# OpenStreetMap Project

*Data Wrangling with MongoDB*

Toshiki Nazikian

12/9/2015

Analysis of OSM Data in San Diego

OVERPASS API: "(node(32.5445, -117.3676, 33.0697, -116.7297);<;);out;"
Map Area: San Diego, California
File size: ~210 MB
Output File size: 97 MB

INTRODUCTION:

This report describes the various locations of San Diego, detailing shops, roads, etc. I chose San Diego because it is the town I am most familiar to. The purpose of this report is to expose myself to Python code involving parsing and manipulating large XML files.

DIFFICULTIES:

There were mapping difficulties while parsing the file, such as different spellings of one name. For example, some 7-Eleven stores recorded as “7-11” or “seven eleven” depending on the user. I have addressed the most obvious issues by standardizing all abbreviations within a dict. This helped greatly in translating abbreviated and misspelled strings to their correct names. A regex was used to arbitrate skipping of street names. Street names that were only numbers or blank spaces were not passed.

Loops and regexes were used to extract relevant address information and data. I did not anticipate there would be no “user” field in any node in the original file. This can be attributed to the fact that OpenStreetMap can be modified anonymously.

Every node element was checked for redundant information, such as country/city attributes, as well as repeated information such as zip codes. This was done by compiling regexes to filter unwanted elements containing colons. The subsequent file assumes that all ways and nodes are in the San Diego area.

Another difficulty in properly recording street and shop names was, capitalizing name fields with correct grammar. A filter was implemented that applied this only to element keys ending with “name”.

Some database posts were updated via. the PyMongo API, such as amenity fields for certain chain cafes and restaurants.

Names:
Queries revealed name fields with either incorrect spelling or numbers only. This was addressed by implementing the following query:

Code:
```
for item in db.small_posts.find({"name":{"$exists":1}}):
   if (item['name']).isdigit():
      db.small_posts.update_one({"_id":item["_id"]}, {"$unset":{item["name"]: ""}})
```

Amenities:
Classifications changes were done to the amenity tags of chain restaurants as a way of standardizing and correction. For example, an aggregation for Starbucks before audit would display:
```
{u'count': 47, u'_id': u'cafe'}
{u'count': 7, u'_id': u'restaurant'}
```

Evidently there are 7 Starbucks entries labeled as restaurants. Using the db.collections.update_many() method I changed all 54 fields to ”cafe”.

Postcodes:
Some postal codes featured unnecessary state abbreviations, or were not five digits long. Fields with ZIP+4 formats were reduced to the first five digits, while the rest were removed. A pipeline aggregation was used to loop through each entry and make adjustments.

```
if not regexnom.match(i["address"]["postcode"]):
      try:
         db.small_posts.update_one({"_id":i["_id"]}, {"$set":{"address.postcode":
(regex2.match(i["address"]["postcode"])).group()}})
      except:
         db.small_posts.update_one({"_id":i["_id"]}, {"$unset":{"address.postcode":""}})
```

INTERESTING OBSERVATIONS:

Top Five Cafes in San Diego:

```
{u'count': 53, u'_id': u'Starbucks'}
{u'count': 3, u'_id': u'Fairbanks'}
{u'count': 3, u'_id': u'Jamba Juice'}
{u'count': 3, u'_id': u"Peet's Coffee & Tea"}
{u'count': 3, u'_id': u'The Coffee Bean'}
```

Pipeline example:

```
pipeline = [
    {"$match": {"name":{"$exists":1},"amenity":{"$in":[re.compile("cafe", re.I),
re.compile("coffee", re.I)]}}},
    {"$group": {"_id": "$name", "count": {"$sum": 1}}},
    {"$sort": {"count":-1}},
    {"$limit":100}
]
```

Top Five “Restaurants” in San Diego:

```
{u'count': 74, u'_id': u"Mcdonald's"}
{u'count': 71, u'_id': u'Jack In The Box'}
{u'count': 62, u'_id': u'Subway Sandwiches Sandwiches'}
{u'count': 37, u'_id': u'Taco Bell'}
{u'count': 32, u'_id': u'Carls Jr.'}
```

Total # Nodes: 1371754

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

Total # Ways: 98438

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

Total Unique Users: 0 (???)

```
db.small_posts.distinct("created.user")
```

Total Documents: 1470192

```
db.small_posts.find().count()
```

# Unique Cuisines: 92
```
len(db.small_posts.distinct("cuisine"))
```

Top Five Ethnic Cuisines:
```
...
{u'count': 169, u'_id': u'mexican'}
...
{u'count': 42, u'_id': u'american'}
...
{u'count': 32, u'_id': u'chinese'}
{u'count': 30, u'_id': u'italian'}
...
{u'count': 23, u'_id': u'sushi'}
{u'count': 21, u'_id': u'thai'}
{u'count': 20, u'_id': u'japanese'}
```
pipeline:
```
[
    {"$match": {"cuisine":{"$exists":1}}},
    {"$group": {"_id": "$cuisine", "count": {"$sum": 1}}},
    {"$sort": {"count":-1}},
]
```

Having lived in this area for some time, I knew some restaurants that were either incorrectly labeled or left out. For example, there are more Korean restaurants than Japanese, most of them barbecue joints or cafes.

CONVOY STREET:

Convoy Street is a lively and popular hang-out area for students and families alike, as well as the site of many authentic asian restaurants and markets. I will compare data from this portion of the map to the rest of San Diego.

Coordinates of interest:
```
32.8383
32.8134
-117.1693
-117.1353
```

Pipeline:
```
[
    {"$match": {"amenity":{"$exists":1}, "$and":[{"pos.lat":{"$gt":32.8134,
"$lt":32.8383}}, {"pos.lon":{"$gt":-117.1693, "$lt":-117.1353}}]}},
    {"$group": {"_id": "$amenity", "count": {"$sum": 1}}},
    {"$sort": {"count":-1}},
    {"$limit":20}
]
```

Top 10 Amenities:
```
{u'count': 18, u'_id': u'restaurant'}
```

```
{u'count': 17, u'_id': u'fast food'}
{u'count': 11, u'_id': u'place of worship'}
{u'count': 9, u'_id': u'cafe'}
{u'count': 7, u'_id': u'fuel'}
{u'count': 6, u'_id': u'bar'}
{u'count': 6, u'_id': u'dentist'}
{u'count': 5, u'_id': u'shelter'}
{u'count': 3, u'_id': u'bank'}
{u'count': 2, u'_id': u'school'}
```

Top 10 Cuisines:

```
{u'count': 8, u'_id': u'sandwich'}
{u'count': 5, u'_id': u'burger'}
{u'count': 4, u'_id': u'mexican'}
{u'count': 4, u'_id': u'chinese'}
{u'count': 3, u'_id': u'korean'}
{u'count': 2, u'_id': u'pizza'}
{u'count': 2, u'_id': u'japanese'}
{u'count': 2, u'_id': u'asian'}
{u'count': 1, u'_id': u'vietnamese'}
{u'count': 1, u'_id': u'chinese;japanese'}
```

Because “sandwich” does not ascribe to any specific ethnic cuisine (all “burger” elements are for generic fast-food joints), I have included the names of the shops:

```
{u'count': 3, u'_id': u'Subway'} - Fast Food
{u'count': 1, u'_id': u'Cali Baguette'} - Vietnamese
{u'count': 1, u'_id': u"Mama's Grill"} - Deli
{u'count': 1, u'_id': u"Jersey Mike's"} - Deli
{u'count': 1, u'_id': u'Subway Sandwiches Sandwiches'}
{u'count': 1, u'_id': u"Arby's"} - Fast Food
```

I recognized many restaurants are neglected in the OSM file, seeing how they number less than the number of generic fast food joints. I was also surprised there was no information of the Mitsuwa or Marukai marketplaces, which are my all-time favorite Japanese establishments.

FILLING IN THE DATA:

From a preliminary review there appears to be many shops not represented in the data. I therefore installed the Yelp API via. Pip to cross-reference information from the Yelp database. This allowed me to convert node entries from Yelp to MongoDB.

The Yelp API pulls search requests from the Yelp database and translates results into Python objects. The library client was used to search for restaurants in San Diego, shaping each element to conform with the dict format of MongoDB elements. I then inserted the documents into a collection named, “small_posts”. Counting the total number of documents confirms the insertion works as intended:

Code:

```
print(db.small_posts.count())
```

Before Request: 1470192

After Request: 1470232

Although limited to 40 documents per query, this method presents an effective and easy way of gathering information.

Code:

```
auth = Oauth1Authenticator(
    consumer_key=YOUR_KEY,
    consumer_secret=YOUR_SECRET,
    token=YOUR_TOKEN,
    token_secret=YOUR_TSECRET
)
yelp_client = Client(auth)

params = {
    'term': 'Restaurant'
}

yelp_results = yelp_client.search('San Diego, CA', **params)
for item in yelp_results.businesses:
    existing = False
    loc = item.location
    if db.small_posts.find_one({"name":item.name,
"pos.lon":item.location.coordinate.longitude,
"pos.lat":item.location.coordinate.latitude}):
        existing = True
    if not item.is_closed:
        try:

            node_fmtd['type'] = 'node'
            node_fmtd['pos'] = {}
            node_fmtd['pos']['lat'] = float(item.location.coordinate.latitude)
            node_fmtd['pos']['lon'] = float(item.location.coordinate.longitude)
            node_fmtd['name'] = item.name
            node_fmtd['amenity'] = 'fast food'
            if node_fmtd['amenity'] in ["restaurant", "cafe"]:
                node_fmtd['cuisine'] = item.categories[0].name
            node_fmtd['address'] = {}
            node_fmtd['address']['street'] =
aligner((item.location.address[0]).split())
            if item.location.cross_streets:
                node_fmtd['address']['cross_streets'] =
aligner((loc.cross_streets).split())
            node_fmtd['address']['postcode'] = item.location.postal_code
            node_fmtd['address']['neighborhoods'] = item.location.neighborhoods
            if existing is True:
                db.small_posts.update_one({"name":node_fmtd['name'],
"pos":node_fmtd['pos']}, {"$set":{"amenity":node_fmtd['amenity']}})
            else:
                db.small_posts.insert_one(node_fmtd)
        except KeyError:
            pass
```

Furthermore, a MongoDB aggregation for restaurants reveals how this function populates the database.
Old Top 3 Amenities:

```
{u'count': 18, u'_id': u'restaurant'}
{u'count': 17, u'_id': u'fast food'}
{u'count': 11, u'_id': u'place of worship'}
{u'count': 9, u'_id': u'cafe'}
{u'count': 7, u'_id': u'fuel'}
```

New:

```
{u'count': 52, u'_id': u'restaurant'}
{u'count': 17, u'_id': u'fast food'}
{u'count': 11, u'_id': u'place of worship'}
{u'count': 9, u'_id': u'cafe'}
{u'count': 8, u'_id': u'grocery'}
```

This is particularly useful because the Yelp API allows the user to specify parameters, reducing the need to map certain elements, such as name, amenity, or location. After running this function while inputting (“Convoy Street, San Diego, CA”, “restaurants”) in the Yelp client's “search” method, the new cuisine aggregation returns:

```
{u'count': 8, u'_id': u'sandwich'}
{u'count': 7, u'_id': u'Chinese'}
{u'count': 5, u'_id': u'Korean'}
{u'count': 5, u'_id': u'Japanese'}
{u'count': 5, u'_id': u'burger'}
{u'count': 4, u'_id': u'mexican'}
{u'count': 2, u'_id': u'Burgers'}
{u'count': 2, u'_id': u'Dim Sum'}
{u'count': 2, u'_id': u'Thai'}
```

These results more accurately represent the influence of asian cultures around Convoy. Despite not counting sushi bars, ramen shops, or other asian food lounges it is clear the number of nodes representing asian restaurants is increased.