OPEN STREET MAP PROJECT
DATA WRANGLING USING MONGODB

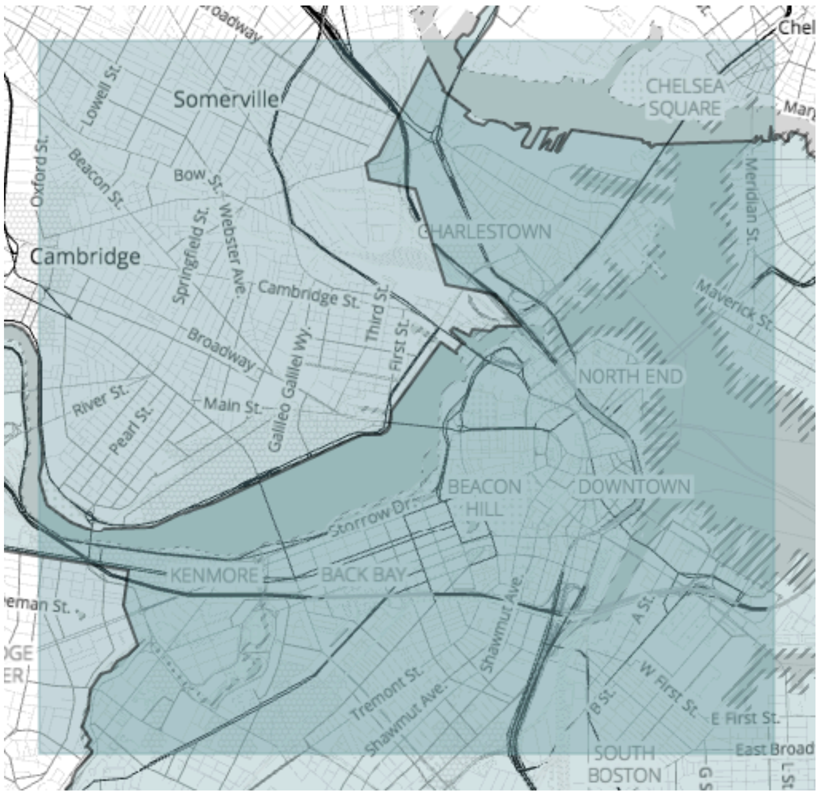
Sohan Samanta

INTRODUCTION

This project deals primarily with the data wrangling processes carried out with the help of MongoDB, for a data set pertaining to street information, downloaded from the OpenStreetMap site.
The map area selected for this project corresponds to a small portion of Boston, Massachusetts.
Map Link:

- https://mapzen.com/data/metro-extracts/your-extracts/c4681d571f3b

My initial choice was to select the map of Rolla, MO, the place where I presently reside. But the data set for this town was too small to carry out a proper investigation. Thus, the selection of the map area for this project was done randomly, only constraint being a dataset large enough for a proper investigation.



The progression of this project can be divided into the following 4 major steps:

1. **Download data and audit**

The data, downloaded as an osm file, is read and an initial audit is performed which consists of looking at the nodes and their tags to identify some issues. Since the most important information in this data set pertains to the street information, we audit them first.

We start by looking at the different types of tags and find the following 8 varieties:

Tag name	***No of tags***	***Tag name***	***No of tags***
osm	*1*	*node*	*249376*
bounds	*1*	*tag*	*165940*
member	*4691*	*way*	*42504*
relation	*561*	*nd*	*328592*

There are also secondary tags, which are identified by the 'k' value inside the tags.
The number of secondary tags found in this data set are: 858

Abbreviation Errors:

Our first order of business is identifying the inconsistencies within the street names. From just looking at the data we can find some of the common street name ending conventions in the Boston area, and they are:
Street, Avenue, Boulevard, Place, Drive, Lane, Court, Square, Lane, Road, Trail, Commons, Broadway, Center, Highway, Park, Plaza
But since the data is user populated, different users use different abbreviations in their street name ends. The different conventions and the correction that we want to make for more uniformity are as follows:

Sl. No	**Existing Name**	**Correction**
1	Ave	Avenue
2	Ave.	Avenue
3	Ext	Exit
4	Pl	Plaza
5	Rd	Road
6	ST	Street
7	St	Street
8	St.	Street
9	St,	Street
10	St	Street
11	Sq	Square
12	Sq.	Square

These corrections to be made to the street names is what we deduce in this audit step.

2. Format data and create a Json file

In the earlier section we audited the data and came up with some corrections for the street name abbreviation errors. In this step we reshape the data so that it can be saved in the Json format, for ease of access and further audit.

Our reshaping of the data will follow the following blueprint:

```
{
"id": "2406124091",
"type: "node",
"visible":"true",
"created": {
          "version":"2",
                    "changeset":"17206049",
          "timestamp":"2013-08-03T16:43:42Z",
          "user":"linuxUser16",
               "uid":"1219059"
              },
"pos": [41.9757030, -87.6921867],
"address": {
    "housenumber": "5157",
      "postcode": "60625",
      "street": "North Lincoln Ave"
           },
"amenity": "restaurant",
"cuisine": "mexican",
"name": "La Cabana De Don Luis",
"phone": "1 (773)-271-5176"
}
```

3. Audit and cleaning of data in the Json file

We have seen to the major corrections for the street name abbreviation errors. Since in the formatting done in the previous section made a separate dictionary to hold all of the address details, we now audit this address dictionary for more errors.

More errors in street names:

Sans than the street name, which we have already corrected, there are 3 more sections that needs correction. These are:

1. city
2. state
3. country

The error in these three sections are similar and are again due to the use of different abbreviations for the city, state and country names.

Our audit of these elements gives the following output:

No of distinct Cities	*:*	*16*
No of distinct Countries	*:*	*2*
No of distinct States	*:*	*5*

However, the correct values for the above should be:

No of distinct Cities	*:*	*10*
No of distinct Countries	*:*	*1*
No of distinct States	*:*	*1*

We look at the data and find out the following corrections:

```
city_replacements = { 'BOSTON' : 'Boston',
                      'Boston, MA' : 'Boston',
                      'Cambridge, MA' : 'Cambridge',
                      'Cambridge, Massachusetts' : 'Cambridge',
                      'boston' : 'Boston',
                      'somerville' : 'Somerville'}

country_replacements = { 'USA' : 'US' }

state_replacements = {'Massachusetts' : 'MA',
                      'MA- MASSACHUSETTS' : 'MA',
                      'Ma' : 'MA',
                      'ma' : 'MA'}
```

Other errors: Postal Codes

Postal codes posed a different sort of problem than the abbreviation issues encountered for the street names. There were primarily 3 issues:

- Characters present before or after the code, as in MA 02116
- Postal code add-ons like 02114-3203
- Only characters instead of a postal code

One of the things that I was surprised to find was that all the postal codes that were present in the data were actually present in the region of the map I selected. Thus, other than the above errors, there were no entries that had a postal code of some region that lay outside of Boston.

I decided to correct the above mistakes in the following ways:

- For the ones with characters present either before or after the code, I just removed the character and kept the rest.
- For the postal codes with add-ons, I created an added field called 'add-on' that contained the 4 add-on digits after the postal code
- There were 2 entries that were missing postal code altogether. I searched their postal code in the rest of the data by matching the city and street to fill the missing value.

Some points about the above solution:

- **Benefits:**
  - Filling of missing values gives a complete address
  - Separation of add-ons from the postal code gives more dynamic and meaning to the address
  - Correcting postal codes by removing characters from it
- **Anticipated issues**
  - Filling of missing values by looking at other data in the set may not always give an output.
  - While add-ons of the postal code give more detail, it increases the number of fields in the address, and may not be of much importance depending upon the type of investigation to be done on the data.
  - Only characters occurring in the front of the postal codes are being removed. The code needs to be revised if characters are present anywhere else for other data sets.

After correction, we save the file as boston_corrected.json.

4. Import data into MongoDB

In this step we simply import the data from the boston_corrected.json file into the mongoDB.

5. Some analysis using MongoDB tools

Here we make some data analysis using the tools of available for mongoDB. The analysis results are detailed in the sections below.

DATA OVERVIEW

File Sizes:

boston.osm	-	58.8 MB
boston.json	-	67.7 MB
boston_corrected	-	67.7 MB

DATA EXPLORATION

No of documents:

```
> db.boston.count()
291880
```

No of way tags:

```
> db.boston.find({'type':'way'}).count()
42489
```

No of way tags:

```
> db.boston.find({'type':'node'}).count()
249276
```

No of unique users: 779

Top user: crschmidt

Additional data explorations:

CODE	RESULT
POSTAL CODE:	
postcode = db.boston.aggregate([{"$match":{"address.postcode":{"$exists":1}}}, {"$group":{"_id":"$address.postcode","count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":10}]) print 'Descending order of postcodes:' for item in postcode: print '{} entries for postal code: {}'.format(item['count'], item['_id'])	***10 most common postal codes:*** 428 entries for postal code: 02139; 94 entries for postal code: 02114; 70 entries for postal code: 02143; 61 entries for postal code: 02116; 57 entries for postal code: 02215; 43 entries for postal code: 02142; 43 entries for postal code: 02145; 37 entries for postal code: 02138; 30 entries for postal code: 02210; 30 entries for postal code: 02111

```
postcode = db.boston.aggregate([
{"$match":{"address.postcode":{"$exists":1}}},
{"$group":{"_id":"$address.postcode","count":{"$sum":1}}},
{"$sort":{"count":-1}}, {"$limit":10}])

print 'Descending order of postcodes:'

for item in postcode:
    print '{} entries for postal code: {}'.format(item['count'],
item['_id'])
```

10 most common postal codes:

```
428 entries for postal code: 02139
94 entries for postal code: 02114
70 entries for postal code: 02143
61 entries for postal code: 02116
57 entries for postal code: 02215
43 entries for postal code: 02142
43 entries for postal code: 02145
37 entries for postal code: 02138
30 entries for postal code: 02210
30 entries for postal code: 02111
```

The code for the rest of the explorations are similar and hence omitted.

COMMON STREETS:

10 most common streets:

```
138 entries for street: Massachusetts Avenue
71 entries for street: Cambridge Street
67 entries for street: River Street
56 entries for street: Boylston Street
53 entries for street: Somerville Avenue
51 entries for street: Fayette Street
40 entries for street: Albion Street
37 entries for street: Beacon Street
36 entries for street: Summer Street
35 entries for street: Main Street
```

PARKING PLACES:

Common Parking places:

```
{u'count': 268, u'_id': None}
{u'count': 94, u'_id': u'surface'}
{u'count': 29, u'_id': u'multi-storey'}
{u'count': 13, u'_id': u'underground'}
{u'count': 2, u'_id': u'metered'}
{u'count': 1, u'_id': u'rooftop'}
```

RESTAURANTS:

Common dishes in Restaurants:

```
{u'count': 160, u'_id': None}
{u'count': 20, u'_id': u'pizza'}
{u'count': 19, u'_id': u'mexican'}
{u'count': 18, u'_id': u'italian'}
{u'count': 18, u'_id': u'american'}
{u'count': 14, u'_id': u'chinese'}
{u'count': 12, u'_id': u'indian'}
{u'count': 11, u'_id': u'asian'}
{u'count': 9, u'_id': u'thai'}
{u'count': 8, u'_id': u'japanese'}
```

RELIGIOUS PLACES:

Common Religious places:

```
{u'count': 111, u'_id': u'christian'}
{u'count': 12, u'_id': None}
{u'count': 5, u'_id': u'jewish'}
{u'count': 2, u'_id': u'unitarian_universalist'}
{u'count': 2, u'_id': u'buddhist'}
{u'count': 1, u'_id': u'muslim'}
```

OTHER IDEAS ABOUT THE DATA SET:

There are a lot of missing data, like the classification of parking spaces and the religions practiced in certain religious places/buildings, that could be filled. Although these data are missing, the corresponding addresses for these locations are available for most cases. It is possible to cross reference these locations from another data set or simply by using the google search engine to find out the missing details.
However, if a downloadable data set is not available consisting of the missing data we want to add, corresponding to their addresses, it will become a very time consuming and tedious job to individually search each location and fill the missing details.

CONCLUSION

In this project we looked at the geographical data for a small part of Boston, that we downloaded from the OpenStreetMap site and performed auditing, cleaning and data analysis using MongoDB.

The wrangling process followed the action of auditing to identify errors, the cleaning to remove those errors and generally insert the correct entries in their place and then perform some data analysis on the cleaned data using MongoDB queries.
The most common error found in the street data was the presence of different abbreviations by different users. Abbreviations for cities and countries were also present which were removed. Next was the missing data. Some were filled using other data points from the set, like the missing postal codes filled using the street and city details.
Other missing data can also be filled by similar as well as more exhaustive measures of individual searches and cross verifications from other data sources.

Overall, I found this data set to be sufficiently well maintained, with not many major areas of error. As with any user populated data, there will always be some errors. However, more depth of knowledge can be provided with some more organized fields of entry.