Wrangling OpenStreetMap Data Project

Window10_64-bit OS, Python 2.7, Anaconda2

June 2017, by Jude Moon

Map Area

Blacksburg, VA, USA

- https://www.openstreetmap.org/relation/206542 (https://www.openstreetmap.org/relation/206542)

I went to school here and am living in this area since then. This is the area where I am the most familiar with street names and locations so I thought it would be a good starting point.

1. Data Audit

In [13]:
```
# Import modules
#import pandoc # to convert ipynb to pdf
import pprint
import audit # modularized py script
```

In [3]:
```
filename = 'bburg_map.xml'
```

In [4]:
```
total_tags = audit.count_tags(filename)
print "How many tag elements?"
print sum(total_tags.values())

print "\nHow many of each tag element?"
pprint.pprint(total_tags)
```

```
How many tag elements?
819599

How many of each tag element?
{'bounds': 1,
 'member': 11083,
 'meta': 1,
 'nd': 281754,
 'node': 270918,
 'note': 1,
 'osm': 1,
 'relation': 387,
 'tag': 233712,
 'way': 21741}
```

In [ ]:
```
tag_keys = audit.count_tag_keys(filename)
print """Among tag elements, there are tag elements with the name 'tag'.
What are the unique key name of 'tag' elements?"""
pprint.pprint(tag_keys)
```

Comment: this output of this cell was cleared because of the length

In [6]:
```
print "Number of unique key names appear at 'tag' elements:"
print len(tag_keys)

print "Number of total keys appear at 'tag' elements:"
print sum(tag_keys.values())

print "Number of total 'tag'' elements:"
print total_tags['tag']
```

```
Number of unique key names appear at 'tag' elements:
367
Number of total keys appear at 'tag' elements:
233712
Number of total 'tag'' elements:
233712
```

Comment: There are uppercase, lowercase, mixedcase keys. And some keys have number, one or two underscore(_), one or two colons(:). Note that a lot of keys with colons share the same words before colons so this implies the hierarchical structures for key names. We will consider breaking down into two columns for key names.

I want to categorize the tag keys into 4 types:

- "whthout_colons" for tags that contain letters(ignore case), number, and underscore and are valid,
- "with_colons" for otherwise with one or more colons in their names,
- "problemchars" for tags with space or problematic characters, and
- "other" for other tags that do not fall into the other three categories

In [7]:
```
tag_keys_types = audit.count_key_types(filename)

print "How many problematic key names?"
pprint.pprint(tag_keys_types)
print '\n'
print 'Number of total keys appear at tag element:'
print sum(tag_keys_types.values())
```

```
How many problematic key names?
{'other': 1, 'problemchars': 1, 'with_colons': 182620, 'without_colons': 51090}


Number of total keys appear at tag element:
233712
```

In [8]:
```
tag_keys_names = audit.key_type(filename)

print "Show me the problematic key names:"
pprint.pprint(tag_keys_names['problemchars'])
print '\n'
print "Show me the other key names:"
pprint.pprint(tag_keys_names['other'])
```

```
Show me the problematic key names:
set(['permanently closed'])


Show me the other key names:
set(['his:1973-:power'])
```

In [9]:
```
print "Show me the street types:"
audit.get_street_types(filename)
```

```
Show me the street types:
```

Out[9]:
```
{'Avenue',
 'Boulevard',
 'Circle',
 'Court',
 'Crossing',
 'Drive',
 'Lane',
 'North',
 'Northeast',
 'Northwest',
 'Pass',
 'Place',
 'Road',
 'Run',
 'South',
 'Southwest',
 'Square',
 'Street',
 'Terrace',
 'Trail',
 'Way'}
```

Street types are consistent in this map, which all are in full name, not in abbreviation

In [10]:
```
print "How many users?"
len(audit.get_user(filename))
```

```
How many users?
```

Out[10]:
```
275
```

2. Problems Encountered in the Map

- Uppercase tag keys: I changed regex to match uppercase as well as lowercase

  ```
  NHD:ComID
  ```

- Number in tag keys : I changed regex to match letters with _number, but this will not be reflected for cleaning process because separating the number from the key does not seem advantageous

  ```
  amenity_1
  ```

- Two colons in the keys: I changed regex to match one or more colons, so first word before the first colon becomes 'type' key and the rest of string becomes 'key'

  ```
  destination:ref:to -> {'type':'destination', 'key':'ref:to'}

  The benefit of this change is that we can analyze and make summary on nested key levels. But
 anticipated issues would be that too many nested keys can be unique keys that will not help t
o summarize any meaningful information, but increase noise on the data.
  ```

3. Data Overview

File sizes:

- bburg_map.xml: 60.851MB
- bburg_nodes.csv: 22.205MB
- bburg_nodes_tag.csv: 5.346MB
- bburg_ways.csv: 1.27MB
- bburg_ways_nodes.csv: 6.148MB
- bburg_ways_tags.csv: 2.086MB
- bburg_map.db: 33.184MB

Number of nodes:

sqlite> SELECT COUNT(*) FROM nodes;

```
270918
```

Number of ways:

sqlite> SELECT COUNT(*) FROM ways;

```
21741
```

Number of unique users:

sqlite> SELECT COUNT(DISTINCT(e.uid))
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e;

```
264
```

---

4. Data Exploration

Top 10 contributing users:

sqlite> 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;

```
mutantmonkey|156773
jumbanho|48958
Evanator|28366
dcat|17114
woodpeck_fixbot|9489
Spesh|4458
jbvejle|2743
Chris Lawrence|2415
Dayton_Poff|2161
bdiscoe|1831
```

Number of users appearing only once (having 1 post):

```
sqlite> 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;
```

```
47
```

Top 10 appearing amenities:

```
sqlite> SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key='amenity'
GROUP BY value
ORDER BY num DESC
LIMIT 10;
```

```
bench|187
recycling|169
bicycle_parking|141
restaurant|51
place_of_worship|43
fast_food|32
school|25
fuel|18
grave_yard|17
shelter|17
```

Biggest religion:

```
sqlite> SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='place_of_worship') i
ON nodes_tags.id=i.id
WHERE nodes_tags.key='religion'
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 1;
```

```
christian|41
```

Most popular cuisines:

```
sqlite> 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;
```

```
chinese|4
mexican|4
american|3
pizza|3
regional|3
italian|2
sandwich|2
Lebanese|1
brazilian|1
ethiopian|1
fine_dining|1
greek|1
hibachi|1
ice_cream|1
japanese|1
sushi|1
thai|1
vegetarian|1
```

5. Other ideas about the dataset

The map data in Blacksburg area were relatively clean overall, but as I looked into summary of columns, there were marginal errors (see other problems).

One thing that I am suspicious about the map data is no postal code of 24061 which is Virginia Tech zip code. Because Blacksburg is Virginia Tech driven town, I would think that there is a good number of nodes associated with the University (about 400). But I found one zip code of 24061-9517, so this totally doesn't make sense. Plus, the total number of postal code info (about 100) is very small with the population of 8 million. This could have happened because users input the address without postal code or they don't separate it into a tag for postal code.

This could be solved by having users' rules or instructions that show requirements of fields and if it does not include all the required information, notify the users or show a red flag of incomplete inputs.

other problems

- Inconsistency in Postal code

```
24060-3348|1
24061-9517|1
VA 24060|1
```

- Numbers in city name

```
10|9
14|9
22|6
24|6
```

- Inconsistency of case in city name

```
CHRISTIANSBURG|2
christiansburg|2
```

- Wrong category in city name

```
Virginia|2
```

Post Codes:

sqlite> SELECT tags.value, COUNT() *as count*
FROM (SELECT FROM nodes_tags
UNION ALL
SELECT * FROM ways_tags) tags
WHERE tags.key='postcode'
GROUP BY tags.value
ORDER BY count DESC;

```
24060|63
24073|58
24060-3348|1
24061-9517|1
VA 24060|1
```

Cities:

sqlite> 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;

```
Blacksburg|16243
Christiansburg|67
10|9
14|9
22|6
24|6
6|6
2|5
4|5
18|3
20|3
1176|2
2316|2
32|2
40|2
44|2
61|2
8|2
CHRISTIANSBURG|2
Virginia|2
christiansburg|2
110|1
```

Operator by Virginia Tech:

sqlite> SELECT nodes_tags.key, nodes_tags.value, COUNT(*) FROM nodes_tags JOIN nodes ON nodes_tags.id=nodes.id WHERE (nodes_tags.key= 'operator') and (nodes_tags.value='Virginia Tech');

```
operator|Virginia Tech|400
```

---

6. Conclusion

Wrangling dataset on the open street map of Blacksburg was interesting process. But I would be more vigorously investigate the dataset if I had particular questions about the data up front. It is hard to tell why I need to wrangle this dataset, what kind of information I need to extract, and which keys and values I should spend more time to investigate. The vague idea of improving the dataset overall was not helpful. If I do this process next time, I will make clear goals in uses of database I am working on.

7. Files

- bburg_map.xml: osm file renamed with xml extension
- audit.py: modularized py script including procedures used in this Jupyter Notebook
- process.py: standalone py script parsing/cleaning/shaping and converting document tree to 5 separated csv files
- map_schema.py: modularized py script including data schema for tabular format, used in process.py
- db.py: standalone py script building database with 5 tables from 5 csv files
- wrangling_osm_project_answers.ipynb: this document answering to the rubric questions
- wrangling_osm_project_answers.html: html version of this document
- wrangling_osm_project_answers.pdf: pdf printed of html answer file

8. Sources

- osm documentation: https://wiki.openstreetmap.org/wiki/Main_Page (https://wiki.openstreetmap.org/wiki/Main_Page)
- unzip osm file: http://www.e7z.org/open-bz2-bzip2.htm (http://www.e7z.org/open-bz2-bzip2.htm)
- ET.iterparse, root.clear: http://effbot.org/zone/element-iterparse.htm (http://effbot.org/zone/element-iterparse.htm)
- install new package in Anaconda: https://www.youtube.com/watch?v=_oNQKcNIBZs (https://www.youtube.com/watch?v=_oNQKcNIBZs)
- familarize with abbreviation of street type in US: http://www.gis.co.clay.mn.us/usps.htm (http://www.gis.co.clay.mn.us/usps.htm)
- reducing memory footprint: https://discussions.udacity.com/t/reducing-memory-footprint-when-processing-large-datasets-in-xml/37571/3 (https://discussions.udacity.com/t/reducing-memory-footprint-when-processing-large-datasets-in-xml/37571/3)
- yield generator: https://pythontips.com/2013/09/29/the-python-yield-keyword-explained/ (https://pythontips.com/2013/09/29/the-python-yield-keyword-explained/)