OpenStreetMap Project
Data Wrangling with MongoDB

Michael Ochs

Map Area: Denver-Boulder, CO, United States
http://www.openstreetmap.org/relation/1411339
https://mapzen.com/data/metro-extracts/metro/denver-boulder_colorado/

Table of Contents

Problems Encountered in the Map

Code from the project summary was used to reduce the XML content with a k=12 value. This reduced the OSM file size from 753MB to 63MB. The codes created during the *Case Study: OpenStreetMap Data* section of the Nanodegree were used to analyze potential data errors in the OSM file. Upon usage of the codes mentioned multiple errors were noticed in the data:

1. Road identifiers (e.g. Street, Boulevard, Circle, etc.) were inconsistent
2. Address content was not standardized and contained errors. Some addresses contained the city or postal code.
3. Numerous Unicode encoding errors when parsing OSM data
4. Inconsistent or redundant addressing standard

Road Identifiers

Road identifiers—like St instead of Street—were throughout the dataset. A dictionary was implemented to maintain consistency of the identifiers.

Address Content

Some v_values did not match a suitable k_values. There were many instances where the *addr:street* was the entire address (e.g. including city and state). There were also instances of where the street value included the main name of the street (e.g. "Lincoln" instead of "Lincoln Street"). A dictionary was implemented to resolve some of the issues of too much or too little data; however, it was quite a bit harder to determine what that address was supposed to be for some nodes. Insufficient data was provided for some nodes to identify it properly.

Unicode Errors

There were 5 instances where Unicode errors triggered during parsing. That was due to special characters—like ñ in Peña Boulevard—that would trip up the converter to JSON. The code below was implemented for the 2 types of Unicode errors:

```
1.  try:
2.      address["address"][addr] = update_name(str(v_value), mapping)
3.  except UnicodeEncodeError:
4.      if str(v_value.encode('ascii', 'ignore'))[0:2] is "Pea":
5.          address["address"][addr] = "Peña Boulevard"
6.      else:
7.          address["address"][addr] = update_name(str(v_value.encode('ascii', 'ignore')),
    mapping)
```

Addressing Standards

A small amount of data included an extra k_value "address" that proved to be redundant. The exact same data was kept in the *addr:xxxxx* k_value's as well. Thus, the extra points were ignored. This would ultimately slow down parsing process if it existed more through the OSM file. It wasn't a main problem for the Denver-Boulder data, but was unnecessary nonetheless.

Data Overview

File Sizes

denver-boulder_colorado.osm is 735MB
denver-boulder_colorado.osm.json is 864MB
MongoDB d*enver* database is 304MB.

Number of Nodes

Code:

```
1.  from pymongo import MongoClient
2.  def mongo_ip():
3.      return MongoClient("192.168.1.15", 32774)
4.
5.  def insert_data(data, db):
6.      db.denver.insert(data)
7.      pass
8.
9.  def data_sources(db, pipeline):
10.     return [doc for doc in db.udacity.aggregate(pipeline)]
11.
12. if __name__ == "__main__":
13.     db = get_db()
14.     pipeline = make_pipeline()
```

```
    result = data_sources(db, [{ "$match" : {"type" : "node"} }])
    pprint.pprint(len(result))
```

Returns 3,471,143.

Number of Ways

Code:

```
result = data_sources(db, [{ "$match" : {"type" : "way"} }])
```

Returns 389,235

Number of Unique Users

Code:

```
def make_pipeline(element = "number_of_unique_users"):
    # complete the aggregation pipeline
    if element is "number_of_unique_users":
        pipeline = [
            { "$group" : { "_id" : "$created.uid",
                           "count" : { "$sum" : 1 } } },
            { "$sort" : { "count" : -1 } }
            ]

    elif element is "number_of_amenities":
        pipeline = [
            { "$match" : { "type" : "node" } },
            { "$group" : { "_id" : "$amenity",
                           "count" : { "$sum" : 1 } } },
            { "$sort" : { "count" : -1 } }
            ]
    #pprint.pprint(pipeline)
    return pipeline

if __name__ == "__main__":
    db = get_db()
    pipeline = make_pipeline("number_of_unique_users")
    result = data_sources(db, pipeline)
    pprint.pprint(len(result))
```

Returns 1,858

Number of Amenities Listed

Code:

```
pipeline = make_pipeline("number_of_amenities")
```

Returns 116

Code is changed to see some of the results from the list of amenities:

```
pipeline = make_pipeline("number_of_amenities")
result = data_sources(db, pipeline)
pprint.pprint(result[0:6])
```

Returns:

```
[{u'_id': None, u'count': 3460357},
 {u'_id': u'restaurant', u'count': 1602},
 {u'_id': u'bicycle_parking', u'count': 843},
 {u'_id': u'school', u'count': 757},
 {u'_id': u'fast_food', u'count': 721},
 {u'_id': u'bench', u'count': 669}]
```

Additional Statistics

Code for viewing the number of streets as well as names listed in the database.

```
def make_pipeline(element = "number_of_unique_users"):
    # complete the aggregation pipeline
    # ...
    elif element is "addresses_listed":
        pipeline = [
            { "$group" : { "_id" : "$address.street",
                           "count" : { "$sum" : 1 } } },
            { "$sort" : { "count" : -1 } }
            ]
    elif element is "names_listed":
        pipeline = [
            { "$group" : { "_id" : "$name",
                           "count" : { "$sum" : 1 } } },
            { "$sort" : { "count" : -1 } }
            ]

if __name__ == "__main__":
    db = get_db()
    pipeline = make_pipeline("addresses_listed")
    result = data_sources(db, pipeline)
    pprint.pprint(len(result))
    pprint.pprint(result[0:6])

    pipeline = make_pipeline("names_listed")
    result2 = data_sources(db, pipeline)
    pprint.pprint(len(result2))
    pprint.pprint(result2[0:6])
```

Returns:

```
1905
[{u'_id': None, u'count': 3827658},
 {u'_id': u'Lipan Street', u'count': 447},
 {u'_id': u'West 32Nd Avenue', u'count': 426},
 {u'_id': u'South Parker Road', u'count': 352},
 {u'_id': u'Emerson Street', u'count': 323},
 {u'_id': u'West 35Th Avenue', u'count': 320}]
50570
[{u'_id': None, u'count': 3737374},
 {u'_id': u'Washington Street', u'count': 420},
 {u'_id': u'RTD Light Rail', u'count': 288},
 {u'_id': u'Sheridan Boulevard', u'count': 250},
 {u'_id': u'Huron Street', u'count': 211},
 {u'_id': u'South Parker Road', u'count': 211}]
[Finished in 18.0s]
```

Other Ideas About the Datasets

Open Source Bots

Many of the users engaged in data entry for OpenStreetMap had “bot” in the user handle. This suggests that the data entries were from an automated script rather than human entry. There was some suggest OSM had an open source bot called *Redaction Bot*; however, there didn’t appear to be a lot of activity around it. There was activity on some third party GitHub OSM bots. Thus, OSM might consider a hackathon around bot creation while simultaneously trying to create a gold standard for an open source bot. Content from a hackathon could be used to derive said standard. This would especially be valuable for new or soon-to-be college graduates to participate in to showcase their skill set as they attempt to enter the workforce.

The benefit of this would be to drive for a more rooted community while also creating a bot to enforce standards. The potential problem would be the difficulty of maintaining a gold standard for a bot. The more people contribute to a code; the more derivative versions exist. It would take some dedication to make sure all relevant changes are being rolled into the gold standard.

Address Naming Convention

Naming convention was consistent across the data sets for the most part. There were issues with “second line” addresses (e.g. addresses that include a suite or building number). There should be a second configuration value reserved for suites, building numbers, apartment numbers, etc.

The benefit is that it would make the analysis of streets much easier. The potential problem could be that any standard would need to be enforced. Even in the data set now, most people tended to follow a common design of laying out data. But, there will almost always be outliers that deviate from a specific standard.