## OpenStreetMap Project in Data Wrangling with MongoDB

Name: Anna Li

Email: annax.li10@gmail.com

Map Area: Auckland, New Zealand

URL: <http://www.openstreetmap.org/export#map=12/-36.8612/174.7839>

I chose my hometown, Auckland, New Zealand, for project 2 because I grew up in this city, and I know the area well. New Zealand is also far away from the rest of the world, and there is little OpenStreetMaps data. I want to contribute to improve the quality of the Auckland map.

The area I chose to wrangle includes the central Auckland city (I live in Epsom) and the greater suburban region, such the North Shore (Takapuna, Birkenhead, and Devonport), all the way west to Henderson and Te Atau and as far south as Mangere Bridge. I have friends living in all of these areas

I downloaded the data in two steps: 1. I found the area I was interested in using OpenStreetMap (<http://www.openstreetmap.org/export#map=12/-36.8612/174.7839>) and made sure it was over 50MB using this website, 2. I ran a mapping query from Overpass API (<http://overpass-api.de/query_form.html>) to download the xml data.

**Table of Contents**

OpenStreetMap Project in Data Wrangling with MongoDB 1

Problems/Points of Interest 1

*Standardization* 1

Appendix 2

## Problems/Points of Interest

After downloading and auditing the original dataset, I discovered three problems in the data set that required wrangling and cleaning as well as two interesting points that merit investigation. The three problems:

* Lack of standardization in street types (e.g. ‘Queen St.' and ‘Queen st’)
* Incorrect values in fields (e.g. ‘Newmarket’ surburb is misplaced in the streetname)
* Missing values (city was often missing from the address)

Two interesting points:

## Standardization

Inconsistent street types

Using the aklaudit.py script, which is modified from the auditing exercise in lesson 6, I discovered inconsistent street types that required cleaning. For example, “Queen St.” and “Queen st” needed to be standardized to “Queen Street”.

Unexpected directional street endings, which are appropriate to New Zealand streets which place the direction at the end of the street, not before like in the U.S. e.g. Victoria Street West is a typical New Zealand street name compared to West El Camino Real which is more typical of a California street name

{'West': set(['Customs Street West',

'Sylvan Avenue West',

'Victoria Street West',

'Wellesley Street West']),

'East': set(['Customs Street East',

'Durham Street East',

'Greenlane East',

'Sylvan Avenue East',

'Victoria Street East',

'Virginia Avenue East',

'Wellesley Street East'])}

Import OSM Auckland json data into MongoDB

mongoimport --db auckland --collection full --file osm-auckland.json --jsonArray

## Appendix

For a list of sources, please refer to the Resources Used.txt file in the Project2 folder in Github (relative path in Github is ../NanoDA/DataWranglingMongoDB/Project2/Resources Used.txt)

“I hereby confirm that this submission is my work. I have cited above the origins of any parts of the submission that were taken from Websites, books, forums, blog posts, github repositories, etc. By including this in my email, I understand that I will be expected to explain my work in a video call with a Udacity coach before I can receive my verified certificate.”

Is there any other important information that you would want your project evaluator to know?

Use this space to communicate with your project evaluator. Is there anything you would like to communicate? Feedback or suggestions?

[*https://www.openstreetmap.org/relation/177415*](https://www.google.com/url?q=https%3A%2F%2Fwww.openstreetmap.org%2Frelation%2F177415&sa=D&sntz=1&usg=AFQjCNHfumZldEV08ImbcbWHb0Xg3Ejlmg)

[*http://metro.teczno.com/#charlotte*](http://www.google.com/url?q=http%3A%2F%2Fmetro.teczno.com%2F%23charlotte&sa=D&sntz=1&usg=AFQjCNGCsHL2If59bTZ7McZVY_nEgQR40A)

[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**

After initially downloading a small sample size of the Charlotte area and running it against a provisional data.py file, I noticed three main problems with the data, which I will discuss in the following order:

* Over­-abbreviated street names (“S Tryon St Ste 105”)
* Inconsistent postal codes (“NC28226”, “28226­0783”, “28226”)
* “Incorrect” postal codes (*Charlotte area zip codes all begin with “282” however a large portion of all* *documented zip codes were outside this region.*)

**Over-­abbreviated Street Names**



Once the data was imported to MongoDB, some basic querying revealed street name abbreviations and postal code inconsistencies. I updated all substrings in problematic address strings, such that “S Tryon St Ste 105” becomes “South Tryon Street Suite 105”.

**Postal Codes**

Postal code strings posed a different sort of problem, forcing a decision to strip all leading and trailing characters before and after the main 5-­digit zip code. This effectually dropped all leading state characters (as in “NC28226”) and 4­-digit zip code extensions following a hyphen (“28226­0783”). This 5­-digit constriction benefits MongoDB aggregation calls on postal codes.

Regardless, after standardizing inconsistent postal codes, some altogether “incorrect” (or perhaps misplaced?) postal codes surfaced when grouped together with this aggregator:

# Sort postcodes by count, descending

db.char.aggregate([{"$match":{"address.postcode":{"$exists":1}}}, {"$group":{"\_id":"$address.postcode", "count":{"$sum":1}}}, {"$sort":{"count":­1}}])

Here are the top two results, beginning with the highest count:

[ {"\_id" : "29732", "count" : 103},

{"\_id" : "28134", "count" : 27}, ...

Considering the relatively few documents that included postal codes, of those, it appears that out of the top ten, seven aren’t even in Charlotte. That struck me as surprisingly high to be a blatant error, and found that the number one postal code and all others starting with “297” lie in Rock Hill, SC. So, I performed another aggregation to verify a certain suspicion...

# Sort cities by count, descending

> db.char.aggregate([{"$match":{"address.city":{"$exists":1}}}, {"$group":{"\_id":"$address.city", "count":{"$sum":1}}}, {"$sort":{"count":­1}}])

And, the results, edited for readability:

[ {"\_id" : "Rock Hill", "count" : 111},...

These results confirmed my suspicion that this metro extract would perhaps be more aptly named “Metrolina” or the “Charlotte Metropolitan Area” for its inclusion of surrounding cities in the sprawl.

So, these postal codes aren’t “incorrect,” but simply unexpected.

**2. Data Overview**

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

File sizes

charlotte.osm ......... 294 MB

charlotte.osm.json .... 322 MB

# Number of documents

> db.char.find().count()

1555851

# Number of nodes

> db.char.find({"type":"node"}).count()

1471349

# Number of ways

> db.char.find({"type":"way"}).count()

84502

# Number of unique users

> db.char.distinct({"created.user"}).length

336

# Top 1 contributing user

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

[ { "\_id" : "jumbanho", "count" : 823324 } ]

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

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

[ {"\_id":1,"num\_users":56} ]

# “\_id” represents postcount

**3. Additional Ideas**

**Contributor statistics and gamification suggestion**

The contributions of users seems incredibly skewed, possibly due to automated versus manual map editing (*the word “bot” appears in some usernames*). Here are some user percentage statistics:

* Top user contribution percentage (“jumbanho”) - 52.92%
* Combined top 2 users' contribution (“jumbanho” and “woodpeck\_fixbot”) - 83.87%
* Combined Top 10 users contribution - 94.3%
* Combined number of users making up only 1% of posts - 287 (about 85% of all users)

Thinking about these user percentages, I’m reminded of “gamification” as a motivating force for contribution. In the context of the OpenStreetMap, if user data were more prominently displayed, perhaps others would take an initiative in submitting more edits to the map. And, if everyone sees that only a handful of power users are creating more than 90% a of given map, that might spur the creation of more efficient bots, especially if certain gamification elements were present, such as rewards, badges, or a leaderboard.

**Additional data exploration using MongoDB queries**

# Top 10 appearing amenities

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

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

[ {"\_id":"place\_of\_worship","count":587},

{"\_id" : "school", "count" : 416}, ….

# Biggest religion (no surprise here)

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

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

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

[ {"\_id":"christian","count":577} ]

# Most popular cuisines

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

[ { "\_id" : "pizza", "count" : 9},

{ "\_id" : "american", "count" : 9} ...

**Conclusion**

After this review of the data it’s obvious that the Charlotte area is incomplete, though I believe it has been well cleaned for the purposes of this exercise. It interests me to notice a fair amount of GPS data makes it into OpenStreetMap.org on account of users’ efforts, whether by scripting a map editing bot or otherwise. With a rough GPS data processor in place and working together with a more robust data processor similar to data.py think it would be possible to input a great amount of cleaned data to OpenStreetMap.org.