Data Wrangling With MongoDB

Project 2

Map Area - Hamilton, Ontario CA

Summary of XML Tags -

{'bounds': 1,
 'member': 15327,
 'nd': 714556,
 'node': 509854,
 'osm': 1,
 'relation': 3386,
 'tag': 385279,
 'way': 87552}

Problems encountered in the map

Just as in the exercise, I found some instances of abbreviated street names as well as incorrectly formatted postal codes.

The postcode cleaning part I found slightly more interesting, very likely because it's different from the street address clean up used in the exercises. I looked up wikipedia on Canadian Postal Codes and learned they follow a pattern of A1A 1A1 (where A can be any alphaphabet, and 1, any number and 6 characters in total). I ran a regex pattern check to see if there are any entries that do not match this pattern and I found two instances where this was the case. Regex used:

postcode = re.compile(r'[a-z][0-9][a-z]\s[0-9][a-z][0-9]', re.IGNORECASE)

The first group of cases where ones where no space was entered between the two groups of codes (e.g A1A1A1 as opposed to A1A 1A1)

The second group includes cases where an additional ON was prefixed to the postal codes to denote that the postal code is an Ontario postal code.

The plan of action was simply to standardize this by having them all in the official format of "A1A 1A1" with the space in

between.

The regex pattern for abbreviated street names with a period used in the exercises would not work in this case. Below is a pprint of a dict showing all street types. Only "street" as "str." is abbreviated. There is both "Road" and "Rd" on the list.
The regex pattern used to search for this is:

streettype = streettype = re.compile(r'\s(\w+\W?)$', re.I)

The plan for auditing the street address is to replace the abbreviated Road and street names with the full "Road" and "Street" respectively.

There are also instances of E. as an abbreviation for east, one address was checked to verify this was indeed used as an alternative for East. This will also be changed to match convention. Regex pattern:

alt = re.compile(r'\b(\w*)\.$')

was then used to isolate instances of street types with period abbreviations so these can be amended.

One interesting thing I noticed is that "tag.attrib['v']" can not only be called, but can also be assigned to. What I mean by that is that I noticed I could assign cleaned up values to the tag.attrib['v'] value instead of keeping the cleaned up value in variable that is then called when converting to JSON. This can come in handy especially when for example, one had to clean up many different fields(e.g address, postcode, phone number, name e.t.c) rather than creating variables for each and everyone of these cleaned up values to store to before we're ready to asign to a dictionary, we can simple assign the cleaned up value to the same variable and then output to a python dictionary at a later time whenever ready. For example

tag.attrib['v'] = cleanpostcode(tag.attrib['v'])

#Building types and count
>>> db.projects.aggregate([{"$match": {"buildingtype": {"$exists": 1}}}, {"$group": {"_id": "$buildingtype", "count": {"$sum": 1}}}])
{u'ok': 1.0, u'result': [{u'count': 2, u'_id': u'shed'}, {u'count': 278, u'_id': u'terrace'}, {u'count': 1, u'_id': u'construction'}, {u'count': 2, u'_id': u'hangar'}, {u'count': 136, u'_id': u'residential'}, {u'count': 4, u'_id':u'manufacture'}, {u'count': 7, u'_id': u'hotel'}, {u'count': 50, u'_id': u'university'}, {u'count': 122, u'_id': u'school'}, {u'count': 1, u'_id': u'train_station'}, {u'count': 42, u'_id': u'roof'}, {u'count': 268, u'_id': u'industrial'}, {u'count': 143, u'_id': u'church'}, {u'count': 4744, u'_id': u'yes'}, {u'count': 12, u'_id': u'public'}, {u'count': 1, u'_id': u'chapel'}, {u'count': 123, u'_id': u'garage'}, {u'count': 2005, u'_id': u'house'}, {u'count': 440, u'_id': u'apartments'}, {u'count': 1, u'_id': u'greenhouse'}, {u'count': 429, u'_id': u'retail'}, {u'count': 2, u'_id': u'block'}, {u'count': 14, u'_id': u'commercial'}, {u'count': 16, u'_id': u'office'}, {u'count': 1, u'_id': u'yes;school'}, {u'count': 1, u'_id': u'yes;apartments'}, {u'count': 1, u'_id': u'mosque'}]}

It's clear that the building types can also use a bit of cleaning as some building types appear more than once but spelled differently.

MongoDB Queries

>>> from pymongo import MongoClient
>>> client = MongoClient('localhost:27017')
>>> db = client.wrangling

File Sizes

08/03/2015 11:49   87.2MB hamilton_canada.osm

13/03/2015 19:17   88.2MB hamilton_canada.osm.json

05/04/2015 03:02   161MB   hamilton_canada.osm_meta.json

#Number of documents
>>> db.projects.find().count()
431256

#Database Statistics
>>> db.command("dbstats")
{u'avgObjSize': 204.1249462172168,
 u'collections': 4,
 u'dataFileVersion': {u'major': 4, u'minor': 22},
 u'dataSize': 335889232.0,
 u'db': u'wrangling',
 u'extentFreeList': {u'num': 0, u'totalSize': 0},
 u'fileSize': 1006632960.0,
 u'indexSize': 53413808.0,
 u'indexes': 2,
 u'nsSizeMB': 16,
 u'numExtents': 27,
 u'objects': 1645508,
 u'ok': 1.0,
 u'storageSize': 418062336.0}

```
#Nbr of Node
>>> db.projects.find({"type": "node"}).count()
372768

Nbr of Way
>>> db.projects.find({"type": "way"}).count()
58488

#Nbr of Unique users
>>> db.projects.distinct("created.user").length
220

#Number 1 contributing user
>>> db.projects.aggregate([{"$group": {"_id": "$created.user", "count": {"$sum": 1}}}, {"$sort": {"count":
-1}}, {"$limit": 1}])
{u'ok': 1.0, u'result': [{u'count': 225822, u'_id': u'andrewpmk'}]}
```

Additional queries

I am actually really glad I for the opportunity to re-visit this project, ideas for additional queries got me thinking maybe creating a kind of meta-data about the data itself specifically the entries by the users. Included now in this project is a mapsmeta.py file containing the code used to extract the info for a new database. It containing in each document, the username, time entry was submitted, node type as well as Lat & Lon where applicable. This new data source gives us an opportunity to really look into the contributors and their submission patterns. Below are a number of queries run on this new database.#To import json to mongodb

```
mongoimport -d wrangling -c mapsmeta --file hamilton_canada.osm_meta.json
```

```
db.mapsmeta.aggregate([{"$group": {"_id": "$username", "count": {"$sum": 1}}},
{"$sort": {"count": -1}},
{"$limit": 5}])

[{u'_id': u'andrewpmk', u'count': 759271},
        {u'_id': u'Kevo', u'count': 111931},
        {u'_id': u'HorseHat', u'count': 61979},
        {u'_id': u'rob5251', u'count': 36156},
        {u'_id': u'a_white', u'count': 33532}}]
```

From this we can see from this that user "andrewpmk" contributes a lot more than other users. Another thing we could do is to plot some sort of map showing the latitudes and longitudes and see what area(s) each contributor is contributing the most or least for, as well as what kind of data is being contributed for, what kind of amenities e.t.c While it's not clear why this might be useful now, it's good data to have.

Another possibility is to add the amount of errors corrected for each entry to the database to get an idea of the errors by the users and maybe even possibly what kind of errors.
And finally, having the timestamp in the data gives us an opportunity to explore contributions over time and the frequency of it, also good data to have.

Problems encountered making the Mapsmeta DB

I learned a lot doing this, The model of the json really matters and should be carefully considered before importing into mongodb. At first, I wanted each document to be a name entry so that multiple entries by the same user are grouped under the user but with the timestamps as the lead element after the usernames. However I noticed some timestamps are also identical meaning the entries were submitted at the exact same time, I decided grouping by timestamp wouldn't properly reflect the amount of contributions made by the user.

When I attempted to import into Mongodb the first time, I got an error that the document size exceeded the 16MB limit, I found this confusing at first since I had previously succesfully imported the wrangled data of over 60MB into Mongo. Reading the documentation online made it clear the size referred to an individual document which mean I was attempting to export the entire json file as one single json document, I amended the code for each user entry to be a single document.