OpenStreetMap Project

Data Wrangling with MongoDB

Map Area: New Delhi, India

INTRODUCTION

In this project, ask was to select any particular area from OpenStreet Maps and explore the data. Post Auditing/ cleansing the data it was required of us to convert the data format into JSON and upload it into MongoDB. The final part consist of exploring the data and doing some analysis, using MongoDB queries.

I would break the problem into following steps and walk you through –

- Selecting and Obtaining the data
- Understand the data, by 'peeking' into it
- Clean the data
- Convert it into JSON and upload it into MongoDB
- Run MongoDB queries to analyze the data

PROCEDURE

STEP 1 – SELECTING AN AREA

I was born and bought up in New Delhi only, thereby selecting this region as basis of my analysis. The domain knowledge helped me in cleaning the data as well.

I used MapZen metro extract to download the ZIPped OSM data

Link to the same is -

STEP 2 – INITIAL PEEKING AND PROBLEMS ENCOUNTERED

To get a sneak peek into data, I wrote a code (Explore_Data.py)

The outputs of this code are the 3 JSON files consisting individual information about Street, City (District) and Zip Codes (Postal code) - (newdelhi_city_data, newdelhi_streets_data, newdelhi_zipcodes_data)

On viewing these 3 files, I observed following anomalies in data -

Cities (Districts) –

1. Incorrect spelling –
    a. Neew Delhi = New Delhi
    b. Gaziabad = Ghaziabad
2. State names are includeed within city –
    a. Noida , Uttar Pradesh = Noida
    b. Ghaziabad, UP, India = Ghaziabad
3. State names are present - Uttar Pradesh, U.P., Haryana
4. Complete names to just district –
    Pratap Colony, Siraspur, Delhi = Siraspur
    Badli Industrial Area, Badli, Delhi = Badli
    Naya Band, Khera = Khera
    Sector- 10, Rohini, Delhi = Rohini
    Sector - 12, Rohini, Delhi = Rohini
    Sector - 17, Rohini, Delhi = Rohini
    Sector - 15, Rohini, Delhi = Rohini
    Sector - 11, Rohini, Delhi = Rohini
    Sector - 28, Rohini, Delhi = Rohini
    Sector - 5, Rohini, Delhi = Rohini
    Hira Colony, Siraspur, Delhi = Siraspur
    West Karawal Nagar, New Delhi = Karawal Nagar

Street Names–

1. Streets are abbreviated –
    a. Rd = Road
    b. St = Street
    c. Nr = Near
    d. NH = National Highway
    e. No = number
    f. Av = Avenue
2. Small / Capital names = Majestic Towers/majestic towers
3. Minor spelling mistakes –
    a. Udhyog = Udyog
    b. G T road = GT road
    c. Pritampura = Pitampura

Zip codes–

1. Length of a pin code is other than standard 6 numbers – 1100049, 10089
2. Alphanumeric pin code- M5G 1C3
3. Non numeric codes - Sunpat House Village
4. Space in between codes- 110 001
5. Misleading – 420420

STEP 3– Data cleansing

Corresponding to above mentioned inconsistencies in Cities, Street and Zip code, I use following approach –

- **City Corrections** – I manually inputted the correct (right hand side) values in the code. Also, wherever only state names are present, they are replaced with blanks
- **Street Corrections-**
  - Mapped the abbreviations and used them in code
  - Manually correct the 3 streets having spelling mistakes
  - For cases where it wasn't clear on what the correct street name should be, I decided to add these to a list and ignore them if I found a matching value.
- **Zip code Correction-** Replaced all numeric code with standard 6 digit codes and remove all non-numeric codes

The above-mentioned corrections along with the requisite data shape can be found under code (Wrangle_Data.py)

STEP 4– Data conversion and insertion into MongoDB

The data is converted to JSON format using code (Wrangle_Data.py) and is pushed into MongoDB by using (insert.py). Here I've used PyMongo package for the same

STEP 5– Writing Queries into MongoDB to analyze the data

Here I wrote a python code (Query.py) to gather information about amenities, nodes, ways, religion, area etc. Also I tried some commands in MongoDB shell as well. I'll be focusing on them in next section of Data Overview

---

Data Overview

The files used for this analysis are –

newdelhi.osm (I've renamed the original file from OS data to this for sake of simplicity) – 598.2 mb

newdelhi.osm.json – 700.1 mb

Some of the queries written in python code (Query.py) and their corresponding output are as follows –

a) Document records related queries-

```
# To count total documents

db.newdelhi.find().count()

-> 3271002


# To find nodes and ways

db.newdelhi.find( {"type":"node"} ).count()

-> 2732250

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

-> 538750

```
# To find unique users

len(db.newdelhi.distinct( "created.user" ) )

-> 759


# To find documents for whom amenity exists

db.newdelhi.count({"amenity" : {"$exists":1}})

-> 3316


   print db.newdelhi.count({"created.version" : "1"})

-> 2788742
```

b) Amenities related queries-

to find top 5 amenities

```
   print list(db.newdelhi.aggregate([{"$match": {"amenity":{"$exists":1}}},
{"$group":{"_id":"$amenity", "count":{"$sum":1}}},{"$sort":{"count":-
1}},{"$limit":5}]))

[{u'count': 879, u'_id': u'school'}, {u'count': 303, u'_id': u'place_of_worship'},
{u'count': 297, u'_id': u'parking'}, {u'count': 198, u'_id': u'fuel'}, {u'count': 177,
u'_id': u'hospital'}]
```

So we have, School – 879, Place of worship – 303, parking – 297, fuel – 198 and

hospital – 177 as top 5 amenities

Though amenities is available for handful of documents but it feels good to see school and hospital among top 5 amenities. In addition, India is known for religion, so it is no surprise to have places of worship as second most amenities

Continuing my exploration with amenity, I wanted to explore top 5 famous schools in New Delhi. Just to share a background, we have many famous schools here having multiple branches in different districts. In addition to private schools, we have state government schools – "Government school" and central government schools – "Kendriya Vidyalaya".

Let's check top 5 schools –

```
print list(db.newdelhi.aggregate([{"$match": {"amenity":"school",
"name":{"$exists":1}}}, {"$group":{"_id":"$name",
"count":{"$sum":1}}},{"$sort":{"count":-1}},{"$limit":5}]))

[{u'count': 5, u'_id': u'Delhi Public School'}, {u'count': 5, u'_id': u'Government
School'}, {u'count': 4, u'_id': u'Kendriya Vidyalaya'}, {u'count': 3, u'_id': u'Modern
School'}, {u'count': 3, u'_id': u'Salwan Public School'}]
```

As expected, Government school and Kendriya Vidayalaya are part of popular schools

New Delhi is famous for its varied variety of food cause we have varied taste buds. This made me to check top 5 cuisines-

Top 5 cuisines

```
 print list(db.newdelhi.aggregate([{"$match": {"amenity":"fast_food",
"cuisine":{"$exists":1}}}, {"$group":{"_id":"$cuisine",
"count":{"$sum":1}}},{"$sort":{"count":-1}},{"$limit":5}]))

[{u'count': 11, u'_id': u'burger'}, {u'count': 8, u'_id': u'indian'}, {u'count': 4, u'_id':
u'pizza'}, {u'count': 3, u'_id': u'sandwich'}, {u'count': 2, u'_id': u'chicken'}]
```

Other Ideas

During initial sneak peak into data (where postal codes JSON is created), I observed ZIP codes outside of New Delhi region. Assuming this data to be of NCR region (includes New Delhi, Gurgaon, Noida, Ghaziabad) it would be interesting to observe that how many of them are only from New Delhi

Background – New Delhi's pin code start with '11'

```
# Total count of postal codes available

    print db.newdelhi.count( {"address.ostcode" : {"$exists":1} } )

-> 610


    # To find count of residents of New Delhi

    # New delhi's area pin code start with "11"

    print db.newdelhi.find( {"address.ostcode" : { "$regex" : "^11.+"} } ).count()

-> 310
```

To find top 5 religions –

```
    # to find top 5 religion

 print list(db.newdelhi.aggregate([{"$match": {"religion":{"$exists":1}}},
{"$group":{"_id":"$religion", "count":{"$sum":1}}},{"$sort":{"count":-
1}},{"$limit":5}]))



[{u'count': 125, u'_id': u'hindu'}, {u'count': 47, u'_id': u'muslim'}, {u'count': 32,
u'_id': u'christian'}, {u'count': 27, u'_id': u'sikh'}, {u'count': 5, u'_id': u'jain'}]
```

Hinduism being most prominent religion in India, so no prize for guessing this result. However the count seems to be quite low. Let's check number of records for which religion field is available

```
print db.newdelhi.count( {"religion" : {"$exists":1} } )

242
```

With recent banking policies, we are witnessing quite a growth in financial sector. It would be interesting case to watch whether public players are taking benefit of it or the private banks are dominant

Top 5 banks

```
 print list(db.newdelhi.aggregate([{"$match": {"amenity":"bank",
"name":{"$exists":1}}}, {"$group":{"_id":"$name",
"count":{"$sum":1}}},{"$sort":{"count":-1}},{"$limit":5}]))

[{u'count': 5, u'_id': u'HDFC'}, {u'count': 5, u'_id': u'ICICI'}, {u'count': 4, u'_id':
u'Citibank'}, {u'count': 4, u'_id': u'HDFC Bank'}, {u'count': 3, u'_id': u'Axis
Bank'}]
```

Unfortunately, none of the top 5 banks belong to public domain

Some final thoughts / additional ideas -

- After running these additional queries, it became clear that amenity record is quite low, contrary to documents available. It could have been more interesting to explore, had there been more amenities
- Telephone number along with other details could have come handy in using these documents (maybe as an api)
- The variety & quantity of documents is huge, so we can potentially use such geo based data for crime against women by tagging location with the safety index for women

Conclusion

It was great activity to clean, convert and analyze the data. From the analysis it became clear that how powerful queries can be. The variety of information included is also great. However, having said so, it would have been better if we could have denser information. Crowdsourcing can be done to include more complete entries. In addition Google API can be applied.

Personally, I think inclusion of data from multiple sources can be much more beneficial as not only it would check veracity of records but also provides quality. For instance, for food related information we can rely on Zomato API. For shops, JustDial API, for school and hospital information – Government database

List of Resources

1. Openstreetmap. Url: http://www.openstreetmap.org
2. Overpass Api. Url: http://wiki.openstreetmap.org/wiki/Overpass– API
3. MongoDB Documentation. Url: http://docs.mongodb.org/manual/core/document/
4. Python Mongo Drivers. Url: http://docs.mongodb.org/ecosystem/drivers/python/
5. Unicode HowTo. Url:https://docs.python.org/2/howto/unicode.html
6. http://stackoverflow.com/questions/30333020/mongodb-pymongo-aggregate-gives-strange-output-something-about-cursor