

Data Wrangling with MongoDB

OpenStreetMap Project

Summary

This project is an exercise in extracting data from an open source map project, assessing the validity of the data, cleaning the data and loading the data into a MongoDB database – a storage format that is optimized for large-scale, semi-structured data.

The data for this project comes from [Map Zen](#)'s pre-defined Raleigh, North Carolina metropolitan area. The data is contributed by users through the [OpenStreetMap project](#). Because the data is manually contributed, it is subject to entry error and inconsistencies.

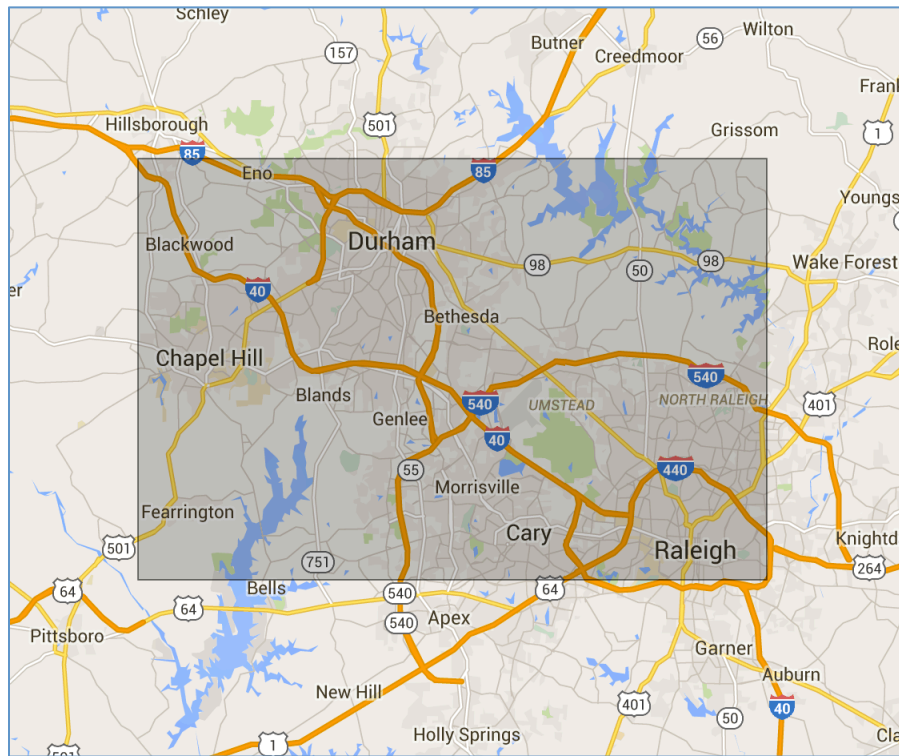


Figure 1: Map Zen's Raleigh metro region - bounded by latitude and longitude max and min values

Data Overview *

The project data is a 465MB OpenStreetMap file in XML format – it is organized into a hierarchical system of elements. Volunteer contributors enter the map data. The Raleigh metropolitan area map data has been edited by 716 different contributors.

The data contains the following elements:

Element	Count	Description
Osm	1	OSM is the top-level element.
Bounds	1	The Bounds element defines the maximum and minimum latitude and longitude boundaries of the OSM file region.
Node	2,252,660	A Node represents a specific point on the earth's surface defined by its latitude and longitude.
Way	224,066	A Way defines linear features and area boundaries.
Tag	827,245	Tags describe the meaning of a particular element.
Relation	797	A Relation documents a relationship between two or more data elements.
Member	7947	The Member elements identify the members of a relation.

For this project, the focus is on nodes and ways and their associated tags because these are the elements that contain the address information – the fields most in need of cleaning and standardization.

The OpenStreetMap project outlines the features of a region. For example, amenities are community facilities used by residents or visitors. The table below shows the top 10 amenities in the Raleigh metropolitan area map data:

Amenity	Count
parking	1884
place_of_worship	535
bicycle_parking	522
restaurant	502
fast_food	253
school	221
fuel	202
bench	127

bank	111
swimming_pool	104

*See data overview source code in appendix.

Data Cleaning Plan

The data cleaning plan centers around the user-specified address information, but also involves standardization of location names, phone numbers and amenities.

Entity	Cleaning Plan
Address Number	Confirm that these are numeric
Zip Code	Confirm against a list of appropriate postal codes
Street	Standardize direction listings Standardize street name Standardize street type
City	Confirm validity
State	Set all to NC
Amenity	Standardize capitalization
Name	Standardize capitalization
Phone	Parse into components and standardize format

Address Number

My assumption is that in general the address number field will contain numeric data, however, the address number could also include a non-numeric suite designation. An audit confirms that 98% of the house number values are in fact numeric.

It will be difficult to determine the validity of the address numbers, however, I recall learning that address numbers are not randomly distributed. Around 30 percent typically begin with the digit 1, 18 percent with 2, and so on, with the smallest percentage beginning with the digit 9. This principle is known as Benford's Law. A review of the distribution of the first digit of the address numbers might be a good starting point to determine whether the address number data seems trustworthy.

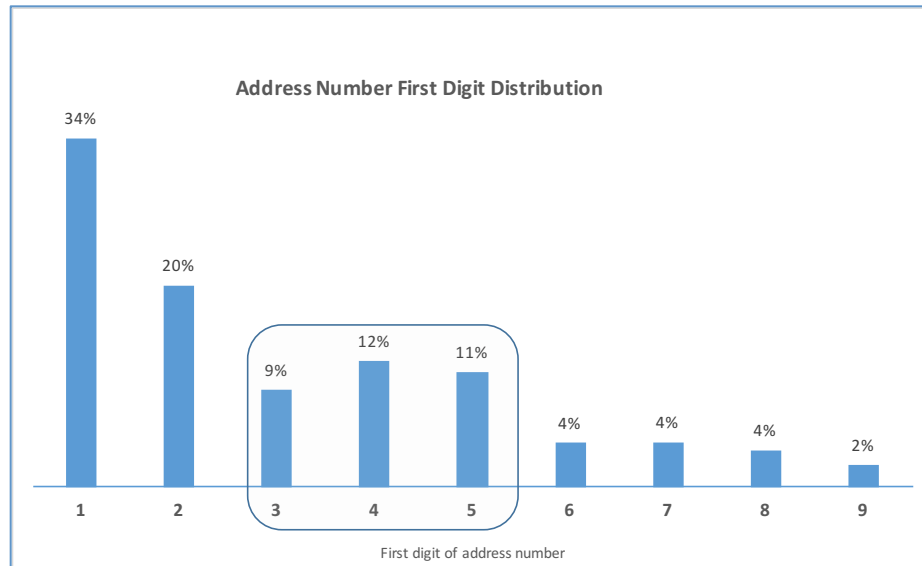


Figure 2: Review of address number first digit distribution

In this dataset the 1 and 2 are distributed as expected according to Benford's Law, but the distributional hump from digits 3 to 5 is not quite as expected. This could be an indication that these data are not trustworthy.

In reviewing the first digit, I also found a handful of records that begin with letters instead of numbers. These records can be flagged for manual review.

Zip Code

The United States Postal Service is the gold standard for zip code data in the United States. A cleaning approach could be to download from the US Postal Service a list of valid zip codes for each of the cities included in the data and compare the entered data against the gold standard data.

Street

Direction

For street names that begin with a directional component: North, South, East, West, some of the listings abbreviate the direction. I standardized this component by converting the abbreviation into the non-abbreviated form.

Street type

There were many different variations of street type. After an audit to identify the variations, I created a mapping function to standardize the street type.

Street name

During the process of auditing the data, I found examples of street names that differed based on punctuation and abbreviation, for example:

Martin Luther King Jr Blvd
Martin Luther King Junior Blvd
Martin Luther King Jr. Blvd

However, I believe that the street names will be difficult to programmatically standardize. It may be possible to get an official listing of streets in a region to use as a gold standard, however, obtaining such a list could be cumbersome.

City

An audit of the cities in the map data revealed inconsistencies in capitalization and spelling


Value	Count
Chapel Hill	470
chapel Hill	2
Chapel Hill NC	1
Durham	1,287
durham	2
Ra	1
Raleigh	874
Ra	1
Raleigh	874
raleigh	2

As with the street type, I created a mapping function to standardize the city values.

State

All addresses are in North Carolina, although the data included variations listed in the table below. I set the state value to NC during the cleaning process.

Value	Count
NC	4,267
NY	1
North Carolina	296
nc	2

**NC**

Phone Number

Create a pattern match to parse phone numbers into their component parts and standardize the format of the phone numbers.

Other Ideas About the Data

Raleigh Data

The OpenStreetMap data for the Raleigh, North Carolina metropolitan area has issues of completeness: the data was certainly not a comprehensive representation of the area and the data has issues of quality: the address data in particular was full of typographical errors and inconsistencies.

In order to obtain more complete and more standardized data, the following approaches could be adopted.

- Use fuzzy matching to compare OSM street names with a gold standard listing of street names for each region.
- Impute zip codes from latitude and longitude entries. Validate existing zip codes with the US Postal Service API.
- Download data from Yelp API to populate a more comprehensive listing of amenities and services.

It could be difficult to obtain gold standard data. I found a website that allowed users to download a listing of street addresses for a given city, however, the API service involved a paid subscription. Once an effort is made to thoroughly clean and standardize the data, it could also be difficult to keep it up to date and to keep user-generated errors from re-occurring. However, the Yelp API does include an indicator to reflect whether a business is permanently closed, which would be useful in keeping the OSM data up to date.

Other Uses

I've addressed above additional ideas for improving the specific Raleigh area map data. However, initially I was unclear why it was actually important for the OpenStreetMap to be comprehensive since location and feature information is available from other sources like Google Maps and Yelp. However, I've now come to understand that the OpenStreetMap project was developed in reaction to a preponderance of proprietary map data and I've realized that for low-infrastructure communities, much of their geographic and feature information is undocumented.

I've learned that mapping is now used as a tool for emergency aid efforts. After the 2010 earthquake in Haiti, volunteers used satellite imagery to map roads, buildings and refugee camps. These maps helped to expedite the delivery and distribution of emergency supplies. However, this so-called "crisis mapping" is done after the disaster happens and relies on motivated volunteers with varying degrees of skill and relies on satellite imagery that may be hard to interpret or out of date.

The RedCross has been leading a Missing Maps initiative that aims to get volunteers involved in a coordinated effort to map the entire globe, rather than relying on a flurry of activity after a disaster happens.

The RedCross reports that last year volunteer mappers mapped areas in West Africa which allowed humanitarian organizations to track the Ebola virus and figure out which areas needed the most help.

All of these efforts in mapping previously unmapped regions of the world seem thus far to have involved distant, skilled volunteers with good access to technology. However, cell phones are nearly ubiquitous all over the world at this point. I wonder if local communities could be involved and incentivized to create and upload map data from their cell phones. By involving local communities, we'd get accurate and feature-rich information. However, people would have to be trained to collect and transmit the information, which would be especially difficult in low-literacy areas. Also, it might be difficult in a low-infrastructure area - outside of a time of crisis - for communities to see the value of the mapping work.

Regardless of the difficulties involved in creating a comprehensive, crowd-sourced map of the world, the benefits are boundless.

Appendix

MongoDB Code

MongoDB – Import and Query Data

```
client = MongoClient('localhost:27017')
db = client.OSM

#JSON formatted data
db.Raleigh.insert_many(data)

#Confirm counts
db.Raleigh.find().count()
#Count is 2476726, which is what is expected

db.Raleigh.find({"type":"node"}).count()
#Count is 2252656, which is not quite what was expected

db.Raleigh.find({"type":"way"}).count()
#Count is 224062, which is not quite what was expected

#Examine type values since count results were not quite as expected
pipeline = [{"type":{"type":{"exists":1}}}, {"group":{"_id":"type","count":{"sum":1}}}, {"sort":{"count":-1}}, {"limit":10}]
```

```

print(list(db.Raleigh.aggregate(pipeline)))
#Result:
[{'_id': 'node', 'count': 2252656}, {'_id': 'way', 'count': 224062}, {'_id': 'sewage', 'count': 3}, {'_id': 'water', 'count': 1}, {'_id': 'route', 'count': 1}, {'_id': 'video', 'count': 1}, {'_id': 'public', 'count': 1}, {'_id': 'audio', 'count': 1}]

#Examine amenities
pipeline = [{"$match": {"amenity": {"$exists": 1}}}, {"$group": {"_id": "$amenity", "count": {"$sum": 1}}}, {"$sort": {"count": -1}}, {"$limit": 10}]

print (list(db.Raleigh.aggregate(pipeline)))
#Result:
[{'_id': 'node', 'count': 2252656}, {'_id': 'way', 'count': 224062}, {'_id': 'sewage', 'count': 3}, {'_id': 'water', 'count': 1}, {'_id': 'route', 'count': 1}, {'_id': 'video', 'count': 1}, {'_id': 'public', 'count': 1}, {'_id': 'audio', 'count': 1}]

#Get count of unique users
len(db.Raleigh.distinct("created.user"))
#Count is 716

```

Data Processing Caveat

The data processing focused exclusively on node elements and way elements. After loading the data into mongodb, I noticed that although the total number of records loaded into the database matched the number of records in the JSON file, the reported counts of node and way types did not match the counts obtained from querying the XML file prior to parsing.

Further investigation revealed that a few of the tags in the XML file have keys called 'type', so during processing, the assigned type='way' or type='node' value was overwritten.

Type	Count
node	2,252,656
way	224,062
sewage	3
water	1
route	1
video	1
public	1
audio	1

The total number of elements with keys called 'type' matches the total expected number of nodes and ways. In order to not overwrite 'node' and 'way' types, perhaps the processing script could catch 'type' tags during processing and rename them as 'subtype'.

References

See: "data wrangling project references".txt