OpenStreetMap Data Wrangling Study

Map Area - Columbus, Ohio, USA

I lived in Columbus, OH in 2015 and 2016, so there is a familiarity with the area. The city was selected to make further discoveries and gain more knowledge about Columbus and its immediate suburbs.

Data Problems and Inconsistencies

Starting with a sample of 5% of the data set and incrementally increasing the size as tendencies became clear, a number of flaws in the Columbus OpenStreetMap were discovered. I was sure that later I would want to query some specific types of landmarks, so those values were especially audited.

- *Address Consistency*: Addresses were formatted using different abbreviations for the same street type.
- *Zip Code Validity and Uniformity*: Some zip codes had invalid entries of only 4 numbers, and other entries included additional shipping codes following the standard 5 numbers.
- *City Name Accuracy and Consistency*: A few typographical errors for city names and a formatting option including the state were discovered.
- *Church Denomination Consistency*: Different formatting for the same church type and unnecessary capitalization were present.
- *Sports Field Uniformity*: Some fields were labeled with more detail, but to query there needed to be the same level of specificity used for each value.
- *Cuisine Consistency and Uniformity*: Similar to sports fields, there were different levels of detail for options at restaurants. There was also periodic capitalization for a small portion of the set.

Address

Though no queries were scheduled to look at specific addresses, it was still determined that correct addresses pertaining to a OpenStreetMap project were vital.

For example, the following three entries were present in the data set. Each represents the street type `'Road'` differently.

- `'Caine Rd'`
- `'E. Dublin Granville Rd.'`
- `'Amity Road'`

To correct these, a regular expression was used along with a dictionary with corrected road types. A few street addresses were explicitly mentioned to correct, but the bulk of the cleaning was done by using the `streetmapping` dictionary and this portion of the `update_name_street` function:

```
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)
m = street_type_re.search(name)
street_type = m.group()
if street_type in streetmapping.keys():
    name = re.sub(m.group(), streetmapping[m.group()],name)
```

The roads in the example above were changed to the following:

- `'Caine Road'`
- `'E. Dublin Granville Road'`
- `'Amity Road'`

Zip Code

Zip codes with only 4 numbers i.e. `'4313'` were deemed to be unfixable since the missing number could come anywhere in the integer. Instead of removing the node, these zip codes were changed to `'Bad Zip'` in case when querying we wanted to look at what they represent. They easily can be removed when the data is ready to be reuploaded back to OpenStreetMap.

To adjust zip codes that contained the extra 4 digit delivery code i.e. `'43215-1430'` a regular expression was used to only take the first five numbers of the code. It was confirmed that this method did not negatively affect the validity of any other zip codes.

```
def update_zips(name):
    if len(name) < 5:
        name = "Bad Zip"
        return name
    elif len(name) > 5:
        if re.search('[0-9]{5}', name):
            updated_name = re.findall('[0-9]{5}', name)
            name = updated_name[0]
            return name
    else:
        return name
```

City Name

On a whole, the city name values were fairly clean. To fix the two spelling errors and remove state identifiers, the `update_city_name` function was written.

```
def update_name_city(name, citymapping):
    if name in citymapping:
        name = citymapping[name]
        if name.islower():
            name = name.capitalize()
        return name
    else:
        if name.islower():
            name = name.capitalize()
        return name
```

Along with the `citymapping` dictionary that adjusts the city to its correct spelling, measures were taken in the code to ensure that each city was formatted with the same capitalization.

Church Denomination and Sports Fields

As with city name, the denominations and sports fields were already pretty clean, so the code only contains some minor tweaks to ensure they each were consistently formatted and void of erroneous detail.

Below is the function used to clean the denominations, given their corrections contained in the `denommapping` dictionary.

```
def update_name_denom(name, denommapping):
    if name.istitle():
        name = name.lower()
        return name
    elif name in denommapping:
        name = denommapping[name]
        return name
    else:
        return name
```

For example, `'united_methodist;methodist'` is updated to `'methodist'`.

Cuisine

Of all the key values looked at for auditing, the cuisine key contained the most problems. A lengthy `cuisinemapping` dictionary was created to hold all of the changes necessary for the set.

A few examples of these are as follows:

```
cuisinemapping = { "Burrito,_Taco, Salad, Chips": "mexican",
                   "Ice_Cream": "ice_cream",
                   "Deli": "sandwich",
                   "Japanese_Sushi_Hibachi": "japanese"}
```

Structuring the `update_name_cuisine` function similar to the functions above, these restaurant categories were audited and cleaned.

Columbus Data Review

File Sizes

```
Columbus.osm      . . .   152.232 MB
nodes.csv         . . .    56.813 MB
nodes_tags.csv    . . .     1.453 MB
ways.csv          . . .     4.488 MB
ways_nodes.csv    . . .    19.053 MB
ways_tags.csv     . . .    13.367 MB
columbusaudit.db  . . .    84.524 MB
```

Total Nodes

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

```
675741
```

Total Ways

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

```
75957
```

Total Unique Contributors to Map Data

```
        SELECT Count(DISTINCT both.user)
        FROM (SELECT user FROM nodes
        UNION ALL SELECT user FROM ways) as both;
```

```
765
```

Top 5 City Tags

```
        SELECT both.value, COUNT(*) as Total
        FROM (SELECT * FROM nodes_tags UNION ALL
        SELECT * FROM ways_tags) as both
        WHERE both.key == 'city'
        GROUP BY both.value
        ORDER BY Total DESC
        LIMIT 5;

                    City
        1         Columbus   803
        2           Dublin   329
        3  Upper Arlington    98
        4         Hilliard    82
        5     Pickerington    58
```

Top 5 Zip Code Tags

```
        SELECT both.value, COUNT(*) as Total
        FROM (SELECT * FROM nodes_tags UNION ALL
        SELECT * FROM ways_tags) as both
        WHERE both.key == 'postcode'
        GROUP BY both.value
        ORDER BY Total DESC
        LIMIT 5;

            Zip Code
        1     43017  319
        2     43221  111
        3     43210   89
        4     43026   86
        5     43215   79
```

Top 5 Church Denominations

```
SELECT value, COUNT(*) as Total
FROM nodes_tags
WHERE key == 'denomination'
GROUP BY value
ORDER BY Total DESC
LIMIT 5;

        Denomination
1            baptist  127
2          methodist   52
3           lutheran   38
4           catholic   21
5       presbyterian   17
```

Top 5 Sports Fields/ Venues

```
SELECT both.value, COUNT(*) as Total
FROM (SELECT * FROM nodes_tags UNION ALL
SELECT * FROM ways_tags) as both
WHERE both.key == 'sport'
GROUP BY both.value
ORDER BY Total DESC
LIMIT 5;

           Sport
1           baseball   311
2             tennis   203
3             soccer   100
4         basketball    94
5  american_football    70
```

Top 5 Restaurant Types

```
SELECT both.value, COUNT(*) as Total
FROM (SELECT * FROM nodes_tags UNION ALL
SELECT * FROM ways_tags) as both
WHERE both.key == 'cuisine'
GROUP BY both.value
ORDER BY Total DESC
LIMIT 5;

       Cuisine
1        burger  77
2       mexican  56
3         pizza  48
4      sandwich  38
5      american  32
```

Additional Improvements

After auditing and querying the data, two major data wrangling principles still loomed large over the OpenStreetMap data as a whole: validity and completeness. Potentially because of the vast number of contributors to the data source, nodes and ways are not entered with the same degree of completeness when considering keys and values.

Example:

```
SELECT both.value, Count(*)
FROM (SELECT * FROM nodes_tags UNION ALL
SELECT * FROM ways_tags) as both
WHERE both.value == 'Starbucks';

Starbucks, 23
```

When querying for locations named Starbucks, there were 23 found. Therefore, one would expect there to be 23 locations of Starbucks when looking at where tags have `key = 'cuisine'` and `value = 'coffee_shop'`.

```
CREATE view both as
SELECT * FROM nodes_tags UNION ALL
SELECT * FROM ways_tags;

CREATE view coffee_shop as
SELECT id
FROM both
WHERE both.key == 'cuisine' and both.value == 'coffee_shop';

SELECT both.value, COUNT(*) from both
JOIN coffee_shop
on both.id = coffee_shop.id
WHERE both.key = 'name' and both.value = 'Starbucks'

Starbucks, 11
```

This leads to the conclusion that although the nodes and ways for the locations may be there, each one is not fully completed to include all relevant keys and values for the location. Though this would make the map much more thorough for querying, the problem lies in the time and effort it would take to complete this task to its full extent.

A second idea for improvement was discovered when looking at values for the key `'maxspeed'`. For the most part, these are speed limits of highways. However, it was very curious to see the sixth most common speed was 105 mph.

```
        SELECT both.value, COUNT(*) as Total
        FROM (SELECT * FROM nodes_tags UNION ALL
        SELECT * FROM ways_tags)
        as both
        WHERE both.key == 'maxspeed'
        GROUP BY both.value
        ORDER BY Total DESC
        Limit 6;

           maxspeed
        1   65 mph  641
        2   55 mph  321
        3   25 mph  249
        4   35 mph  180
        5   45 mph  168
        6  105 mph   74
```

Further investigation yielded that 2 contributors were mainly responsible for these tags.

```
    SELECT ways.user, COUNT(*) as Total
        FROM both JOIN ways
        on both.id = ways.id
        WHERE both.key = 'maxspeed' and both.value = '105 mph'
        GROUP BY ways.user
        ORDER BY Total DESC;

                user
        1   Vid the Kid  37
        2           NE2  32
        3          cl94   3
        4  Sunfishtommy   2
```

These tags were placed on highway locations, and it seems curious as to why these two users would add 105 mph values other than to potentially mark these locations as their own. Usually, the username is enough to mark a submission for users, but there could possibly be an epidemic that data contributors are leaving their mark in other ways. Ideally our data set would be clean of these potential markers, but the difficult part is identifying these repeated marks and differentiating them from other valid values.

Conclusion

Overall, the Columbus data is a robust XML document containing a large amount of useful information but also a large amount of faulty values. The specific keys chosen in this audit are now clean and should be accurate to query, but it bares mentioning that there are 481 total unique keys for ways and nodes that comprise this document. A full audit and cleaning of this would be extremely lengthy, but it is possible if following a structure similar to what is layed out above. It would also be worth checking out samples from other cities across the world, for in a best case scenario the OpenStreetMap.org data entry protocol would be uniform throughout the entire map.