P3 - Data Wrangling with MongoDB

Open Street Map (OSM) is a huge geographical information database, constantly updated by a large community of contributors all over the world. The idea of having constant input from developers always makes the data up to date, but at the same time, it is always prone to be not error free. In this project, I am going to explore for common errors and use the data wrangling techniques to fix some of the common errors programmatically. Initially our data is in XML format and then it is converted to Json data and then populated to MongoDB database.

Chosen Area

In this project, I used the open street map of Santa Cruz, a small town located in the state of California, USA. I chose it, because it is my first city I dwelled in the States.

1. Problems encountered

In this section, I will attempt to audit and clean all the common errors that come from human input. As mentioned earlier, OSM being open makes it error prone as different developers have different means of giving names for different amenities. In this section, I am going to address the common errors in street names and zip-codes and then try to fix them in the commonly agreed names and formats. I will try to address all abbreviated street names and write them in their full text format. I will also attempt to get postal codes different from those 5 digit number in Santa Cruz.

i. Zip-Code

In the United States, Zip code is composed of 5 digit numbers and each area has its unique zip-code starting identifier. For instance the Zip code for Santa Cruz starts with 95. Having this in mind, I will attempt to identify zip codes other than those and try to fix them. For this particular dataset, we can broadly categorize the data errors in to 3 main formats:

- Post codes other than 5 digit numbers (1982, '95062-4205'
- Post codes with characters other than integers: most likely in string format
- With text at the front (CA 95065)

To address the above related errors: a function that handles all these error types is defined and gets all these errors as shown below.

```
[17]: pprint.pprint(dict(san_zipcode))
```

```
{'19': set(['1982']),
 '95': set(['95002',
            '95003',
            '95010',
            '95018',
            '95041',
            '95060',
            '95062',
            '95062-4205',
            '95064',
            '95065',
            '95065-1711',
            '95066',
            '95066-4024',
            '95066-5121',
            '95073']),
 'CA': set(['CA 95065'])}
```

As we can see from the above list of zip code errors, most of the list looks like proper Santa Cruz zip codes. But as mentioned earlier, it is most likely that they were in a string type data format instead of integer type, so in this case, I will not alter or drop them. For the other types of errors I used the function (update_name) that strips the prefix text (CA in our case) and trims the zip code extension 4 digits from the end as shown below.

```
for zip, ways in san_zipcode.iteritems():
    for name in ways:
        better_name = update_name(name)
        print name, "-->", better_name
```

```
1982 --> 1982
CA 95065 --> 95065
95073 --> 95073
95065-1711 --> 95065
95065 --> 95065
95064 --> 95064
95041 --> 95041
95066 --> 95066
95060 --> 95060
95062 --> 95062
95018 --> 95018
95066-5121 --> 95066
95062-4205 --> 95062
95010 --> 95010
95003 --> 95003
95002 --> 95002
95066-4024 --> 95066
```

As we can see from the above figure, there still exits one zip code error on first line of errors table, which is not yet altered. But that is completely wrongly input (not even in 5 digit format) and left as it is, since it is very difficult to guess what that exact value is.

ii. Street names

Most of the street names are written in full text with some abbreviations as displayed down.

```
# updating the street names in the expected format
for street_type, ways in san_street_types.iteritems():
    for name in ways:
        updated_street_name = update_street_name(name, street_type_mapping, street_type_re)
        print name, "-->", updated_street_name
```

```
pprint.pprint(dict(san_street_types))  # pri
{'245': set(['245']),
 'Ave': set(['220 Sylvania Ave', '41st Ave',
 'Rd': set(['Mount Hermon Rd']),
 'St': set(['225 Rooney St'])}
```

To get the above corrected list of street names, I need to have the list of expected list of street names and then a built in function called regex that maps the abbreviated street names to the list of possible expected street names.

2. Data overview

After converting the XML data in to JSON types, the mongodb database is populated in order for data query and visualization to be simpler. Here is some characters of our data populated to the mongodb.

File sizes:
The original OSM file is 53 MB
The JSON file is 55 MB

# of users

len(db.Santa_osm.distinct('created.user'))
436

# of nodes and ways:
Number of nodes: 251833
Number of ways: 21122

Top 10 Amenities (Public Service Areas):

```
[{u'count': 953, u'_id': u'parking'}, {u'count': 263, u'_id': u'bicycle_pa
rking'}, {u'count': 250, u'_id': u'restaurant'}, {u'count': 232, u'_id': u
'toilets'}, {u'count': 202, u'_id': u'bench'}, {u'count': 141, u'_id': u'p
lace_of_worship'}, {u'count': 105, u'_id': u'school'}, {u'count': 96, u'_i
d': u'recycling'}, {u'count': 91, u'_id': u'cafe'}, {u'count': 68, u'_id':
u'drinking_water'}]
```

Top 6 Restaurants:

```
[{u'Food': None, u'Count': 80}, {u'Food': u'mexican', u'Count': 32}, {u'Fo
od': u'chinese', u'Count': 22}, {u'Food': u'pizza', u'Count': 21}, {u'Food
': u'italian', u'Count': 14}, {u'Food': u'japanese', u'Count': 13}]
```

The above figure shows the population diversity of this small city.

Building Types:

```
[{u'count': 3857, u'_id': u'yes'}, {u'count': 144, u'_id': u'commercial'},
{u'count': 131, u'_id': u'house'}, {u'count': 123, u'_id': u'residential'}
, {u'count': 114, u'_id': u'apartments'}]
```

3. Additional data exploration using MongoDB queries

I was curious to see how the Zip code statistics looks like after populating it to the mongodb database. Here is the top 10 list of zip-codes:

Top 10 Freqent Post codes

[{u'count': 406, u'_id': u'95064'}, {u'count': 97, u'_id': u'95060'}, {u'count': 31, u'_id': u'95066'}, {u'count': 10, u'_id': u'95062'}, {u'count': 10, u'_id': u'95018'}, {u'count': 9, u'_id': u'95003'}, {u'count': 9, u'_id': u'95010'}, {u'count': 6, u'_id': u'95073'}, {u'count': 4, u'_id': u'95065'}, {u'count': 1, u'_id': u'95019'}]

Top 10 cities

```
[{u'count': 508, u'_id': u'Santa Cruz'}, {u'count': 35, u'_id': u'Scotts V
alley'}, {u'count': 9, u'_id': u'Felton'}, {u'count': 9, u'_id': u'Capitol
a'}, {u'count': 7, u'_id': u'Aptos'}, {u'count': 4, u'_id': u'Soquel'}, {u
'count': 1, u'_id': u'SANTA CRUZ'}, {u'count': 1, u'_id': u'capitiola'}, {
u'count': 1, u'_id': u'CAPITOLA'}, {u'count': 1, u'_id': u'Bonny Doon'}]
```

I was trying to see if there is any misplaced city outside of our area of map, but I have found it to be that all of them are within the Santa Cruz County.

4. Conclusion

As stated in the introductory part, even though it is very essential to have an up to date data, the way we update our data should attempt to minimize (if possible avoid) errors. OSM is getting fast popularity. As mentioned earlier, being open to be updated by different users makes it easy to include up to date information like if a new business is opened or changed its functionality. Unlike other commercial developers, OSM data is free for users to contribute in the development of OSM data which creates work collaboration. Its drawback is that, users can make errors while updating. As far as we standardized the way users update data, its advantages is much greater than its weaknesses and thus it should be implemented especially in poor countries like mine, where it's not easy to get up to date data.

By examining the Santa Cruz data, I have noticed that most of the errors are data entry type of errors, where different users put data in the way they like it. I am glad to hear that, the documentation of OSM is on its way to be published, which I believe will solve the aforementioned weakness while updating information. So once we have documentation manuals, can update data based on common standards, which avoids user bias.

Generally speaking the Santa Cruz data is fairly clean, but I suggest the zip code should be all numeral form and also adjusted to 5 digits. We need to have standard naming conventions in order our data to be uniform, which gives desired correct aggregates.

Citation

1. http://blog-en.openalfa.com/how-to-query-openstreetmap-using-the-overpass-api
2. http://www.openstreetmap.org/about
3. http://searchdatamanagement.techtarget.com/definition/MongoDB
4. http://stackoverflow.com/questions/2577236/regex-for-zip-code
5. http://stackoverflow.com/questions/30327508/mongodb-osm-street-maps-unique-users
6. http://www.ciclt.net/sn/clt/capitolimpact/gw_ziplist.aspx?ClientCode=capitolimpact&State=ca&StName=california&StFIPS=&FIPS=06087
7. https://www.youtube.com/watch?v=ZdDOauFIDkw
8. http://www.webilop.com/openstreetmap-an-alternative-to-google-maps/
9. https://www.e-education.psu.edu/geog585/node/738