OpenStreetMap Data Case Study

A. Map Area

Denver, CO United States

- https://www.openstreetmap.org/relation/25375
- https://mapzen.com/data/metro-extract/your-extract/fb04e1c2cd98

I love Denver, Colorado so I chose to see what the Denver map reveals when I preform database queries. I would like to improve OpenStreetMap.org in the Denver area.

GPS Location

- S: 39.4850
- W: -105.1488
- N: 40.0434
- E: -104.561

B. Data Overview

The file I downloaded from Mapzen was a .bz2 file. I had challenges unzipping the file. I posted on the forum and learned that 7-zip would help me unzip the OSM file, which was incredibly helpful. Using 7-zip, I was able to unzip the .bz2 file and start wrangling data. I ran the input in a Jupyter notebook. This section contains basic statistics about the dataset.

File sizes

```
Audit.ipynb         341 KB
Data.ipynb          10 KB
DataBase.ipynb      4 KB
Query.ipynb         5 KB

DenverCO.db         340 MB

schema.py           3 KB
tags.py             2 KB

DenverCO.osm        450 MB
DenverCO_sample.osm45 MB
nodes.csv           172 MB
nodes_tags.csv      7 MB
ways.csv            16 MB
ways_nodes.csv      58 MB
ways_tags.csv       38 MB
```

Number of tags

```
{'member': 2117,
 'nd': 240131,
 'node': 200867,
 'osm': 1,
 'relation': 119,
 'tag': 132958,
 'way': 26565}
```

Count of Problematic Characters

I ran a count to see how many had lowercase letter and were valid, has lower case and colon in the name, and ones that had problematic character. I was surprised that there were no problematic characters.

```
{'lower': 286485,
'lower_colon': 238128,
'other': 17437,
'problemchars': 0}
```

Number of unique users

```
def process_map(filename):
    users = set()
    for __, element in ET.iterparse(filename):
        for e in element:
            if 'uid' in e.attrib:
                users.add(e.attrib['uid']
    return users
users = process_map(SAMPLE_FILE)
len(users)

768
```

C. Problems Encountered in the Map

After initially downloading a sample of the Denver, Colorado map and running it against a provisional data.py file, I noticed three main problems with the data:

1. Unclean street names
2. Over abbreviated street names *("S Niagra Ct" & "E Fair Ave")*
3. Incorrect zip codes

1. Unclean Street Names

I ran a script that shows the unclean street names.

```
{'100': set(['Sheridan Boulevard #100']),
 '2': set(['Colorado SH 2', 'Colorado SR 2']),
 '287': set(['US Highway 287']),
 'Ave': set(['E 6th Ave',
             'E 72nd Ave',
             'E Caley Ave',
             'E Fair Ave',
             'E Maplewood Ave',
             'W 84th Ave']),
 'Ave.': set(['W. Alameda Ave.']),
 'Blvd': set(['745 Colorado Blvd',
              'East Academy Blvd',
              'Federal Blvd',
              'Green Valley Ranch Blvd',
              'S University Blvd',
              'South Colorado Blvd',
              'Wadsworth Blvd',
              'Wadworth Blvd']),
 'Broadway': set(['Broadway', 'South Broadway']),
 'Center': set(['Garden Center']),
 'Cir': set(['E Flatiron Cir', 'S Lake Cir']),
 'Colfax': set(['East Colfax']),
 'Crescent': set(['Interlocken Crescent']),
 'Ct': set(['S Niagra Ct']),
 'Dr': set(['Community Circle Dr']),
 'Highway': set(['North Valley Highway']),
 'Lincoln': set(['Lincoln']),
 'Pl': set(['E Maplewood Pl', 'E Orchard Pl']),
 'Rd': set(['Coalton Rd', 'E Arapahoe Rd', 'S Parker Rd']),
 'Row': set(['Colony Row']),
 'St': set(['Dayton St',
            'S Broadway St',
            'S Clayton St',
            'S Columbine St',
            'S Elizabeth St',
            'S Josephine St',
            'S Newport St',
            'S Peoria St',
            'S Poplar St',
 'St.': set(['Pearl St.']),
 'Tennyson': set(['Tennyson'])
```

2. Over Abbreviated Street Names

Next, I took the over abbreviated names and updated the spelling of them. The script to the abbreviated versions like “Pearl St” and made it “Pearl Street”.

```
East Colfax => East Colfax
Park Avenue West => Park Avenue West
Pearl St. => Pearl Street
E Arapahoe Rd => E Arapahoe Road
Coalton Rd => Coalton Road
S Parker Rd => S Parker Road
North Valley Highway => North Valley Highway
Colorado SH 2 => Colorado SH 2
Colorado SR 2 => Colorado SR 2
E Orchard Pl => E Orchard Place
E Maplewood Pl => E Maplewood Place
Community Circle Dr => Community Circle Drive
W. Alameda Ave. => W. Alameda Avenue
Garden Center => Garden Center
Tennyson => Tennyson
S Sherman St => S Sherman Street
S Clayton St => S Clayton Street
Wright St => Wright Street
S Poplar St => S Poplar Street
South Grant St => South Grant Street
Dayton St => Dayton Street
Wewatta St => Wewatta Street
S Newport St => S Newport Street
S Elizabeth St => S Elizabeth Street
S Peoria St => S Peoria Street
S Broadway St => S Broadway Street
S Josephine St => S Josephine Street
S Columbine St => S Columbine Street
E Flatiron Cir => E Flatiron Circle
S Lake Cir => S Lake Circle
W 84th Ave => W 84th Avenue
E 6th Ave => E 6th Avenue
E Caley Ave => E Caley Avenue
E Maplewood Ave => E Maplewood Avenue
E 72nd Ave => E 72nd Avenue
E Fair Ave => E Fair Avenue
Wadworth Blvd => Wadworth Boulevard
S University Blvd => S University Boulevard
Green Valley Ranch Blvd => Green Valley Ranch Boulevard
South Colorado Blvd => South Colorado Boulevard
Wadsworth Blvd => Wadsworth Boulevard
S Niagra Ct => S Niagra Court
```

3. Zip Codes

Next, I examined the zipcodes to ensure they were all at least 5 digits long. It appeared the zip code field only contained zip codes. However, one of the zip codes had an extra digit.

```
{'80': set(['80002',                             '80207',
            '80003',                             '80209',
            '80004',                             '80210',
            '80010',                             '80211',
            '80011',                             '80212',
            '80012',                             '80214',
            '80013',                             '80214-1801',
            '80014',                             '80214-1833',
            '80015',                             '80215',
            '80016',                             '80216',
            '80017',                             '80218',
            '80018',                             '80219',
            '80020',                             '80220',
            '80021',                             '80221',
            '80022',                             '80222',
            '80023',                             '80223',
            '80027',                             '80224',
            '80030',                             '80226',
            '80031',                             '80229',
            '80033',                             '80230',
            '80045',                             '80231',
            '80110',                             '80232',
            '80111',                             '80233',
            '801111',                            '80234',
            '80112',                             '80235',
            '80113',                             '80237',
            '80120',                             '80238',
            '80121',                             '80239',
            '80122',                             '80241',
            '80123',                             '80246',
            '80127',                             '80247',
            '80202',                             '80249',
            '80203',                             '80260',
            '80204',                             '80601',
            '80205',                             '80602',
            '80206',                             '80640'])}
```

Additional Data Exploration Ideas

Nodes & Ways Information

```
def number_of_nodes():
     result = cur.execute('SELECT COUNT(*) FROM nodes')
     return result.fetchone()[0]

def number_of_ways():
     result = cur.execute('SELECT COUNT(*) FROM ways')
     return result.fetchone()[0]
```

```
Number of nodes
```

- 2008669

```
Number of ways
```

- 265642

User Information

```
def number_of_unique_users():
     result = cur.execute('SELECT COUNT(DISTINCT(e.uid)) \
          FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM
          ways) e')
     return result.fetchone()[0]

def top_contributing_users():
     users = []
     for row in cur.execute('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'):
          users.append(row)
     return users

def number_of_users_contributing_once():
     result = cur.execute('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')
     return result.fetchone()[0]
```

```
Number of unique users:
   • 1161

Top contributing users:
   • u'Your Village Maps', 561562
   • u'chachafish', 551036
   • u'GPS_dr', 152106
   • u'woodpeck_fixbot', 139217
   • u'Stevestr', 127754
   • u'DavidJDBA', 87372
   • u'CornCO', 73601
   • u'jjyach', 47854
   • u'TheAverageNomad', 38628
   • u'SeanMaday', 37249

Number of users contributing once:
   • 228
```

Top appearing amenities

```
def common_ammenities():
      for row in cur.execute('SELECT value, COUNT(*) as num \
            FROM nodes_tags \
            WHERE key="amenity" \
            GROUP BY value \
            ORDER BY num DESC \
            LIMIT 10'):
             return row
```

```
Restaurant
   • 1,237
```

Most Popular Denomination of Religion

```
def biggest_religion():
     for row in cur.execute('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;'):
     return row
```

```
Christian:
```

- 485

Most Popular Cuisines

```
def popular_cuisines():
     for row in cur.execute('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'):
          return row
```

```
American:
```

- 119

Conclusion

Having lived in Denver for over 10 years, I believe that restaurants are the most common amenities with American cuisine being the most popular. I also believe the most common religion of Christianity sounds correct as well. I would be interested to explore other areas of Colorado outside of the metro area and would like to dig deeper to the types of establishments in the area (coffee shop, brewery, gas stations, etc). The number of unique users appears to be high. My data exploration told me that 1,161 unique users contributed to the Denver OpenStreetMap data, which explains why there were very few problematic characters.