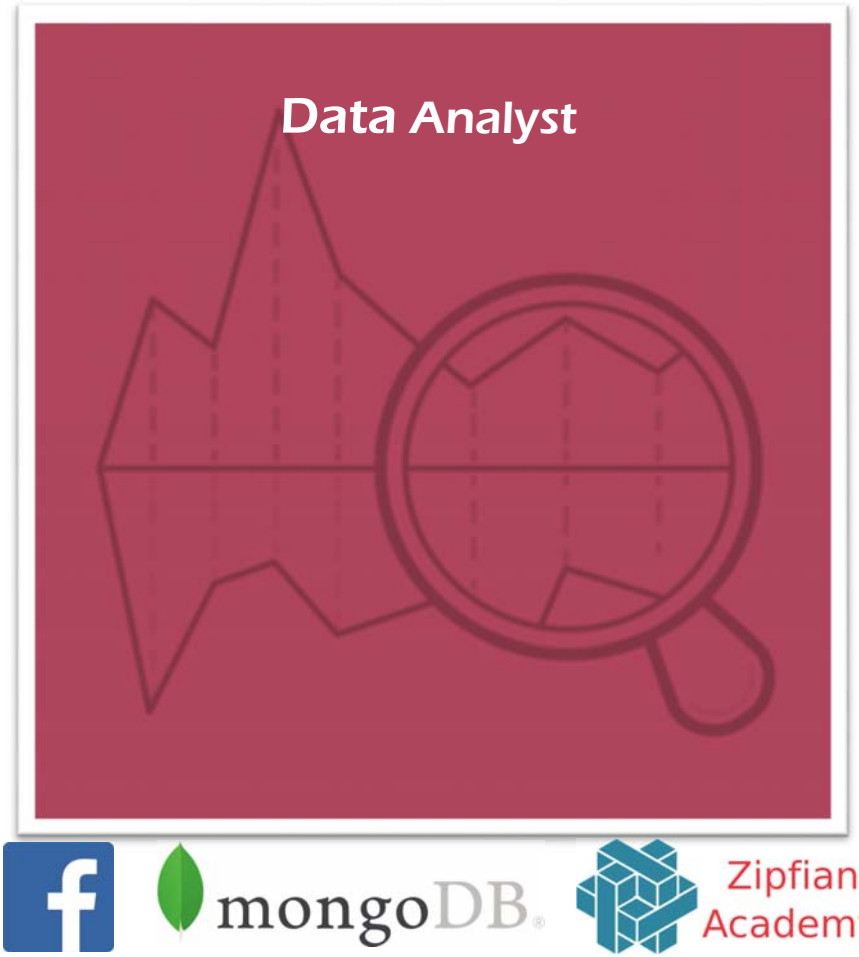


Data Analyst Nano Degree

P3 – WRANGLE OPENSTREETMAP DATA

Harshit Trivedi | Data Analyst Nano Degree | March 15, 2016

Background Information

OpenStreetMap is a community built map of the world, similar in nature to how content on GitHub, Wiki, CodePlex and many other websites are generated and maintained. Here, users can map roads, buildings, areas of interest or landmarks, which can be further tagged with details such as street addresses or amenity type. I read about the detailed preview, features and map format on https://wiki.openstreetmap.org/wiki/Main_Page.

Highlights of OSM XML format

- It is a list of instances of data primitives (nodes, ways and relations) found within a given bounds.
- Nodes represent dimensionless points on the map.
- Ways contain node references to form either a polyline or a polygon on the map.
- Nodes and Ways both contain children tag elements that represent key-value pairs of descriptive information about a given node or way.
- Relations consist of one or more tags and also an ordered list of one or more nodes, ways and/or relations defining logical or geographical relations between other elements.

Choosing The Map Area

Having been born and brought-up in **Delhi**, I'm a pure Delhiite to the core. So, choosing the map area as Delhi (NCT region of Delhi) was an obvious choice.

Delhi is the capital territory of India. It is the largest city in India in terms of geographical area - about 1,484 square kilometers (573 sq. mi). It is also the second most populous city in India with a population of about 16.3 million with a population density of about 11,297/km^2. Delhi features an atypical version of the humid subtropical climate experiencing a wide variety of weathers throughout the year.

I used Overpass API to download the OpenStreetMap XML for the corresponding bounding box: https://www.openstreetmap.org/export#map=12/28.6469/77.2504

Auditing The Map Data

With the OSM XML map file downloaded, I started the process of auditing the data. This is an essential step before cleaning and analyzing the data.

Identifying the Tags

Here, I have 3 regular expressions: lower, lower_colon and problemchars.

- lower: matches strings containing lower case characters.
- lower_colon: matches strings containing lower case characters and a single colon within the string.
- problemchars: matches characters that cannot be used within keys in MongoDB.

Input:

```
import xml.etree.cElementTree as ET
import pprint
import re
from collections import defaultdict

lower = re.compile(r'^([a-z]|_)*$')
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')

def key_type(element, regex_keys, keys):
    if element.tag == "tag":
        if lower_colon.search(element.attrib['k']):
            regex_keys['lower_colon'] += 1
        elif problemchars.search(element.attrib['k']):
            regex_keys['problemchars'] += 1
        elif lower.search(element.attrib['k']):
            regex_keys['lower'] += 1
        else:
            regex_keys['other'] += 1
    keys[element.attrib.get('k')].add(element.attrib.get('v'))
    return regex_keys, keys
def process_map(filename):
    regex_keys = {"lower": 0, "lower_colon": 0, "problemchars": 0, "other":
0}
    keys = defaultdict(set)
    for _, element in ET.iterparse(filename):
        regex_keys, keys = key_type(element, regex_keys, keys)
    return regex_keys, keys
regex_keys, keys = process_map('map')
print 'RegEx Keys:'
pprint.pprint(regex_keys)
```

Output:

```
RegEx Keys:
{'lower': 674454, 'lower_colon': 8009, 'other': 243, 'problemchars': 7}
```

Identifying the Unique Users

To find out the number of unique users who have contributed to this map's data, we can build a set of these unique User IDs (uid), and then find out the length of this set.

Input:

```
import xml.etree.cElementTree as ET
import pprint
import re
from collections import defaultdict

def process_map(filename):
    users = defaultdict(int)
    for _, element in ET.iterparse(filename):
        uid = element.attrib.get('uid')
```

```
        if uid != None:
            users[uid] += 1
    return users

users = process_map('map')
print "No. Of Unique Users :"
pprint.pprint(len(users))
```

Output:

```
No. Of Unique Users :
883
```

Problems With the Data

Street Names

I found many street names to be incorrectly named, abbreviated or incomplete. To find out and clean these, I parsed through the XML looking for tag elements with k="addr:street" attributes. Then I compared these with a prepared regex dictionary to return a clean string for these.

Input:

```
import xml.etree.cElementTree as ET
from collections import defaultdict
import re
import pprint

OSMFILE = "map"
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)

expected = ["Road", "Marg", "Place", "Lane", "Kunj", "Colony", "Estate",
"Nagar",
            "Lines", "Enclave", "Expressway", "Market", "Extension", "Area",
"Chowk", "Complex"]

mapping = { 'road' : "Road",  'Rd' : "Road", 'Abulane' : "Abu Lane",
'vihar' : "Vihar",
           'Mkt' : "Market", 'colony' : "Colony" }

def audit_street_type(street_types, street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type not in expected:
            street_types[street_type].add(street_name)

def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")

def audit(osmfile):
    osm_file = open(osmfile, "r")
    street_types = defaultdict(set)
```

```
    for event, elem in ET.iterparse(osm_file, events=("start",)):
        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_street_name(tag):
                    audit_street_type(street_types, tag.attrib['v'])
    elem.clear()    # Clear from memory
    return street_types

def process_match(m):
    if mapping.get(m.group()) != None:
        return mapping.get(m.group())
    else:
        return m.group()
    return

def update_name(name, mapping, regex):
    # Update each street name with the replacement ending in the mapping
dictionary
    m = regex.search(name)
    if m:
        match = m.group()
        if match in mapping:
            name = re.sub(regex, mapping[match], name)
       return name

def test():
    st_types = audit(OSMFILE)
    for st_type, ways in st_types.iteritems():
        if st_type in mapping:
            for name in ways:
                better_name = update_name(name, mapping, street_type_re)
                print name, "=>", better_name.title()

if __name__ == '__main__':
    test()
```

Output (Fragment):

```
Arya Samaj Rd => Arya Samaj Road
swami Narayan Marg, Ashok vihar => Swami Narayan Marg, Ashok Vihar
Ring road => Ring Road
Abulane => Abu Lane
Khora colony => Khora Colony
Dr. Bishamber das marg => Dr. Bishamber Das Marg
```

Besides dirty data, there was also an apparent lack of data on street addresses altogether. To count the total number of nodes and ways that contain a tag child with k="addr:street".

Input:

```
import xml.etree.cElementTree as ET
from collections import defaultdict
import re
import pprint

osm_file = open("map", "r")
```

```
address_count = 0

def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")

for event, elem in ET.iterparse(osm_file, events=("start",)):
    if elem.tag == "node" or elem.tag == "way":
        for tag in elem.iter("tag"):
            if is_street_name(tag):
                address_count += 1
print address_count
```

Output:

```
917
```

Postal Codes

I found many postal codes (PIN codes) to be incorrect. They had a variety of errors, i.e. string entries, 4, 5 or 7-digit numeric entries, alphanumeric entries, and space/- separated entries.

Input:

```
def fixed_postal_code(code):
    if VALID_POSTAL.match(code):
        return (code, code)
    elif INVALID_POSTAL_NON_BREAKING_HYPHEN.match(code):
        fixed = code[0:2] + '-' + code[2:]
        return (code, fixed)
    elif INVALID_POSTAL_MISSING_HYPHEN.match(code):
        fixed = code[0:2] + '-' + code[2:]
        return (code, fixed)
    else:
        print code
        msg = u'Invalid postal code: {0}'.format(code)
        raise Exception(msg)


def find_postal_issues(osmfile):
    """Returns dictionaries of valid and invalid postal codes; original
values as dictionary keys, fixed values as values."""
    osm_file = open(osmfile, "r")
    valid = {}
    invalid = {}

    for event, elem in ET.iterparse(osm_file, events=("start",)):
        if elem.tag == 'node':
            for tag in elem.iter('tag'):
                k = tag.get('k')
                if k == "addr:postcode":
                    v = tag.get('v')
                    v, fixed = fixed_postal_code(v)
                    if v == fixed:
                        valid[v] = v
```

```
                    else:
                        invalid[v] = fixed
    return valid, invalid


def report_postal_issues():
    #Prints a report about valid and invalid postal codes to stdout.
    valid_postal, invalid_postal = find_postal_issues(OSMFILE)
    print '# of valid postal codes: ', len(valid_postal)
    print '# of invalid postal codes: ', len(invalid_postal)
    print
    print 'Invalid postal codes:'
    pprint.pprint(invalid_postal)
```

Output:

```
Sunpat House Village
Exception: Invalid postal code: Sunpat House Village
```

Exporting to MongoDB

To load the XML data into MongoDB, we have to first transform the audited and corrected data into JSON format.

Input:

```
import xml.etree.cElementTree as ET
import pprint
import re
import codecs
import json
from collections import defaultdict

lower = re.compile(r'^([a-z]|_)*$')
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)


CREATED = [ "version", "changeset", "timestamp", "user", "uid"]
ST_TYPE_EXPECTED = ["Road", "Marg", "Place", "Lane", "Kunj", "Colony",
"Estate", "Nagar",
            "Lines", "Enclave", "Expressway", "Market", "Extension", "Area",
"Chowk", "Complex"]

ST_TYPE_MAPPING = { 'road' : "Road",  'Rd' : "Road", 'Abulane' : “Abu Lane”,
'vihar' : "Vihar",
           ‘Mkt’ : “Market”, 'colony' : "Colony" }

def shape_base(element):
    node = defaultdict()
    created_keys = ["version", "changeset", "timestamp", "user", "uid"]
    root_keys = ['id', 'visible']
    node['type'] = element.tag
```

```
    node['created'] = defaultdict()

    # Probably could use the .get() method on element here
    if 'lat' in element.attrib and 'lon' in element.attrib:
        node['pos'] = [float(element.attrib['lat']),
float(element.attrib['lon'])]
    for k, v in (element.attrib).iteritems():
        if k in created_keys:
            node['created'][k] = v
        elif k in root_keys:
            node[k] = v
    return shape_node(node, element)

def regex_key(k):
    l = lower.search(k)
    lc = lower_colon.search(k)
    pc = problemchars.search(k)
    return l, lc, pc

def shape_node(node, element):
    node_refs = []
    address = defaultdict()
    for t in element:
        k = t.attrib.get('k')
        v = t.attrib.get('v')
        r = t.attrib.get('ref')
        if k:
            l, lc, pc = regex_key(k)
            if pc == None:
                if k.startswith('addr'):
                    if lc:
                        # FIXME The v in this line need to be run through
data
                        # cleaning for street types
                        address[k.split(':')[1]] = v
                    else:
                        continue
                else:
                    node[k] = v
        if r:
            node_refs.append(r)
        if len(address) > 0:
            node['address'] = address
        if len(node_refs) > 0:
            node['node_refs'] = node_refs
    return node

def shape_element(element):
    node = {}
    if element.tag == "node" or element.tag == "way" :
        node = shape_base(element)
        return dict(node)
    else:
        return None
```

```
# Instructor code from audit.py
def audit_street_type(street_types, street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type not in ST_TYPE_EXPECTED:
            street_types[street_type].add(street_name)

# Instructor code from audit.py
def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")

# Instructor code from audit.py
def audit(osmfile):
    osm_file = open(osmfile, "r")
    street_types = defaultdict(set)
    for event, elem in ET.iterparse(osm_file, events=("start",)):
        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_street_name(tag):
                    audit_street_type(street_types, tag.attrib['v'])
    return street_types

def process_match(m):
    if ST_TYPE_MAPPING.get(m.group()) != None:
        return ST_TYPE_MAPPING.get(m.group())
    else:
        return m.group()
    return

def update_name(name):
    return street_type_re.sub(process_match, name)


def process_map(file_in, pretty = False):
    # You do not need to change this file
    file_out = "{0}.json".format(file_in)
    data = []
    with codecs.open(file_out, "w") as fo:
        for _, element in ET.iterparse(file_in):
            el = shape_element(element)
            if el:
                data.append(el)
                if pretty:
                    fo.write(json.dumps(el, indent=2)+"\n")
                else:
                    fo.write(json.dumps(el) + "\n")
    return data

def test():
    correct_first_elem = {
        "id": "261114295",
        "visible": "true",
        "type": "node",
```

```
        "pos": [41.9730791, -87.6866303],
        "created": {
            "changeset": "11129782",
            "user": "bbmiller",
            "version": "7",
            "uid": "451048",
            "timestamp": "2012-03-28T18:31:23Z"
        }
    }
if __name__ == "__main__":
    test()
```

Output:

```
[ 'map.json' file generated in the destination folder ]
```

File Sizes

Input:

```
import os

print "The downloaded file is {} MB".format(os.path.getsize('map')/1.0e6)
print "The JSON file is {} MB".format(os.path.getsize('map.json')/1.0e6)
```

Output:

```
The downloaded file is 634.742102 MB
The JSON file is 977.213704 MB
```

Exporting JSON to MongoDB

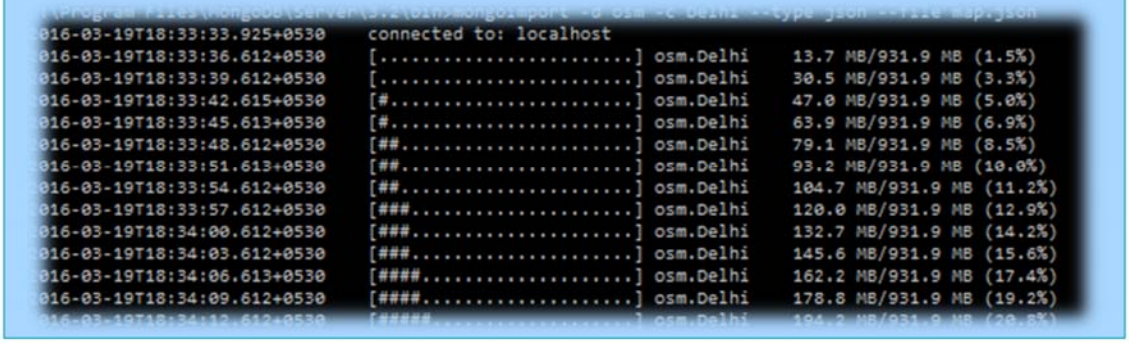
To export this JSON to MongoDB, I first started the MongoDB database process 'mongod.exe'. After the database process connected to localhost, I ran the command.

Input:

```
mongoimport –d osm –c Delhi –type json –file map.json
```

Output:

```
Imported 3391410 documents
```



Overview Of The Data

No. of nodes and ways in the map data

Input:

```
db.Delhi.find({“type”:”node”}).count()
```

Output:

```
2840416
```

Input:

```
db.Delhi.find({“type”:”way”}).count()
```

Output:

```
550847
```

Basic amenities like Banks, Gas Stations, Cinemas and Cafes in the map area.

Input:

```
db.Delhi.find({“amenity”:”bank”}).count()
```

Output:

```
115
```

Input:

```
db.Delhi.find({“amenity”:”fuel”}).count()
```

Output:

```
194
```

Input:

```
db.Delhi.find({“amenity”:”cinema”}).count()
```

Output:

```
53
```

Input:

```
db.Delhi.find({“amenity”:”cafe”}).count()
```

Output:

```
54
```

The top 10 amenities in the city:

Input:

```
db.Delhi.aggregate([{“$match” : {“amenity”:{“$exists”:1}}}, {“$group” :
{“_id”:”$amenity”, “count”:{“$sum”:1}}}, {“$sort”: {“count”:-1}},
{“$limit”:10}])
```

Output:

```
{“_id” : “school”, “count” : 893}
{“_id” : “place_of_worship”, “count” : 320}
{“_id” : “parking”, “count” : 303}
{“_id” : “fuel”, “count” : 194}
```

```
{“_id” : “hospital”, “count” : 187}
{“_id” : “restaurant”, “count” : 153}
{“_id” : “atm”, “count” : 134}
{“_id” : “college”, “count” : 129}
{“_id” : “bank”, “count” : 115}
{“_id” : “police”, “count” : 85}
```

Top fast food joints in the city based on no. of outlets in OpenStreetMaps data.

Input:

```
db.Delhi.aggregate([{"$match": {"amenity": "fast_food"}}, {"$group": {"_id":
"$name", "count":{"$sum":1}}}, {"$sort":{"count":-1}}])
```

Output:

```
{“_id” : “McDonald’s”, “count” : 10}
{“_id” : “Domino’s Pizza”, “count” : 5}
{“_id” : “McDonalds”, “count” : 5}
{“_id” : “Subway”, “count” : 5}
{“_id” : “KFC”, “count” : 3
... ... ...
```

All religions in the city based on Places of Worship in OpenStreetMaps data.

Input:

```
db.Delhi.aggregate([{“$match”:{“amenity”:”place_of_worship”}}, {“$group”:
{“_id”:”$religion”, “count”: {“$sum”:1}}}, {“$sort”:{“count”:-1}},
{“$limit”:1}])
```

Output:

```
{“_id” : “hindu”, “count” : 129}
```

User with the highest contributions

Input:

```
db.Delhi.aggregate([{"$group" : {"_id": "$created.user", "count":
{"$sum":1}}}, {"$project":{"_id":0, "user":"$_id", "count":1}},
{"$sort":{"count":-1}}, {"$limit":1}])
```

Output:

```
{“count” : 129, “user” : “Oberaffe”}
```

No. of distinct sources of data

Input:

```
db.Delhi.distinct("source").length
```

Output:

```
53
```

Problems With The Data

Incomplete Street Addresses

Input:

```
db.Delhi.find({"address.housenumber" : {"$exists":1, "$regex":/^[0-9]+$/},
"address.street" : {"$exists":0}}, {"address":1, "_id":0}).count()
```

Output:

```
352
```

Uncharted/Unidentified Data

In less-travelled areas or areas with potentially less awareness of the OpenStreetMap service, like the eastern part of my selected area, the data editing experience can vary widely - some areas might be very nearly uncharted territory, whereas others might be mostly filled out, with the occasional error.

One way to help users prioritize and edit more efficiently might be to display any incomplete addresses within a given radius on the map. Someone looking at "Rohini", for example, would then be shown all the amenities or other locations geographically near Rohini that lack a street name in their address.

Invalid Postal Codes

Input:

```
db.Delhi.aggregate([{"$match" :  {"address.postcode": {"$exists":1}}},
{"$group": {"_id":"$address.postcode", "count": {"$sum":1}}}, {"$sort":
{"count":1}}])
```

Output:

```
{"_id" : "122105", "count" : 1}
{"_id" : "110049", "count" : 1}
{"_id" : "420420", "count" : 1}
{"_id" : "110 067", "count" : 1}
{"_id" : "121004", "count" : 1}
{"_id" : "121008", "count" : 1}
{"_id" : "78703", "count" : 1}
{"_id" : "201007", "count" : 1}
{"_id" : "Sunpat House Village", "count" : 1}
{"_id" : "110037", "count" : 1}
{"_id" : "1100002", "count" : 1}
{"_id" : "121002", "count" : 1}
{"_id" : "110 001", "count" : 1}
{"_id" : "110010", "count" : 1}
{"_id" : "110022", "count" : 1}
{"_id" : "110031v", "count" : 1}
{"_id" : "203207", "count" : 1}
{"_id" : "110031", "count" : 1}
{"_id" : "200005", "count" : 1}
{"_id" : "2010", "count" : 1}
```

```
{"_id" : "20134", "count" : 1}
{"_id" : "110064", "count" : 1}
{"_id" : "110030", "count" : 1}
{"_id" : "110025", "count" : 2}
{"_id" : "110052", "count" : 2}
{"_id" : "122011", "count" : 2}
{"_id" : "110007", "count" : 2}
{"_id" : "110094", "count" : 2}
```

In many cases (highlighted in yellow), contributors have mistyped postal codes as 4 or 5 digit codes instead of a 6-digit valid postal code. In some cases, I also encountered string values pertaining to the street address or the locality in the 'addr:postcode' tag which threw an exception to my python code.

I think that the postal codes should be validated at the time of submission, so that they are received in the correct format as per the requirements of the region.

Invalid/Incorrect City Names

Similarly, for city names, many entries have been mistyped or incorrect.

Input:

```
db.Delhi.aggregate([{"$match" : {"address.city": {"$exists":1}}}, {"$group":
{"_id":"$address.city", "count": {"$sum":1}}}, {"$sort": {"count":1}}])
```

Output:

```
{"_id" : "Neew Delhi", "count" : 1}
{"_id" : "Sector-11, Rohini, Delhi", "count" : 1}
{"_id" : "DELHI", "count" : 1}
{"_id" : "Village- Barola, Sector- 49, Noida", "count" : 1}
{"_id" : "Uttar Pradesh", "count" : 1}
{"_id" : "Noida (U.P)", "count" : 1}
{"_id" : "Noida , Uttar Pradesh", "count" : 1}
{"_id" : "Austin", "count" : 1}
{"_id" : "Sector - 10, Rohini,, Delhi", "count" : 1}
{"_id" : "Sector - 12, Rohini, Delhi", "count" : 1}
{"_id" : "Old Delhi", "count" : 1}
{"_id" : "Muradnagar", "count" : 1}
{"_id" : "ad", "count" : 1}
{"_id" : "Nueva Delhi", "count" : 1}
{"_id" : "Gaziabad", "count" : 1}
{"_id" : "Ghaziabazd", "count" : 1}
{"_id" : "Sahibabad, Ghaziabad", "count" : 1}
{"_id" : "Janakpuri", "count" : 1}
{"_id" : "New delhi", "count" : 2}
{"_id" : "Sector - 28, Rohini, Delhi", "count" : 2}
{"_id" : "NOIDA", "count" : 2}
{"_id" : "Rohtak", "count" : 3}
{"_id" : "Dwarka", "count" : 3}
{"_id" : "Bahadurgarh", "count" : 3}
{"_id" : "delhi", "count" : 4}
... ... ... ...
```

Population Data

A query to retrieve the population data by township didn't result in any data. I suspect this is because no population data has been added by any contributor to date.

Input:

```
db.Delhi.aggregate([{"$match": {"population":{'$exists': 1}}}, {'$group':
{'_id': '$name', 'total': {'$sum': '$population'}}}, {'$sort': {'total': -
1}}, ])
```

Output:

```
{“_id” : “New Delhi”, “total” : 0}
{“_id” : “Faridabad”, “total” : 0}
{“_id” : “Delhi”, “total” : 0}
{“_id” : “Meerut”, “total” : 0}
... ... ... ...
```

I don't blame them for this. Even the census committee finds it hard to provide an accurate report in this regard because of the rising population density in the past few years.

Conclusion

After reviewing the data, it is clear that the OpenStreetMap data for my city isn't complete. Even the data that is populated, isn't very accurate.

Overall, I would be hesitant to rely on this dataset for drawing any general inferences on the total population because it is not yet populated with enough data points as other popular services like Google Maps, Sygic or MapMyIndia. I think it would do OpenStreetMap pretty good to partner with local search and discovery services to improve their OSM data. They can also partner with such services to review or audit their data.

OpenStreetMap should also put a check on the data input by the contributors and validate the inputs. I think checkbox, auto-suggested lists, drop-downs and radio-button inputs would be a better option in some cases rather than text-based input methods as this will restrict the user's input to preset values and hence will improve the overall quality of input received by OpenStreetMap. But, I can see that this is a pretty difficult task gathering the input range and formats of various fields which would differ for various cities, countries and regions throughout the world.

I think it would be an interesting future assignment to compare and plot the population, amenities and overall contributed data of the rural and urban areas of the city. Investigating and analyzing data of this magnitude is a real time-consuming process and requires a lot of patience.

Resources

1. OpenStreetMap – http://www.openstreetmap.org
2. OpenStreetMap Wiki - https://wiki.openstreetmap.org/wiki/Main_Page
3. MongoDB Documentation - http://docs.mongodb.org/manual/ core/document
4. Python Mongo Drivers - http://docs.mongodb.org/ecosystem/ drivers/python