# OpenStreetMap Data Case Study

**Author:** Kevin Vo

## Map Area

San Jose, CA, USA

Dataset contains information for San Jose Area. I'm curious to see the map contribution of San Jose, which is my hometown.

## Main Procedure to identify Problems in the Map

- Since the the original file `sanjose.osm` is 600 MB, which is very large to view, I use `sample_region.py` to subset data.(This is the hint that Udacity provide). The new subset is stored in `sample.osm`, which is about 4MB (Set k = 100, it means pick every 100th top level element).
- It is hard to use `less` command in Unix to view file. So I switch to use Sublime Text to view data.
- Codes in `audit.py` are used to view and fix street names, phone numbers and postal codes.
- Run `process_osm` to clean data and export necessary information into CSV files( processed with schema in `schema.py` ).
- Choose a smaller k value to get a bigger `sample.osm` file. Then modify `audit.py` to be able to screen and audit more cases.

## Problems Encountered In the Map

**Overabbreviated and Inconsistent street names:** some street types and directions are abbreviated such as `St, Rd, Ave, Blvd, Cir, Ln, Ct` or `1st, 2nd,.... 10th`

For example: `N 1st St.` should be changed to `North First Street`

However, if I change `1st` to `First`, the road names are not consistent when I figured that there are some road names such as `13th Street` or `128th Street`. So I think instead of changing from `1st` to `First`, I will change `First` to `1st`.

```
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)
expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "La
ne", "Road",
            "Trail", "Parkway", "Commons", "Circle", "Crescent", "Gate", "Terrace", "
Grove", "Way"]

mapping = {
            "St": "Street",
            "St.": "Street",
            ...
            "W.": "West",
            "W": "West"
          }


word_number_mapping = {
                     "First": "1st",
                     "first": "1st",
                     ...
                     "ninth": "9th",
                     "Tenth": "10th",
                     "tenth": "10th"
                   }
```

I have not fixed extreme cases such as `(408) 738-CHEF` or placing website address in phone number. Fortunately, there are very few extreme cases in this data.

**Incorrect phone number format:**

Convert phone number to correct format: `+1 (408)###-####` (in fact, beside 408, 699 is another phone area code for San Jose,CA. However, I have not seen. For example:

`+1 408 123 4567` to `+1 (408)123-4567`

`+1 408-123-4567` to `+1 (408)123-4567`

`408.123.4567` to `+1 (408)123-4567`

`408.123-4567` to `+1 (408)123-4567`

`123-4567` to `+1 (408)123-4567`

First of all, I have to remove special characters such as `+1` , `-` , `()` , whitespace, `.` or `+` from phone number.

```
phone_num = re.sub("\+1", "", phone_num)
phone_num = re.sub("-", "", phone_num)
phone_num = re.sub("[()]", "", phone_num)
phone_num = re.sub("\s", "", phone_num)
phone_num = re.sub("\\.", "", phone_num)
phone_num = re.sub("\\+", "", phone_num)
```

Now most of phone numbers do not contain special chracters. We can check whether it is 7 digits long, 10 digits long or 11 digits long.

```
PHONE_7_NUM = re.compile(r'^\d{7}$')
PHONE_10_NUM = re.compile(r'^\d{10}$')
PHONE_11_NUM = re.compile(r'^\d{11}$')
```

Based on each case, we can modify phone number to standard format `+1 (408)###-####`

**Incorrect postal code format:**

Convert postal code to correct format: `CA #####` or `CA #####-####` .

I have used regular expression to classify all these cases:

```
POSTCODE = re.compile(r'^\d{5}$|\d{5}-\d{4}$')
```

If postal code are not 5-digits number or 9-digits number, it will be classified as wrong postal code then will be excluded from our data.

For example: these are excluded from our data

```
WRONG POSTAL CODE: 95014-218
WRONG POSTAL CODE: 95014-321
WRONG POSTAL CODE: 9404
```

## Create SQL Tables & Import Data Into Tables

```
sqlite> CREATE TABLE Nodes(id INTEGER PRIMARY KEY, lat REAL, lon REAL, user TEXT, uid
 INTEGER, version INTEGER, changeset INTEGER, timestamp DATETIME);
sqlite> CREATE TABLE nodesTags(id INTEGER, key TEXT, value TEXT, type TEXT, FOREIGN K
EY (id) REFERENCES Nodes (id));
sqlite> CREATE TABLE Ways(id INTEGER PRIMARY KEY, user TEXT, uid INTEGER, version INT
EGER, changeset INTEGER, timestamp DATETIME);
sqlite> CREATE TABLE waysNodes(id INTEGER, node_id INTEGER, position INTEGER, FOREIGN
 KEY(id) REFERENCES Nodes(id));
sqlite> CREATE TABLE waysTags(id INTEGER, key TEXT, value TEXT, type TEXT, FOREIGN KE
Y(id) REFERENCES Ways(id));
sqlite> .table
Nodes      Ways       nodesTags  waysNodes  waysTags
sqlite> .mode csv
sqlite> .import nodes.csv Nodes
sqlite> .import nodes_tags.csv nodesTags
sqlite> .import ways.csv Ways
sqlite> .import ways_tags.csv waysTags
sqlite> .import ways_nodes.csv waysNodes
```

## Data Overview

This section contains basic statistics about San Jose OpenStreetMap dataset and the SQL queries used to gather them.

### File sizes

```
sample.osm          237 MB
sanjose.db          117.3 MB
nodes.csv           90.7 MB
nodes_tags.csv      1.7 MB
ways.csv            8.1 MB
ways_nodes.cv       30.1 MB
ways_tags.csv       14.3 MB
```

### Number of nodes

```
sqlite> SELECT COUNT(*) FROM Nodes;
```

1048575

### Number of ways

```
sqlite> SELECT COUNT(*) FROM Ways;
```

136514

### Average number of nodes per day

```
sqlite> SELECT ROUND(number_of_nodes/(max_day - min_day)) AS average_nodes_per_day
        FROM (SELECT COUNT(*) AS number_of_nodes,
                julianday(MAX(timestamp)) AS max_day,
                julianday(MIN(timestamp)) AS min_day
                FROM Nodes);
```

285.0

## Users:

### Number of unique users

```
sqlite> SELECT COUNT(DISTINCT(ListOfUserId.uid))
        FROM (SELECT uid
                FROM Nodes UNION ALL
                    SELECT uid
                    FROM Ways) ListOfUserId;
```

1399

### Top 5 contributing users

```
sqlite> SELECT ListOfUser.user, COUNT(*) as num
        FROM (SELECT user FROM Nodes
                UNION ALL SELECT user FROM Ways) ListOfUser
        GROUP BY ListOfUser.user
        ORDER BY num DESC
        LIMIT 5;
```

```
nmixter,182215
andygol,113211
mk408,100515
karitotp,62823
RichRico,57426
```

### First contributor

```
sqlite> SELECT user, timestamp FROM Nodes
                UNION ALL SELECT user, timestamp From Ways
        ORDER BY timestamp
        LIMIT 1;
```

mikelmaron,2007-03-08T02:02:46Z

#### Number of Contributions by Year

```
sqlite> SELECT strftime('%Y', timestamp) AS year, count(*)
        FROM Nodes
        GROUP BY year;
```

```
2007,78
2008,11366
2009,49405
2010,103494
2011,59975
2012,39351
2013,57133
2014,100063
2015,241547
2016,266813
2017,119350
```

### Places To Eat:

### Most popular cuisine

```
sqlite> SELECT nodesTags.value, COUNT(*) as num
        FROM nodesTags
            JOIN (SELECT DISTINCT(id) FROM nodesTags WHERE value='restaurant') GetRes
taurantId
            ON nodesTags.id = GetRestaurantId.id
        WHERE nodesTags.key = 'cuisine'
        GROUP BY nodesTags.value
        ORDER BY num DESC
        LIMIT 10;
```

```
chinese,42
vietnamese,34
pizza,33
mexican,32
japanese,21
indian,18
italian,15
american,14
thai,14
sushi,12
```

## Coffee:

### Number of Cafe stores

```
sqlite> SELECT COUNT(*) FROM nodesTags WHERE value = 'cafe';
```

143

### Number of Starbucks

```
sqlite> SELECT COUNT(*) FROM nodesTags WHERE value LIKE '%Starbucks%';
```

50


## Banking:

### 5 Most popular bank

```
sqlite> SELECT nodesTags.value, COUNT(*) as num
        FROM nodesTags
            JOIN (SELECT DISTINCT(id)
                    FROM nodesTags
                    WHERE value='bank') GetBankId
            ON nodesTags.id=GetBankId.id
        WHERE nodesTags.key='name'
        GROUP BY nodesTags.value
        ORDER BY num DESC
        LIMIT 5;
```

```
Chase,12
"Bank of America",9
"Wells Fargo",7
Citibank,4
"US Bank",2
```


## Religion:

### Number of Place of Worship for each religion

```
sqlite> SELECT nodesTags.value, COUNT(*) as num
        FROM nodesTags
            JOIN (SELECT DISTINCT(id) FROM nodesTags WHERE value='place_of_worship')
GetPlaceOfWorshipId
            ON nodesTags.id=GetPlaceOfWorshipId.id
        WHERE nodesTags.key='religion'
        GROUP BY nodesTags.value
        ORDER BY num DESC;
```

```
christian,108
buddhist,1
caodaism,1
jewish,1
rosicrucian,1
shinto,1
zoroastrian,1
```

## Data Improvement Ideas:

- Working with this dataset, I find that there are still some wrong postal codes such as 9404 when the postal code are 5 digits or 7 digits number.
- Based on the number of contribution by year, we have realized that the number of contribution has been increasing every year. It means that the map data is going to grow larger. Also it means that the number of wrong postal codes will also increase.
- One way we could find the address for this issue is that I can use Reverse Geocoding from Google Geocoding API. Reverse Geocoding is the process of converting geographic coordinates into a human-readable address. In other words, I can use longitude and latitude to retrieve postal code.

## Conclusion

San Jose is a small town but its dataset is quite large. Therefore it is impossible to clean all the mess inside San Jose CA OpenStreetMap data. However, I believe the dataset is sufficiently cleaned in this project. Most street types abbreviation are replaced by more approriate street type without abbreviation. Phone number and postal code are handled nicely in the format of `+1 (408)###-####` and `CA ##### or CA #####-####` respectively. Via SQL query, I learned a few new things about my hometown. Throughout this project I have a great time learning how to clean data with python and SQL. However, my code still take nearly 30 minutes to handle 200MB osm file. Faster processing is the next approach I have to achieve for this project.