Project: Wrangle OpenStreetMap Data ¶

1) Gather Data

Austin, TX

https://www.openstreetmap.org/export#map=11/30.3080/-97.7488
(https://www.openstreetmap.org/export#map=11/30.3080/-97.7488)

This area contains the city of Austin, TX. I chose it b/c I've lived here for many years and I am somewhat familiar with the streets.

2) Audit Data

Auditing a sample of my city data via audit.py revealed some opportunity to cleaning up the street name abbreviations in my area, as well as perhaps some oppotunity in the original provided audit routines. I accounted for these by adjusting the logic a bit via streetnames.py, as well as making an adjustment in the excepted street-types.

Some abbreviations were being picked up by the code even when they occured at the beginning of a complete word, and the original update_names function was still performing a replacement, so I was ending up with names like Driveive, or Avenuenue. I accounted for this by checking whether a street name containing more than one word, ignoring those which did. However, if more than one word was found, I would only send that last word to the cleaning function, and then I would check that the entire word matched an entry in the street-name 'mapping' dictionary, rather than replacing a portion of the entire street name.

I found a lot of the data in my city to be very clean, but I chose to tackle cleaning up values in the "height" tags, as well as removing the last-four from longer zip-code values in the "postcode" tags.

As recommended, I made a sample osm file for use when developing. The routine for doing so is in the file make_example.py

I used the audit.py file, after making some minor adjustments, to review the contents of the sample data, however I did not use audit.py itself when actually writing to the CSV files. Instead I made custom cleaning functions in separate files.

3) Clean Data & Generate CSVs

In my data.py routine, I outsourced the cleaning routines to external files. I had to make a few minor updates for Python 3 in the code given. I think the approach to shape_element could be a little less verbose, but this gets the job done.

streetnames.py addresses variation in street-type abbreviations

heights.py removes string values from the height tag. Common strings found in the field included trailing apostrophes (indicating the unit of measurement to be feet), or the field ending in 'm' (indicating meters). It would be better to handle the values and units in separate tags, and there is no constency or validation of what unit is given in most values.

4) Import to DB

Using db.py,I passed the CSV files' values into an SQLite database, austin_osm.db.

5) Assess Data w/ SQL

I'm moving into this Jupyter Notebook now to complete the rest of the data assessment and documentation.

Following are some general metrics on the dataset

In [1]:
```
import sqlite3
OSM_DB = "austin_osm.db"

#connect to database
cnn = sqlite3.connect(OSM_DB)
cur = cnn.cursor()

#gather some stats
unique_users = cur.execute('select count(distinct user) from (select user from
nodes union all select user from ways) un').fetchone()[0]
number_nodes = cur.execute('select count(*) from nodes').fetchone()[0]
number_ways = cur.execute('select count(*) from ways').fetchone()[0]
number_address_nodes = cur.execute("select count(*) from nodes_tags where type
= 'addr'").fetchone()[0]
number_address_ways = cur.execute("select count(*) from ways_tags where type =
'addr'").fetchone()[0]

top_contributor = cur.execute('select u.user, count(*) as recs from (select us
er from nodes union all select user from ways) u group by user order by recs d
esc limit 1').fetchone()[0]
top_contributor_recs = cur.execute('select u.user, count(*) as recs from (sele
ct user from nodes union all select user from ways) u group by user order by r
ecs desc limit 1').fetchone()[1]

#output osm data stats
print(str(unique_users) + ' Unique Users')
print(str(number_nodes) + ' Nodes')
print(str(number_ways) + ' Ways')
print(str(number_address_nodes) + ' Address Nodes')
print(str(number_address_ways) + ' Address Ways')
print('Top Contributor is ' + top_contributor + ', with ' + str(top_contributo
r_recs) + ' records')

#close database
cur.close()
cnn.close()
```

```
2404 Unique Users
6598166 Nodes
718069 Ways
222721 Address Nodes
522386 Address Ways
Top Contributor is patisilva_atxbuildings, with 2640851 records
```

Following are some counts of some specific chosen types of nodes

In [17]:
```
#connect to database
cnn = sqlite3.connect(OSM_DB)
cur = cnn.cursor()

#gather count of some chosen node types
waterways = cur.execute("select count(*) from (select * from nodes_tags union
 all select * from ways_tags) u where key = 'waterway'").fetchone()[0]
footways = cur.execute("select count(*) from (select * from nodes_tags union a
ll select * from ways_tags) u where key = 'footway'").fetchone()[0]
cycleways = cur.execute("select count(*) from (select * from nodes_tags union
 all select * from ways_tags) u where key = 'cycleway'").fetchone()[0]
railways = cur.execute("select count(*) from (select * from nodes_tags union a
ll select * from ways_tags) u where key = 'railway'").fetchone()[0]
shops = cur.execute("select count(*) from (select * from nodes_tags union all
 select * from ways_tags) u where key = 'shop'").fetchone()[0]

print(str(waterways) + ' Waterways')
print(str(footways) + ' Footways')
print(str(cycleways) + ' Cycleways')
print(str(railways) + ' Railways')
print(str(shops) + ' Shops')

#all_nodes = cur.execute('select key, count(*) from (select * from nodes_tags
 union all select * from ways_tags) u group by key order by count(*) desc')
#for row in all_nodes.fetchall():
#    print(row)

#close database
cur.close()
cnn.close()
```

```
2325 Waterways
6431 Footways
2183 Cycleways
803 Railways
2041 Shops
```

And finally, some statis on the size of the various datasets.

In [18]:
```
#connect to database
cnn = sqlite3.connect(OSM_DB)
cur = cnn.cursor()

#gather db stats
nodes_size = cur.execute('select sum(pgsize) from dbstat where name="nodes"').
fetchone()[0]
nodes_tags_size = cur.execute('select sum(pgsize) from dbstat where name="node
s_tags"').fetchone()[0]
ways_size = cur.execute('select sum(pgsize) from dbstat where name="ways"').fe
tchone()[0]
ways_tags_size = cur.execute('select sum(pgsize) from dbstat where name="ways_
tags"').fetchone()[0]
ways_nodes_size = cur.execute('select sum(pgsize) from dbstat where name="ways
_nodes"').fetchone()[0]

#output db stats
print('nodes table size: ' + str(nodes_size) + ' bytes')
print('nodes_tags table size: ' + str(nodes_tags_size) + ' bytes')
print('ways table size: ' + str(ways_size) + ' bytes')
print('ways_tags table size: ' + str(ways_tags_size) + ' bytes')
print('ways_nodes table size: ' + str(ways_nodes_size) + ' bytes')

#close database
cur.close()
cnn.close()
```

```
nodes table size: 563580928 bytes
nodes_tags table size: 12005376 bytes
ways table size: 45637632 bytes
ways_tags table size: 75624448 bytes
ways_nodes table size: 157782016 bytes
```

6) Additional Documentation

Problems Encountered

In addition to the actual scrubbing of the data above, I had to make a number of minor adjustmenst to account for the changes from Python2 to Python3. Here's a brief ad hoc list of some issues I ran into, and the solutions to those.

```
make_map_sample throwing errors passing strings where bytes expected
    added .encode('utf8')
iteritems() changed to items() to accomodate Python 3
print 'text' changed to print('text') to accomodate Python 3
unicode changed to str to accomodate Python 3
UnicodeDecodeError: 'charmap' codec can't decode byte 0x8f in position 9380: charac
ter maps to <undefined>
    osm_file = open(osmfile, "r", encoding="utf8")
```

Additional Ideas

I did not spend a lot of time perusing the data elements in the tags, however it seemed that there was a sharp decline in the amount of interesting information when looking beyong just the streed addresses. It could be interesting to try bouncing this data up against other sources to acquire additional information about what is located at the address, ratings of those places, etc. Obviously, this is a large task, and it would present a good deal of difficulty attempting to match the datasets up together, though I imagine if that other dataset included the same geographical coordinates data, that would be the element used for this.

Within the data, I did note a lack of consistency in some basic number fields, like Height tags, where there was no indication whether the height was given in inches, feet, meters, etc, except for the occasional indication w/in the value field itself. It would be better to handle the value and the value's unit of measure either in separate, or in compound tags such as "height:feet".

In terms of working with the data available, to better analyze it, I believe developing out the database a bit more would be of benefit. A view to union-ize the tags tables would be handy for generating some overall stats (as seen above, I did the unions on the fly in subqueries). Some more logging information during the shape_element routine could also be leveraged to gather and store stats into additional datasets instead of just the raw data. It would also help for reporting out on the amount of reshaping actually done. That in turn could lead to insights for optimizing the routine, or reveal possible problems introduced by the routine.