OpenStreetMap Data Case Study - Saginaw, Texas

Map Area

Saginaw, TX, United States

- https://www.openstreetmap.org/relation/6571681 (https://www.openstreetmap.org/relation/6571681)
- https://www.google.com/maps/place/Saginaw,+TX (https://www.google.com/maps/place/Saginaw,+TX)

This is a map of my hometown and I thought it would be interesting to see what the data revealed. I also thought the familiarity with street names and various locations would help me see problems in the dataset.

Problems Encountered in the Map

After downloading my initial OSM file of the Saginaw area, I condensed it into a smaller, more workable size to look at some problems in the dataset. There were a few problems areas that I wanted to address.

Inconsistent Street Naming Conventions

The first problem I encountered was that the dataset has inconsistent street naming conventions (indicated by a 'k' value of 'addr:street' in my OSM file). For one example, here are the counts from various ways 'Boulevard' was input in the dataset:

```
Blvd: 16
BLVD: 17
Blvd.: 4
BLVD.: 2
Boulevard: 664
```

To correct for these inconsistencies, I created a dictionary called `mapping` that held the incorrect street name endings as keys and their corrected names as its values. I then created a list called `expected` that held the proper name endings. If a street name ending did not exist in my expected list, it was replaced by its associated value from the `mapping` dictionary. I used a regular expression to extract the street name ending for this comparison. I choose to ignore street endings that were numeric because there are valid streets that end in a number, for example 'Highway 287'. Below is a code snippet from my `update_street` naming function located in `update_functions.py` .

```
#this function will update street name endings to format all data to a standard typ
e of street address
def update_street(name, mapping):
    m = street_type_re.search(name)
    if m:
        street_type = m.group()
        try:
            if int(street_type):
                return name.title()
        except:
            if street_type not in expected:
                name = re.sub(street_type, mapping[street_type], name)
                return name.title()
            else:
                return name.title()
```

Below is a query from my finalized database, `SaginawDB.db` , that shows we now have 0 abbreviated versions for street endings that should be 'Boulevard'.

In [1]:
```
import sqlite3
import os
import math
db=sqlite3.connect("SaginawDB.db")
```

In [2]:
```
c=db.cursor()
streets="""
Select Count(*)
From (Select value From nodes_tags where key="street"
Union Select value From ways_tags where key="street") AS v
Where value LIKE "%BLVD" or value LIKE "%Blvd" or
value LIKE "%Blvd." or value LIKE "%BLVD.";
"""
c.execute(streets)
for c in c.fetchall():
    print(c[0])
```

```
0
```

Inconsistent Location Naming Conventions

Several categories have unusual naming conventions that included an underscore '_' in place of a space (indicated by the following 'k' values in my OSM file: 'denomination', 'amenity', 'tourism', 'leisure'). These categories also used all lower case to identify proper nouns like places of business. Here are a few examples:

```
'church_of_christ'
'roman_catholic'
'community_centre'
'charging_station'
'waste_disposal'
'dog_park'
'golf_course'
```

To correct these irregularities and properly format using a standard naming convention, I replaced all occurrences of an underscore with a space and capitalized the first letter of each word following normal proper noun naming conventions. My `fix_name` function below (located in `update_functions.py`), executes these changes and utilizes the built in `title()` function for capitalization. I also used my `is_proper_noun` function (located in `helper_functions.py`), to correct this naming convention for the 'denomination', 'leisure', 'tourism', and 'amenity' categories.

```
def is_proper_noun(elem):
    return (elem.attrib['k'] == "denomination" or elem.attrib['k'] == "leisure" or
elem.attrib['k'] == "tourism" or \
           elem.attrib['k'] == "amenity")
```

```
def fix_name (name):
    if '_' in name:
        return name.replace('_',' ').title()
    else:
        return name.title()
```

Here is one query from my finalized database that shows the corrected naming conventions for the denomination category.

In [3]:
```
c=db.cursor()
denomination="""
Select distinct v.value
From (Select value From nodes_tags where key="denomination"
Union ALL Select value From ways_tags where key="denomination") AS v
LIMIT 10;
"""
c.execute(denomination)
for c in c.fetchall():
    print(c[0])
```

```
Baptist
Pentecostal
Methodist
Seventh Day Adventist
Mormon
Jehovahs Witness
Church Of Christ
Evangelical
Roman Catholic
```

Inconsistent Units of Speed

There are inconsistent units for speed limits (indicated by a 'k' value of 'maxspeed' in my OSM file). Most records used the American standard 'mph' for miles per hour, but others just indicated a number without an associated unit. Here is an example:

```
{'30 mph', '70 mph', '65 mph', '10 mph', '45 mph', '30', '75 mph', '5 mph', '60 mp
h', '50 mph', '35 mph', '15 mph', '20 mph', '40 mph'}
```

To correct for this, I first verified the category was for 'maxspeed', using the `is_maxspeed` function (located in `helper_functions.py`). Then I created a `verify_speed` function (located in `update_functions.py`) and used a regular expression to extract the units from the 'maxspeed' category. If the category could be converted to an 'int' and therefore did not include a unit of speed, I then added the 'mph' unit.

```
#function will test if tag has a 'k' attribute that indicates it is a speed limit
def is_maxspeed(elem):
    return (elem.attrib['k'] == "maxspeed")

#this function will verify the maxspeed attribute is the standard mph
def verify_speed(name):

    m = speed_type_re.search(name)
    if m:
        speed_type = m.group()
        try:
            if int(speed_type):
                speed_type = speed_type + ' mph'
                return speed_type
        except:
            return name
```

Here is a query from my finalized database that shows the corrected speed limits.

In [4]:
```
c=db.cursor()
maxspeed="""
Select distinct v.value
From (Select value From nodes_tags where key="maxspeed"
Union Select value From ways_tags where key="maxspeed") AS v
LIMIT 10;
"""
c.execute(maxspeed)
for c in c.fetchall():
    print(c[0])
```

```
10 mph
15 mph
20 mph
30 mph
35 mph
40 mph
45 mph
5 mph
50 mph
60 mph
```

Overview of the Data

This section contains basic statistics about the dataset and the SQL queries used to gather them.

File Sizes

In [5]:
```
def get_file_size(filename):
    file_size = math.ceil(os.path.getsize(filename)/1024)
    return str(file_size)

print('Saginaw.osm {:>12} KB'.format(get_file_size('Saginaw.osm')))
print('SaginawDB.db {:>11} KB'.format(get_file_size('SaginawDB.db')))
print('ways.csv {:>13} KB'.format(get_file_size('ways.csv')))
print('nodes.csv {:>13} KB'.format(get_file_size('nodes.csv')))
print('nodes_tags.csv {:>6} KB'.format(get_file_size('nodes_tags.csv')))
print('ways_tags.csv {:>8} KB'.format(get_file_size('ways_tags.csv')))
print('ways_nodes.csv {:>8} KB'.format(get_file_size('ways_nodes.csv')))
```

```
Saginaw.osm       144353 KB
SaginawDB.db      158119 KB
ways.csv          5593 KB
nodes.csv         59920 KB
nodes_tags.csv    245 KB
ways_tags.csv     7840 KB
ways_nodes.csv    17877 KB
```

Number of Unique Users

In [6]:
```
c=db.cursor()
unique_users='Select Count(uniq_users.uid) From (Select uid From nodes UNION S
elect uid From ways) AS uniq_users;'
c.execute(unique_users)
print(c.fetchone()[0])
```

```
609
```

Number of Nodes

In [7]:
```
nodes_count='Select Count(*) From nodes;'
c.execute(nodes_count)
print(c.fetchone()[0])
```

```
640401
```

Number of Ways

In [8]:
```
ways_count='Select Count(*) From ways;'
c.execute(ways_count)
print(c.fetchone()[0])
```

```
80008
```

Top 10 Leisures

In [9]:
```
c=db.cursor()
top_leisure="""
Select distinct v.value, COUNT(*) as num
From (Select value From nodes_tags where key="leisure"
Union ALL Select value From ways_tags where key="leisure") AS v
GROUP BY value
ORDER BY num DESC
LIMIT 10;
"""
c.execute(top_leisure)
for c in c.fetchall():
    print(c[0], c[1])
```

```
Pitch 114
Park 60
Playground 25
Swimming Pool 17
Garden 10
Bleachers 6
Sports Centre 5
Golf Course 3
Common 3
Picnic Table 2
```

Top 10 Amenities

In [10]:
```
c=db.cursor()
top_amenities="""
Select distinct v.value, COUNT(*) as num
From (Select value From nodes_tags where key="amenity"
Union ALL Select value From ways_tags where key="amenity") AS v
GROUP BY value
ORDER BY num DESC
LIMIT 10;
"""
c.execute(top_amenities)
for c in c.fetchall():
    print(c[0], c[1])
```

```
Parking 454
Place Of Worship 103
Restaurant 92
Fast Food 78
School 68
Fuel 48
Grave Yard 47
Waste Disposal 23
Bank 21
Shelter 13
```

Top 10 Areas of Tourism

In [11]:
```
c=db.cursor()
tourism="""
Select distinct v.value, COUNT(*) as num
From (Select value From nodes_tags where key="tourism"
Union ALL Select value From ways_tags where key="tourism") AS v
GROUP BY value
ORDER BY num DESC
LIMIT 10;
"""
c.execute(tourism)
for c in c.fetchall():
    print(c[0], c[1])
```

```
Hotel 26
Motel 19
Artwork 8
Attraction 7
Picnic Site 6
Museum 3
Information 2
Trail Riding Station 1
```

Top 10 Denominations

In [12]:
```
c=db.cursor()
denomination="""
Select distinct v.value, COUNT(*) as num
From (Select value From nodes_tags where key="denomination"
Union ALL Select value From ways_tags where key="denomination") AS v
GROUP BY value
ORDER BY num DESC
LIMIT 10;
"""
c.execute(denomination)
for c in c.fetchall():
    print(c[0], c[1])
```

```
Baptist 40
Methodist 7
Pentecostal 6
Roman Catholic 3
Seventh Day Adventist 2
Mormon 2
Jehovahs Witness 1
Evangelical 1
Church Of Christ 1
```

Most Common Fast Food Restaurant

In [13]:
```
c=db.cursor()
most_fast_food="""
SELECT ways_tags.value, COUNT(*) as num
FROM ways_tags
JOIN (SELECT DISTINCT(id) FROM ways_tags WHERE value='Fast Food') i
ON ways_tags.id=i.id
WHERE ways_tags.key='name'
GROUP BY ways_tags.value
UNION
SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='Fast Food') j
ON nodes_tags.id=j.id
WHERE nodes_tags.key='name'
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 10;
"""
c.execute(most_fast_food)
for c in c.fetchall():
    print(c[0], c[1])
```

```
Whataburger 7
McDonald's 6
Chick-fil-A 3
Panda Express 3
Sonic 3
Taco Bell 3
Taco Cabana 3
Braum's 2
Firehouse Subs 2
In-N-Out Burger 2
```

Most Common Dine-In Restaurant

In [14]:
```
c=db.cursor()
most_restaurant="""
SELECT ways_tags.value, COUNT(*) as num
FROM ways_tags
JOIN (SELECT DISTINCT(id) FROM ways_tags WHERE value='Restaurant') i
ON ways_tags.id=i.id
WHERE ways_tags.key='name'
GROUP BY ways_tags.value
UNION
SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='Restaurant') j
ON nodes_tags.id=j.id
WHERE nodes_tags.key='name'
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 10;
"""
c.execute(most_restaurant)
for c in c.fetchall():
    print(c[0], c[1])
```

```
Waffle House 3
Chili's 2
Denny's 2
Newk's Eatery 2
On The Border 2
Alba's Italian Restaurant 1
An Zen Asian Dining 1
BoomerJack's Grill and Bar 1
Boopa's Bagel and Deli 1
Bosses Pizza and Pasta 1
```

Additional Ideas

Of the 609 unique users that contributed to the dataset, I was curious who were the top contributing users. I explored this in the code below.

In [15]:
```
c=db.cursor()
top_user_contributions = """
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;
"""
c.execute(top_user_contributions)
for c in c.fetchall():
    print(c[0],c[1])
```

```
Andrew Matheny_import 574348
Andrew Matheny 20183
woodpeck_fixbot 13019
Thea Clay 13011
houston_mapper1 10591
Юкатан 5097
RoadGeek_MD99 4746
karl-marx 4497
bournkev12 3757
T Smiley 2932
```

I would say it is safe to assume 'Andrew Matheny_import' and 'Andrew Matheny' are probably the same person. Meaning this one user contributed to 594531 of the total pieces of data in the dataset. So then I wanted to determine what percent of the total this one user contributed.

In [16]:
```
c=db.cursor()
total='Select Count(*) From (SELECT user FROM nodes UNION ALL SELECT user FROM
ways) total;'
c.execute(total)
print(c.fetchone()[0])
```

```
720409
```

Out of the 720409 entries in total, Andrew Matheny contributed 594531, which is approximately 82.53% of the entire dataset. If we take the contributions from the top 5 contributors, we see that more than 88% of our data was collected from 5 sources, one of which has the name 'fixbot'. Considering a majority of our dataset is from a handful of users, it would be beneficial if more users were contributing towards a more complete and accurate OpenStreetMap data set. Limiting the data to a handful of users can lead to more inaccuracies in the dataset. If one of these predominate users continuously added erroneous data, then a large percentage of the dataset would reflect these inaccuracies. Having a more diversified set of users could help eliminate this issue. More users entering and editing data that was incorrect would mean the overall dataset was more accurate. Also having more users contributing would mean a wider range of streets and places would be included. Additional users, each contributing a small amount, would lead to less work for everyone and more attention could be spent on accuracy and minute details.

The challenge is finding enough people who are willing to do stuff for the advancement of technology and the greater whole, without being financially compensated. One idea I had for this would be to try to involve the local community colleges or universities. When I was attending my local community college, I had a professor who was very involved in helping the community, while simultaneously providing her students with opportunities to get hands on experience with web development and general IT exposure. This type of collaborative effort from a local community college could be exactly what the OpenStreetMap project could benefit from. If some entry level computer science courses had their students register for the OSM project and contribute to the data around the areas they live, work, and go to school, it would further the OSM project and provide more accurate data. It could be beneficial for the students as well by having real life exposure to some coding assignments. The data cleaning aspect of any category I refined in my project would be perfect coding projects or assignments for entry level computer science students. I'm sure most professors could find a way to integrate some aspect of the OSM project into their curriculum while simultaneously contributing to the OSM project. I know there would be challenges in finding local colleges and universities to participate; as well as challenges integrating such a project into a normal curriculum, but it could be mutually beneficial to all parties involved.

Conclusion

After spending a decent amount of time looking through and refining this dataset, I can see there is quite a bit of work to be done. There are numerous inconsistencies for both categories and the formatting of data in these categories. Numerous fields exist for something a simple as a zipcode: zip_left, zip_right, and addr:postcode just to name a few. Not to mention inconsistencies within these categories, such as variations in units used. It was interesting to dig into some of this data and see categories like the number of fast food restaurants, with my favorite, Whataburger coming out on top. It was also not a surprise to me to see very little in the means of tourism. The top two from this category were hotels and motels. Saginaw is a fast-growing suburb of Fort Worth, but it is still small in the grand scheme of things.

References

https://stackoverflow.com/questions/6591931/getting-file-size-in-python (https://stackoverflow.com/questions/6591931/getting-file-size-in-python)

https://www.w3schools.com/python/ref_string_format.asp (https://www.w3schools.com/python/ref_string_format.asp)

https://gist.github.com/carlward/54ec1c91b62a5f911c42#file-sample_project-md (https://gist.github.com/carlward/54ec1c91b62a5f911c42#file-sample_project-md)

In [17]:
```
db.close()
```