Wrangle OpenStreetMap Data

Prepared by: Jeff Daniels
Sep. 18th 2017

Map Area

Washington, District of Columbia, United States of America was chosen as the dataset because of my years spent living in and around this area. I chose to only study the area within the boundaries of the city because addresses generally follow a consistent naming convention based on a grid system. Because of this labelling system, I thought the data would simpler to examine and easier to spot and correct misnamed elements. That is why I chose not to study the nearby metropolitan area which has different sets of naming conventions.

The dataset can be found in various formats here:

http://download.geofabrik.de/north-america/us/district-of-columbia.html (http://download.geofabrik.de/north-america/us/district-of-columbia.html)

Problems Encountered in the Map

The dataset was sampled at a 1% rate to create a smaller set of data to develop auditing and cleaning functions.

Using the audit.py script to group each "addr:street" value by the last word, I found that most of the addresses spelled out the quadrant and street name, ie "1600 Pennsylvania Avenue Northwest" rather than "1600 Pennsylvania Ave NW" which is what one would typically see on a street sign. If needed, all values were updated to spell out the street type and the quadrant.

There were some parenthetically named streets such as "K Street Northwest (access road)" which I decided should just be contain the value of the main road, "K Street Northwest". All parenthesis and their contents were therefore deleted from the street names.

Finally, there were some vexing values which only occured once such as "5601 E Captitol St SE, Washington, DC 20019" which I didn't systematically correct. I just added a line in script that would correct this one specific value to be "East Capitol Street Southeast".

The script update.py, partially shown below, was implemented to clean the xml elements before they were written to .csv files.

In [ ]:
```
def update_name(name):
    old_name = name
    # Get rid of any parenthetical descriptions, ie. "K Street NW (access road)"
    name = re.sub(r'\s\(.*\)', '', name)

    # Get rid of that pesky full address
    name = re.sub(r'\, Washington, DC \d*', '', name)
    name = re.sub(r'5601 E Capitol', 'East Capitol', name)

    # Update Quadrant and Street Abbreviations
    name = name.split(' ')
    for i in range(len(name)):
        if name[i] in quadrant_mapping:
            name[i] = quadrant_mapping[name[i]]
        if name[i] in street_mapping:
            name[i] = street_mapping[name[i]]
    name = ' '.join(name)
```

House numbers in Washington, DC range from 1 to about 6000. Thus we can ignore housenumbers that have more than 4 digits. Housenumbers were frequently suffixed by letters and apartment numbers which can also be cleaned from the dataset.

Similar cleaning can be performed on the postcodes so that they are all 5 digits long and beginning with the digits '20' which is common to all Washington, DC zip codes.

Finally, phone numbers were all converted to the '+1 202 555 1234' format.

These operations were performed using regular expression functions shown below. I should also state that if a value could not be cleaned well enough to fit the proper format for a field, the value remained unaltered. Therefore the cleaned dataset contains values that my functions were unable to clean as well as values that do not belong in the dataset such as out of state zip codes.

In [ ]:
```
PHONE_FORMAT = re.compile(r'\+1\s\d{3}\s\d{3}\s\d{4}')
POSTCODE_FORMAT = re.compile(r'20\d{3}')
HOUSENUMBER_FORMAT = re.compile(r'\d{1,4}')

def update_phone(phone):
    m = PHONE_FORMAT.match(phone)
    if m is None:
        # grab all the digits and plus sign
        digits = re.findall(r'\+?\d', phone)
        if digits[0] != '+1':
            digits.insert(0,'+1')
        phone_update = (''.join(digits[0]) + ' ' +''.join(digits[1:4]) + '
 '
                        + ''.join(digits[4:7]) + ' ' +
''.join(digits[7:11]))
        # Verify that phone_update is valid. If it isn't, phone is returne
d unchanged
        if PHONE_FORMAT.match(phone_update) is not None:
            return phone_update

    return phone

def update_postcode(postcode):
    m = POSTCODE_FORMAT.match(postcode)
    if m:
        return m.group()
    return postcode

def update_housenumber(housenumber):
    m = HOUSENUMBER_FORMAT.match(housenumber)
    if m:
        return m.group()
    return housenumber
```

Overview of the data

Size of File

district-of-columbia.osm 379.5 MB
dc_sample.osm 3.8 MB
dc.db 212.4 MB
nodes.csv 133.4 MB
nodes_tags.csv 5.8 MB
ways.csv 12.1 MB
ways_nodes.csv 45.3 MB
ways_tags.csv 42.4 MB

Number of Unique Users

```
In [ ]: sqlite> SELECT COUNT(DISTINCT(uid))
                FROM (SELECT uid FROM nodes
                      UNION ALL
                      SELECT uid FROM ways);
        1392
```

Number of Nodes

```
In [ ]: sqlite> SELECT COUNT(*)
                FROM nodes;
        1682569
```

Number of Ways

```
In [ ]: sqlite> SELECT COUNT(*)
                FROM ways;
        198963
```

Number of places to buy alcohol

```
In [ ]: sqlite> SELECT count(*)
                FROM nodes_tags
                WHERE value = 'alcohol';
                189
```

Additional Queries and Statistics

The top 10 contributing users

In [ ]:
```
sqlite> SELECT user, COUNT(*) as num
        FROM (SELECT user FROM nodes
          UNION ALL
          SELECT user FROM ways)
          GROUP BY user
          ORDER BY num DESC
          LIMIT 10;
aude|710081
DavidYJackson_import|449188
wonderchook|170422
emacsen|75526
RoadGeek_MD99|67374
woodpeck_fixbot|40673
sejohnson|29545
westendguy|26251
asciiphil|23975
Will White|17735
```

List all the nodes that list a cuisine by amenity value

In [ ]:
```
sqlite> SELECT nodes_tags.value, COUNT(*) as num
        FROM nodes_tags
        JOIN (SELECT id FROM nodes_tags WHERE key = 'cuisine') as food
        ON nodes_tags.id = food.id
        WHERE nodes_tags.key = 'amenity'
        GROUP BY nodes_tags.value
        ORDER BY num DESC;
restaurant|432
fast_food|215
cafe|96
pub|8
bar|7
bbq|1
food_cart|1
ice_cream|1
ice_cream;cafe|1
```

The top 10 cuisines

Cuisines listed for all nodes that have cuisines

In [ ]:
```
sqlite> SELECT value, COUNT(*) as num
        FROM nodes_tags
        WHERE nodes_tags.key = 'cuisine'
        GROUP BY nodes_tags.value
        ORDER BY num DESC
        LIMIT 10;
sandwich|65
pizza|58
coffee_shop|54
mexican|51
american|49
burger|42
chinese|35
italian|34
thai|31
indian|25
```

Most Popular Fast Food by Locations

In [ ]:
```
sqlite> SELECT nodes_tags.value, COUNT(*) as num
        FROM nodes_tags
        JOIN (SELECT id FROM nodes_tags WHERE value = 'fast_food') as food
        ON nodes_tags.id = food.id
        WHERE key = 'name'
        GROUP BY value
        ORDER BY num DESC
        LIMIT 10;
Subway|38
McDonald's|23
Chipotle|12
Taylor Gourmet|6
Five Guys|5
Pizza Hut|4
Potbelly Sandwich Shop|4
Sweetgreen|4
Chick-fil-A|3
Chop't|3
```

Additonal ideas about the dataset

Querying the number of parks

I tried to learn more about the parks in DC by querying the nodes tags that had the word 'park' in them. These were all tags where the key was 'leisure' and the value was 'park'. What I found was underwhelming. Though there appeared to be 147 parks, there were very few attributes listed for these nodes. I for one would like to know if there are water fountains and toilets at these parks.

In [ ]:
```
sqlite> SELECT nodes_tags.key, COUNT(*) as num
        FROM nodes_tags
        JOIN(SELECT * FROM nodes_tags
            WHERE value = 'park') as parks
        ON nodes_tags.id = parks.id
        GROUP BY nodes_tags.key
        ORDER BY num DESC
        LIMIT 20;
leisure|147
name|147
county_id|146
created|146
ele|146
feature_id|146
state_id|146
tourism|3
artwork_type|2
fixme|2
historic|2
amenity|1
edited|1
memorial|1
nrhp|1
wheelchair|1
```

Consulting the openstreetmap wiki, I discovered that parks are actually supposed to be described by ways elements. Parks are supposed to be ways with tags with the following keys and values: "leisure = park" and "area = yes". With this knowledge I queried the ways tags table for parks and found much more information about the parks. Also, there were more parks listed.

In [ ]:

```
sqlite> SELECT ways_tags.key, COUNT(*) as num
        FROM ways_tags
        JOIN(SELECT * FROM ways_tags
            WHERE value = 'park') as parks
        ON ways_tags.id = parks.id
        GROUP BY ways_tags.key
        ORDER BY num DESC
LIMIT 20;
leisure|883
name|836
source|363
website|363
dataset|358
pubdate|357
address|355
propid|355
res_num|355
use_type|355
ward|355
adj_school|354
area|354
zone_|354
active|352
bbcourt|351
diamond90|351
drinkfount|351
plygrd|351
tennisct|351
```

Listing parks as ways is not what I expected which is probably why some users contributed data as nodes. While I am sure there are good reasons for documenting parks in this manner, it highlights how easily the rules of OpenStreetMap data can be broken. Unlike fixing simple typos and street names, converting incorrect nodes into ways may require more complex human intervention if the information contained in the tags is to be preserved. Otherwise, it just seems easier to delete these incorrect nodes.

Some sort of vertification of key values, in either nodes and ways, would be useful. Contribruting users could have to undergo training and testing to verify that they understand how to make only valid entries into the dataset. Certifying users would probably be costly to undertake for users and be met with much resistance. Accepting contributions only from certified users would probably reduce the contributions and size of the OpenStreetMap dataset.

Automated data cleaning after submission seems to be the best way to produce a relatively clean dataset for now.

Knowing what keys and values are valid for nodes and ways can be confusing. Many of the nodes which had values of 'park' had names that indicated that they were 'Recreation Center's. They could be labeled as <key = 'amenity' value = 'community_centre'> or <key = 'leisure' value = 'fitness_centre'>. They should also be labelled as <key = 'building' value = 'civic'>.

It can get quite bureaucratic knowing what keys to apply. A labelling system like this could be general and lax in its rules so that it can be used worldwide where the land and facillities vary. It could also be very specific and require vast more definitions to cover all the features the dataset hopes to contain. Either way, creating and revising international standards seems difficult.

Conclusion

The Washington, DC OpenStreetMap dataset contains lots of useful information about the region. There seems to be a wealth of information about restaurants, one of the top amenities. The dataset has errors in spelling out street names and quadrants which is probably because abbreviations are used so often outside of OpenStreetMap datasets.

I found errors in how parks were entered into the dataset which highlights how easily rules are broken by users not having a comprehensive knowledge of dataset conventions.

Still, the dataset remains a work in progress and I look forward to seeing how it improves.