# Data Wrangling with MongoDB - OpenStreetMap

*Swaroop Oggu*

Map Area: Edison, NJ, United States

*https://www.openstreetmap.org/export#map=12/40.5381/-74.3781*

## 1. Problems Encountered in the Map

After initially downloading a small sample size of the Edison, NJ area and running it against a provisional data.py file, I noticed three main problems with the data, which I will discuss in the following order:

- Inconsistent postal codes (“07095-0887”, “NJ 07095”)
- Street names inconsistent with uppercase / lower case (“NICHOL AVE”, ”St. George”, ”St George’s”)
- Street name abbreviation and spacing issues
- Tag with Key = ‘type’ overriding type attribute

### 1.1. Inconsistent postal codes

Postal codes have a format other than 5 digits like (“07095-0887”, “NJ 07095”) which lead to update of code to strip anything before and after the five digits of postal code.

Also more than 50% of the records have postal codes outside Edison area (088 series), Falling more on Staten Island NY region

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

**Result**

```
{ "_id" : "10312", "count" : 10568 }
{ "_id" : "10309", "count" : 9295 }
```

When tried to check which city 10312 refers to, We found that the city is Staten Island , NY and also most of the records 10566 to be precise have null values as city name

```
> db.openstreetedison.find(
...   {
...     "address.postcode": "10312",
...  "address.city" : { $ne: null }
... }
... ).count()
2
> db.openstreetedison.find(
...   {
...     "address.postcode": "10312",
...  "address.city" : { $ne: null }
... },
...   {
... _id:0,
... 'address.city': 1
...   }
... )
{ "address" : { "city" : "Staten Island" } }
{ "address" : { "city" : "Staten Island" } }
```

Tried with modifying the query to see where both city and postcode are present to see if it returns valid values, Turns out it did

```
> db.openstreetedison.aggregate([
...                   { $match: { "address.city":{$exists:1},"address.postcode":{$exists:1} } },
...                   { $group: { _id: "$address.city", total: { $sum: 1 } } },
...                   { $sort: { total: -1 } }
...                  ])
```

**Result**

```
{ "_id" : "New Brunswick", "total" : 369 }
{ "_id" : "Piscataway", "total" : 258 }
{ "_id" : "Edison", "total" : 53 }
```

```
>    db.openstreetedison.aggregate([
...                   { $match: { "address.postcode":{$exists:1},"address.city":{$exists:1} } },
...                   { $group: { _id: "$address.postcode", total: { $sum: 1 } } },
...                   { $sort: { total: -1 } }
...                  ])
{ "_id" : "08901", "total" : 368 }
{ "_id" : "08854", "total" : 259 }
{ "_id" : "08823", "total" : 42 }
```

## 1.2. Street names uppercase/lowercase issues

Street names when observed in the Edison osm data reflected a varied casing across. Code was updated to handle the casing to be consistent across by applying camel casing

## 1.3. Street name abbreviation/spacing issues

Street names when observed in the Edison osm data reflected a varied abbreviation across nodes and ways E.g.(‘Ave.’ ,’St.’ ,’BLDG’ ,’Pl’). Code was updated to handle the discrepancies. Also few of the street names had spacing issue E.g.(‘ Bayard St’,’ Wood Ave South’) which were resolved by defining a updated value in the dictionary

```
edison_street_corrections = { 'St Georges Ave': "Saint George's Avenue",
           "St George's Ave": "Saint George's Avenue",
           'St. Georges Avenue':"Saint George's Avenue" ,
           "St. George's Avenue":"Saint George's Avenue" ,
           "St. Georges Avenue ":"Saint George's Avenue",
           ' Bayard St': 'Bayard Street',
           ' Wood Avenue South': 'Wood Avenue South',
            'GIBBONS CIRCLE BLDG E':'GIBBONS CIRCLE BUILDING-E',
            'NICHOL AVENUE BLD E' : 'NICHOL AVENUE BUILDING-E'
          }
```

```
mapping = {'Ave': 'Avenue',
           'Ave.': 'Avenue',
           'AVE.':'AVENUE',
           'AVE':'AVENUE',
          'av': 'Avenue',
           'Blvd': 'Boulevard',
           'Blvd.': 'Boulevard',
           'BLD' : 'BUILDING',
           'BLDG' : 'BUILDING',
           'Cir': 'Circle',
           '(NJ)' : 'NJ',
           'Dr': 'Drive',
           'Dr.': 'Drive',
           'Hwy': 'Highway',
           'HWY.': 'Highway',
           'Pl': 'Place',
           'RD': 'Road',
           'Rd': 'Road',
           'Rd.': 'Road',
           'Rt.': 'Route',
           'Rt': 'Route',
           'Road,': 'Road',
           'U.S.':'US',
           'U.S':'US',
           'S.': 'South',
            "W.": "West",
            "N.": "North",
            "S.": "South",
            "E": "East",
           'St': 'Street',
           'St.': 'Street',
           'road': 'Road',
           'st': 'Street',
           'street': 'Street',
           'Pky': 'Parkway',
           'Pkwy': 'Parkway',
           'Plz':'Plaza',
           'Ct':'Court',

           }
```

## 1.4. Tag with Key = ‘type’ overriding type attribute

Edison OSM data has tag elements that have key as ‘Type’ which overlapped with the root element tag ‘Type’ and created records with Type being MultiPolygon , Gas etc, which is incorrect. Data wrangled should only contain node and way elements. Code is updated to ignore the Tag with key = Type

**Problem when identified**

```
db.openstreetedison.distinct("type")
[ "node", "way", "multipolygon", "gas", "Public" ]
db.openstreetedison.count()
515822
> db.openstreetedison.find({type:"node"}).count()
453141
> db.openstreetedison.find({type:"way"}).count()
62596
```

**After fixing the code**

```
db.openstreetedison.distinct("type")
[ "node", "way" ]
> db.openstreetedison.count()
515822
> db.openstreetedison.find({type:"node"}).count()
453141
> db.openstreetedison.find({type:"way"}).count()
62681
```

# 2. Data Overview

Data downloaded was for Edison NJ area. I chose the region as i live close by and know the area around. Data set was downloaded using Overpass API as openstreet export didn't work for the region I queried

## 2.1. Data retrieved on

Aug 29, 11 AM EST

## 2.2. Data co-ordinates

Minimum Latitude: 40.4221
Maximum Latitude: 40.6543
Minimum Longitude: -74.6476
Maximum Longitude: -74.1086

## 2.3. Overpass API query

(node (40.4221,-74.6476,40.6543,-74.1086);<;);out meta

## 2.4. File Size

Input/Initial OSM file for Edison region: 111 MB
Final JSON for Edison region: 155MB

## 2.5. MongoDB Queries

### 2.5.1. Number of total documents

```
> db.openstreetedison.count()

515822
```

### 2.5.2. Number of Nodes

```
> db.openstreetedison.find({type:"node"}).count()

453141
```

### 2.5.3. Number of Ways

```
> db.openstreetedison.find({type:"way"}).count()
62681
```

### 2.5.4. Top contributing user

```
> db.openstreetedison.aggregate([
...                     { $match: { "created.user":{$exists:1} } },
...                     { $group: { _id: "$created.user", total: { $sum: 1 } } },
...                     { $sort: { total: -1 } },
...  { $limit:1}
])
{ "_id" : "ebrelsford_nycbuildings", "total" : 107861 }
```

### 2.5.5. Top sources for Edison maps

```
db.openstreetedison.aggregate([
...                     { $match: { "source":{$exists:1}} },
...                     { $group: { _id: "$source", total: { $sum: 1 } } },
...                     { $sort: { total: -1 } },
                        { $limit:3}
...                    ])
{ "_id" : "NJ2002LULC", "total" : 2088 }
{ "_id" : "Bing", "total" : 1889 }
{ "_id" : "Rutgers", "total" : 616 }
```

### 2.5.6. Top amenities in Edison

```
> db.openstreetedison.aggregate([
...                   { $match: { "amenity":{$exists:1} } },
...                   { $group: { _id: "$amenity", count: { $sum: 1 } } },
...                   { $sort: { count: -1 } },
…                    { $limit:5}
...                  ])
{ "_id" : "parking", "count" : 1057 }
{ "_id" : "school", "count" : 352 }
{ "_id" : "place_of_worship", "count" : 312 }
{ "_id" : "restaurant", "count" : 108 }
{ "_id" : "fast_food", "count" : 67 }
```

### 2.5.7. Top fast food restaurants in Edison

```
> db.openstreetedison.aggregate([
...                   { $match: { "amenity":"fast_food" } },
...                   { $group: { _id: "$name", count: { $sum: 1 } } },
...                   { $sort: { count: -1 } },
...                    { $limit:5}
...                  ])
{ "_id" : "McDonald's", "count" : 12 }
{ "_id" : "Burger King", "count" : 9 }
{ "_id" : "Subway", "count" : 6 }
{ "_id" : "Wendy's", "count" : 6 }
{ "_id" : "Dunkin Donuts", "count" : 6 }
```

### 2.5.8. Metropark Stations in Edison

```
> db.openstreetedison.find( { name: { $regex: /^Metropark.*/ } } ).count()
3
```

### 2.5.9. Top Residential areas in Edison

```
> db.openstreetedison.aggregate([
...                     { $match: {"highway":"residential" ,"name":{$exists:1} }},
...                     { $group: { _id: "$name", total: { $sum: 1 } } },
...                     { $sort: { total: -1 } }
...                    ])
{ "_id" : "Madison Avenue", "total" : 23 }
{ "_id" : "Church Street", "total" : 21 }
{ "_id" : "John Street", "total" : 17 }
```

## 3. Additional Ideas

### 3.1. Other Ideas about the datasets

Since most of the data is user submitted/edited, I believe that validation from actual users for the user submitted data would be a better check point. Some of the ways this can be achieved are outlined below

### 3.1.1. Appify the openstreet data on Mobile

Create an app that pops up the detail of a node/way eg like an amenity in a certain address/location when a user in particular location like (user is at 76 lat and 44 lon) and confirm from the user, if the location has indeed the amenity or service listed in the map,

If user confirms, they can get an incentive from the local business/amenty/service like a coupon or discount. If the validation reaches like 70 percent or so then it can be confirmed that the location is

valid and can be removed from the confirmation list, All the confirmed locations can be tagged or flagged with a diff attribute and indicate to user with start or a flag.

Similarly if a user confirms that the location is not the one listed in update the data in the openstreet data set. This approach will be more active since it's mobile and incentive driven

### 3.1.2. Cross validation

Cross validating the openstreet position data against google/apple maps, Waze, Four Square, Yelp and Facebook. This needs extensive wrangling and grouping on the openstreet data set to work against the disparate sources.

When a user checks in using facebook or foursquare pick the geo coordinates and look up the open street data , if found check if the name matches any detail from facebook or foursquare.

This helps in **accuracy of the data in very short span of time**.

### 3.1.3. Gamification

Gamification can also generate a more user confirmed data set, Active participation can be derived from more game rewards and titles Like EXPLORER, CONQUEROR who confirm and validate most regions. Also region wise competitions can be held to attract more user participation. This approach will be accurate as well but might lead to delay due to niche segment of users engaged

### 3.1.4. Appify the openstreet data on the automobile

This is same as the mobile, but creating the same as an automobile map app will lead to more customer engagement. This also is a niche segment since tech advanced automobiles are pricey

### 3.1.5. Imputing missing values

This approach is dependent on the other sources in the openstreet data which might not be accurate or reliable . This can be done in the current

code like, if address . city is not found see if the city is there in the same node,way ,tag or under different node object like TIGER , NHD,GNIS

## 3.2. Use custom mongodb "_id "

If observed carefully we can notice that the dataset itself has a unique, id attribute as part of the osm , We can use the same to map the _id object attribute in mongodb instead of the mongodb generated value. This involves creation of _id attribute in shape element method. I haven't done this change in the actual code base

So instead of the below mentioned line in the code we can change node[id] to node[_id] and import in mongodb

```
node["id"] = element.attrib["id"]
```

## 3.3. Use valid tags and key names

If observed carefully we can notice that the dataset has some misleading key names that give a different perspective. For E.g. Highway is a key which has values like BusStop, residential etc. If proper keys are used the dataset makes more sense and adds lot of value

## 3.4. Additonal TIGER ,NHD and GNIS JSON objects

Going through the dataset lead to identification of potential JSON object nodes . Edison NJ data set has more data from TIGER , GNIS and NHD. Hence it made sense to group the information under respective nodes, so that it would result in meaningful insights

TIGER NODE

```
"tiger": {
    "name_base": "Library",
    "zip_left": "08820",
    "cfcc": "A41",
    "reviewed": "no",
    "county": "Middlesex, NJ",
    "name_type": "Pl",
    "zip_right": "08820"
  },
```


GNIS NODE


```
  "gnis": {
    "feature_id": "2055407",
    "import_uuid": "57871b70-0100-4405-bb30-
88b2e001a944",
    "reviewed": "no",
    "county_name": "Union"
  },
```

NHD NODE


```
"nhd": {
   "Zip_1": "08854-5602",
   "BldgName_1": "Administrative Services Building
(ASB)",
   "City_1": "Piscataway",
   "CampusCo_1": "20.000000",
   "BldgNum": "3751.000000",
   "AUXCODE": "Acad/Adm",
   "State_1": "NJ",
   "Long_1": "-74.455625",
   "BldgAddr_1": "65 DAVIDSON ROAD",
   "Lat_1": "40.523487"
  },
```


## 3.5. Additonal MongoDB Queries

### 3.5.1. Top Cuisine in Edison

```
> db.openstreetedison.aggregate([
...                   { $match: { "amenity":"restaurant" ,"cuisine":{$exists:1}} },
...                   { $group: { _id: "$cuisine", total: { $sum: 1 } } },
...                   { $sort: { total: -1 } }
...                  ])
{ "_id" : "pizza", "total" : 9 }
{ "_id" : "american", "total" : 9 }
{ "_id" : "burger", "total" : 4 }
{ "_id" : "italian", "total" : 4 }
{ "_id" : "chinese", "total" : 4 }
{ "_id" : "indian", "total" : 3 }
```

### 3.5.2. Shops by count in Edison

```
> db.openstreetedison.aggregate([
...                     { $match: { "shop":{$exists:1},"name":{$exists:1} } },
...                     { $group: { _id: "$name", total: { $sum: 1 } } },
...                     { $sort: { total: -1 } }
...                   ])
{ "_id" : "ShopRite", "total" : 7 }
{ "_id" : "Home Depot", "total" : 5 }
{ "_id" : "Walmart", "total" : 5 }
{ "_id" : "Target", "total" : 5 }
{ "_id" : "Wawa", "total" : 4 }
{ "_id" : "Costco", "total" : 3 }
```

## 4. Conclusion

After a well formed JSON structure by data wrangling and deep analysis of the Edison data set for the open street data, It can be concluded that data is incomplete and inconsistent across elements, tags and nodes. Eg if there is city , there is no poscode . Although its incomplete the data set has provided nearly approximate results when queried considering the unknown or null values , Like querying with both city and post code present revealed that New Brunswick is the Top City near by Edison