Proj3.md

OpenStreetMap Data Project

Siddharth Kumar

Map Area
Columbus, Ohio, USA

- https://www.openstreetmap.org/relation/182706#map=11/39.9511/-82.9176

This area is of the city where I went to college. While I'm already familiar with campus area and downtown Columbus, I chose this map to learn more about surrounding areas and less popular spots. I also used my previous knowledge of the area to check the validity of the dataset.

Problems Encountered in Map

Keys:

- Repetitive/Similar Key Names
  When keys had similar names, I changed them to the more common one.
  ex: ('County', 'county_name'), ('ST_num', 'state_id'), ('url, website')
- Nested Tag Type
  For keys that started with a word followed by a ':', I set the first word to the 'type' attribute for all tags
  ex: 'addr:street', 'contact:phone', 'surveillance:webcam'
- Nested Naming Authority
  If the type attribute was 'tiger' or 'gnis', I set it to an attribute called 'naming_auth' (Naming Authority). The 'Geographic Names Information System' (GNIS) contains information on geographic features in the US, and 'Topologically Integrated Geographic Encoding and Referencing' (TIGER) is used by the US Census to describe land attributes. Both these naming authorites have integrated their data with OSM.
  ex: 'tiger:county', 'tiger:zip_right', 'gnis:state_id'

Values:

- Unstandardized Values:
  All unstandardized values were converted to standard formats. Unstandardized values were most likely due to different naming conventions used by GNIS and TIGER, as well as human input. I did not see any problems related to location specific formatting. Most formats, such as 9 digit postal codes and (555) 555-5555 phone number style were easily recognizable and easy to fix.

  - Street Names
    unstandard: 'W. Woodruff Ave', 'National Road SW'
    standard: 'West Woodruff Avenue', 'National Road Southwest'
  - State Names
    unstandard: 'OH', 'OH - Ohio'
    standard: 'Ohio'
  - City Names
    unstandard: 'Columbus, OH', 'Columbus, Ohio', 'columbus'
    standard: 'Columbus'
  - County Names
    unstandard: 'OH:Franklin'
    standard: 'Franklin'

- Postcodes
  unstandard: '43201-3247', 'OH 43201', '4320'
  standard: '43201'
- Phone Numbers
  unstandard: '(614) 555-5555', '614-555-5555', '+1 (614) 555-5555'
  standard: '6145555555', '16145555555'
- Speed Limits
  unstandard: '55'
  standard: '55 mph'
- Restaurant Cuisines
  unstandard: 'ice cream', 'Ice_Cream'
  standard: 'ice_cream'

- Nested Values in Same String
  Values that contained multiple nested values were separated and treated as individual tags.
  ex: 'bar;restaurant', '43201:43210', 'Franklin, Licking, Delaware'

Separating Nested Values

I dealt with nested values by assuming that all values had multiple nested values. By calling the ```process_value``` method for each value, I split the value by its delimeter and returned a list of values. A tag was created for each value in the returned list. Most of the time, the list only contained one value, therefore only one tag was created.

```
def process_value(key, value):
    split_symbols = [';', ',_', ',', ':', '_/_']
    split_vals = list()

    if key in ['ref', 'county', 'destination', 'phone', 'exit_to', 'cuisine', 'amenity'] or key.startswith('zip'):
        for symbol in split_symbols:
            if symbol in value:
                for item in value.split(symbol):
                    split_vals.append(item.strip())
                break
        if len(split_vals) == 0:
            split_vals.append(value)
    else:
         split_vals.append(value)
    return split_vals
```

Nodes Tags Defined as Ways Tags

While auditing street names in ways tags, I noticed obscure street names such as *'CVS Pharmacy', 'Riverside Methodist Hospital', 'Huber Ridge Elementary School'*, and many others. Knowing that these did not belong as way tags, I programatically removed them. I accomplished this with the ```is_valid_way_element``` method. It basically checked to see if the element contained keys like 'building', 'leisure', and 'shop'. If it did, I assumed that the tag was not describing what a way tag is supposed to be, and returned ```False```. Before appending the first level element, I ran this method to ascertain its validity as a way tag.

```
def is_valid_way_element(tags):
    unwanted_way_keys = ['building', 'leisure', 'shop', 'amenity', 'tourism', 'landuse', 'housename']
    for way_tag in tags:
        if (way_tag['key'] in unwanted_way_keys) or (way_tag['type']=='addr'):
            return False
    return True
```

Auditing ZipCode Accuracy

After auditing data validity in Python, I was curious to test accuracy using gold standard data. I figured zipcode data would be a good place to start. I downloaded a list of zipcodes in the US from aggdata.com. The original source of the data is from geonames.org, a free geographical database. I ran a query that compared city names from OSM and GeoNames, matching on zipcode. Most cities were the same, so I only displayed those that were not.

```
sqlite> SELECT a.value AS zip, b.value AS osm_city, c.city AS geonames_city
   ...> FROM nodes_tags a, nodes_tags b
   ...> LEFT JOIN zip_codes c
   ...> ON zip = c.zipcode
   ...> WHERE a.id = b.id
   ...> AND a.key = 'postcode'
   ...> AND b.key = 'city'
   ...> AND (osm_city != geonames_city OR geonames_city IS NULL)
   ...> GROUP BY zip, osm_city;
```

```
| zip   | osm_city          | geonames_city |
|-------|-------------------|---------------|
| 43081 | Columbus          | Westerville   |
| 43085 | Worthington       | Columbus      |
| 43207 | Obetz             | Columbus      |
| 43209 | Bexley            | Columbus      |
| 43212 | Grandview Heights | Columbus      |
| 43213 | Grove City        | Columbus      |
| 43213 | Whitehall         | Columbus      |
| 43220 | Upper Arlington   | Columbus      |
| 43221 | Upper Arlington   | Columbus      |
| 43230 | Gahanna           | Columbus      |
| 43328 | Columbus          |               |
```

I checked the validity of this table by searching the zipcodes on usps.com and found that many of the OSM cities were listed as "recognized" cities under that zipcode, however not the default. The GeoNames data matched all zipcodes to their default city. Two cases where OSM data is wrong are 43081 and 43328. According to usps.com, 43081 is only used for Westerville, OH and 43328 is not valid. These are confirmed in the GeoNames data. As a result, I updated the database to show only the GeoNames cities and removed the invalid zipcode.

Overview of the Data

File Sizes

- ColumbusOhio.osm -- 82 MB
- columbus_osm.db -- 44 MB
- nodes.csv -- 30 MB
- nodes_tags.csv -- 0.95 MB
- ways.csv -- 1.9 MB
- ways_tags.csv -- 7.7 MB
- ways_nodes.csv -- 6.5 MB
- us_postal_codes.csv -- 2.3 MB

Number of Unique Users

```
sqlite> SELECT COUNT(nodes_and_ways.uid)
   ...> FROM (SELECT uid FROM nodes
   ...>       UNION
   ...>       SELECT uid FROM ways
   ...>       GROUP BY uid) nodes_and_ways;
```

514

Number of Nodes

```
sqlite> SELECT COUNT(*) FROM nodes;
```

375132

Number of Ways

```
sqlite> SELECT COUNT(*) FROM ways;
```

33331

Top 10 Cuisines

As someone who loves to eat, I'm curious to see what the most popular cuisine is in Columbus.

```
sqlite> SELECT value, COUNT(value)
   ...> FROM nodes_tags
   ...> WHERE key = 'cuisine'
   ...> GROUP BY value
   ...> ORDER BY COUNT(value) DESC
   ...> LIMIT 10;
```

```
| cuisine     | count |
|-------------|-------|
| mexican     | 20    |
| pizza       | 20    |
| american    | 19    |
| burger      | 19    |
| sandwich    | 19    |
| coffee_shop | 16    |
| italian     | 14    |
| chinese     | 12    |
| asian       | 9     |
| ice_cream   | 9     |
```

Streets With Pizza

In the last query, I found that 'mexican' and 'pizza' are the 2 most popular cuisines. As a result, I decided to find out what streets had the most options for pizza.

```
sqlite> SELECT a.value, COUNT(a.value)
   ...> FROM nodes_tags a, nodes_tags b
   ...> WHERE a.id = b.id
   ...> AND a.key = 'street'
   ...> AND b.key = 'cuisine'
   ...> AND b.value = 'pizza'
   ...> GROUP BY a.value
   ...> ORDER BY COUNT(a.value) DESC;
```

```
| street            | count |
|-------------------|-------|
| North High Street | 5     |
| Indianola Avenue  | 1     |
| South High Street | 1     |
| West Fifth Avenue | 1     |
| West Lane Avenue  | 1     |
```

Pizza Places on North High Street

According to the last query, North High Street had the most pizza places than any other street in Columbus. My last step was to find the names of these places, and make my final lunch decision!

```
sqlite> SELECT a.value
   ...> FROM nodes_tags a, nodes_tags b, nodes_tags c
   ...> WHERE a.id = b.id
   ...> AND b.id = c.id
```

```
    ...> AND a.key = 'name'
    ...> AND b.key = 'street'
    ...> AND b.value = 'North High Street'
    ...> AND c.key = 'cuisine'
    ...> AND c.value = 'pizza';
```

```
| name              |
|-------------------|
| Hound Dog's Pizza |
| Papa John's       |
| Cottage Inn Pizza |
| Anges Pizza       |
| Blaze Pizza       |
```

Ideas For Improving the Dataset

When analyzing the data in Python, I found that there were multiple instances of node data listed as ways. I originally thought of converting these invalid ways to nodes, however I found it to be impossible due to the different conventions used for nodes and ways. (*eg: nodes require latitude and longitude data, while ways do not*) As a result, I had to remove all of these invalid ways programmatically. Since there was a lot of valuable node information being removed, I would suggest a way for OSM to safeguard against careless user entry. I was able to discern valid ways from invalid ways by checking its key values. Generally, if ways had key names like 'leisure', 'shop', or 'building', it was safe to assume that it wasn't actually a way. Perhaps OSM can check key names before adding them to their dataset? If this were implemented, it would ensure that valuable node data isn't wasted. However, A drawback of checking way information programmatically is that it may prevent users from entering valid way data.

Conclusion

While this OSM dataset is nowhere near complete, it's a good start toward providing developers a free, open source method of mapping the world. I liked the idea of integrating TIGER data into the map; if OSM were able to integrate with more GPS mapping services (*i.e. Garmin, TomTom, etc.*), it could build a much more informative and accurate datset.