OpenStreetMap Data Wrangling

Introduction

This project is intended to demonstrate an understanding of data cleaning processes. A subset of OpenStreetMap data was cleaned, converted into a .csv file, and imported into a SQL database.

The data itself consists of six cities in Southwest Idaho; Boise, Meridian, Nampa, Eagle, Kuna, and Caldwell. It is approximately 68 MB in size and bounded by the coordiantes (43.4596, -116.7517, 43.7552, -115.9765).

Data Overview

The OpenStreetMap data for this region is predictably sparse. Idaho is not a populous state, holding a population around 1.6 million (560 thousand of which came to the state between 1990 and 2010). The cumulative data for the six cities in question only amounts to 68 MB. For contrast, data for the single city of Charlotte, North Carolina is nearly 300 MB.

General Information

- 455 Unique User ID's
- 281,196 Nodes
- 33,401 Ways
- 165 Schools
- 20 Cafes
- 13 Subway Sandwhich Shops

The main roads for each city appear to be well-documented, with smaller in-streets like those in neighborhoods being ignored. The data is surprisingly clean in this section. It seems likely that much of it has been previously cleaned.

Nodes (that is, objects like resaurants and schools) are noticably messier. The information appears out-of-date and poorly organized, where it isn't absent altogether. This section would doubtlessly benefit from some attention.

Problems Encountered

Questionable Spelling and Abbreviations

The data regarding street names was surprisingly well kept. Only a couple dozen names needed to be corrected, out of a pool of 357 unique street names (3353 non-unique). To that end, a lightweight Python script was used.

```
def update_street_name(name):
    fix = name.split()
    first = True
    for word in fix:
        if first and 'Mc' not in word and 'ID' not in word:
            name = name.replace(word, word.capitalize())
            word = word.capitalize()
            first = False
        if word in suffix_mapping.keys():
            name = name.replace(word, suffix_mapping[word])
        elif word in prefix_mapping.keys() and len(word) <= 2:
            name = name.replace(word, prefix_mapping[word])
    name = name.replace('.', '')
    return name
```

```
E Barber Valley Dr -> East Barber Valley Drive
```

The data on restaurants, however, was a mess. Nearly all restaurants with more than a single instance had an alias. Many had issues with punctuation or unusual symbols.

```
'Carl_s Jr': 1,
'Carl_s Jr.': 1,
'Carl_s Jr. / Green Burrito': 1,
'Carl_s Junior': 1,
'Carl_s Junior Green Burrito': 1,
'Carls Jr.': 1,
```

This was crudely repaired via a regular expression that removed non-alphanumeric characters (except spaces). The solution is imperfect but much improved. A more complex mapping might be able to standardize the various restaurants names by recognizing permuations like 'jr' or 'robinn'.

```
problemchars_x = re.compile(r'[=\+/&<>;\'"\?%#$@\,\.\t\r\n]')
```

```
'Carls Jr': 3,
'Carls Jr Green Burrito': 1,
'Carls Junior': 1,
'Carls Junior Green Burrito': 1,
```

Sparse and Outdated Information

To illustrate the problem, consider the below query:

```
# Determine number of various places
SELECT value,COUNT(value) FROM Nodes_Tags WHERE key IS 'amenity' \
GROUP BY value ORDER BY COUNT(value) DESC
```

```
[('restaurant', 166),
 ('school', 165),
 ('place_of_worship', 93),
 ('fast_food', 82),
 ('fuel', 62),
 ('bank', 57),
 ('parking', 25),
 ('cafe', 20),
 ('bar', 13),
 ('pharmacy', 13),
 ('car_wash', 11),
 ('cinema', 11),
 ('doctors', 11),
 ('grave_yard', 10),
 ('hospital', 9),
 ('toilets', 9),
 ('bicycle_repair_station', 8),
 ('fire_station', 8),
 ('library', 8),
 ('post_office', 8),
 ('dentist', 7),
 ('drinking_water', 7),
 ('veterinary', 7),
 ('bench', 5),
 ('pub', 5),
 ...]
```

A few points immediately stand out. Eleven doctor's offices seems awfully small, and eight bicycle repair stations is oddly specific. A little more thought might lead one to wonder about the integrity of the more reasonable statistics. Restaurants seem especially suspect, as the list seems incomplete to the eyes of a long-term Idaho resident.

Likewise, information regarding street names appears smaller than expected.

```
# Investigate street data
SELECT value from Ways_Tags WHERE key IS 'street' UNION ALL \
SELECT value from Nodes_Tags WHERE key IS 'street'

SELECT DISTINCT value from Ways_Tags WHERE key IS 'street' UNION ALL \
SELECT DISTINCT value from Nodes_Tags WHERE key IS 'street'
```

```
Number of named streets in data: 3353
Number of unique street names in data: 357
```

There is unfortunately little that can be done about an absence of data. No amount of cleaning can conjure new information. This leads to the consideration of possible solutions...

Additional Ideas for Improvement

The most significant problems with the data set are its sparsity and lack of accuracy. Fixing these may require manually gathering data on cities in Idaho. Another alternative would be to import significant amounts of data from another service (as had been previously done with TIGER).

If a digital copy of the local yellow pages exists, a script could be developed that would parse the information into a format readable by OpenStreetMap. This solution would keep OSM from relying on Google Maps, but I'm unsure of the legality. Web crawlers could offer similar functionality by digging through review sites like Yelp.

This certainly wouldn't be the first time automation had been used to implement wide reaching changes. A bit of research shows that one bot was responsible for 110,000 changes to the Southwest Idaho data set in 2009 alone. For reference, the sum total of all changes across eleven years for the data set is just under 315,000.

```
# Investigate participation by user, in 2009
cur.execute("SELECT user,timestamp FROM Nodes WHERE timestamp LIKE '2009%' \
UNION ALL SELECT user,timestamp FROM Ways WHERE timestamp LIKE '2009%';")
x = cur.fetchall()

# Aggregate users
user_participation = defaultdict(int)
for item in x:
    user_participation[item[0]] += 1
```

```
defaultdict(<class 'int'>,
            {'C Bentz': 34,
             'Chris Lawrence': 13357,
             'Minh Nguyen': 920,
             'Tony Speer': 5,
             'amillar': 17,
             'barkerid': 2349,
             'chokeyou': 3,
             'h4ck3rm1k3': 14,
             'haveaswiss': 14,
             'iandees': 70,
             'isotope': 14,
             'jrglasgow': 322,
             'linuxuser728': 2,
             'ml11013': 7,
             'nmixter': 290,
             'pdunn': 259,
             'woodpeck_fixbot': 111950,
             'xybot': 14})
```

Additional Queries

```
# Determine number of nodes
SELECT * FROM Nodes
```

```
# Determine number of ways
SELECT * FROM Ways
```

```
# Investigate resturaunts
food_ids= []
```

```
food_places = defaultdict(int)
cur.execute("SELECT DISTINCT id FROM Nodes_tags \
WHERE value IS 'restaurant' OR value is 'fast_food' OR value is 'cafe';")
x = cur.fetchall()
for tup in x:
    food_ids.append(tup[0])

cur.execute("SELECT * FROM Nodes_tags WHERE key is 'name'")
y = cur.fetchall()
for tup in y:
    if tup[0] in food_ids:
        food_places[tup[2]] += 1
```

```
# Investigate participation by year
cur.execute("SELECT timestamp FROM Nodes UNION ALL SELECT timestamp FROM Ways;")
x = cur.fetchall()

yearly_participation = defaultdict(int)
for item in x:
    dt = item[0].split('-')
    yearly_participation[dt[0]] += 1
```