Final Project for Data Wrangling with MongoDB

Contents

1. Project Summary

This is the final project from udacity's online course "Data Wrangling with MongoDB". The task is to choose any area of the world from https://www.openstreetmap.org and use data munging techniques, such as assessing the quality of the data for validity, accuracy, completeness, consistency and uniformity, to clean the OpenStreetMap data.

Map Area : Great New York City Area

Open Street Map URL:

https://www.openstreetmap.org/export#map=10/40.7202/-73.8638

Data Source (OSM XML):

https://mapzen.com/data/metro-extracts/metro/new-york_new-york/

File Sizes:

ny.osm: 2.13GB (original OSM XML file)

ny.json: 2.73GB (after converting original OSM XML file to json file)

References:

OSM tag keys: http://wiki.openstreetmap.org/wiki/Map_Features

US town names (US.zip): http://download.geonames.org/export/dump

Prefix of postcodes: https://en.wikipedia.org/wiki/List_of_ZIP_code_prefixes

2. Data Cleaning

1. overview of the dataset

To get an idea of the size of the dataset, elem_count.py was used to count the element, attribute and tagkeys present in this dataset. With the result showing below. (Only node, way and the corresponding subelements will be processed later on to mongodb)

Element Count:
{'bounds': 1,
'member': 98515,
'nd': 11821306,
'node': 9125861,
'osm': 1,
'relation': 8081,
'tag': 8630865,
'way': 1503605}

Attrib Count:
{'node_changeset': 9125861,
'node_id': 9125861,
'node_lat': 9125861,
'node_lon': 9125861,
'node_timestamp': 9125861,
'node_uid': 9125861,
'node_user': 9125861,
'node_version': 9125861,
'tag_k': 8630865,
'tag_v': 8630865,
'way_changeset': 1503605,

Total Number of Distinct Tagkeys: 1469

Tagkey Types Count:
{ 'dots': 24991,
'lower': 3392866,
'others': 27,
'problemchars': 7,
'single_colon': 5109466,
'UpperNums': 33138,
'UpperNums_colon': 70370}

Tagkeys Count
(stored in data/tagkeys_stats.txt):

'way_id': 1503605, 'way_timestamp': 1503605, 'way_uid': 1503605, 'way_user': 1503605, 'way_version': 1503605}	{'building': 1197472, 'height': 1102643, 'nycdoitt:bin': 1079114, 'addr:street': 927273, ...}

2. problematic tag keys

From the result above in point 1, every tag key was counted. It was found that many of tag keys appear only once. In order to pick up resonable tag keys for processing, functions in fix_tagkeys.py were used to fix tagkeys.

First, a distribition of the count of the tag keys were calculated (stored in "distrib" variable)

'total num of distinct keys = 1469',

'total num of keys = 8630865',

'num of keys count more than 86308 times (0.01*total_count) = 16',

'num of keys count more than 8630 times (0.001*total_count) = 51',

'num of keys count more than 863 times (0.0001*total_count) = 148',

'num of keys count more than 86 times (0.00001*total_count) = 322',

'num of keys count more than 8 times (0.000001*total_count) = 593'

Based on the result, only keys appear more than 0.00001*total_count times (86 times) will be processed (stored inside variable "keys_v1"). The other keys are so rare, or they might have the meaning with one of the keys appear more often. For example, "alt_name_1" which appear 4 times most likely to have the same meaning as "alt_name" (appear 1442 times). Also, something like "to" (appear 57 times) is ambiguous and something like "name:wa" (appear 1 time) is so rare.

Second, for the 322 tag keys that appear more than 86 times (stored inside variable "keys_v1"), they are compared with common standard keys from http://wiki.openstreetmap.org/wiki/Map_Features (the standard keys on this web page were crawled using "BeautifulSoup" lib and stored inside "official_keySet" variable). Of the 322 tag

keys, 102 of them exists as standard keys and will be processed without change, the other 220 keys (stored inside "key_need_check" variable) will need further fix.

Third, for the other 220 keys need to check, their types were investigated (if they contain colon, if they contain specific characters, etc), the result was stored in "prob_key_types" variable. Based on the findings, for keys only contain lower case characters, if they appear more than 0.0001*total_count times (863 times), they are saved. For keys contain colon, they are saved if the first part before the colon was a standard key (except the meaning of the second part is ambigous), or if the first part appear more than 0.0001*total_count times (863 times) on average. Some of the keys are problematic, like "cityracks.housenum", in this case, the dot was changed to colon to keep consistency. of the 220 keys, the saved keys were stored in "fixed_key" variable. (The combined fix from this paragraph and last paragraph were stored in "keys_v2" variable)

Fourth, it was found that something like "building" and "building:levels" both exist. Such keys exist both as itself and as the first part in a colon-containing key. This will bring trouble to process into a dictionary because of the key confilict (cannot have dict[building] = value and dict[building] = {"levels":value} both exist). Therefore, for such keys, they are changed to "keys:keys" format (for example, "building" changed to "building:building")

After all the four steps fix, the final keys and the counts were stored in "final_keys" variable. old key to new key mapping was stored in "key_map" variable, it could also be found in data/key_map.txt.

3. put to mongodb

After initial fixation of the tag keys, before further cleaning on other fields, the dataset was loaded into mongodb (Only Node, Way and respective sub-elements were loaded into database) using put_to_db.py. The following structure was used to import the dataset.

```
Node Elem:
{u'types': u'node',
 u'created': {u'changeset': unicode,
              u'timestamp': datetime.datetime,
              u'uid': unicode,
```

```
Way Elem:
{u'types': u'way',
 u'created': {u'changeset': unicode,
              u'timestamp': datetime.datetime,
              u'uid': unicode,
```

```
                u'user': unicode,                                   u'user': unicode,
                u'version': unicode},                               u'version': unicode},
                u'id': unicode,                                     u'id': unicode,
                u'pos': [float(lon), float(lat)] }                  u'nd': [unicode, unicode]}
```

"tag" were added to the corresponding "node" and "way" element with the attribute k/v as key/value pairs :

```
tag keys do not have colon: Element[k] = v

tag keys having one colon: Element[k_first_section] = {k_second_section:v}

tag keys having two colon: Element[k_first_section] = {k_second_section:{third_section:v}}

example:

<tag k="addr:street:name" v="Lincoln"/>

<tag k="amenity" v="pharmacy"/>

<tag k="addr:housenumber" v="5158"/>

should be turned into:

{...

"addr": {"housenumber": 5158,

            "street":{"name":"Lincoln"}},

"amenity": "pharmacy",

...}
```

4. problematic addr:state

fix_state.py was used to audit the "addr:state" field and fix the state values. Using fix_state.py, distinct values of "addr.state" in the database were obtained (stored in "dist_state" variable). It was found that the state names exist in several different forms, like "New York", "NJ - New Jersey",

"ct", etc. To make the values consistent, these forms were all converted to the abbreviation of the corresponding state with upper letters, ie, "NY", "NJ" and "CT".

Also, unexpected state names like "ON", "BY", "TX", "CA", "10009" and so on were found. The corresponding instances were extracted from the database and turns out that the state names were all mistake inputs caused by user. Therefore, they were updated to the right names ("NY", "NJ", or "CT").

The mapping of the old state name to new state name was stored in "state_map" variable.

5. problematic addr:city

fix_city.py was used to audit the "addr:city" field and fix the city values. Upon getting the distinct city names (stored in "dist_city"), it was found that some of the city names contain state or postcode information. like "Merrick, New York", 'New Brunswick, NJ 08901', 'Fresh Meadows NY' and so on. In order to fix them, these values were updated to the correct city names, and the state or postcode info were also moved the right field ("addr:state" and "addr:postcode").

The city names were also compared to the standard town database (could be found in folder "data/US

_town.txt". This txt file was downloaded from http://download.geonames.org/export/dump/ as tab-delimited text, a description of the database could be found on the website as well), the standard town names were processed into a variable "ny_town" as a list. 45 of the 412 city names could not be found from "ny_town". some of them are due to typos, like "Brookklyn" (should be "Brooklyn"), some are not in the right format, like "Hasbrouck Hts" (should be "Hasbrouck Heights"), some are valid names but not in "ny_town" that was processed, like "Bronx" (in this work, the value for the "feature class" field processed is "P", but "Bronx" is in "A" catagory, see description of the database for detail), others are ambigous, like "M", "2", etc.The problematic city names were either updated, kept unchanged, or deleted. Final mapping of original city name to the updated city name could be found in variable "city_map"

6. problematic addr:street

fix_street.py was used to audit the "addr:street" field and fix the street values.There are 9452 distinct street in this dataset. (stored in "dist_street" variable).

First, street names were converted to a uniform format using "street_name.strip().title()".

Second, it was found that some street names contain unusual characters, like "#", ",", "\" and so on. Some of the characters were used to indicate alternative names, like "US 1 (Brunswick Pike)"; Some of them were used for housenumbers, like 'H Highway 34 #120'; Some are ambiguous, like '37th Avenue, 14th Street, 21st Street'; Some of them were valid, but different formats exists, like "George's Road" and "Georges Road", they are the same road but with different formats. In order to fix, all special characters were removed to keep unification ("George's Road" convert to "Georges Road", "U.S. 1" convert to "Us 1", 'NJ-36' convert to "Nj 36", '102 St.' convert to "102 St", etc). Redundant info (like alternative names, housenumbers, etc) were removed. And if the street names were ambiguous, they were deleted.

Third, it was found that in some cases, the street name suffixs were abbreviated (like "10th Ave"); in other cases, full suffixs were used (like '10th Avenue'); To fix the problem: for the formal case where the suffixs were abbreviated, they were converted to the full suffixs ('Harbor Dr' to "Harbor Drive", "10th Ave" to "10Th Avenue", etc).

Fourth, for the street names do not end with an "expected suffixs" ("Parkway", "Drive", "Expressway", etc), there are several cases:

In one case, the street names contain the "expected suffix" in the middle instead of at the end and redundant info exist (like 'West 80th Street NYC 10024'). To fix, the redundant info were removed ('West 80th Street NYC 10024' to "West 80Th Street");

In second case, the street names were valid and they are highways, like 'NJ 35', 'NJ Route 35', 'South New Jersey 17Th South', 'State Route 36', 'US 1', 'US Highway 1 North', 'US Highway 1', etc. There are multiple formats of the highways. To make the format consistent, direction info were removed (remove "North", "South", etc), and only "State Route", "Route" and "Us" were used before the numbers (the above sample highway names were converted to 'State Route 35', 'State Route 35', "State Route 17", 'State Route 36', 'Us 1', 'Us 1' and 'Us 1' respectively. See "street_map" and code for detail);

In third case, street names are valid, like "Avenue Of Puerto Rico", they are saved without change;

In fourth case, street names are not valid and they are deleted, like "Washington Square Village", 'ROAD 1', etc.

Fifty, some street names contain housenumber info at the front, like "40 W 94Th Street", '505Th 8Th Avenue'. In this case, the housenumber info were removed ("40 W 94Th Street" to "W 94Th Street"; '505Th 8Th Avenue' to "8Th Avenue").

Sixth, some street names do not have the right number format, like '7 Avenue', '102 Street', 'Third Avenue', 'Fifth Avenue', etc. They were converted to '7Th Avenue', '102Th Street', '3Rd Avenue' and '5Th Avenue' respectively.

Seventh, it was found that some street names start with directions ("East", "West", etc). For street names starting with directions, some of them have full direction, like 'East 73rd Street', some of them have abbreviated directions, like 'E 73rd Street'. To fix, the abbreviated directions were converted to full word ('E 73rd Street' to "East 73Rd Street").

The mapping of the original street names to the updated street names were stored in variable "street_map"

7. problematic addr:postcode

fix_postcode.py was used to audit the "addr:postcode" field and fix the postcode values.

It was expected that postcode should be five digit numbers. However, it was found that nine digit postcode exist ('11201-2483'), and postcode with state info exist ('NY 10533'). To fix, the nine digit postcodes were convert to five digit postcodes ('11201-2483' to '11201') and for postcodes with state info, the state info was deleted ('NY 10533' to '10533').

Other unexpected postcodes including something like '100014' (should be "10014"), which is due to typos; 'New York, NY 10065' (should be "10065") which contain redundant info; '40299', which is not a valid postcode in NJ, NY or CT (https://en.wikipedia.org/wiki/List_of_ZIP_code_prefixes); "(718) 778-0140", which is abviously a phone number instead of postcode. Such problematic postcodes were all corrected (in the case of invalid postcode like "40299", they are deleted).

The mapping of old postcodes to new postcodes were stored in "postcode_map" variable.

8. problematic addr:housenumber

fix_housenumber.py was used to audit the "addr:housenumber" field and fix the housenumber values.

First, it was found that in some cases, the "addr.housenumber" field has a value, but the "addr.street" field does not have values. Such housenumbers exist without any street names. They were considered invalid and were deleted (except in cases where the housenumber include street names, like '359 van Brunt'. In such cases, the correct housenumber ("359") were extracted and the street names ('Van Brunt') were put to "addr.street" field, mapping is stored in variable "housenumber_map_1").

Second, it was found that in the field of "addr.housenumber", values like '502 9th Avenue', '30 Vesey St' and so on exists. Such values were a mix of housenumber and street names. To fix such values, housenumbers were kept and street names were deleted. The mapping is stored in variable "housenumber_map_2".

9. problematic amenity

`fix_amenity.py` was used to audit the "amenity" field and fix the amenity values.

153 distinct amenity values exist (stored in "dist_amenity" variable). After checking all the values, several problems were found: First, some values have the same meaning, like 'waste_disposal' and "waste_basket" both exist and they have the same meaning. To fix, 'waste_disposal' was converted to "waste_basket", since the latter one appear more times. Second, some values were misunderstanding, like 'user_defined', 'school;place_of_worship', etc. they were either deleted (formal case), or updated (latter case,'school;place_of_worship' convert to 'school' according to detailed descriptions in this piece of data). Third, some of the values do not have the right format or contain typos, like "Liquor_Store", 'Family health clinic', 'pakring', etc. They were converted to the right format ("liquor_store", 'family_health_clinic' and 'parking' respectively).The mapping of the change was stored in "amenity_map" variable.

10. output to json

cmd command `mongoexport -d osm -c NY_osm -o ny.json` was used to export the mongodb dataset to json file.

3. Data Overview

Number of documents

```
> db.NY_osm.find().count()
```

```
10629466
```

Number of nodes

```
> db.NY_osm.find({"types":"node"}).count()

9125861
```

Number of ways

```
> db.NY_osm.find({"types":"way"}).count()

1503605
```

Nodes do not have tags:

```
> cursor = collection.find({"types":"node"})

> simple_node = 0

> for doc in cursor:

    if len(doc) == 5:

        simple_node += 1

> simple_node

8811816
```

Ways do not have tags:

```
> cursor = collection.find({"types":"way"})

> simple_way = 0

> for doc in cursor:

    if len(doc) == 5:

        simple_node += 1

> simple_way
```

```
6703
```

Number of unique users

```
> db.NY_osm.distinct("created.user").length

3591
```

Top 5 contributing user

```
> db.NY_osm.aggregate([{"$group":{"_id":"$created.user",

                                  "count":{"$sum":1}}},

                     {"$sort":{"count":-1}},

                     {"$limit":5}])

{ "_id" : "Rub21_nycbuildings", "count" : 4891073 }

{ "_id" : "ingalls_nycbuildings", "count" : 935873 }

{ "_id" : "woodpeck_fixbot", "count" : 640476 }

{ "_id" : "ediyes_nycbuildings", "count" : 272014 }

{ "_id" : "lxbarth_nycbuildings", "count" : 234580 }
```

Number of users contributing only once

```
> db.NY_osm.aggregate([{"$group":{"_id":"$created.user",

                                  "count":{"$sum":1}}},

                     {"$group":{"_id":"$count",

                                "num_users":{"$sum":1}}},

                     {"$sort":{"_id":1}},

                      {"$limit":1}])

{ "_id" : 1, "num_users" : 920 }
```

Top 5 amenities:

```
> db.NY_osm.aggregate([{"$match":{"amenity":{"$exists":1}}},
                                {"$group":{"_id":"$amenity",
                                           "count":{"$sum":1}}},
                                {"$sort":{"count":-1}},
                                {"$limit":5}])
{ "_id" : "parking", "count" : 6774 }
{ "_id" : "bicycle_parking", "count" : 4868 }
{ "_id" : "school", "count" : 4693 }
{ "_id" : "place_of_worship", "count" : 4641 }
{ "_id" : "restaurant", "count" : 3154 }
```

Top 5 cuisines:

```
> db.NY_osm.aggregate([{"$match":{"amenity":"restaurant",
                                   "cuisine":{"$exists":1}}},
                                {"$group":{"_id":"$cuisine",
                                           "count":{"$sum":1}}},
                               {"$sort":{"count":-1}},
                                {"$limit":5}])
{ "_id" : "italian", "count" : 221 }
{ "_id" : "american", "count" : 173 }
{ "_id" : "pizza", "count" : 163 }
{ "_id" : "mexican", "count" : 120 }
```

```
{ "_id" : "chinese", "count" : 98 }
```

Top 5 postcodes:

```
> db.NY_osm.aggregate([{"$match":{"addr.postcode":{"$exists":1}}},
                                    {"$group":{"_id":"$addr.postcode",
                                                "count":{"$sum":1}}},
                                    {"$sort":{"count":-1}},
                                    {"$limit":5}])
{ "_id" : "10314", "count" : 23066 }
{ "_id" : "11234", "count" : 20139 }
{ "_id" : "10312", "count" : 17841 }
{ "_id" : "10306", "count" : 16186 }
{ "_id" : "11236", "count" : 15225 }
```

Top 5 street names:

```
> db.NY_osm.aggregate([{"$match":{"addr.street":{"$exists":1}}},
                                    {"$group":{"_id":"$addr.street",
                                                "count":{"$sum":1}}},
                                    {"$sort":{"count":-1}},
                                     {"$limit":5}])
{ "_id" : "Broadway", "count" : 3554 }
{ "_id" : "3Rd Avenue", "count" : 2485 }
{ "_id" : "5Th Avenue", "count" : 2474 }
{ "_id" : "79Th Street", "count" : 2414 }
```

```
{ "_id" : "78Th Street", "count" : 2340 }
```

4.Additional Ideas

Since majority of "node" do not have tags (8811816/9125861=95.6%), which means the only useful info (if the application is mostly interested in "places" rather than who created the dataset) in such "nodes" is "pos" coordinates. Thus, it would be possible to incoorperate the "node" directly into "way", removing the "nd" field in way and removing all "node".

Since the data is only from one source (OpenStreetMap) and it is user defined, it might be necessary to validate the dataset and fill in the missing data (street,postcode, etc) using other tools, like Google Maps APIs.