P3: Wrangle OpenStreetMap Data

My submission has the following files:

1. Sample_sj_ca.osm
    - Sample osm file taken from the larger .osm file for ease in processing.
2. Sample_sj_ca.osm.json
    - JSON file generated after processing the sample osm file.
3. San-jose_california.osm
    - Complete .osm data (not included due to size)
4. San-jose_california.osm.json
    - JSON file generated from the complete osm file. (not included due to size)
5. Process_file.py
    - Script to process the osm file and perform certain cleaning and auditing operations as well find information about the data like, number of users and tags.
6. Query_file.py
    - Script to connect to MongoDB and perform analysis on the data by querying the database.
7. Lesson6_files:
    - This directory contains the solutions for Lesson 6 from the attached course.

How to run this project:

1. Make sure you have MongoDB installed.
2. If your DB is locked and you are not able to access it, it can be solved by doing:
    a. `sudo chown -R `id -u` /data/db`
3. Navigate to the project directory and run, this will take a few minutes due to the large size of the osm file:
    a. `python process_file.py`
4. Once that is complete you should see a JSON file generated in your project directory.
5. Start your DB by navigating to your bin folder inside your mongodb directory and typing in:
    a. `./mongod`
6. Once you have your instance up and running, on a separate terminal window, navigate to the project directory and load the JSON file onto the DB by:
    a. `mongoimport --db sanjose1 --collection openstreetmaps --type json --file san-jose_california.osm.json`
    b. Here the name of the db is sanjose1 and the name of the collection is openstreepmaps and the file we are loading into the db is the JSON file we generated in Step 3.
7. Once the data has been loaded onto the DB, run the following:
    a. `python query_file.py`
8. You should now see the results of the queries being run on your terminal.

Which city did I pick?

I picked San Jose, California -
http://www.openstreetmap.org/relation/112143#map=11/37.3846/-121.7007
as I am familiar with the city and I live nearby in San Francisco, California. It is the third-largest city by population in California, the tenth-largest by population in the United States. It is known as the 'Capital of Silicon Valley'.

Problems encountered in your map:

The data from Openstreetmaps was quite clean but there were a few exceptions. Street name abbreviations were inconsistent and while auditing the dataset, I noticed that some streets had abbreviations while others didn;t as can be seen below(I used a sample while studying the structure of the data). For example: Street names with 'Cir' and 'Circle'. During the data audit, I found a few field names with double colons and so added an extra regex to handle that as it seemed unlikely that the lower_colon regex would catch them.

```
defaultdict(<type 'set'>,
            {'Rd': set(['San Antonio Valley Rd', 'Saratoga Los Gatos
Rd', 'Mt Hamilton Rd']),
             'Way': set(['Firebird Way', 'Dublin Way', 'Flicker Way',
'Larry Way', 'Barnsley Way', 'Whitney Way', 'Las Ondas Way', 'Bianchi
Way', 'Clifden Way', 'Willowbrook Way', 'Tangerine Way', 'Cartwright
Way', 'Arata Way', 'Fife Way', 'Tomlin Way', 'Hunter Way', 'Flamingo
Way', 'Altair Way', 'Erin Way', 'Normandy Way', 'Sakura Way', 'John
Way', 'Poppy Way', 'Cheshire Way', 'Dorset Way', 'Nandina Way',
'Brahms Way', 'Terry Way', 'Lilac Way', 'Primrose Way', 'Gillick
Way', 'Revelstoke Way', 'Longfellow Way', 'Alderbrook Way', 'Tonita
Way', 'Durshire Way', 'Dunnock Way', 'Squirewood Way', 'Carlisle
Way', 'Innovation Way', 'All America Way', 'Forge Way', 'Devonshire
Way', 'Prince Edward Way', 'Pyrus Way', 'Kingfisher Way', 'Connemara
Way', 'Allison Way', 'Robindell Way', 'Felton Way', 'Flin Way',
'Nelson Way', 'Senate Way', 'Mallard Way', 'Humewick Way', 'Big Basin
Way', 'Duncardine Way', 'Westlynn Way', 'Martinwood Way', 'Miette
Way', 'Dartshire Way', 'Milky Way']),
             'Circle': set(['Murano Circle', 'Bobolink Circle',
'Carriage Circle', 'Vista Club Circle', 'Calabazas Creek Circle',
'Gardenside Circle', 'vista Club Circle']),
             'East': set(['Vanderbilt Court East', 'Park Circle
East']),
             'Alameda': set(['The Alameda']),
             'Real': set(['El Camino Real', 'West El Camino Real',
'East El Camino Real']),
             'Expressway': set(['Almaden Expressway', 'San Tomas
Expressway', 'Lawrence Expressway', 'West Capitol Expressway']),
'Paviso': set(['Via Paviso']),
             'Marino': set(['Via San Marino']),
             'Volante': set(['Via Volante']),
             'Napoli': set(['Via Napoli']),
             'Sq': set(['Evergreen Village Sq']),
             'Plaza': set(['Portal Plaza']),
             'Barcelona': set(['Calle de Barcelona']),
             'Cir': set(['Celadon Cir']),
             'Palamos': set(['Via Palamos']),
             'Presada': set(['Paseo Presada']),
             'Loop': set(['Village View Loop']),
             'Sorrento': set(['Via Sorrento']),
             'Portofino': set(['Via Portofino']),
             'Terrace': set(['Holthouse Terrace', 'Riorden Terrace',
'Yellowstone Terrace', 'Oak Point Terrace', 'Windsor Terrace', 'Costa
Mesa Terrace', 'Seneca Terrace', 'Satsuma Terrace', 'Avon Terrace',
'Pinnacles Terrace', 'Raines Terrace', 'Manet Terrace', 'Pyracantha
Terrace', 'Brea Terrace', 'Hogarth Terrace', 'Fontana Terrace',
'Avoset Terrace', 'Lautrec Terrace', 'Markham Terrace', 'Anaheim
Terrace', 'Amherst Terrace', 'Panache Terrace', 'Cedarbrook Terrace',
'Reston Terrace', 'Miller Terrace', 'Bristol Terrace', 'Isla Vista
Terrace']),
             'Blvd': set(['Palm Valley Blvd']),
             'Ave': set(['Foxworthy Ave', 'Cherry Ave', 'Seaboard
Ave', 'Meridian Ave'])})
```

There was only 1 problemchars in the audit of the full dataset which is very impressive!

I also implemented a check for the zip codes using regex as follows in my auditing process: `re.match(r'^\d{5}$', zipcode)` to ensure that zipcodes followed the 5 consecutive integer format.

The San Jose dataset had zipcodes in the format of ‘xxxxx’ where x is an integer, for example: ‘95120’.
I implemented a check for the zipcodes in this dataset and found a few instances where the zipcode was of the format ‘xxxxx-xxxx’. For example: `95014-2029`. Such instances were audited to pick up only the first 5 digits. Therefore `‘95014-2029’` was turned into `‘95014’`.

Overview of the Data(the structure of each query can be viewed in query.py):

File Sizes:
San-jose_california.osm - 276 MB
San-jose_california.osm.json - 399 MB

Total number of nodes:
```
db.openstreetmaps.find({"type": "node"}).count()
1254292
```

Total number of ways:
```
db.openstreetmaps.find({"type": "way"}).count()
163780
```

Total number of total documents:
```
db.openstreetmaps.find().count()
1418110
```

Top 5 contributing users:
```
db.openstreetmaps.aggregate(top_user())
[{u'_id': u'nmixter', u'count': 285524},
 {u'_id': u'mk408', u'count': 153200},
 {u'_id': u'Bike Mapper', u'count': 82373},
 {u'_id': u'samely', u'count': 77516},
 {u'_id': u'RichRico', u'count': 71377}]
```
The top 5 users have very strong numbers and do not appear to be bots but its hard to be sure, nonetheless they have done a great job in maintaining a relatively clean dataset.

Number of single contributing users:
```
db.openstreetmaps.aggregate(single_post_users())
[{u'_id': 1, u'num_users': 226}]
```
There are about 226 contributors which is a pretty high number and probably explains why the dataset has been well maintained.

Top twenty building types:
```
db.openstreetmaps.aggregate(most_common_buildings())
```

```
[{u'_id': u'yes', u'count': 82536},
 {u'_id': u'house', u'count': 5867},
 {u'_id': u'residential', u'count': 4029},
 {u'_id': u'apartments', u'count': 466},
 {u'_id': u'roof', u'count': 355},
 {u'_id': u'school', u'count': 282},
 {u'_id': u'commercial', u'count': 218},
 {u'_id': u'office', u'count': 188},
 {u'_id': u'terrace', u'count': 162},
 {u'_id': u'garage', u'count': 123},
 {u'_id': u'retail', u'count': 122},
 {u'_id': u'garages', u'count': 65},
 {u'_id': u'church', u'count': 56},
 {u'_id': u'industrial', u'count': 51},
 {u'_id': u'modular', u'count': 41},
 {u'_id': u'shed', u'count': 37},
 {u'_id': u'grandstand', u'count': 20},
 {u'_id': u'public', u'count': 16},
 {u'_id': u'hangar', u'count': 16},
 {u'_id': u'greenhouse', u'count': 14}]
```
The unusually high number of ‘yes’ categories is actually buildings that have been classified in the ‘other’ category, which explains their number. Getting an idea of the most popular building types helps in understanding the general layout of the city.

Top ten amenities:

```
db.openstreetmaps.aggregate(top_amenities())
[{u'_id': u'parking', u'count': 1707},
 {u'_id': u'restaurant', u'count': 875},
 {u'_id': u'school', u'count': 539},
 {u'_id': u'fast_food', u'count': 442},
 {u'_id': u'place_of_worship', u'count': 340},
 {u'_id': u'cafe', u'count': 228},
 {u'_id': u'fuel', u'count': 223},
 {u'_id': u'bench', u'count': 181},
 {u'_id': u'bank', u'count': 173},
 {u'_id': u'toilets', u'count': 168}]
```
I was curious to see what amenities are most commonly found in the city and interestingly restaurants, schools and fast food restaurants are the most common(besides parking spots which I would think would be high anyway)

Top banks:

```
[{u'_id': u'Bank of America', u'count': 25},
 {u'_id': u'Chase', u'count': 24},
 {u'_id': u'Wells Fargo', u'count': 23},
 {u'_id': u'Citibank', u'count': 9}]
```
I was curious to find out which banks had a strong presence in the city and it looks like BofA, Chase and Wells Fargo are a head and shoulders above the rest.

Top cafes:
```
db.openstreetmaps.aggregate(top_cafes())
[{u'_id': u'Starbucks', u'count': 70},
 {u'_id': u'Subway', u'count': 6},
 {u'_id': u'Philz Coffee', u'count': 5},
 {u'_id': u'Jamba Juice', u'count': 4}]
```
It’s interesting that the number of Starbucks outnumbers its competitors by such a high margin. Cafes with ‘None’ values did not have names entered for their establishment.

Top fast food chains:
```
db.openstreetmaps.aggregate(top_fast_foods())
[{u'_id': u'Subway', u'count': 32},
 {u'_id': u"McDonald's", u'count': 30},
 {u'_id': u'Taco Bell', u'count': 18},
 {u'_id': u'Burger King', u'count': 16}]
```
No surprises here with the Subway and McDonalds having the highest numbers in the city which is in line with their national spread as well. Fast food chains with ‘None’ values did not have names entered for their establishment.

Top restaurants:
```
db.openstreetmaps.aggregate(top_restaurants())
[{u'_id': u'Pizza My Heart', u'count': 7},
 {u'_id': u"Denny's", u'count': 7},
 {u'_id': u'Panera Bread', u'count': 6},
 {u'_id': u'Round Table', u'count': 6}]
```
I was curious to see if there were any popular restaurant chains considering the high number of restaurants in the city and turns out the popular ones could just as easily have been classified as fast food spots. Restaurants with ‘None’ values did not have names entered for their establishment.

Conclusion

The data for San Jose while relatively clean has a lot of ground for improvement like looking at the phone numbers, zipcodes and other number based information which can be easily cleaned with more regex codes. There are ample more opportunities to clean this data beyond what we did for ‘node’ and ‘way’ as we did not consider the other tags like ‘nd’ in our analysis. I am interested to see how these results would compare with Google Maps' API as OpenStreepMap is crowdsourced and is therefore dependant on users' input, which could lead to incomplete or outdated information. After auditing and cleaning street names and zipcodes it is clear that using OSM data has its drawbacks but at the same time is open source. With Google Maps there is a limit to the number of times you can call the API after which you have to pay to use the service. But otherwise the Google Maps API has very good documentation and is fairly easy to use as well. There is also the added advantage of the data being clean for the most part as it is actively being taken care of and is not reliant upon the benevolence of the free users. Auditing and cleaning the OSM data may not be everyone’s piece of cake as it is a time consuming and fairly tedious process. But at the same time it is a fun experiment to work with.

Bibliography:

http://overpass-turbo.eu/
https://docs.mongodb.com/
http://www.tutorialspoint.com/mongodb/mongodb_insert_document.htm
http://www.tutorialspoint.com/mongodb/mongodb_query_document.htm