OpenStreetMap Project Data Wrangling with MongoDB

*Ye Xu*

Map Area: Boston, MA, United States

<https://mapzen.com/metro-extracts/>

[1. Problems Encountered in the Map](https://docs.google.com/document/d/1F0Vs14oNEs2idFJR3C_OPxwS6L0HPliOii-QpbmrMo4/pub#h.ueey7dly83g7)

[Over-­abbreviated Street Names](https://docs.google.com/document/d/1F0Vs14oNEs2idFJR3C_OPxwS6L0HPliOii-QpbmrMo4/pub#h.l5zk1sr6iqsy)

[Postal Codes](https://docs.google.com/document/d/1F0Vs14oNEs2idFJR3C_OPxwS6L0HPliOii-QpbmrMo4/pub#h.1gvukyc8hthj)

[2. Data Overview](https://docs.google.com/document/d/1F0Vs14oNEs2idFJR3C_OPxwS6L0HPliOii-QpbmrMo4/pub#h.ql5hndj5vh2a)

[3. Additional Ideas](https://docs.google.com/document/d/1F0Vs14oNEs2idFJR3C_OPxwS6L0HPliOii-QpbmrMo4/pub#h.30qfugxkfikk)

[Contributor statistics and gamification suggestion](https://docs.google.com/document/d/1F0Vs14oNEs2idFJR3C_OPxwS6L0HPliOii-QpbmrMo4/pub#h.vtjnh5x9putq)

[Additional data exploration using MongoDB](https://docs.google.com/document/d/1F0Vs14oNEs2idFJR3C_OPxwS6L0HPliOii-QpbmrMo4/pub#h.7ygo1ggwcb7)

[Conclusion](https://docs.google.com/document/d/1F0Vs14oNEs2idFJR3C_OPxwS6L0HPliOii-QpbmrMo4/pub#h.rc7ls9o2vj3)

**1. Problems Encountered in the Map**

I chose the Boston as the city for my project. Initially I looked at a small sample size of the data with the data.py file provisioned in the last lesson of the course. Other than the mappings I finished in the lecture, there are some other abbreviated street types that were not included in the mapping. I iteratively ran this file and added more mappings in the dictionary, finally the mapping dictionary looks something like the follows:

mapping = { "St": "Street",

"St.": "Street",

"Ave": "Avenue",

"Rd.": "Road",

"ave": "Avenue",

"avenue": "Avenue",

"rd." : "Road",

"st" : "Street",

"street" : "Street",

"Street." : "Street",

"Sq." : "Square",

"Pkwy" : "Parkway",

"Ct" : "Court",

}

In the provisioned “data.py” file, I did the following things:

1. For each tag with the value “node” and “way”, the data.py file would process it and convert the tag into a python dictionary.
2. All attributes of "node" and "way" were turned into regular key/value pairs, except the following:
3. Attributes in the CREATED array were added under a key "created"
4. Attributes for latitude and longitude were added to a "pos" array,
5. The values inside "pos" array are floats
6. If second level tag "k" value contains problematic characters, it was ignored
7. If second level tag "k" value starts with "addr:", it was added to a dictionary "address"
8. If second level tag "k" value does not start with "addr:", but contains ":", process it the same as any other tag.
9. If there is a second ":" that separates the type/direction of a street, the tag was ignored.
10. For the address sub field, I used the mapping I created before to store the data.
11. for "way" specifically:

<nd ref="305896090"/>

<nd ref="1719825889"/>

were turned into: "node\_refs": ["305896090", "1719825889"]

1. I did each tag element with the above steps, and turn it into a json file element.
2. Then I inserted all of the converted data (.json) into MongoDB.

**2. Data Overview**

This section contains basic statistics about the dataset and the MongoDB queries used to gather them.

File sizes

boston\_massachusetts.osm ......... 361 MB

boston\_massachusetts.osm.json.... 412 MB

# Number of documents

> db.boston\_open.find().count()

1849216

# Number of nodes

> db.boston\_open.find({"type":"node"}).count()

1603318

# Number of ways

> db.boston\_open.find({"type":"way"}).count()

245898

# Number of unique users

> db.boston\_open.distinct({"created.user"}).length

336

# Top 1 contributing user

> db.boston\_open.aggregate([{"$group":{"\_id":"$created.user", "count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":1}])

{'ok': 1.0, 'result': [{'\_id': 'crschmidt', 'count': 1060051}]}

# Number of users appearing only once (having 1 post)

> db.boston\_open.aggregate([{"$group":{"\_id":"$created.user", "count":{"$sum":1}}}, {"$group":{"\_id":"$count", "num\_users":{"$sum":1}}}, {"$sort":{"\_id":1}}, {"$limit":1}])

{'ok': 1.0, 'result': [{'\_id': 1, 'num\_users': 215}]}

# “\_id” represents postcount

**3. Additional Ideas**

**Contributor statistics and gamification suggestion**

Another problem in the data is that some nodes already have an “address” attribute. For example, some nodes have “address” = “S Tryon St”. In such a case, there are two options, one is to convert it to the consistent address dictionary, as other nodes, another one is just to leave it like that. I converted it to be consistent with other addresses.

**Additional data exploration using MongoDB queries**

# Top 10 appearing amenities

> db.boston\_open.aggregate([{"$match":{"amenity":{"$exists":1}}}, {"$group":{"\_id":"$amenity",

"count":{"$sum":1}}}, {"$sort":{"count":­1}}, {"$limit":10}])

{'ok': 1.0,

'result': [{'\_id': 'parking', 'count': 933},

{'\_id': 'bench', 'count': 740},

{'\_id': 'school', 'count': 686},

{'\_id': 'restaurant', 'count': 446},

{'\_id': 'parking\_space', 'count': 444},

{'\_id': 'place\_of\_worship', 'count': 368},

{'\_id': 'library', 'count': 324},

{'\_id': 'bicycle\_parking', 'count': 211},

{'\_id': 'cafe', 'count': 170},

{'\_id': 'fast\_food', 'count': 152}]}

# Biggest religion (no surprise here)

> db.boston\_open.aggregate([{"$match":{"amenity":{"$exists":1}, "amenity":"place\_of\_worship"}},

{"$group":{"\_id":"$religion", "count":{"$sum":1}}},

{"$sort":{"count":­1}}, {"$limit":1}])

{'ok': 1.0, 'result': [{'\_id': 'christian', 'count': 321}]}

# Most popular cuisines

> db.boston\_open.aggregate([{"$match":{"amenity":{"$exists":1}, "amenity":"restaurant"}}, {"$group":{"\_id":"$cuisine", "count":{"$sum":1}}},        {"$sort":{"count":­1}}, {"$limit":2}])

{'ok': 1.0,

'result': [{'\_id': 'pizza', 'count': 29},

{'\_id': 'american', 'count': 26}]}

**Conclusion**

In this final project I downloaded the Boston area street data map from the open street map website. Before doing any database storing and querying, I cleaned up the raw data in the form described in this report. Then I used mongodb python module to store these data into a mongodb database. These data files are huge, insertion each element into the database took quite a while. But since it is a No-SQL database, the query afterwards was not too slow. Although the data size was a big as 1849214, most of the queries in this report just took less than 5 seconds to finish.

Published by [Google Drive](https://docs.google.com/)–[Report Abuse](https://docs.google.com/abuse?id=1F0Vs14oNEs2idFJR3C_OPxwS6L0HPliOii-QpbmrMo4)–Updated automatically every 5 minutes