OpenStreetMap Data Wrangling with MongoDB

Zanete Ence

Map Area: Crawley, West Sussex, United Kingdom

Cleaning

Audit - iteration #1

Initial look at downloaded data sample suggests a certain structure. I've made a note of some assumptions about it in Appendix along with a report of a data audit which confirms that the data set roughly confirms.

Audit - iteration #2

In iteration 2 I focused on the most rich yet unstructured content, found in the tag elements. After inspecting an export of all key value pairs, I found quite a few issues. Below are a those which I'm addressing as part of the cleaning process:

- Parts of address are scattered around under different keys, not just under the 'addr' namespace, e.g. in keys such as 'street', 'is_in', 'postal_code', 'name'
- Numeric values followed by units vs not, e.g. "maxspeed": ['10 mph', '90 mph', '97'], "min_height: ['144m','35', '75'], "maxheight": ['10\'6"','17\'0"','4.8']
- Lowercase and uppercase mixed in keys and values, duplicating meaning, .e.g "fixme" and "FIXME", "CRAWLEY", "crawley", "Crawley"

Cleaning and Shaping

Address

I created a mapping of the different tag keys to a unified address and contact information format, so that values from tags with the same meaning would be saved in the same field in the database, e.g. 'addr:street' and 'street' both end up being saved as "address.street", "contact:phone" and "phone" - as "contact.phone", etc.

Numerical values

To handle the nonuniformity of speed, weight and length values, I created a set of lists comprising of keys from tags that contained these values. During the cleaning process I identified values that needed treatment and converted all to one specific unit and store just the numerical value, e.g. "length": "145km" became "length" : 14,500 (in meters), etc.

Other

Wherever possible I converted an encountered date or datetime value in any format (such as "1848" or "2011-04-07T11:47:23Z") to isoformat, t.i: "2016-12-05T23:36:19". In addition, I converted tags of flags with values such as "yes" or "no" to boolean True or False.

Exploration

File sizes

crawley.osm - 58.1 MB, data.json - 49.9 MB

```
# Total documents
db.map_elements.find().count()

# Nodes, Ways, Relations
db.map_elements.aggregate(
  { $group:
    {"_id": "$element_type","count": {"$sum": 1} }
},
  { $sort: { "count": -1 }})
```

```
>>> 289,897

>>> { "_id" : "node",
    "count" : 258241.0
}{
    "_id" : "way",
    "count" : 31218.0
}{
    "_id" : "relation",
    "count" : 438.0}
```

```
# Unique Users
db.map_elements.distinct("uid").length

# Oldest record
db.map_elements.find({},
  {"_id": 0, "created": 1}).sort({"created":
1}).limit(1)
```

```
>>> 442

>>> {"created" : "2006-02-07T03:28:47"}
```

Other Analysis

1. How many and what types of leisure spots are around?

```
db.map_elements.find({"leisure":
{"$exists":1}}).count()

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

```
>>> 354


>>>_id                count
pitch                156
playground           66
garden               44
park                 39
swimming_pool        13
golf_course          13
recreation_ground    9
sports_centre        6
common               3
fitness_centre       1
maze                 1
stadium              1
fishing              1
slipway              1
```

2. How many and what types of restaurants or cafes?

```
db.map_elements.aggregate(
  { $match:
    { $or: [
      { $text:
```

```
>>>_id               count
restaurant           59
fast_food            46
cafe                 35
```

```
        { $search: "cafe restaurant pub bar",
          $caseSensitive: false }},
      { "amenity":
        {"$in":
          ["cafe", "restaurant", "fast_food",
"bar"]} },
      {"cuisine": {$exists: 1}}
    ]}},
    { $group:
      {"_id": "$amenity","count": {"$sum": 1} }
    },
    { $sort: { "count": -1 }})
```

```
bar                     9
parking                 1
```

3. What gyms are in the area?

```
db.map_elements.find(
  { $or: [
    { $text: { $search: "gym fitness",
$caseSensitive: false }},
    { "leisure": {$in: ['fitness_centre',
'sports_centre']}}]},
  {"_id":0, "name": 1, "leisure":1, "element_type":
1}).sort({"name": 1})
```

```
>>> { "leisure" : "sports_centre",
    "element_type" : "node"
}{ "leisure" : "sports_centre",
    "element_type" : "way"
}{ "leisure" : "sports_centre",
    "element_type" : "way"
}{ "leisure" : "sports_centre",
    "element_type" : "way",
    "name" : "K2 Leisure Centre"
}{ "element_type" : "node",
    "name" : "Kwik-fit"
}{ "element_type" : "way",
    "name" : "LA Fitness"
}{ "leisure" : "sports_centre",
    "element_type" : "node",
    "name" : "Nuffield Health"
}{ "leisure" : "sports_centre",
    "element_type" : "node",
    "name" : "The Gym"
}{ "leisure" : "fitness_centre",
    "element_type" : "way",
    "name" : "Virgin Active"}
```

Further Data Corrections

Some examples of further data cleaning ideas:

- The address field mapping should be reviewed as one of the mapped fields to “city” is producing unwanted results. Quite a few records hold a city value that is, in fact, the county. The values in ‘is_in’ tags need closer inspection before an appropriate field can be determined

```
db.map_elements.find(
  {
    "type": "node",
    "address.city": {"$exists":1},
    "address.postcode":{"$exists":0}
  }, {
    "_id": 0,
    "name": 1,
    "address": 1
  })
```

```
>>> { "address" : {"city" : "West Sussex, England,
UK"},
    "name" : "Crawley"
}{
    "address" : {"city" : "Sussex, England"},
    "name" : "Handcross"
}{
    "address" : {"city" : "West Sussex, England,
UK"},
    "name" : "Pound Hill"
}{
    "address" : {"city" : "West Sussex, England,
UK"},
    "name" : "Three Bridges"}
```

- Phone numbers in different formats, with and without international calling code, with and without spaces, even arrays with extra text information. However, there are only 36 records containing phone numbers, therefore, perhaps not the most impactful problem to solve.

```
db.map_elements.find({
    "type": "node",
    "contact.phone": {"$exists":1}}).count()

db.map_elements.find(
  {
    "type": "node",
    "contact.phone": {"$exists":1}
  }, {
    "_id": 0,
    "name": 1,
    "contact.phone": 1
  })
```

```
>>> 36


>>> { "contact" : {"phone" : "01293 852146"}
}{ "contact" : {"phone" : "0345 671 9251"}
}{ "contact" : {"phone" : "+44 1293 579199"}
}{ "contact" : {"phone" : "0844 477 5115"}
}{ "contact" : {"phone" : [
            "Reservations 01293512666",
            "Take Away 01293518118"]}}
```

- As data is submitted by users, the same or similar entities have been added many times with different names (as abbreviations, mixed case letters, etc). However, this kind of cleaning would be a fairly large manual undertaking as there's no easy way to automate.

```
db.map_elements.find(
  { $text: { $search: "Lloyds -pharmacy sainsbury
-petrol costa", $caseSensitive: false }},
  {"_id": 0, "name": 1}
).sort({"name": 1})
```

```
>>> { "name" : "Costa"
}{ "name" : "Costa"
}{ "name" : "Costa Coffee"
}{ "name" : "Lloyds"
}{ "name" : "Lloyds"
}{ "name" : "Lloyds Bank"
}{ "name" : "Sainsbury Supermarket"
}{ "name" : "Sainsbury's West Green"
}{ "name" : "Sainsburys"}
```

- Standardising the opening hours which currently come in varying formats, e.g. "opening_hours": ['00:00-24:00','05:00-00:00','08:00-dusk', '24/7', 'Mo 11:00-19:30; Tu-Su 10:00-18:30']

Further Data Analysis

1. How many and what type of Schools are in the area?

```
db.map_elements.find(
  { "element_type": "node", "amenity":
"school"}).count()
```

```
>>> 4
```

That's an unexpectedly low number as while I was looking at the data manually, I saw a lot more elements with the word School in the contents.

```
db.map_elements.find({ $text:
    { $search: "school", $caseSensitive: false
}}).count()
```

```
>>> 80
```

That's a lot more, however, when inspecting the values in more detail, most of the elements were ways, and quite a few were present twice, once as a node and once as a way

```
db.map_elements.find(
  { $text:
    { $search: "school", $caseSensitive: false
}},
  {"_id":0, "name": 1, "amenity":1,
"element_type": 1}
).sort({"name": 1})
```

```
>>> { "element_type" : "node",
    "name" : "Handcross Primary School"
}{ "amenity" : "school",
    "element_type" : "way",
    "name" : "Handcross Primary School"
}{ "element_type" : "node",
    "name" : "Hazelwick School"
}{ "amenity" : "school",
    "element_type" : "way",
```

	<pre>"name" : "Hazelwick School"
	}{
	"amenity" : "school",
	"element_type" : "way",
	"name" : "Hilltop Primary School"
	}{
	"element_type" : "way",
	"name" : "Hilltop Primary School"
	}...</pre>

In addition, the only indication of the type of school it is, e.g. Primary, Secondary etc, is usually in the name, if at all. To fully answer the question, would require a more data pre-processing on these records and adding another field such as “school_type” or similar.

Further Data Improvements

As I live in the area, I’m realising how incomplete the data is. The gym I go to isn’t found among the results or anywhere in the original xml file. In fact, none of their neighbouring businesses, e.g. Tesla store or even the street isn’t mentioned. Similarly, a couple of the listed gyms have been closed for more than a year. I suspect, the situation is similar with other institutions such as restaurants, bars and leisure spots. This means that no matter how much data cleansing on this data set I would do, it would be very difficult, if not impossible, to get reliable answers to questions about places in the area.

One possible way to get a more accurate picture would be to obtain location data available in Google maps via their Google Places API web service. A new script could:

- poll the Nearby Search web service endpoint for a list of places for a given lat/lon,
- transform to OpenStreetMap format
- submit nodes to OpenStreetMap API

This solution would require quite a bit of coding, however, the as a result OpenStreetMap data would be a lot more complete, enable better analysis and benefit other OSM users.

However, there are a a few limitations and things to consider:

- Google APIs have usage restrictions - only 1000 free requests for every 24h. That would need to be worked around by throttling requests or choosing to pay.
- Google often update / deprecate their APIs so the web service might become unavailable or stop working as expected.
- Some additional effort would be required if we wanted to minimise place duplication as it’s quite tricky to determine if a certain place already exists in OpenStreetMap and might just need an update.
- The places that no longer exist in reality would still be part of OSM, addressing this issue would need additional work to explicitly remove places that don’t show up on Google.
- This suggestion enables supplementing OSM data with node elements only, not relations or ways.

Appendix

Schema Rules

Rules that script in audit.py is verifying and producing a report of any irregularities.

- There are 3 types of main elements: **nodes**, **ways** and **relations**
- All of these have a set of mandatory attributes, for brevity I’ll focus on a set of these:
  - **id** - always numeric, int, e.g. 527862206
  - **uid** - always numeric, e.g 19799
  - **user** - text, e.g. tilsch

    - **timestamp** - timestamp "2008-02-09T11:34:42Z"
- **nodes** have an additional couple of mandatory attributes:
    - **lat** - always numeric, float, e.g. 51.1323794
    - **lon** - always numeric, float, e.g. -0.1598410
- All main elements have a child element **tag** with two mandatory attributes:
    - **k** - any value, e.g. highway
    - **v** - any value, e.g. crossing
- **ways** must have at least a couple of child **nd** elements with one mandatory attribute:
    - **ref** - always numeric, reference to an existing node, e.g. 4214523323
- **relations** must have at least a couple of child **member** elements with three mandatory attributes:
    - **ref** - always numeric, reference to an existing node, e.g. 4350382290
    - **role** - text, can be blank, e.g. stop
    - **type** - text, one of [node, way, relation], e.g. way

Report

The outcome of running audit.py on data in original input file crawley.osm

```
{
    'member': {'count': 20885, 'errors': []},
    'nd': {'count': 332470, 'errors': []},
    'node': {'count': 258242, 'errors': []},
    'relation': {'count': 438,
                 'errors': ['Not enough member elements within relation 5220',
                            'Not enough member elements within relation 240194',
                            'Not enough member elements within relation 329027',
                            'Not enough member elements within relation 330274',
                            'Not enough member elements within relation 330287',
                            'Not enough member elements within relation 330290',
                            'Not enough member elements within relation 330300',
                            'Not enough member elements within relation 330337',
                            'Not enough member elements within relation 330341',
                            'Not enough member elements within relation 1253759',
                            'Not enough member elements within relation 2526064',
                            'Not enough member elements within relation 2975573',
                            'Not enough member elements within relation 5311323',
                            'Not enough member elements within relation 5506435',
                            'Not enough member elements within relation 5506436',
                            'Not enough member elements within relation 5563948',
                            'Not enough member elements within relation 5767790',
                            'Not enough member elements within relation 5862634',
                            'Not enough member elements within relation 7334649']},
    'tag': {'count': 101730, 'errors': []},
    'unsuported_elements': defaultdict(<function <lambda> at 0x7f2f4e951c08>, {'note': 1, 'meta': 1,
    'osm': 1, 'bounds': 1}),
    'way': {'count': 31218, 'errors': []}
}
Total count: 744987
```