## 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 with metadata:

(node(-36.9469,174.6243,-36.7755,174.94355);<;);out meta;

**Table of Contents**

OpenStreetMap Project in Data Wrangling with MongoDB 1

Problems in the OSM Auckland Data 1

*Standardization* 2

*Incorrect values in fields* 2

*Missing values* 3

Overview of OSM Auckland Data 4

*Data file sizes* 4

MongoDB descriptive statistics 4

Appendix 4

## Problems in the OSM Auckland Data

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’ suburb is misplaced in the street)
* Missing values (city was often missing from the address)

Command line function to import OSM Auckland json data into MongoDB

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

## 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”. I updated inconsistent street types to match the standardized format and removed leading and trailing whitespaces, for example:

Gillies Ave => Gillies Avenue

Erson Ave => Erson Avenue

Queen st => Queen Street

Queen St. => Queen Street

New North Rd => New North Road

Note: New Zealand streets frequently have the street direction appended at the end of the street name, such as “Customs Street West”. I decided not to remove the direction because they are meaningful. This happens because very long streets stretch across the city and “90 Green Lane East” is in a different suburb (Remuera) than “90 Green Lane West” (Epsom).

After importing the JSON-formatted data into MongoDB, I wrote queries with the pymongo connector to retrieve the street names and cities from the Auckland data.

db.auckland.aggregate( [{'$match': {'address.street': {'$exists': 1}}},'$group': {'\_id': '$address.street','count': { '$sum': 1} }},

{'$sort': {'count': -1}}, {'$limit': 100}] )

I was able to look at the different street names to find incorrect values, such as “Hurstmere” which is missing “Road” or “Waverley” which is missing “Avenue” (I fixed this in aklaudit.py). I noticed some streets have unusual names and types, such as “The Strand”. But this is a correct street name (The Strand is an arcade-style square in the Auckland city center so its street type is an unusual exception) and does not require cleaning.

## Incorrect values in fields

City values frequently contain additional data like house number and street

After exploring the addresses, I audited the city data to determine where what kinds of inconsistent formatting and values occurred, and therefore what needed cleaning. I modified the MongoDB aggregation pipeline query for street to work on city:

db.auckland.aggregate( {'$match': {'address.city': {'$exists': 1}}}, {'$group': {'\_id': '$address.city', 'count': { '$sum': 1} }}, {'$sort': {'count': -1}}] )

The city field had two main problems:

* Inconsistent or incorrect spellings of “Auckland”, such as “auckland”, “1 Auckland”, and “Auckalnd”
* Most of the city values were suburbs in Auckland, such as “Parnell”, “Takapuna”, and “Morningside”. Sometimes there was a suburb concatenated with the city, such as “Birkenhead, Auckland” and “New Lynn, Auckland”.

The city ‘Auckland’ appears correctly 460 times in the OSM data set. There are 28 instances where it requires cleaning are listed below (city and number of occurrences):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 'Waterview' | 1 | 'North Shore': | 3 | 'Kelston, Auckland' | 1 | 'Takapuna' | 5 |
| 'auckland' | 1 | 'Royal Oak' | 1 | 'Birkenhead, Auckland' | 1 | 'Mt Wellington' | 1 |
| 'Waitemata' | 5 | 'Morningside' | 2 | 'New Lynn, Auckland' | 3 | 'Auckalnd' | 1 |
| 'St Heliers' | 1 | '1 Auckland' | 1 | 'Glen Eden' | 1 |  |  |

## I fixed the dirty city data by setting all city values to “Auckland” and extracting a suburb with the help of regular expressions (see the script aklcitycleaning.py).

suburb\_re = re.compile(r'^([a-zA-Z ]\*)(,|$)')

auck\_re = re.compile(r'^([Auckland])\*$')

## If there were a suburb in the city field, I saved it as a value to the key “suburb” in the JSON data. I hardcoded all the city keys with the value “Auckland” deliberately because I had picked the Auckland metro area so I was confident that all the city values should be “Auckland”.

A similar but less prevalent problem existed in the street field. After standardizing the street types, I discovered a small number of streets contained the house number.

"address": {

"city": "Auckland",

"street": "161 Halsey Street",

"housename": "Viaduct Events Centre",

"postcode": "1010",

"housenumber": "161"

}

I decided to fix the street by removing the house number from the street but there was no need to extract the house number and create a new key-value pair like what I did for city and suburb because the value of ‘housenumber’ already existed. Here is the regular expression I used with a re.sub() command to remove the house number from the street:

housenumber\_re = re.compile(r'(^[0-9 ]+)')

## Missing values

City is missing from address data that has at least a street

Realizing that many city values were actually suburbs made me wonder how many city fields were missing data. I wrote a query to uncover how many address documents that had at least a street were missing a city:

db.auckland.aggregate ( [{'$match': {'address': {'$exists': 1}}}, {'$match': {'address.street': {'$exists': 1}}}, {'$match': {'address.city': {'$exists': False}}}, {'$group': {'\_id': '$address.city','count': {'$sum': 1}}}] )

The result showed there were 801 address documents missing a city.

{u'ok': 1.0, u'result': [{u'\_id': None, u'count': 801}]}

The way I cleaned the suburb data also allowed me to take care of adding a missing city to the address document since I hardcoded every address document with a key-value pair of {‘city’: ‘Auckland’}. Therefore, I filled in the 801 address documents missing a city with “Auckland”.

## Overview of OSM Auckland Data

This section provides statistics about the Auckland dataset and the MongoDB queries used to retrieve these statistics.

## Data files

OSM Data File Size Description

auckland.osm 114.6 MB (original Auckland dataset from OpenStreetMap)

osm-auckland.json 159.2 MB (cleansed dataset converted to JSON for MongoDB)

Data Wrangling scripts Description

aklstreetaudit.py Audits and standardizes street types and names

aklcitycleaning.py Audits and standardizes city data and extracts suburb

akldata.py Converts XML to JSON and performs final cleaning

aklMongoAddressQueries.py Exploratory data analysis in MongoDB for cleaning

aklMongoExploreDataset.py Queries for analyzing data after importing clean dataset in MongoDB

## 

## Descriptive statistics on Auckland Dataset

Exploratory analysis in Mongo shell

Number of documents

> db.auckland.count()

Output: 586017

Number of nodes

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

Output: 515431

Number of ways

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

Output: 70586

Exploratory analysis in using Pymongo-based queries

Number of unique users

db.auckland.aggregate ([{'$group': {'\_id': '$created.user'}},

{'$group': {'\_id': 'Users', 'count': {'$sum': 1}}}])

Output: 396

Top 3 contributors to the Auckland dataset

db.auckland.aggregate ( [{'$group': {'\_id': '$created.user', 'count': {'$sum': 1}}},{'$sort': {'count': -1}},{'$limit': 3}] )

Output: {u'ok': 1.0, u'result': [{u'\_id': u'Rudy355', u'count': 187094},

{u'\_id': u'Robert Ancell', u'count': 159313},

{u'\_id': u'myfanwy', u'count': 124607}]}

Amenities occurring more than 100 times

db.auckland.aggregate([{'$match': {'amenity': {'$exists': 1}}}, {'$group': {'\_id': '$amenity', 'count': {'$sum': 1}}},

{'$match': {'count': {'$gte': 100}}}, {'$sort': {'count': -1}}])

Output: u'result': [{u'\_id': u'parking', u'count': 1362},

{u'\_id': u'post\_box', u'count': 390},

{u'\_id': u'telephone', u'count': 333},

{u'\_id': u'school', u'count': 270},

{u'\_id': u'place\_of\_worship', u'count': 261}, ...

(Output was condensed for space)

## 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?

# 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.