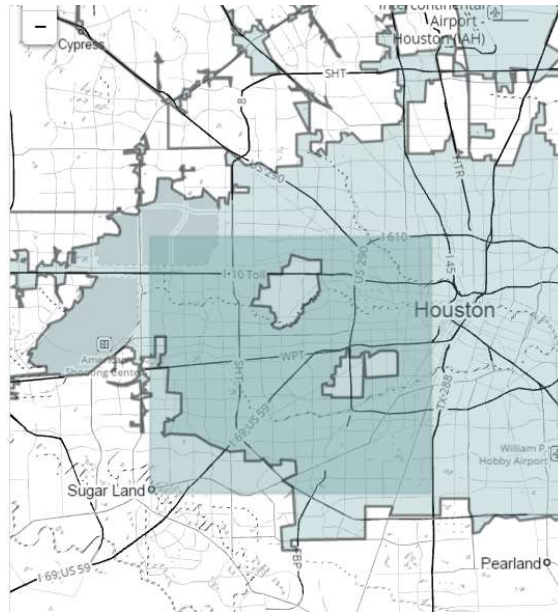


DAND P3: Data Wrangling of OpenStreet Map

Map Area:

The area studied in this project was taken from Houston, TX, United States. Since Houston is a very spread-out city, only southwest area of the city is included in the file. This is because I reside in this area and it would be easier for me to check accuracy of addresses and names.



Problems of Data in Map

A short version of OSM file was created using the python code provided by the project. The auditing procedure was tested on the sample OSM file, and then applied to the original file. Several problems were found.

Unexpected Key Names

The key names were checked against the four categories: “lower”, “lower_colon”, “problemchar” and “other”. No problematic string was found, but there were some in the “other” category:

```
{'lower': 1512, 'lower_colon': 1707, 'other': 61, 'problemchars': 0}
```

```
gnis:ST_alpha  
gnis:County_num  
FIXME  
name_1  
tiger:name_base_1  
tiger:name_type_1  
NHS  
tiger:name_base_1  
note:toll:hov
```

By listing out some of the key names in “other” category, we get an idea of what these are and how they look like inside of the XML file. They are mostly nodes generated by GPS such as “gnis” or “tiger”, and were divided into segments.

```
<tag k="gnis:ST_alpha" v="TX" />
<tag k="gnis:County_num" v="201" />
```

Various Types of Problematic Street Names

1. Some street names ended with numerical characters such as follows:

```
'704': set(['Memorial City Way #704']),
'77027': set(['77027']),
'77096': set(['Meyerland Plaza, Houston, TX 77096']),
'90a': set(['Hwy 90a']),
'925': set(['Katy Freeway Suite 925']),
```

These seem to be three types of inputs:

- Unit number, which is correct info and should be kept.
- The street name is a number indeed, such as ‘90a’, which should be kept as is.
- Zip codes due to incorrect input of addresses such as the following:

```
<tag k="name" v="Bikram Yoga Meyerland Plaza"/>
<tag k="leisure" v="sports_centre"/>
<tag k="alt_name" v="BYMP"/>
<tag k="addr:street" v="Meyerland Plaza, Houston, TX 77096"/>
<tag k="designation" v="Yoga Instruction"/>
<tag k="addr:postcode" v="77096"/>
<tag k="addr:housename" v="Bikram Yoga Meyerland Plaza"/>
<tag k="addr:housenumber" v="410"/>
```

2. Some street types were abbreviated or omitted, such as in the following:

```
'Ave.': set(['Bertner Ave.']),
'B': set(['Richmond Ave, Ste B', 'W Holcombe Blvd #B']),
'Beechnut': set(['Beechnut']),
'Blossom': set(['Blossom']),
'Blvd': set(['John Freeman Blvd', 'Post Oak Blvd']),
```

Incorrect Postal Code

There were 2 instances of incorrect postal code as follows:

```
Incorrect ZipCode:
['Weslayan Street', '7-']
```

The first one was an input error which swapped the street name and postal code:

```
<tag k="atm" v="yes"/>
<tag k="name" v="Chase"/>
<tag k="amenity" v="bank"/>
<tag k="addr:street" v="77027"/>
<tag k="addr:postcode" v="Weslayan Street"/>
<tag k="addr:housenumber" v="2900"/>
```

The second one is a missing information. After research, the missed postal code should be “77478”.

```
<tag k="name" v="Berryhill Tacos Sugarland"/>
<tag k="amenity" v="restaurant"/>
<tag k="cuisine" v="mexican"/>
<tag k="addr:city" v="Sugar Land"/>
<tag k="addr:state" v="TX"/>
<tag k="addr:street" v="Southwest Freeway"/>
<tag k="addr:postcode" v="7-"/>
<tag k="addr:housenumber" v="13703"/>
```

Data Cleaning Methods

Street Names

Three types of corrections had been done to the street names:

1. Fixing abbreviation in street types: using the mapping method
2. Adding missing street types
3. postal code mistaken as street names

The method used is to first match the individual cases to replace them with correct ones. Then expend the abbreviations against the common usages. The corrected names were obtained by replacing the original string with the mapped strings in the mapping dictionary.

```
def update_street(name, mapping_streetName, mapping_streetType):
    # Initialize Return
    corr_name=name
    # Correct idiosyncratic cases:
    if name in mapping_streetName:
        corr_name=mapping_streetName[name]
    #Expanding Common Abbr.
    else:
        words=name.split()
        for i in range(len(words)) :
            if words[i] in mapping_streetType:
                words[i]=mapping_streetType[words[i]]
        corr_name=" ".join(words)
    return corr_name
```

Postal Code








were idiosyncratic errors and had to be treated specifically for each individual case. This was done through mapping.

```
def update_postcode(postcode, mapping_postcode):
    corr_postcode=postcode
    if not postcode_re.match(postcode):
        if postcode in mapping_postcode:
            corr_postcode=mapping_postcode[postcode]
    return corr_postcode
```

Database in SQL

File Sizes

The following “csv” files were created with the schema as in “schema.py”. The “csv” files had been imported with python DB-API calls. A SQL database has been created as ‘HoustonSW_OSM.db’. The sizes of the files are shown as follows:

 HoustonSW.osm	10/31/2016 5:35 PM	OSM File	72,281 KB
 nodes.csv	11/4/2016 9:25 PM	Microsoft Excel Co...	23,564 KB
 nodes_tags.csv	11/4/2016 9:25 PM	Microsoft Excel Co...	970 KB
 ways.csv	11/4/2016 9:25 PM	Microsoft Excel Co...	3,398 KB
 ways_nodes.csv	11/4/2016 9:25 PM	Microsoft Excel Co...	9,201 KB
 ways_tags.csv	11/4/2016 9:25 PM	Microsoft Excel Co...	10,657 KB
 HoustonSW_OSM.db	11/4/2016 11:46 PM	Data Base File	43,415 KB

Number of Nodes

Query was sent in DB-API Playground as follows:

```
c.execute('''
SELECT COUNT(*) FROM nodes;
''')
all_rows = c.fetchall()
print 'number of nodes: ', all_rows
```

number of nodes: [(285952,)]

Number of Ways

Query was sent in DB-API Playground as follows:

```
c.execute('''
SELECT COUNT(*) FROM ways;
''')
all_rows = c.fetchall()
print 'number of ways: ', all_rows
```

number of ways: [(57639,)]

Number of Unique Users

```
c.execute('''
SELECT COUNT(DISTINCT(u.uid))
FROM (SELECT uid FROM nodes
UNION ALL SELECT uid FROM ways) as u;
''')
all_rows = c.fetchall()
print 'number of unique users: ', all_rows
```

number of unique users: [(427,)]

Number of Fast Food Restaurants

```
c.execute('''
SELECT COUNT(*)
FROM nodes_tags
WHERE key='amenity'
AND value='fast_food';
''')
all_rows = c.fetchall()
print 'number of fast_food restaurants: ', all_rows

number of fast_food restaurants: [(121,)]
```

Additional Data Exploration

Number of Used Nodes

Nodes were either used by node tags or to define a way. The following produce the total number of nodes that are being used.

```
c.execute('''
SELECT COUNT(DISTINCT(used_nodes.nodeId))
FROM (SELECT nodeId from nodes_tags
UNION ALL SELECT nodeId FROM ways_nodes) as used_nodes;
''')
all_rows = c.fetchall()
print 'number of used nodes: ', all_rows

number of used nodes: [(285946,)]
```

Number of Shared Nodes

The following show the number of nodes that are used both in node tags and ways:

```
c.execute('''
SELECT COUNT(DISTINCT(nodeId))
FROM nodes_tags
WHERE nodeId in (SELECT DISTINCT(nodeId) FROM ways_nodes);
''')
all_rows = c.fetchall()
print 'number of shared node ', all_rows

number of shared node [(8483,)]
```


What type of nodes are these?

```
c.execute('''
SELECT nodes_tags.value, COUNT(DISTINCT(nodeId)) as num
FROM nodes_tags
WHERE nodeId in (SELECT DISTINCT(nodeId) FROM ways_nodes)
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 10;
''')
all_rows = c.fetchall()
print 'shared nodes '
pprint.pprint(all_rows)
```

```
shared nodes
[(u'traffic_signals', 1996),
 (u'turning_circle', 1663),
 (u'tower', 1618),
 (u'crossing', 885),
 (u'pole', 742),
 (u'yes', 519),
 (u'gate', 392),
 (u'level_crossing', 285),
 (u'uncontrolled', 215),
 (u'motorway_junction', 168)]
```

These are mostly element on the roads, such as traffic signals, junctions or gates.

What street has the most traffic lights?

```
# This part of the code take a long time to finish
c.execute('''
SELECT ways_tags.value, COUNT (ways_nodes.nodeId) as num
FROM (ways_nodes JOIN nodes_tags
ON ways_nodes.nodeId=nodes_tags.nodeId)
JOIN ways_tags
ON (ways_tags.wayId=ways_nodes.wayId)
WHERE nodes_tags.value=="traffic_signals" AND
      ways_tags.key="name"
GROUP BY ways_tags.value
ORDER BY num DESC
LIMIT 10;
''')
all_rows = c.fetchall()
print 'Roads with most traffic Lights '
pprint.pprint(all_rows)
```

```
Roads with most traffic Lights
[(u'Westheimer Road', 198),
 (u'Richmond Avenue', 181),
 (u'Bellaire Boulevard', 147),
 (u'Bissonnet Street', 146),
 (u'Beechnut Street', 144),
 (u'Fondren Road', 100),
 (u'South Braeswood Boulevard', 98),
 (u'Southwest Freeway Frontage Road', 96),
 (u'Chimney Rock Road', 91),
 (u'Westpark Drive', 88)]
```

Is it possible to have 198 traffic lights on 'Westheimer'? It must have counted multiple lights at each cross.
(Please note this code take a long time to run.)

Other Ideas

Data Cleaning

1. It is interesting to notice that the number of contributed users found in XML file is 437, while the number got from SQL query is 427. This means there have been 10 users whose contributed data was discarded during the data cleaning. These were the entries that had key words that was problematic. They could be in a different language or script or simply input errors. These users can be notified to correct these entries.
2. The total number of nodes is 285952, while the number of nodes used either with a tag or in defining a way is 285946. This means there are 6 nodes that are useless in the map, which could have been deleted.
3. The cleaning process in this project only selected information about "node" and "way". The third major data type "relation" were excluded. As a future work, the relation data can be included into the database.

Data Exploration

If the relation data are cleaned and included in the dataset, more practical questions like the following can be answered:

1. Which neighborhood has the most running trails.?
2. Which plaza has the most restaurants?
3. Which bus stop is closest to the post office?

Conclusion

There are a lot of data in the area studies. This only covers about a quarter of the Houston Metropolitan area. Future work are needed to improve the accuracy and functionality of the map.