**P3: Wrangle OpenStreetMap Data**

**Problems Encountered in the Map**

After downloading the sample .osm data of Raleigh from Map Zen, I came across several issues in the dataset.

**1. Inconsistent House Numbers**  
 Most house numbers contained only numbers, but several problems appeared when encountering alphanumeric house numbers. For a given address, the house number could be in the form 123a, 123A, or 123-A. In addition, sometimes the house number and the street would be combined in the ‘housenumber’ field. Sometimes, only the house number would be provided without a street value. For buildings with multiple house numbers, there is not a consensus on how to represent this, so some use commas or semicolons to separate the numbers while other use a hyphen to represent the range. However, using a hyphen creates extra confusion because some addresses utilize a hyphen to further specify a location such as a store located inside a shopping center. Regarding apartments, some house numbers included “Suite” while in other entries this specification was moved to the address string.

I wrote a function to update the house numbers to remove hyphens from certain values and convert lowercase letters to uppercase (ex: “123-a becomes 123A).

**def** fix\_housenumber(housenumber):  
 *# search housenumbers for one to many digits, hyphen, one to many letters (ignore case)* housenumber\_re = re.compile(**r'^\d+-[a-zA-Z]+$'**)  
 h = housenumber\_re.search(housenumber)  
 **if** h:  
 housenumber = housenumber.replace(**"-"**, **""**)  
 housenumber = housenumber.upper()  
 **return** housenumber

*# Sift out house numbers with hyphens and letters in the housenumbers field*

[{**"$project"**: {**"address.housenumber"**: {**"$substr"**: [**"$address.housenumber"**, 0, -1]}}},  
 {**"$sort"**: {**"address.housenumber"**: 1}},  
 {**"$match"**: {**"address.housenumber"**: {**"$regex"**: **'[-a-zA-Z]'**}}}]

Here is a small sample of the results:

[{u'\_id': ObjectId('5668bae8d5e9469a08c8f9b3'),

u'address': {u'housenumber': u'101 E. Morgan St'}},

{u'\_id': ObjectId('5668bae8d5e9469a08c95818'),

u'address': {u'housenumber': u'Suite #205'}},

{u'\_id': ObjectId('5668baf5d5e9469a08cea644'),

u'address': {u'housenumber': u'5410-8'}},

{u'\_id': ObjectId('5668baf0d5e9469a08cbfc26'),

u'address': {u'housenumber': u'500-506'}}

**2. Extended Postal Codes**

Some postcodes contained a four-digit extension. I removed the extension and combined similar postcodes to mitigate consistencies and to allow for meaningful observations such as the most popular postcode.

**def** fix\_postcode(postcode):  
 *# search postcode for five digits, hyphen, four digits* postcode\_type\_re = re.compile(**r'^\d{5}-\d{4}$'**)  
 n = postcode\_type\_re.search(postcode)  
 **if** n:  
 **return** postcode.split(**"-"**)[0]  
 **else**:  
 **return** postcode

*# Group and sort postcodes by descending count, limit to 1*

[{**"$match"**: {**"address.postcode"**: {**"$exists"**: 1}}},  
 {**"$group"**: {**"\_id"**: **"$address.postcode"**, **"count"**: {**"$sum"**: 1}}},  
 {**"$sort"**: {**"count"**: -1}}, {**“$limit”**: 1}]

The most popular postcode: [{u'\_id': u'27612', u'count': 1726}].

**3. Abbreviated or Uncommon Streets:**

From running the code several times, I was able to locate abbreviated street names and include them in the mapping. In addition, legitimate street names that I did not account for due to their infrequency (“Hills”, “Circle”) were added to the **expected** array. Any truncated forms of the street name, with or without periods, were fixed to reflect the full name (Pl 🡺 Place).

#list of expected street names

expected = [**"Street"**, **"Avenue"**, **"Boulevard"**, **"Drive"**, **"Court"**, **"Place"**, **"Square"**, **"Lane"**, **"Road"**, **"Trail"**, **"Parkway"**, **"Commons"**, **"Plaza"**, **"Center"**, **"Bypass"**, **"Circle"**, **"Way"**, **"54"**, **"55"**, **"Suite"**, **"Hill"**, **"Hills"**, **"Extension"**, **"West"**, **"East"**, **"North"**, **"South"**, **"Fork"**]

**Overview of the Data**

Here are some basic statistics about the data:

|  |  |
| --- | --- |
| File | Size |
| raleigh\_north-carolina.osm | 513.7 MB |
| raleigh\_north-carolina.osm.json | 794.8 MB |

#Number of documents:

> db.getCollection("raleigh\_north-carolina.osm").find().count()

> 2750050

#Number of nodes:

> db.getCollection("raleigh\_north-carolina.osm").find({"type": "node"}).count()

> 2526439

#Number of ways

> db.getCollection("raleigh\_north-carolina.osm").find({"type": "way"}).count()

> 223611

#Number of unique users

> db.getCollection("raleigh\_north-carolina.osm").distinct("created.user").length

>720

#Top Contributing User:

> {**"$group"**:{**"\_id"**:**"$created.user"**, **"count"**:{**"$sum"**:1}}},  
 {**"$sort"**:{**"count"**: -1}}, {**"$limit"**:1}

[{u'\_id': u'jumbanho', u'count': **1989463**}]

*#number of users contributing equal to or less than 10 times*> pipeline = [{**"$group"**: {**"\_id"**: **"$created.user"**, **"count"**: {**"$sum"**:1}}},  
 {**"$group"**: {**"\_id"**: **"$count"**, **"num\_users"**: {**"$sum"**:1}}},  
 {**"$match"**: {**"\_id"**: {**"$lte"**: 10}}},  
 {**"$group"**: {**"\_id"**: None, **"total"**: {**"$sum"**: **"$num\_users"**}}}]

[{u'\_id': None, u'total': **360**}]

**Ideas About the Dataset**

To improve on the quality of the dataset, certain standards should be instilled when appending the dataset with new entries. Several decisions must be made, including whether to write complete street names or abbreviate, house number formatting, and whether postcodes should be specified up to the extra four digits. Perhaps two fields within addr:postcode could be created, with one containing the 5 digit code, and the other showing an optional extra 4-digit specification.

Automated edits from bots are more preferable than human input simply because humans are much more prone to mistakes in data entry. From the Raleigh dataset, there were several instances where postcodes were either 4 or 6 digits, or letters such as ‘NC’. In extreme cases, perhaps human contributors are not contributing to OSM at all and are instead providing faulty and ridiculous entries. Bots are also beneficial in this regard, an example being “General Dreedle”, a Bot who runs revert scripts to modify edits made by established “vandals” in OSM. A more generalized Bot that could edit datasets with corrections and updates would vastly improve datasets; country-specific Bots that perform these tasks do exist such as “Czechreg (Czech administrative boundaries)”.

Several issues arise with the implementation of bots, especially in the realm of automated edits. A bot can only perform what it is commanded to do, and the existence of multi-faceted problems in a dataset can nullify the potential usefulness a bot could provide. A poorly programmed bot could quickly scramble data to an extremely large scale, or add bunches of nonsensical data in an “update,” compounding the original issue many times over. As a result, bots should only be trusted with rudimentary edits such as whitespace errors. It is also important to note that bots would probably not be able to mitigate the problems in OSM maps such as overlapping or self-crossing ways since these issues do not have an established correct way of fixing and could be multifaceted (misplaced or misplaced nodes). Currently, bots are only trusted to perform simple edits in OSM; much more development and precision is required for bots to handle more complex problems.

**More Aggregation Pipeline Queries**

*#number of cafes*> pipeline = [{**"$match"**:{**"amenity"**:{**"$exists"**:1}, **"amenity"**:**"cafe"**}},  
 {**"$group"**:{**"\_id"**: **None**, **"count"**:{**"$sum"**:1}}}]

[{u'\_id': None, u'count': **94**}]

*#number of chinese restaurants*> pipeline = [{**"$match"**:{**"cuisine"**: **"chinese"**}},  
 {**"$group"**:{**"\_id"**: None, **"count"**:{**"$sum"**: 1}}}]

[{u'\_id': None, u'count': **24**}]

*# list of amenities (Raleigh is a great place to park. )*> pipeline = [{**"$match"**:{**"amenity"**:{**"$exists"**:1}}},  
 {**"$group"**:{**"\_id"**:**"$amenity"**, **"count"**:{**"$sum"**:1}}},  
 {**"$sort"**:{**"count"**: -1}}, {**“$limit”: 3**}]

[{u'\_id': u'parking', u'count': 1941},

{u'\_id': u'place\_of\_worship', u'count': 543},

{u'\_id': u'bicycle\_parking', u'count': 524}]

**Conclusion**

From observing the data in Raleigh, it is apparent that these OSM datasets could use significant cleaning such as homogenizing street names and postcodes. In addition, the use of bots could reduce the amount of data cleaning needed since mistakes would be drastically reduced. The code took a lengthy time to run (~5 minutes) due to conversion to JSON, so it would be interesting to see what improvements I could make to my hardware or the code to reduce processing time. Using a GUI such as Robomongo would provide for an even better experience for data visualization, presentation, and even editing.