Project 3 Data Wrangling

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:

1. Street name end with letter but not its type(From 'Avenue A' to 'Avenue Z')
2. Street name with different case for same type ('AVENUE':"Avenue")
3. “Incorrect” postal codes('NY 10533')

Street name case and ending problem

I find the problems with function “audit_street_type” and “audit”.

There are 23 not expected street types. Some of them are over abbreviated, and some are case problem, such as “St.” or “DRIVE”.

I fixed these problems with “update_name” and “mapping”.

Mapping from “St.” to “Street”, and from “DRIVE” to “Drive”.

Postal Codes problem

I find the problems with function “audit_postcode”. I find that, there are some postcodes with “NY”, as an abbreviation of New York, attached in front of the postcode. In the sample, there are only 4 'NY 10533', but there must be more in the original osm file.

I fixed these problems with “update_postcode”, which strip the “NY” from the “NY 10533”.

Memory management problem

At first i tried to use insert method to fill the collection, but a lot of wired error and failure occurred. Most important is that, the memory consumption rises rapidly. Then under the instruction from couch, I used mongoimport, everything works fine then.

```
# Sort users by count, descending

aggregate_by_user=db.map_collection.aggregate([{"$group":{"_id":"$created.user","count":{"$
sum":1}}},{"$sort":{"count":-1}}])
```

result:

```
In [21]: for i,user in enumerate(aggregate_by_user):
    ...:     if i <10:
    ...:         print user
    ...:
{u'count': 4194995, u'_id': u'Rub21_nycbuildings'}
{u'count': 865056, u'_id': u'ingalls_nycbuildings'}
{u'count': 650227, u'_id': u'woodpeck_fixbot'}
{u'count': 241181, u'_id': u'ediyes_nycbuildings'}
{u'count': 227852, u'_id': u'NJDataUploads'}
{u'count': 207273, u'_id': u'lxbarth_nycbuildings'}
{u'count': 156898, u'_id': u'CoreyFarwell'}
{u'count': 123478, u'_id': u'celosia_nycbuildings'}
{u'count': 117105, u'_id': u'ebrelsford_nycbuildings'}
{u'count': 110352, u'_id': u'smlevine'}
```

This result shows that almost half of the nodes and ways are created by user "Rub21_nycbuildings"

Sort postcode by count, descending

aggregate_by_postcode=db.map_collection.aggregate([{"$group":{"_id":"$address.postcode","count":{"$sum":1}}},{"$sort":{"count":-1}}])

```
In [23]: for i,postcode in enumerate(aggregate_by_postcode):
    ...:     if i <10:
    ...:         print postcode
    ...:
{u'count': 8841902, u'_id': None}
{u'count': 2101, u'_id': u'11234'}
{u'count': 1820, u'_id': u'11101'}
{u'count': 1793, u'_id': u'10314'}
{u'count': 1724, u'_id': u'11203'}
{u'count': 1693, u'_id': u'11211'}
{u'count': 1664, u'_id': u'11235'}
{u'count': 1613, u'_id': u'10461'}
{u'count': 1608, u'_id': u'11375'}
{u'count': 1571, u'_id': u'11206'}
```

This result shows that, almost 98.46% documents do not have a postcode.

2. Data Overview

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

File size

```
In [3]: %run "D:\github\udacity_data_analyse\project3\new_project3.py"
D:\github\udacity_data_analyse\project3\report/new-york_new-york.osm ------ 2123.74 MB
D:\github\udacity_data_analyse\project3\report/new-york_new-york.osm.json ------ 2283.62 MB
```

Number of documents

```
In [12]: db.map_collection.find().count()
Out[12]: 8980000
```

Number of nodes

```
In [13]: db.map_collection.find({"type":"node"}).count()
Out[13]: 8833086
```

Number of ways

```
In [14]: db.map_collection.find({"type":"way"}).count()
Out[14]: 146914
```

Number of unique users

```
In [8]: len(db.map_collection.distinct("created.user"))
Out[8]: 3165
```

Top 1 contributing user

```
In [14]: for user in db.map_collection.aggregate([{"$group":
    ...: {"_id":"$created.user", "count":{"$sum":1}}},
    ...: {"$sort":{"count":-1}}, {"$limit":1}]):
    ...:     print user
    ...:
{u'count': 4194995, u'_id': u'Rub21_nycbuildings'}
```

Number of users appearing only once (having 1 post)

```
In [16]: for a in db.map_collection.aggregate([{"$group":
    ...:     {"_id":"$created.user", "count":{"$sum":1}}},
    ...:     {"$group":{"_id":"$count", "num_users":{"$sum":1}}},
    ...:     {"$sort":{"_id":1}}, {"$limit":1}]):
    ...:     print a
    ...:
{u'num_users': 746, u'_id': 1}
```

3. Additional Ideas

Additional data exploration using MongoDB queries

#Aggregate the collection by created year. It shows that when the project is hot or ignored.

group_by_year=db.map_collection.aggregate([{"$match":{"created.year":{"$exists":1}}},{"$group":{"_id":"$created.year", "count":{"$sum":1}}},{"$sort":{"_id":-1}}])

```
In [18]: for group in group_by_year:
    ...:     print group
    ...:
{u'count': 263516, u'_id': 2016}
{u'count': 536399, u'_id': 2015}
{u'count': 3139400, u'_id': 2014}
{u'count': 3538443, u'_id': 2013}
{u'count': 172771, u'_id': 2012}
{u'count': 95462, u'_id': 2011}
{u'count': 177452, u'_id': 2010}
{u'count': 1027914, u'_id': 2009}
{u'count': 22867, u'_id': 2008}
{u'count': 5776, u'_id': 2007}
```

Other ideas about the datasets

One big problem about the dataset is that, we do not know how much can we believe in it.

I think we need an additional attribute for every tag. We can call it “endorsement”, just like the Linkedin.

It is based on the assumption that, the nodes and ways with more endorsements from users are more reliable.

The implementation may require more resources. But the biggest anticipated problem is that, some potential vicious user may register a lot of accounts, and use them to endorse their own fake nodes or not existing address.

Conclusion

A lot of errors occurred during my implementation. Sampling, auditing, update fields, shape element, query and aggregate. I think I learned a lot in this lesson. But there is still some pity.

I don’t know how to write an appropriate regular expression according to my requirements.

I need more practice in this field.

All update method for the fields should be integrated into the „shape_element” method.

“shape_element” is the key point between the mongodb and xml file.