Bindu Paul

1. Problems Encountered in the Map

After downloading and auditing the open street map data from Lincoln, Nebraska I realized the following:

-Not all addresses are referred to by an expected street (‘102’: {‘NW 1st St #102’}
-Phone numbers are not correct {u'_id': u'001/402/ 797-7700', u'count': 1}
-Questionable “type” appear {u'_id': u'clay_pigeon', u'count': 1}

Incorrect Reference to Street Names

After running the audit() function I found that some of the address values had keys that referred to apartment numbers, for example: “110”: {“Pioneer Woods Drive #110”}. I corrected the keys by manually changing the mapping function and adding the key and corrected value, for example: mapping = {“110”: “Drive”}. Then I ran the update_name() function which corrected the keys. Unfortunately some values did not list any expected street, for example: “1st”: {“NW 1st”}. I decided to change those values to NoneType. I also had some single letter Key, Value pairs such as “K”: {“K”}, “O”: {“O”}, “P”: {“P”}, which I also changed to NoneType.

Incorrect Phone Numbers

Once I managed to get my data into mongodb I started using the aggregate()and make_pipeline() functions to group my data for analysis. I found that only one phone number was duplicated {u'_id': u'402-441-8376', u'count': 2} and many did not follow the ###-###-#### syntax. After isolating all phone numbers to a “results” variable, I created 2 functions: find_numbers(results) and fix_numbers() (See function 1 line 217)

Questionable Type

After checking the type of tags present with make_pipeline() function, I discovered 2 tags that were questionable {u'_id': u'clay_pigeon’} and {u'_id': u'public'}. I removed the document from mongodb as I do not know what they refer to. (See function 2 line 238)

2. Data Overview

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

File sizes

lincoln_nebraska.osm ……. 70.3 MB
lincoln_nebraska.osm.json …. 79.3 MB

#Number of documents

- db.street.find().count()

- 344534

#Number of nodes

- db.street.find({“type”: “node”}).count()

- 309915

#Number of ways

- db.street.find({“type”: “node”}).count()

- 34617

#Number of unique users

- ```
  def make_pipeline():
      pipeline = [{'$group': {"_id": "$created.user",
                              "count": {"$sum": 1}}}]
      x = [doc for doc in db.street.aggregate(pipeline)]
      return len(x)
  ```

- 206

#Top 1 contributing user

- ```
  def make_pipeline():
  pipeline = [{'$group': {"_id": "$created.user",
                          "count": {"$sum": 1}}},
              {'$sort': {'count': -1}},
              {'$limit': 1}]
  return [doc for doc in db.street.aggregate(pipeline)]
  ```

- [{u'_id': u'Your Village Maps', u'count': 148592}]

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

- ```
  def make_pipeline():
      pipeline = [{"$group":{"_id":"$created.user",
                             "count":{"$sum":1}}},
                  {"$group":{"_id":"$count",
                             "num_users":{"$sum":1}}},
                  {"$sort":{"_id":1}},
                  {"$limit":1}]
  ```
  ```

```
    return [doc for doc in db.street.aggregate(pipeline)]
```

- [{u'_id': 1, u'num_users': 40}]

3. Additional Ideas

Cuisine Clean Up

Grouping by cuisine (See function 3 line 246), it can be seen that the categories are not well thought-out. For example, there is a category for international but also Thai, Vietnamese, Japanese, Chinese, and Brazilian that are all technically international. It would make sense if the grouping was based off nationality and associated subgroups for food type, {American: {Burger: Cheeseburger}, {Sandwich: BLT}, {Ice_Cream: Vanilla}}.

One possible way to create this fix would be to create a new dictionary before importing the data to MongoDB and changing the key/value. Just like the "created" dictionary under the "shape_element()" function, I would create a series of "if" statements that would .replace() the string.

Some foreseeable difficulties with implementing the fixes would be that the data has to be sifted manually to create the "if" statements. For example, if string equals Burger then append to American dictionary, if string equals Pad Thai append to Thai dictionary. This is not a problem until you get to larger data sets with thousands of different cuisine types. And since the user is deciding what the correct categories are, this becomes a daunting process. There is also the problem of ambiguity, when a cuisine type says "noodles" is that Asian noodles or Italian noodles (I'd say Asian noodles and Italian Pasta).

The benefit of going through the work is so the data will be very comprehensive and unambiguous. Also, once you have the code written to "sift" through the cuisine types via "if" statements, it can be applied to multiple other cities, as it is likely they will have similar issues.

Additional Exploration on MongoDB

The percentage for the top unique sandwich shop is Subway at 33% (See function 4)

The most common zip code is 68508 with count 130. (See function 5)

Out of 276 schools, 50 are elementary schools, 28 are middle schools, and 30 are high schools. That leaves 168 schools that were unspecified, which is another problem with the dataset. (See function 6)

Conclusion

After review of the Lincoln, Nebraska Street map dataset it can be concluded that it is fairly clean. I assume most of the incorrect data comes from user inputs seeing as how not all phone numbers are 10 digit structures, not all schools are specific enough to state the level of the school, and some categories of cuisines can be combined for better quality. Regardless, it is possible to analyze the data.