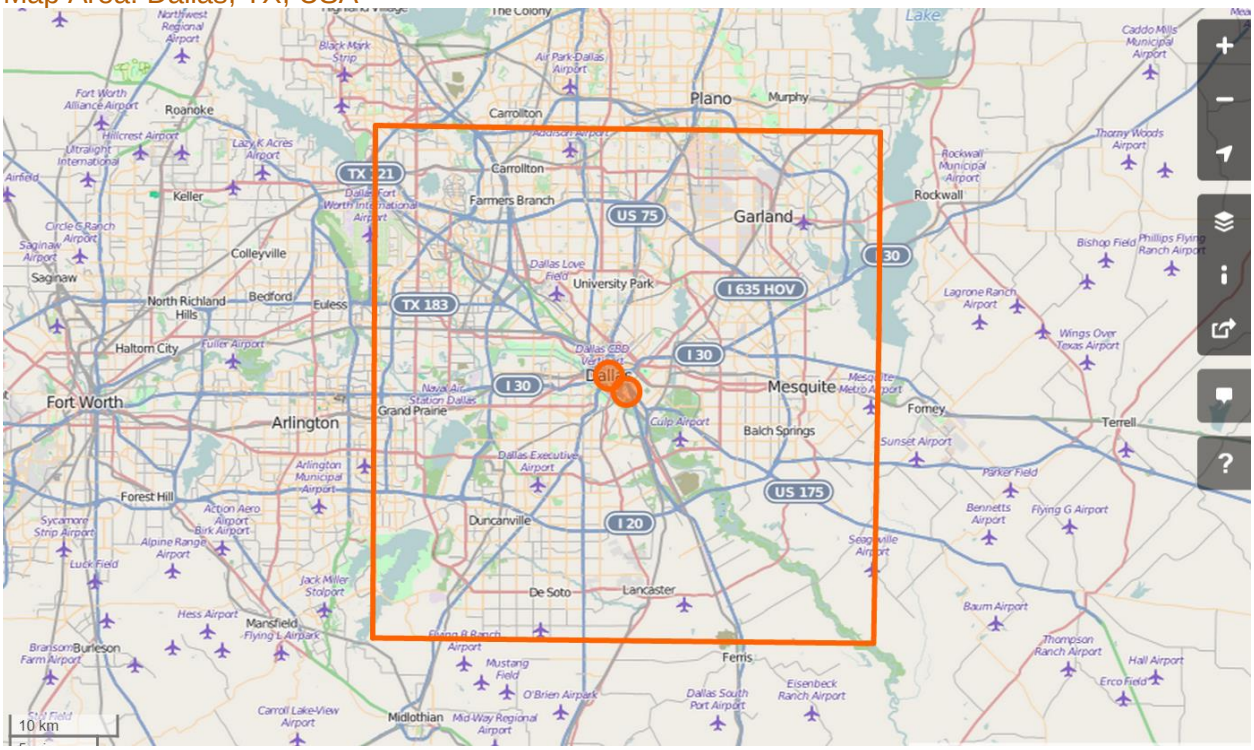
OpenStreetMap Sample Project
Data Wrangling with MongoDB
Anh Chu

Map Area: Dallas, TX, USA



I chose to wrangle the Dallas map because it is where I am currently living and I want to learn about the area. At first I looked for Dallas County from openstreetmap.org. However, the data set downloaded from mapzen.com appeared to encompass a much larger area than Dallas County. I realized it is supposed to be the data for Dallas Fort Worth Metroplex (DFW), which includes 19 counties, 2 metropolitan areas and 7 micropolitan areas, population of 6,8 millions…

1. Problems Encountered in the Map

The osm file is too large (563MB) to be opened by Notepad++ and cannot parse by Tree-based XML parsing. So to have a glimpse of the data structure, I parsed one tag at time using iterparse and wrote every 10 top level elements to a sample.osm file. I recognized that there are noises in the data set such as strange tag type: <tag k = “tiger” (US Census Tiger road dataset) under the “way” tag and <tag k="gnis:id" under the “node” tag. Due to the scope of the project, I focus on cleaning up some data of interest: street names, postal codes, phone numbers and population

After the data has been cleaned up and changed to json format, I use mongoimport in command prompt to import the json file to MongoDB. Then make some queries about the data using some MongoDB aggregation operators

Street name

First I identified the problematic street types:

```
expected = ["Street", "Avenue", "Boulevard", "Drive", "Court",
"Place", "Square", "Lane", "Road", "Walk", "Trail", "Parkway",
"Commons",
"Circle","Highway","Suit","Way","Turnpike","Trace","Tollway","Center"
,"Bay","Expressway","Freeway","Run"]

def audit_street_type(street_types, street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type not in expected:
            street_types[street_type].add(street_name)

def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")

def audit(osmfile):
    osm_file = open(osmfile, "r")
    street_types = defaultdict(set)
    for event, elem in ET.iterparse(osm_file, events=("start",)):

        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_street_name(tag):
                    audit_street_type(street_types, tag.attrib['v'])

    return street_types
```

The problem with street names are:

- Over-abbreviated street names (“N Murphy Rd”), abbreviated direction(N, S, W, E)
- Same road with different names (TX 78, Highway 78)
- Street name include house number and zip code (5223 alpha road dallas tx 75240)
- Unidentified street names (Avenue North Ste 3),
- Typo errors (Sinder Plaza, Snider Plaza)

However, the abbreviations are found not only at the end of street names but also in the middle or beginning of the street names (For example: E FM 544). I used mapping to change the most common abbreviations

```
mapping = { "St": "Street","St.": "Street",
            "Rd":"Road","Rd.":"Road","RD":"Road",
            "Ave":"Avenue","Av":"Avenue", "Ave.":"Avenue",

"BLVD":"Boulevard","BLVD.":"Boulevard","Blvd":"Boulevard","Blvd.":"Bo
ulevard","blvd":"Boulevard",
            "Dr":"Drive","Dr.":"Drive", "dr":"Drive", "Trl":"Trail",
            "pkwy":"Parkway", "Pkwy":"Parkway",
            "Hwy":"Highway","Hwy78":"Highway 78", "SH":"State
Highway","TX":"Highway",
            "Fwy":"Freeway",
            "Cir":"Circle",
            "Ln":"Lane",
            "Pl":"Place",
            "Expessway":"Expressway","Expy":"Expressway",

"N":"North","N.":"North","S":"South","S.":"South","W":"West","W.":"We
st","E":"East","E.":"East",
            "I":"Interstate", "i":"Interstate","FM":"Farm to
Market","Farm-to-Market":"Farm to Market","Fm":"Farm to
Market","CR":"County Road","U.S.":"US",
            "Sinder":"Snider"
            }

def update_name(name, mapping):

    m = street_type_re.search(name)
    if m:
        street_type = m.group()
    if street_type in mapping:
        new_street_type = mapping[street_type]
        name = name.replace(street_type, new_street_type)
    #Identify abbreviations in the beginning or middle of street names
    for key in mapping.keys():
        if key in name:
            for word in name.split():
                if word == key:
                    name = name.replace(word,mapping[key])
            for word in name.split("-"):
                if word == key:
                    name = name.replace(word,mapping[key])
                    name = name.replace("-"," ")

    return name
```

And this is the result of street names before and after cleaned up:

E FM 544=>East Farm to Market 544 W FM 544=>West Farm to Market 544 FM 544=>Farm to Market 544

Postal Codes & Phone number:

Problems with postal codes and phone numbers are inconsistency

- Inconsistent postal codes (TX 75229, 76010-1183).
- Inconsistent phone number (+1 817 626 7131, 972-473-2289); invalid phone number (+1 (469) 215 - SEO1; Main: (817) 310-5600 Catering: (817) 310-5610)

The preferred format for postal codes is a 5-digit block (75229) so it becomes favorable for MongoDB to aggregate the most common postal codes. Invalid post codes will be set to Null

```
def update_zipcode(zip_code):
    if zip_code.startswith("7")==False:
       zip_code = zip_code.replace("TX ","")
    if len(zip_code) > 5:
       zip_code = zip_code[:5]
    if len(zip_code) <5:
       zip_code = "NULL"
    return zip_code

for event,element in ET.iterparse(OSMFILE, events=("start",)):
    if element.tag == "node":
        for tag in element.iter("tag"):
                if k =="addr:postcode" or k=='tiger:zip_left' or k
=='tiger:zip_right':
                        tag.attrib['v']                         =
update_zipcode(tag.attrib['v'])
```

Post codes after cleaned:

76010-1183=>76010 TX 76013=>76013

Similarly to phone number, I decide to have every phone number with the format like this xxx-xxx-xxxx (972-473-2289) and trim out unrelated information

```
def update_number(number):
    num_list = re.findall(r'\d+',number)
    if num_list[0] == '1' or num_list[0] == '01' or num_list[0]=="011":
        num_list.remove(num_list[0])
    number = '-'.join(num_list)
    if len(number)>12:
        number = number[:12]
```

```
    if len(number)<12:
        number = “NULL”
    return number

for event,element in ET.iterparse(OSMFILE, events=("start",)):
    if element.tag == "node":
        for tag in element.iter("tag"):
                if k =="addr:postcode" or k=='tiger:zip_left' or k
=='tiger:zip_right':
                        tag.attrib['v']                         =
update_number(tag.attrib['v'])
```

Phone number after cleaned up

Main: (817) 310-5600 Catering: (817) 310-5610 => 817-310-5600 +1 (469) 215 - SEO1 => NULL (817) 788-1688 => 817-788-1688

Population:

Population has string type instead of integer type. This will cause difficulty to future query. So I change population data to integer

```
for event,element in ET.iterparse(OSMFILE, events=("start",)):
    if element.tag == "node":
        for tag in element.iter("tag"):
            k = tag.attrib['k']
            if k == "population”:
               tag.attrib[“v”] = int[tag.attrib[“v”]
```

Population after cleaned:

‘127672’ => 127672 ‘1299542’ => 1299542

2. Data Overview

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

File sizes

dallas_texas.osm..... 563 MB
dallas_texas.osm.json.... 594 MB

```
# Number of documents
documents = db.dallas.find().count()
```
⇨ 2725048
```
# Number of nodes
nodes = db.dallas.find({"type":"node"}).count()
```
⇨ 2460477

# Number of ways
db.dallas.find({"type":"way"}).count()
⇨ 264527
#Top 10 counties:
```
Pipeline= [{"$match":{"county":{"$exists":True}}},
                          {"$group":{"_id":"$county","count":{"$sum":1}}},
                          {"$sort":{"count":-1}},{"$limit":5}]
db.dallas.aggregate(pipeline)
```

{u'_id': u'Dallas, TX', u'count': 39729}
{u'_id': u'Tarrant, TX', u'count': 33822}
{u'_id': u'Denton, TX', u'count': 20548}
{u'_id': u'Collin, TX', u'count': 16521}
{u'_id': u'Ellis, TX', u'count': 13440}

Number of counties:
counties = db.dallas.distinct("county")
print len(counties)
⇨ 79
Top street names
```
pipeline=[{"$match":{"address.street":{"$exists":True}}},
{"$group":{"_id":"$address.street","count":{"$sum":1}}},
{"$sort":{"count":-1}},{"$limit":5}]

db.dallas.aggregate(pipeline)
```

{u'_id': u'Preston Road', u'count': 485}
{u'_id': u'Main Street', u'count': 221}
{u'_id': u'Hickory Street', u'count': 190}
{u'_id': u'Lebanon Road', u'count': 144}
{u'_id': u'Alexandria Drive', u'count': 136}

3. Additional Ideas

Incomplete dataset
Top 5 cities
```
pipeline=[{"$match":{"address.city":{"$exists":True}}},
{"$group":{"_id":"$address.city","count":{"$sum":1}}},
{"$sort":{"count":-1}},{"$limit":5}]

db.dallas.aggregate(pipeline)
```

{u'_id': u'Frisco', u'count': 52682}
{u'_id': u'Plano', u'count': 2997}
{u'_id': u'Cedar Hill', u'count': 615}
{u'_id': u'Dallas', u'count': 339}
{u'_id': u'McKinney', u'count': 289}
Number of unique city names
cities = db.dallas.distinct("address.city")
print len(cities)
⇨ 151
Total number of city

```
Pipeline=[{"$match":{"address.city":{"$exists":True}}},
{"$group":{"_id":"$id","count":{"$sum":1}}},
{"$group":{"_id":"city_total", "total":{"$sum":"$count"}}}]

db.dallas.aggregate(pipeline)
```

{u'total': 58225, u'_id': u'city_total'}
Among 58225 documents with address.city, Frisco appears 52682 times, account for 90.5%. Clearly, the dataset is incomplete with missing information of other cities.

Top 5 post codes

```
Pipeline=[{"$match":{"address.postcode":{"$exists":1}}},

{"$group":{"_id":"$address.postcode","count":{"$sum":1}}},
{"$sort":{"count":-1}},{"$limit":10}]

db.dallas.aggregate(pipeline)
```

{u'count': 617, u'_id': u'75104'}
{u'count': 312, u'_id': u'75093'}
{u'count': 177, u'_id': u'75070'}
{u'count': 113, u'_id': u'75051'}
{u'count': 89, u'_id': u'75069'}

To test the match between postcodes and cities, I find cities based on two top postcodes

```
pipeline=[{"$match":{"address.postcode":{"$exists":1},
"address.postcode":"75104"}},
{"$group":{"_id":"$address.city","count":{"$sum":1}}},
{"$sort":{"count":-1}},{"$limit":5}]

db.dallas.aggregate(pipeline)
```

{u'count': 612, u'_id': u'Cedar Hill'}
{u'count': 5, u'_id': None}

```
pipeline=[{"$match":{"address.postcode":{"$exists":1},
"address.postcode":"75093"}},
{"$group":{"_id":"$address.city","count":{"$sum":1}}},
{"$sort":{"count":-1}},{"$limit":5}]

db.dallas.aggregate(pipeline)
```

{u'count': 310, u'_id': u'Plano'}
{u'count': 2, u'_id': None}

It comes to my attention that although Frisco is the most-appearing city, the most appearing post codes don't match it. So I made a reverse query to find post codes based on city Frisco

```
pipeline=[{"$match":{"address.postcode":{"$exists":1},
"address.city":"Frisco"}},
{"$group":{"_id":"$address.postcode","count":{"$sum":1}}},
{"$sort":{"count":-1}},{"$limit":5}])

db.dallas.aggregate(pipeline)
```

{u'count': 43, u'_id': u'75034'}
{u'count': 6, u'_id': u'75035'}
{u'count': 6, u'_id': u'75033'}
{u'count': 1, u'_id': u'NULL'}

It seems like many documents relating to the Frisco city are missing of postcodes

Contributor information & Gamification

```
# Number of unique users
users = db.dallas.distinct("created.user")
print len(users)
```

⇨ 1534

```
# Top 5 contributing user
pipeline=[{"$group":{"_id":"$created.user","count":{"$sum":1}}},{"$so
rt":{"count":-1}},{"$limit":5}]

db.dallas.aggregate(pipeline)
```

{u'_id': u'woodpeck_fixbot', u'count': 1135251} => Percentage – 41.7%
{u'_id': u'fmmute', u'count': 99391} => Percentage – 3.65%
{u'_id': u'TexasNHD', u'count': 88499} => Percentage – 3.25%
{u'_id': u'brianboru', u'count': 61028}
{u'_id': u'Chris Lawrence', u'count': 60850}

Among 1534 contributors to the dataset, woodpeck_fixbot accounts for 41.7%. Other users account for a very small percentage of contribution. As fixbot runs occasionally and cleans data based on predefined algorithm. This implies that the dataset is mostly created automatically and can explain why it is incomplete and inaccurate. Local knowledge and ground truth are important to ensure the data quality of openstreetmap project. Hence, incentives can be used to attract local contributors.

After reviewing the gamification of openstreetmap, I think using mobile app games to attract local knowledge is a very good idea. For instance, the Kort game is fun and easy to scale. Some methods to make the game more popular to many players are adding sharing function, making the game available on social network, promoting incentives: badges, points, leaderboard…

Unexpected unclean data

One of the biggest problem I encountered in this project is unforeseen dirty data. As the dataset is large, it is difficult to discover all messy data at the first place. For instance, while making some queries on the data, I realized that there are further areas to clean up to improve consistence across dataset as well as increase accuracy and convenience when querying:

- names of restaurant and fast food chain are not consistent
- city and county names are not in the correct format
- population field is string, not number

As MongoDB doesn't provide tools to clean data after the data has already been loaded to the database, the process can be very painful. For example, I wanted to make some query from the population field. Then I realized the data type of the field is not integer (but string) and it limits the tools I can use to calculate the data. I needed to clean the data again then loaded it to MongoDB

once more time. Consequently, it would take more time, more effort and would possibly cause more errors.

Suggestions for future wrangling project

It is important to ensure data quality: validity, accuracy, completeness, consistency and uniformity before loading the dataset to the database. If the data is not clean enough, the query and statistics drawn from the data will be inaccurate and misleading, which poses risk to the decision making process.
I believe the good place to start is to iterparse a small sample dataset and have a sense of the data structure, think ahead about the areas to query, clean up those areas (for example, if I want to query number of Chick-fil-A in the dataset I will need to make sure the restaurant names are written in a consistent format), change the data types (for example: convert strings to integers in any fields that require calculation: amount, quantity, age…)

Furthermore, I realize I can use cross-check from other fields to input missing values. For example, matching the postcodes and cities, create a new cuisine_type field in every restaurant node based on restaurant names to query the most popular cuisine type…

Cleaning up the data beforehand can take time and require the ability to foresee the data of interest and future requirement. However, it would become beneficial after the dataset has been loaded to database and ready to query.

Additional data exploration using MongoDB queries

```
# Top 10 appearing amenities
pipeline[{"$match":{"amenity":{"$exists":True}}},
{"$group":{"_id":"$amenity","count":{"$sum":1}}},
{"$sort":{"count":-1}},{"$limit":10}]

db.dallas.aggregate(pipeline)
```

{u'_id': u'parking', u'count': 3293}
{u'_id': u'place_of_worship', u'count': 2842}
{u'_id': u'school', u'count': 2006}
{u'_id': u'fast_food', u'count': 1015}
{u'_id': u'restaurant', u'count': 865}
{u'_id': u'fuel', u'count': 424}
{u'_id': u'grave_yard', u'count': 291}
{u'_id': u'bank', u'count': 199}
{u'_id': u'fire_station', u'count': 162}
{u'_id': u'post_office', u'count': 153}

```
# Top 1 religion
pipeline =[{"$match":{"amenity":{"$exists":1},
                      "amenity":"place_of_worship"}},
{"$group":{"_id":"$religion","count":{"$sum":1}}},
{"$sort":{"count":-1}},{"$limit":1}])

db.dallas.aggregate(pipeline)
```

{u'_id': u'christian', u'count': 2758}

Most popular fast_food

```
pipeline=[{"$match":{"amenity":{"$exists":1},
"amenity":"fast_food"}},
{"$group":{"_id":"$name","count":{"$sum":1}}},
{"$sort":{"count":-1}}, {"$limit":5}])

db.dallas.aggregate(pipeline)
```

{u'_id': u'Whataburger', u'count': 142}
{u'_id': u"McDonald's", u'count': 98}
{u'_id': u'Chick-fil-A', u'count': 72}
{u'_id': u"Wendy's", u'count': 52}
{u'_id': u'Taco Bell', u'count': 50}

Max population

```
pipeline=[{"$match":{"population":{"$exists":1}}},
{"$group":{"_id":"$name", "population":{"$max":"$population"}}

db.dallas.aggregate(pipeline)
```

{u'_id': u'Dallas',, u'population': 1299542}

Conclusion

I believe the dataset is incomplete and should have more manual map editing to ensure completeness. The openstreetmap project should encourage more users to manual edit and contribute to the map data because automatic editing is surely insufficient. Gamification, prizes and badges can be use as incentives to attract contributors to the map editing projects. As for the data wrangling process, we should spend more time and effort to clean all the possible messy data before loading them to the database.

Reference:

http://docs.mongodb.org/manual/reference/operator/aggregation/group/
https://docs.python.org/2/howto/regex.html
http://api.mongodb.org/python/current/tutorial.html
https://blog.rainforestqa.com/2012-11-05-mongodb-gotchas-and-how-to-avoid-them/
http://blog.physalix.com/datas-manipulation-in-mongodb-rename-field-change-type-add-sub-document/
http://stackoverflow.com/questions/27825985/count-the-number-of-times-object-an-property-exists-in-mongo-array-of-objects
https://www.quora.com/How-can-I-change-a-field-type-from-String-to-Integer-in-mongodb
https://www.youtube.com/watch?v=Ld1aDHMbP-k
https://www.youtube.com/results?search_query=project+aggregation+in+mongodb
https://www.youtube.com/watch?v=W-WihPoEbR4
https://www.youtube.com/results?search_query=project+aggregation+in+mongodb
http://stackoverflow.com/questions/15285795/mongodb-aggregation-framework-group-project
http://stackoverflow.com/questions/22819303/mongodb-aggregation-divide-computed-fields
http://docs.mongodb.org/manual/core/cursors/
https://en.wikipedia.org/wiki/Dallas%E2%80%93Fort_Worth_metroplex
https://en.wikipedia.org/wiki/OpenStreetMap