

OpenStreetMap Data Wrangling

Luiz Schiller

Map Area: Rio de Janeiro, Brazil

- https://mapzen.com/data/metro-extracts/metro/rio-de-janeiro_brazil/ (https://mapzen.com/data/metro-extracts/metro/rio-de-janeiro_brazil/)

This area contains three cities that had a great part in my history. I lived about one third of my life on each: Petrópolis (where I was born), Niterói and Rio de Janeiro. That said, I would like to explore this extract a little bit and see what interesting data I can find.

Problems Encountered in the Map

After the initial cleaning on the data from the downloaded xml file, it was imported into mongodb using the following command:

```
mongoimport --db osm --collection rio --file rio-de-janeiro_brazil.osm.json
```

Analyzing a sample of the data, some problems showed up:

- Tags with k="type";
- String 'bicycle_parking' capacities
- pprint.pprint method not printing Unicode characters properly

Tags with k="type"

Second level 'k' tags with the value 'type' were overriding the node['type'] field, which should equal 'node' or 'way' only. These tags were mapped to the node with the 'type_tag' key before being imported to mongodb.

String 'bicycle_parking' capacities

Nodes representing bicycle parkings had their capacity fields as strings, which did not allow numeric operations I was willing to make with them. All of them represented numbers, except for one '§0' value. To solve this, I iterated over the mongodb collection, updating the values with the parsed integer values. Whenever the parsing failed, the 'capacity' field was removed. The code used for the removal is shown below:

```
In [1]: from pymongo import MongoClient

def get_db(db_name):
    client = MongoClient('localhost:27017')
    db = client[db_name]
    return db

db = get_db('osm')

nodes = db.rio.find({'amenity': 'bicycle_parking', 'capacity': {'$exists': 1}})

for node in nodes:
    try:
        capacity = int(node['capacity'])
        db.rio.update_one(
            {'_id': node['_id']},
            {'$set': {'capacity': capacity}})
    except ValueError:
        db.rio.update_one(
            {'_id': node['_id']},
            {'$unset': {'capacity': 1}})
```

pprint.pprint method not printing Unicode characters properly

This problem is not related to the data itself, but it was harming the wrangling process. When printing the results of some queries with the pprint.pprint method, characters out of the ascii table showed as their Unicode representation, making it hard to read. To solve this I had to instantiate my own printer, witch encoded unicode objects to utf-8, making it possible to read. Check the code below:

```
In [2]: import pprint

class MyPrettyPrinter(pprint.PrettyPrinter):
    def format(self, object, context, maxlevels, level):
        if isinstance(object, unicode):
            return (object.encode('utf8'), True, False)
        return pprint.PrettyPrinter.format(self, object, context, maxlevels, level)
```

Data Overview

This section contains basic statistics about the dataset and the MongoDB queries used to gather them.

File Sizes

```
rio-de-janeiro_brazil.osm ..... 323 MB
rio-de-janeiro_brazil.osm.json ..... 369 MB
```

```
In [3]: db.rio.find().count()
```

```
Out[3]: 1737174
```

Nodes Count

```
In [4]: # node count
        db.rio.find({'type': 'node'}).count()
```

```
Out[4]: 1550716
```

Ways Count

```
In [5]: # way count
        db.rio.find({'type': 'way'}).count()
```

```
Out[5]: 186458
```

Number of Distinct Users

This query uses the following 'aggregate' method:

```
In [6]: def aggregate(db, pipeline):
        return [doc for doc in db.rio.aggregate(pipeline)]

        distinct_users = [
            {'$group': {'_id': '$created.user'}},
            {'$group': {'_id': 'Distinct users:', 'count': {'$sum': 1}}}]
        result = aggregate(db, distinct_users)
        MyPrettyPrinter().pprint(result)

        [{_id: Distinct users:, count: 1239}]
```

Top 10 Contributing Users

```
In [7]: top_10_users = [
        {'$group': {'_id': '$created.user', 'count': {'$sum': 1}}},
        {'$sort': {'count': -1}},
        {'$limit': 10}]
result = aggregate(db, top_10_users)
MyPrettyPrinter().pprint(result)

[{'_id': Alexandrecw, count: 374621},
 {'_id': ThiagoPv, count: 186562},
 {'_id': smaprs_import, count: 185690},
 {'_id': AlNo, count: 169678},
 {'_id': Import Rio, count: 85129},
 {'_id': Geaquinto, count: 69987},
 {'_id': Nighto, count: 63148},
 {'_id': Thundercel, count: 55004},
 {'_id': Márcio Víncius Pinheiro, count: 35985},
 {'_id': smaprs, count: 31507}]
```

Users Appearing Only Once

```
In [8]: users_appearing_once = [
        {'$group': {'_id': '$created.user', 'count': {'$sum': 1}}},
        {'$group': {'_id': '$count', 'num_users': {'$sum': 1}}},
        {'$sort': {'_id': 1}},
        {'$limit': 1}]
result = aggregate(db, users_appearing_once)
MyPrettyPrinter().pprint(result)

[{'_id': 1, num_users: 274}]
```

Additional Ideas

Variety.js

The open-source tool Variety (<https://github.com/variety/variety>) allows the user get a sense of how the data is structured in a MongoDB schema. It does so by showing the number of occurrences for each key on documents returned by a query. It is a useful ally when analysing datasets like Open Street Map, which does not define an allowed key set.

Most Common Amenities

```
In [9]: most_common_amenities = [
        {'$match': {'amenity': {'$exists': 1}}},
        {'$group': {'_id': '$amenity', 'count': {'$sum': 1}}},
        {'$sort': {'count': -1}},
        {'$limit': 10}]
result = aggregate(db, most_common_amenities)
MyPrettyPrinter().pprint(result)

[{'_id': 'school', 'count': 1818},
 {'_id': 'bicycle_parking', 'count': 1409},
 {'_id': 'restaurant', 'count': 1080},
 {'_id': 'parking', 'count': 976},
 {'_id': 'fast_food', 'count': 890},
 {'_id': 'fuel', 'count': 678},
 {'_id': 'place_of_worship', 'count': 562},
 {'_id': 'bank', 'count': 534},
 {'_id': 'pub', 'count': 400},
 {'_id': 'pharmacy', 'count': 368}]
```

Statistics on Bike Parking Capacity

```
In [10]: bike_parkings_capacity = [
        {'$match': {'amenity': 'bicycle_parking', 'capacity': {'$exists': 1}}},
        {'$group': {
            '_id': 'Bike parking stats:',
            'count': {'$sum': 1},
            'max': {'$max': '$capacity'},
            'min': {'$min': '$capacity'},
            'avg': {'$avg': '$capacity'}}}]
result = aggregate(db, bike_parkings_capacity)
MyPrettyPrinter().pprint(result)

[{'_id': 'Bike parking stats:',
  'avg': 11.487840825350037,
  'count': 1357,
  'max': 700,
  'min': 1}]
```

10 Most Common Cuisines

```
In [11]: top_10_cuisines = [
    {'$match': {'amenity': 'restaurant', 'cuisine': {'$exists': 1}}},
    {'$group': {'_id': '$cuisine', 'count': {'$sum': 1}}},
    {'$sort': {'count': -1}},
    {'$limit': 10}]
result = aggregate(db, top_10_cuisines)
MyPrettyPrinter().pprint(result)

[{'_id': 'pizza', 'count': 88},
 {'_id': 'regional', 'count': 83},
 {'_id': 'japanese', 'count': 38},
 {'_id': 'italian', 'count': 38},
 {'_id': 'steak_house', 'count': 20},
 {'_id': 'barbecue', 'count': 18},
 {'_id': 'brazilian', 'count': 16},
 {'_id': 'international', 'count': 12},
 {'_id': 'chinese', 'count': 8},
 {'_id': 'seafood', 'count': 8}]
```

10 Most Common Religions

```
In [12]: most_common_religions = [
    {'$match': {'amenity': 'place_of_worship', 'religion': {'$exists': 1}}},
    {'$group': {'_id': '$religion', 'count': {'$sum': 1}}},
    {'$sort': {'count': -1}},
    {'$limit': 10}]
result = aggregate(db, most_common_religions)
MyPrettyPrinter().pprint(result)

[{'_id': 'christian', 'count': 495},
 {'_id': 'spiritualist', 'count': 7},
 {'_id': 'jewish', 'count': 6},
 {'_id': 'buddhist', 'count': 3},
 {'_id': 'religion_of_humanity', 'count': 1},
 {'_id': 'umbanda', 'count': 1},
 {'_id': 'macumba', 'count': 1},
 {'_id': 'muslim', 'count': 1},
 {'_id': 'seicho_no_ie', 'count': 1}]
```

The vast majority is christian. Among them, which are the most common denominations?

```
In [13]: christian_denominations = [
    {'$match': {'amenity': 'place_of_worship', 'religion': 'christian', 'denomination': {'$exists': 1}}},
    {'$group': {'_id': '$denomination', 'count': {'$sum': 1}}},
    {'$sort': {'count': -1}},
    {'$limit': 10}]
result = aggregate(db, christian_denominations)
MyPrettyPrinter().pprint(result)

[{'_id': catholic, count: 157},
 {'_id': baptist, count: 33},
 {'_id': roman_catholic, count: 31},
 {'_id': evangelical, count: 27},
 {'_id': spiritist, count: 20},
 {'_id': pentecostal, count: 19},
 {'_id': protestant, count: 14},
 {'_id': methodist, count: 10},
 {'_id': presbyterian, count: 3},
 {'_id': assemblies_of_god, count: 2}]
```

Fast-food Sites Near the Sugar Loaf

Consider you are visiting the Sugar Loaf in Rio and suddenly you are starving! Where to go? MongoDB Geospatial Index to the rescue!

```
In [14]: from pymongo import GEOS2D

db.rio.create_index([('pos', GEOS2D)])

sugar_loaf = db.rio.find_one({'name': 'Pão de Açúcar', 'tourism': 'attraction'})

result = db.rio.find(
    {'pos': {'$near': sugar_loaf['pos']}, 'amenity': 'fast_food'},
    {'_id': 0, 'name': 1, 'cuisine': 1}).skip(1).limit(3)

MyPrettyPrinter().pprint([item for item in result])

[{'cuisine': corn, name: Tino},
 {'cuisine': sandwich, name: Max},
 {'cuisine': popcorn, name: França}]
```

Luckily there are Tino's corn, Max's sandwich and França's popcorn to satisfy your hunger!

Conclusion

Data inserted by humans is almost certain to show inconsistencies. And even though a big part of it is inserted by bots, different bots may insert data using different patterns, and the inconsistency remains. On the other hand, this freedom on the data input grants a lot of flexibility to users, and because of that, the representation of the map may be even more faithful to the real world than if there were key constraints or limitations.

Anyway, for the purposes of this wrangling exercise the data has been well cleaned.

References:

pprint Unicode

- <http://stackoverflow.com/questions/10883399/unable-to-encode-decode-pprint-output>
(<http://stackoverflow.com/questions/10883399/unable-to-encode-decode-pprint-output>)

MongoDB Geospatial Index

- <https://docs.mongodb.com/v3.2/tutorial/build-a-2d-index/>
(<https://docs.mongodb.com/v3.2/tutorial/build-a-2d-index/>)
- <https://docs.mongodb.com/v3.2/tutorial/query-a-2d-index/>
(<https://docs.mongodb.com/v3.2/tutorial/query-a-2d-index/>)
- http://api.mongodb.com/python/current/api/pymongo/collection.html?_ga=1.25837502.2095208423.1476211996#pymongo.collection.Collection.create_index
(http://api.mongodb.com/python/current/api/pymongo/collection.html?_ga=1.25837502.2095208423.1476211996#pymongo.collection.Collection.create_index)

Variety Open Source Tool

- <https://github.com/variety/variety> (<https://github.com/variety/variety>)