OpenStreetMap Project

Wrangle OpenStreetMap Data

City: Bangalore, India

1. Introduction

OpenStreetMap is a wikipedia like map that can be edited by any user. Being an opensource map that is unbiased and uncontrolled, OpenStreetMap does face some problems with data consistency.

This project will perform data wrangling of the OpenStreetMap data for the city of Bangalore. I have been living in Bangalore for the last 9 years, and it would be fun to see what the OpenStreetData for Bangalore says about this city.

To start with, I downloaded the OSM XML for Bangalore from the Metro Extracts on **OpenStreetMap website (https://mapzen.com/data/metro-extracts/metro/bengaluru_india/ (https://mapzen.com/data/metro-extracts/metro/bengaluru_india/))**

The entire dataset was 630MB and was bigger than what my laptop memory could handle. So I created a sample of the data, taking every 5th element of the dataset. The sampling code from udacity website was used to create this file. The below analysis was performed on this sample.

2. Problems encountered with the map

Python script: process_map.py

2.1 Postal Codes

The Postal Codes in India are of 6 digits, contain only numbers and do not contain spaces. Looking at the postal code in the data, I could see some inconsistencies.

The entire code to audit postcodes is in cleaning_for_database.py

In [11]:
```
# For each postcode element, we classify it as good or bad postcode
for event, element in ET.iterparse(filename):
    if is_postcode(element):
        audit_postcode(postcode_good,postcode_bad,element.attrib["v"])

print postcode_bad
```

```
set(['5600041', '560001ph', '5600109', '560 001', '560 002', '560 025', '560
 080', '560 064', '560 055', '380068', '560 077', '- 560068', '5600037', '- 5
60001'])
```

Here we can see the problems with the postcodes.

1. The most common problem is formatting. Some postcodes, seem to have an additional space in between. This is easy to fix.
2. The second type of problem is the presence of a special character, like a "-". This again can be addressed.
3. The third type of problem is the postcodes being more than 6 digits and starting with a digit other than 5. These are harder to fix programmatically as we do not know which digit to remove. So we will not be addressing this issue in this code.

We will address the first 2 issues in the next section.

2.2 Street Types

Let's look at the street types next.

There are a few expected street types for a Bangalore city. These are: "Road","Stage","Layout","Street","Nagar","Circle","Cross","Main","Block","Phase","Colony"

I decided to first understand how other street types are present in the data. So I collect the street names in a set called all_streets and the street types that are not in expected in a set called street_types.

The entire code to audit and collect street types is in cleaning_for_database.py

In [3]:
```
## Go through each street element, collect all the street names and street typ
es
all_streets = set()
street_types = defaultdict(set)
for event, element in ET.iterparse(filename):
    if is_street(element):
        collect_streetnames(all_streets,element.attrib["v"])
        audit_streetname(street_types,element.attrib["v"])

print len(all_streets)
print len(street_types)
```

```
841
203
```

So there are about 203 street types that do not belong in the expected street types. That's huge for 841 street names!

Let's see a few examples from the street_types set.

In [21]:
```
print list(street_types.keys())[3:7]
```

```
['Bangalore-66', 'Krishnarajapuram', 'btm', 'Bellandur']
```

The problem here is that the area names are part of the addr:street field. This is a common way of writing addresses in India, where street names alone do not always suffice to capture the address.

Unfortunately, it is highly difficult to fix this issue programmatically as it is hard to identify all valid area names and street names.

For this excercise, I am choosing to leave this field as is and provide some additional suggestions in a further section.

2.3 Phone Number

Fix all phone numbers to follow the format +91 followed by number in one of these formats +91xxxxxxxxxx (for all mobile numbers) +91xxxxxxxxxxx (for landline numbers starting with city code 080 for Bangalore) 1800xxxxxxx (for toll free numbers)

In [23]:
```
#Print a small subset so we can see a few discrepancies
print list(all_phone)[100:120]
```

```
['+91 80 40926283', '(91)-80-41572230', '+91 9845080075', '+91-080 6741 778
5', '800560076', '+91-7676751499', '080 4120 3999', '+91-80-25589333;+91-80-5
1783344;+91-80-25588697', '+918028029548', '+918040515253', '+91-80-2344197
4', '080 4210 0812', '080 2558 8776', '+977607 77999', '+91 80 41487799;+91 8
0 41725222', '9343223223', '080 2221 0540', '+918068886888', '080 4117 1552',
'080 6569 7600']
```

The phone field has a lot of formatting issues. This is typical of user entered data. Some of the common formatting issues in phone field are:

1. Use of ":". "()", "-" in phone number
2. Use of spaces for formatting
3. Country code missing
4. Use of a leading 0 in the phone number

We will fix these again in the next section.

2.4 Website

The data for the website URLs are inconsistent with some specifying http or https, and other just specifying the URL as "aaa.com".

In order to bring uniformity in the data, we will consider valid website format as http://www.aaa.com (http://www.aaa.com) or https://www.aaa.com (https://www.aaa.com) or http:aaa.com or https://aaa.com (https://aaa.com)

```
In [ ]: # Website URLs must start with http or https
        website_re = re.compile(r'^\bhttp[s]?\b')
```

Here are some samples of valid and invalid URLs from the dataset. We will fix them in the next section

Valid URLS ['http://www.library.iisc.ernet.in/ (http://www.library.iisc.ernet.in/)', 'http://www.littleangelsclinic.com/ (http://www.littleangelsclinic.com/)', 'http://www.morestore.com (http://www.morestore.com)', 'http://www.goldenpalmshotel.com/ (http://www.goldenpalmshotel.com/)', 'http://highgateshotel.com (http://highgateshotel.com)', 'http://www.themarkboulevard.com/ (http://www.themarkboulevard.com/)', 'https://www.axisbank.com (https://www.axisbank.com)', 'http://www.ryaninternational.org (http://www.ryaninternational.org)', 'http://www.goldfinchhotels.com (http://www.goldfinchhotels.com)']

Invalid URLS ['www.karunashraya.org', 'stamfordresidency.com', 'apollospectra.com', 'www.aabsweets.in', 'www.malabargoldindia.com', 'www.ITforchange.net', 'www.faasos.com', 'www.karthavya.com', 'ushalexushotels.com']

2.5 House Number

On collecting all the house numbers and looking through the data, we can spot some issues:

1. House numbers are confused with full address including street name.
2. Some house numbers do not have any numbers

In India, house numbers always contain numbers, they may not may not contain an alphabet, like 491/A for example is a valid house number. But H is not a house number.

In [ ]:
```
# house numbers must end with a digit or a single alphabet.
house_numbers_re = re.compile(r'\d+[/-]?[a-zA-Z]?$')
```

We will use this regex to identify the invalid house numbers. Below are some samples:

Valid House Numbers ['100', '30', '60', 'P-4', '956', '95/3', '38/3', '13', '125']

Invalid House Numbers ['92/3, 80 Feet Road', '1st floor', 'Hosur road', 'RMZ Infinity', '9 ', 'H', 'Aditya Layout, RR Nagar', 'CareerNet Technologies', '60th Main road']

Unfortunately, we cannot easily fix the invalid house numbers because there are a variety of errors - street addresses with and without house numbers are found in this field. So for my analysis, I am going to ignore these values.

3. Cleaning the data programmatically

Python script: cleaning_for_database.py

3.1 Postal codes

I found 2 issues with the postal codes:

1. Some postcodes, seem to have an additional space in between.
2. Presence of a special character, like a "-".

To fix the postal code, I decided to just take a subset of numbers from the postal code.

In [32]:
```
postcode = '560 001'
postcode_correct = re.sub('[^0-9]','',postcode)
print postcode_correct
```

```
560001
```

3.2 Phone Number

I wrote a function called fix_phone_number to correct all issues with the phone number field.

The function is part of cleaning_for_database.py script.

In summary, this is the behavior of the function:

- Remove : in phone
- Remove () in phone
- Remove - in phone
- Remove spaces in phone
- Remove quotes in phone
- If phone starts with 91, append a + sign before the number
- If phone starts with 0, remove 0
- If phone starts with 1800, do not append +91
- Add +91 to the rest of the phone numbers

3.3 Website

To fix the issues with website URLs, we first identify incorrect URL with the regular expression from the previous section.

If the website URL starts with http or https, return the value. Else append http:// before the URL.

In [ ]:
```
def check_url(website):
    if re.search(website_re,website):
        return website
    else:
        return ("http://" + website)
```

At this stage, after fixing these issues, the data is ready to be convered to JSON and imported into MongoDB

Command to import the JSON file from the Mongo Shell

mongoimport --db OSM --collection bangalore --type json --file bengaluru_sample_k05.osm.json

4. Overview of the data

Python script: data_overview.py

4.1 Statistical Overview

File Sizes:

- bengaluru_sample_k05.osm --> 127 MB
- bengaluru_sample_k05.osm.json --> 148 MB

Count of documents in the DB

```
In [4]: db.bangalore.count()
Out[4]: 704971
```

Number of nodes and ways

In Mongo Shell

```
db.bangalore.find({type:"node"}).count()
573594

db.bangalore.find({type:"way"}).count()
131377
```

4.2 Additional Statistics

Top 10 users by contribution

In [6]:
```
users = db.bangalore.aggregate([
    {"$group" : {"_id" : "$created.uid",
                    "count" : {"$sum" : 1}}
    },
    {"$sort" : {"count" : -1}
    },
    {"$limit" : 10}
])
for user in users:
    print user
```

```
{u'count': 25199, u'_id': u'2900516'}
{u'count': 23849, u'_id': u'2910514'}
{u'count': 23136, u'_id': u'2901478'}
{u'count': 23010, u'_id': u'2900470'}
{u'count': 20021, u'_id': u'2900616'}
{u'count': 18874, u'_id': u'1306'}
{u'count': 18859, u'_id': u'2901753'}
{u'count': 17649, u'_id': u'2901493'}
{u'count': 17417, u'_id': u'2905518'}
{u'count': 17054, u'_id': u'2905508'}
```

How many unique users?

In Mongo Shell:

db.bangalore.distinct("created.uid").length
1317

First user entry

In [8]:
```
entries = db.bangalore.aggregate([
    {"$sort" : {"created.timestamp" : 1}
    },
    {"$project" : {"_id":0,"created.user":1 ,"created.timestamp":1, "created.u
id":1 }
    },
    {"$limit" : 1}
])
for entry in entries:
    print entry
```

```
{u'created': {u'timestamp': u'2007-03-14T13:56:35Z', u'user': u'Scooter_DE',
 u'uid': u'2179'}}
```

Most common postal code

In [14]:
```
postcode = db.bangalore.aggregate([
    {"$match" : {"address.postcode" : {"$exists" : 1}}
    },
    {"$group" : {"_id" : "$address.postcode",
                "count" : {"$sum" : 1}}
    },
    {"$sort" :{"count" : -1}
    },
    {"$limit" : 1}
])
for code in postcode:
    print code
```

```
{u'count': 60, u'_id': u'560066'}
```

There are quite a lot of documents with no postal code. In fact, only 907 documents have a postal code. We can find that if we remove the match stage in the pipeline above.

Top 5 listed amenities

In [12]:
```
amenities = db.bangalore.aggregate([
    {"$match" : {"amenity" : {"$exists" : 1}}
    },
    {"$group" : {"_id" : "$amenity",
                "count" : {"$sum" : 1}}
    },
    {"$sort" : {"count" : -1}
    },
    {"$limit" : 5}
])
for amenity in amenities:
    print amenity
```

```
{u'count': 375, u'_id': u'restaurant'}
{u'count': 216, u'_id': u'place_of_worship'}
{u'count': 161, u'_id': u'atm'}
{u'count': 145, u'_id': u'bank'}
{u'count': 145, u'_id': u'school'}
```

Restaurants are the most tagged amenities followed by places of worship.

Most popular cuisines

In [13]:
```
restaurants = db.bangalore.aggregate([
    {"$match" : {"amenity" : {"$eq" : "restaurant"}}
    },
    {"$match" : {"cuisine" : {"$exists" : 1}}
    },
    {"$group" : {"_id" : "$cuisine",
                "count" : {"$sum" : 1}}
    },
    {"$sort" : {"count" : -1}
    },
    {"$limit" :5}
])
for cuisine in restaurants:
    print cuisine
```

```
{u'count': 76, u'_id': u'regional'}
{u'count': 49, u'_id': u'indian'}
{u'count': 16, u'_id': u'chinese'}
{u'count': 13, u'_id': u'vegetarian'}
{u'count': 10, u'_id': u'pizza'}
```

5. Other ideas about the dataset

Python script: additional_statistics.py

5.1 How many houses near schools?

With Bangalore infamous for its traffic woes, it is common for people to stay closer to schools so that children cut down on commute time. So I wanted to see how many residential buildings are near the schools.

To accomplish this, first I will collect the school names and positions as well as the residential building positions in dictionaries and lists respectively.

The data for schools is spread between nodes and ways. Nodes have a position data in them, hence it is straightforward to extract the school name and position from the nodes. For ways, the process is a two step one. Ways do not have position data, but they do have node_refs, which point to nodes. To get the position of a way, we can take the node_ref, match it to a node id and get the position from the node.

In [5]:
```
schools_dict = {}

# Filter for all schools with a name
schools = db.bangalore.aggregate([
    {"$match" : {"amenity": "school",
                   "name" : {"$exists": 1}}
    }
])
for sc in schools:
    # Nodes have Pos data, so we take the nodes and add the Pos information to
 a dictionary with school name as key
    if sc["type"] == "node":
        schools_dict[sc["name"]] = sc["pos"]
    # Ways do not have Pos data. But they have a node reference and the that i
n turn can give us Pos data
    else:
        # Collect all node_refs with the school names
        schools_node_db = db.bangalore.aggregate([
           {
            "$match" : {"amenity": "school",
            "name" : {"$exists": 1}}
           },
           {
            "$match" : {"node_refs" : {"$exists" : 1}}
           },
           {
            "$project" : {"name" : "$name", "node" : "$node_refs"}
           }
        ])
        schools = {}
        # Loop thorugh the pipeline result object and add the node_refs to a l
ist in a dictionary with school names as keys
        for school in schools_node_db:
            n = school["node"]
            values = []
            for i in n:
                values.append(i)
            schools[school["name"]] = values
```

In [6]:
```
# Here we iterate through the dictionary and look for a matching node. Then we
 get the node Pos
for sc,value in schools.iteritems():
    sc_pos_db =  db.bangalore.aggregate([
    {
    "$match" : {"type" : "node",
               "id" : {"$in" : value}
                }
    },
    {"$project" : {"name" : sc, "pos" : "$pos", "_id" : 0}}
    ])
    # A school can have more than one node and each of these nodes give a slig
htly different Pos. In case of multiple values,
    # only one of them gets added to the dictionary
    for s in sc_pos_db:
        schools_dict[s["name"]] = s["pos"]

print len(schools_dict)
```

```
121
```

So there are a 121 schools with position data.

Next step is to get the list of buildings. All the building information is in ways. The process for getting the buildings location is very similar to that of schools.

We first filter the amenity type to reflect houses and then proceed to get the node_refs, using which we will get the node ids and positions.

In [7]:
```
# Collect all residential buildings information
buildings_node_db = db.bangalore.aggregate([
    {
    "$match" : {
                "$or" :
               [
                   {"building" : "Residential House"},
                   {"building" : "apartment"},
                   {"building" : "apartments"},
                   {"building" : "residential"},
                   {"building" : "house"}
               ]
    }
    },
    {
        "$match" : {"node_refs" : {"$exists" : 1}}
    },
    {"$project" : {"node" : "$node_refs"}}
])

# All buildings are in ways. So they do not have pos information. We take the
 node_refs information here and add them to a list
building_nodes = []
for b in buildings_node_db:
    n = b["node"]
    for i in n:
        building_nodes.append(i)
print len(building_nodes)
```

```
2745
```

In [8]:
```
# Find a matching node for each node_ref from the list
building_pos_db =  db.bangalore.aggregate([
    {
    "$match" : {"type" : "node",
               "id" : {"$in" : building_nodes}
                }
    }
])

# Get the latitude and longitude values for the buildings
building_pos = []
for i in building_pos_db:
    lat = i["pos"][0]
    lon = i["pos"][1]
    building_pos.append([lat,lon])
print len(building_pos)
```

```
505
```

There are 505 buildings with position data.

Now we can move on to the last step, which is to compare positions.

I wanted to create a to create a square shaped range around each school and look for number of houses in that area. The side of a square is 0.08.

In [19]:
```
# Create a square shaped range around each school and look for number of house
s in that area. The side of a square is 0.08.
houses_near_schools = {}
for school in schools_dict.keys():
    count=0
    lat = schools_dict[school][0]
    lon = schools_dict[school][1]
    for build in building_pos:
        if build[0] >= lat - 0.04 and \
        build[0] <= lat+0.04 and \
        build[1] >= lon - 0.04 and \
        build[1] <= lon + 0.04:
            count+=1
    houses_near_schools[school] = count

#Sort the dictionary keys based on values
sorted_values= sorted(houses_near_schools, key = lambda x:
houses_near_schools[x], reverse=True)

for school in sorted_values:
    print("There are {} houses near {}".format(houses_near_schools[school] , s
chool))
```

Sample of the top 5 results
There are 140 houses near Bangalore Montessori High School
There are 133 houses near Dr Ambedkar Nursery Primary & High School
There are 131 houses near Bangalore Public School
There are 127 houses near St. Thomas Public School
There are 125 houses near Poorna Prajna Education Centre

There indeed are a huge number of houses near schools and there is also quite a bit of overlap. A lot of houses are located close to multiple schools, which again makes sense as densely populated residential areas tend to have more schools next to each other.

5.2 Improving the dataset

The dataset for Bangalore city is by no means complete but is a good place to start. The dataset has a huge number of nodes and ways but the details are missing. For example, there are only 907 postal code data available.

One problem with this dataset that is hard to solve programatically is the street addresses. Since there is no consistent way of writing addresses, a better option would be to provide an option to pick up the address data from a GPS location. There are a lot of local services, like mapmyindia that provide GPS location data. Using these create addresses would ensure a lot more uniformity in addresses.

Incentives for updating OSM information:
A bigger problem with the Bangalore data is the lack of information too. For a city as huge as Bangalore the information on say restaurants, schools etc are very sparse. Information on Metro services are completely missing. One way to get people to participate more and provide updates would be make the process more engaging. Some lessons from gamification maybe a good way to do this. Providing points for updates, badges, user levels etc may increase participation.

Increase awareness to OSM:
Increasing awareness to OSM would encourage more people to update and use OSM. As the ecosystem of OSM users increases, more updates will follow. Conducting OSM meetups in various cities is one way to spread the word. There is a lot that can be done with OSM data both in data analysis as well as offline mapping.

Advantages and Disadvantages of the improvement suggestions ¶

Advantages

1. OSM data is crowd sourced and as with any project that is open it has a lot of data issues. If we could address some of these issues with the help of automation, like using GPS data for location, it would make the data cleaner to use. OSM is in a space that is dominated by products like Google Maps which provide much cleaner information.
2. OSM map coverage will improve with more user participation and quality information will in turn increase awareness to OSM.

Disadvantages

1. Relying on a third party service to pick up location data may introduce errors due to incorrect information in the service. For example, if mapmyIndia has a location tagged incorrectly or has out of date information, the error would creep into OSM too.
2. Getting users to contribute is easier said than done.

6. References

- https://mapzen.com/data/metro-extracts/metro/bengaluru_india/ (https://mapzen.com/data/metro-extracts/metro/bengaluru_india/)
- https://www.theguardian.com/technology/2014/jan/14/why-the-world-needs-openstreetmap (https://www.theguardian.com/technology/2014/jan/14/why-the-world-needs-openstreetmap)
- https://docs.mongodb.com/manual/aggregation/ (https://docs.mongodb.com/manual/aggregation/)
- https://discussions.udacity.com/t/passing-output-of-one-aggregation-to-another/275096 (https://discussions.udacity.com/t/passing-output-of-one-aggregation-to-another/275096)