# P3 Wrangle OpenStreeMap data for San Francisco with MongoDB

March 22, 2017

**By: Piyush Neupane**

**Data Analyst NanoDegree**

---

**Map Area:** San Francisco Metro, California, United States
Link to MapZen Metro Extracts page for San Francisco.

---

## 1 Problems Encountered in the Map data

During the Audit of the OpenStreetMap data, following problems were encountered:

1. Inconsistent address types: > Address values can have more than more variations, including abbreviations. For example, Address can either end with Road or Rd.
2. Inconsistent county value: > County names are either stand-alone or might be followed by the state abbreviation after a comma.
3. Inconsistent zip-code values: > Zip codes or Postal Codes are sometimes concatenated together as a single string. They're usually separated by semicolon(;). For example: 94605; 94619
4. Keys prefixed with either 'tiger', 'gnis' or no prefix at all: > Fields representing similar concepts might have different prefixes.

### 1.1 FIXES PERFORMED

**Perform Street update** Different variations of street types were standardized by building a mapping between non-standard values and desired values. Then the non-standard values (such as Rd, St, Ave, Blvd, etc.) were replaced with standard values (Road, Street, Avenue, Boulevard, respectively).

**Perform county names update** County Names sometimes were followed by the state abbreviation (CA is this case). So, the state abberviation was removed from the county name.

**Perform zip code update** Some zip-code values were concatenated together separated by a semi-colon. For these cases, first zip-code in the string seemed like the primary zip. So, only the first zip-code was retained and remaining was discarded.

**Group all address-related values** All fields related to address were lumped together in a nested dictionary 'address' to be later uploaded into the database.

**Loading the cleaned data into MongoDB** After cleaning up the data in Python, it is saved in JSON format. This file is then uploaded to MongoDB. Here is the MongoDB instance:

```
In [2]: import pprint

        # Get db function
        def get_db(db_name):
            from pymongo import MongoClient
            client = MongoClient("mongodb://localhost:27017")
            db = client[db_name]
            return db

        db = get_db('osm_col')
```

---

## 2 Data Overview

This part lists some basic statistics of the OSM data uploaded into MongoDB:

**File size statistics**

```
san-francisco_california.osm --> 1.17 GB
sf_sample.osm --> 119 MB
sf_sample.json --> 137 MB
```

** Total number of Documents **

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

Out[3]: 1265388
```

** Total number of nodes **

```
In [4]: db.osm_col.find({"type": "node"}).count()

Out[4]: 1129186
```

** Total number of ways **

```
In [5]: db.osm_col.find({"type": "way"}).count()

Out[5]: 136172
```

**Top 10 Users with most contributions**

```
In [6]: def make_pipeline():
            # complete the aggregation pipeline
            pipeline = [
                {"$group"    :   {"_id"     :   "$created.user",
                                  "count"     :   {"$sum" :   1}
                                 } },
                {"$sort"     :   {"count"     :   -1} },
                {"$limit"    :   10}
                ]
            return pipeline

        def aggregate(db, pipeline):
            return [doc for doc in db.osm_col.aggregate(pipeline)]

        if __name__ == '__main__':
            db = get_db('osm_col')
            print 'db: ',db
            print 'osm collection :',db.osm_col
            print '---------------------------------------'
            pipeline = make_pipeline()
            result = aggregate(db, pipeline)
            #print result
            pprint.pprint(result)
```

```
db:  Database(MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=F
osm collection : Collection(Database(MongoClient(host=['localhost:27017'], document
---------------------------------------
[{u'_id': u'ediyes', u'count': 183580},
 {u'_id': u'Luis36995', u'count': 142022},
 {u'_id': u'andygol', u'count': 128228},
 {u'_id': u'RichRico', u'count': 79884},
 {u'_id': u'Rub21', u'count': 78942},
 {u'_id': u'dannykath', u'count': 71492},
 {u'_id': u'calfarome', u'count': 37134},
 {u'_id': u'oldtopos', u'count': 33742},
 {u'_id': u'KindredCoda', u'count': 29948},
 {u'_id': u'karitotp', u'count': 26924}]
```

**Top 10 most commonly occuring Street Addresses**

```
In [7]: def make_pipeline():
            # complete the aggregation pipeline
            pipeline = [
                {"$match"    :   {"address.street" : {"$ne" : None} } } ,
                {"$group"    :   {"_id"     :   "$address.street",
                                  "count"     :   {"$sum" :   1}
```

```
                                } },
                    {"$sort"    :   {"count"    :   -1} },
                    {"$limit"   :   10}
                    ]
        return pipeline

    def aggregate(db, pipeline):
        return [doc for doc in db.osm_col.aggregate(pipeline)]

    if __name__ == '__main__':
        db = get_db('osm_col')
        pipeline = make_pipeline()
        result = aggregate(db, pipeline)
        #print result
        pprint.pprint(result)
```

```
[{u'_id': u'Irving Street', u'count': 164},
 {u'_id': u'9th Avenue', u'count': 116},
 {u'_id': u'El Camino Real', u'count': 96},
 {u'_id': u'Page Street', u'count': 94},
 {u'_id': u'Broadway', u'count': 94},
 {u'_id': u'14th Avenue', u'count': 84},
 {u'_id': u'12th Avenue', u'count': 82},
 {u'_id': u'11th Avenue', u'count': 82},
 {u'_id': u'Church Street', u'count': 78},
 {u'_id': u'10th Avenue', u'count': 76}]
```

**Which street has most number of selected amenity types?**

```
In [8]: def make_pipeline():
            # complete the aggregation pipeline
            pipeline = [
                {"$match"   :   {
                                "amenity" :{"$in": ["restaurant","place_of_worship"
                                "address.street" : {"$ne" : None}
                                }
                },
                {"$group"   :   {"_id"    :   {"amenity": "$amenity", "street" : "$
                                 "count"    :   {"$sum" :   1}
                                } },
                {"$sort"    :   {"_id.amenity": 1, "count" : -1}},
                {"$group"   :   {"_id" : "$_id.amenity",
                                 "address" : {"$first" : "$_id.street"},
                                 "count" : {"$first" : "$count"},

                                }}
                ]
```

```
        return pipeline

    def aggregate(db, pipeline):
        return [doc for doc in db.osm_col.aggregate(pipeline)]

    if __name__ == '__main__':
        db = get_db('osm_col')
        pipeline = make_pipeline()
        result = aggregate(db, pipeline)
        pprint.pprint(result)
```

```
[{u'_id': u'school', u'address': u'19th Avenue', u'count': 4},
 {u'_id': u'restaurant', u'address': u'El Camino Real', u'count': 12},
 {u'_id': u'place_of_worship', u'address': u'Ralston Avenue', u'count': 2},
 {u'_id': u'fast_food', u'address': u'Haight Street', u'count': 6},
 {u'_id': u'cafe', u'address': u'Irving Street', u'count': 8},
 {u'_id': u'bar', u'address': u'19th Street', u'count': 4}]
```

**What are the most commonly occuring building-levels**

```
In [9]: def make_pipeline():
            # complete the aggregation pipeline
            pipeline = [
                {"$match"   :   {"building:levels" : {"$ne" : None} } } ,

                {"$group"   :   {"_id"    :   "$building:levels",
                                 "count"    :   {"$sum" :   1}
                                } },
                {"$sort"    :   {"count"    :   -1} },
                {"$limit"   :   10}
                ]
            return pipeline

        def aggregate(db, pipeline):
            return [doc for doc in db.osm_col.aggregate(pipeline)]

        if __name__ == '__main__':
            db = get_db('osm_col')
            pipeline = make_pipeline()
            result = aggregate(db, pipeline)
            #print result
            pprint.pprint(result)
```

```
[{u'_id': 2, u'count': 138},
 {u'_id': 3, u'count': 116},
 {u'_id': 4, u'count': 76},
 {u'_id': 1, u'count': 72},
```

```
 {u'_id': 5, u'count': 24},
 {u'_id': 7, u'count': 16},
 {u'_id': 10, u'count': 14},
 {u'_id': 6, u'count': 12},
 {u'_id': 9, u'count': 10},
 {u'_id': 8, u'count': 8}]
```

**How many stories in average does each type of building have?**

```
In [10]: def make_pipeline():
             # complete the aggregation pipeline
             pipeline = [
                 {"$match"    :    {"building:levels" : {"$gt" : 0},
                                    "building" : {"$nin": [None,"Yes"]}
                                   } } ,


                 {"$group"    :    {"_id"     :    "$building",
                                    "avg"     :    {"$avg" :  "$building:levels"}
                                   } },
                 {"$sort"     :    {"avg"     :    -1} }
                 ]
             return pipeline

         def aggregate(db, pipeline):
             return [doc for doc in db.osm_col.aggregate(pipeline)]

         if __name__ == '__main__':
             db = get_db('osm_col')
             pipeline = make_pipeline()
             result = aggregate(db, pipeline)
             #print result
             pprint.pprint(result)
```

```
[{u'_id': u'hotel', u'avg': 36.0},
 {u'_id': u'office', u'avg': 9.0},
 {u'_id': u'dormitory', u'avg': 9.0},
 {u'_id': u'yes', u'avg': 6.285714285714286},
 {u'_id': u'condominiums', u'avg': 6.0},
 {u'_id': u'hangar', u'avg': 5.0},
 {u'_id': u'hospital', u'avg': 4.666666666666667},
 {u'_id': u'university', u'avg': 4.5},
 {u'_id': u'mixed_use', u'avg': 4.0},
 {u'_id': u'apartments', u'avg': 3.3333333333333335},
 {u'_id': u'museum', u'avg': 3.0},
 {u'_id': u'UCB_Recreation_Offices', u'avg': 3.0},
 {u'_id': u'residential', u'avg': 3.0},
```

```
{u'_id': u'commercial', u'avg': 2.888888888888889},
{u'_id': u'terrace', u'avg': 2.7777777777777777},
{u'_id': u'house', u'avg': 2.2413793103448274},
{u'_id': u'school', u'avg': 2.0},
{u'_id': u'church', u'avg': 2.0},
{u'_id': u'Urban_Pioneer', u'avg': 2.0},
{u'_id': u'industrial', u'avg': 1.6666666666666667},
{u'_id': u'retail', u'avg': 1.5},
{u'_id': u'abandoned', u'avg': 1.3333333333333333},
{u'_id': u'public', u'avg': 1.0}]
```

---

## 3 Additional Ideas

To reduce instances of inconsistent address data, there should be an automatic check to accept or reject the change. For example, if the user enters street address with abbreviated 'Rd', the system identifies this and either suggests a corrected value 'Road', or rejects it altogether.

OpenStreetMaps can also introduce novel ideas that gets people excited about this app. It can introduce geo-tagging or treasure-hunt type of games that involves the players solving location-based puzzles, as individuals, or as groups. It can award more points to users/teams that contribute the most. A mobile app can be provided that users can 'tag' the location with and add other details about that location. This will create a truly crowd-sourced effort to improve its data.

The document also provides location for various types of amenities (restaurants, schools, etc.). It will be helpful to include average ratings provided by users for these places, and other useful information such as Business Hours, price-range, etc. This way, more and more people will start using this app and they will also have incentive to start contributing more.

Here are some problems with this approach, and some potential ways to address the problem:

- User might lack willingness to make submission: If the user submission is rejected, the user might not have motivation to submit next time.
- Potential resolution: To ensure data quality, user should be given potential correct choices before they make the final submission. This is why data submitted by multiple users for the same location should be compiled, and some kind of weighing mechanism should be used to identify submission with higher quality. If the task is made 'fun' and a point system is used as reward, I think users will be more motivated to achieve higher scores. This system works with video-games, where there is no physical reward, yet people contribute immense amount of time and energy to achieve virtual award.
- Data quality issues: If multiple users are allowed to contribute to same data-field, there is greater chances of conflicts which might affect quality of the data. Also, most users are not interested in filling out all relevant information.
- Potential resolution: OpenStreetMaps can develop a system that puts more weight on contribution based on how often a submission was made by unique users, or based on 'merit' of the user. For example, if 10 users claim that the given location is '10 Main Street', vs. 2 users that claim it to be '100 Main Street', more weight can be placed on the first value (along with

other relevant considerations). Merit can be based on how many high-quality submission the user has made in the past, how often does the user submit, etc..

---

## 4 Conclusion

Given competition from other 'free' map services from Google, Apple, etc, OpenStreeMaps need to incentivize its users to contribute more. There are handful of uses that seem to contribute bulk of the map data. However, given the scope and size of this project, it needs to attract attention of wider variety of population. It also need to take steps to ensure correctness and validation of the data.

I think OpenStreetMaps has the potential to carve out its own niche, and stand out among other well-established mapping services.