
[('West Little York Road', 67), ('West Road', 65), ('Mason Road', 65), ('Cypresswood Drive', 64), ('North Eldridge Parkway', 63), ('Westheimer Road', 62), ('Eastex Freeway Frontage Road', 62), ('Eastex Freeway', 61), ('Barker Cypress Road', 61), ('TC Jester Boulevard', 58)]
```

Average ways nodes

In [10]:

```
#Average of ways
QUERY = '''
SELECT AVG(Count)
FROM
    (SELECT COUNT(*) as Count
    FROM ways
    JOIN ways_nodes
    ON ways.id = ways_nodes.id
    GROUP BY ways.id);
'''

cur.execute(QUERY)
all_rows = cur.fetchall()
print(all_rows)
```

```
[(8.823025665871752,)]
```

Top 10 amenities

In [11]:

```
#Amenities
QUERY = '''
SELECT value, COUNT(*) as Count
FROM nodes_tags
WHERE key='amenity'
GROUP BY value
ORDER BY Count DESC
LIMIT 10;
'''

cur.execute(QUERY)
all_rows = cur.fetchall()
print(all_rows)
```

```
[('parking_space', 404), ('place_of_worship', 372), ('fountain', 303), ('restaurant', 243), ('fast_food', 177), ('bench', 157), ('parking', 87), ('waste_basket', 71), ('fuel', 71), ('school', 59)]
```

Top religions

In [12]:

```
#Religion
QUERY = '''
SELECT nodes_tags.value, COUNT(*) as Count
FROM nodes_tags
JOIN
    (SELECT DISTINCT(id)
    FROM nodes_tags
    WHERE value='place_of_worship') as Sub
ON nodes_tags.id=Sub.id
WHERE nodes_tags.key='religion'
GROUP BY nodes_tags.value
ORDER BY Count DESC;
'''

cur.execute(QUERY)
all_rows = cur.fetchall()
print(all_rows)
```

```
[('christian', 359), ('muslim', 6), ('buddhist', 5), ('jewish', 1), ('hindu', 1)]
```

Conclusion

As a result of going through this data, I cleaned up a number of things, including inconsistent street types and inconsistent postal codes. Furthermore, while cleaning these two fields, I also observed that the city field could use data auditing and cleaning as well if I had additional time. Finally, I iteratively wrote the cleaned data to CSVs, loaded the CSVs into a SQL database, and used a variety of queries to understand the structure and nature of the data set. I would like to further investigate the data set to see different types or cuisine and popular places to visit with children.