Wrangle OpenStreetMap Data: San Jose

Table of Contents

Introduction

In this project, I will explore an audit the Open Street Map data for San Jose area in California United States. Then, I will clear couple of the problems that I find during the auditing process, and write the cleaned data into five csv files according to a predetermined schema. Finally, I import these CSV files into a sqlite3 data base and do some further exploration of data using SQL queries.

Map Area

I Downloaded the XML file of the preselected metro area for San Jose from MapZen; Here is the address:

- https://mapzen.com/data/metro-extracts/metro/san-jose_california/ (https://mapzen.com/data/metro-extracts/metro/san-jose_california/)

The OSM file size is 402.5 MB and the specific bounds of this area are:

- Minimum (Latitude, Longitude): ("37.125", "-122.046")
- Maximum (Latitude, Longitude): ("37.469", "-121.589")

Using the Open Street Map Export Tool, I also downloaded a smaller version of San Jose XML. This sample OSM file size is 4.6 MB (almost 1% of the original file) with these boundaries:

- Minimum (Latitude, Longitude): ("37.2942", "-121.996")
- Maximum (Latitude, Longitude): ("37.316", "-121.9397")

I live in San Jose area; so I'm interested in exploring the OSM data of this area. I also like the opportunity to contribute to its improvement on OpenStreetMap.org

Getting Familiarize With Data

> Before processing the data and add it into my database, I explore the data a little bit more in order to get familiarize with it. First, I find and count all of the existing tags in the xml file using 'count_tags' function. Secondly, I find out how many unique users have contributed to the map in this particular area

In [3]:
```
def count_tags(filename):
    tag_dic = {}
    for event, elem in ET.iterparse(filename, events=("start",)):
        if elem.tag not in tag_dic.keys():
            tag_dic[elem.tag] = 1
        else:
            tag_dic[elem.tag] += 1
    return tag_dic

pprint.pprint(dict(count_tags(osm_sanjose)))
```

```
{'bounds': 1,
 'member': 21926,
 'nd': 2160823,
 'node': 1850355,
 'osm': 1,
 'relation': 2631,
 'tag': 768186,
 'way': 244809}
```

In [111]:
```
def get_user(element):
    tags = ["way", "node"]
    if element.tag in tags:
        return element.attrib['uid']
    else:
        return False

def process_map(filename):
    users = set()
    for _, element in ET.iterparse(filename):
        if get_user(element):
            users.add(get_user(element))
        pass

    return users

print "number of users:", len(process_map(osm_sanjose))
```

```
number of users: 1472
```

Problems Encountered in the Map

After getting familiarize with data, It's time to audit the OSM data and see if there is any problem. I will audit the address tags; more specifically, the 'tag' elements in which the 'k' attribute has these values: "addr:city", "addr:street" and "addr:postcode"

If I find any problem, I will try to clean the data as much as possible before importing it into my Database.

you can find all the source codes for this section in 'osm_audit.py'

Auditing City Names

In the 'audit_city' function, I will add the value of the tags with "addr:city" to a Set. These values are names of the cities in San Jose area.

In [6]:
```
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 tag.attrib['k'] == "addr:city":
                city_name = tag.attrib['v']
                city_names.add(city_name)
```

```
City Names: set(['cupertino', 'Sunnyvale, CA', 'San jose', 'Santa Clara',
'Moffett Field', 'Felton', 'campbell', 'Los Gato', 'Milpitas', 'Mountain
View', 'san Jose', 'Fremont', 'Campbelll', 'Coyote', 'San Jose', 'SUnnyva
le', u'San Jos\xe9', 'Saratoga', 'san jose', 'Los Gatos, CA', 'Sunnyval
e', 'Alviso', 'Mt Hamilton', 'Santa clara', 'Cupertino', 'los gatos', 'sa
nta clara', 'santa Clara', 'Morgan Hill', 'New Almaden', 'Los Gatos', 'su
nnyvale', 'Campbell', 'Redwood Estates'])
```

The list of cities in San Jose area are very limited as expected. There are couple of problems with city names:

- Typos: "Los Gato" instead of "Los Gatos" or "Campbelll" instead of "Campbell"
- Names with their first letter in lowercase: "san jose"

I use following mapping dictionary and 'update_city' function to modify the incorrect city names:

In [7]:
```
city_mapping={'cupertino':'Cupertino', 'Sunnyvale, CA':'Sunnyvale', 'campbel
              'Los Gato': 'Los Gatos','San jose': 'San Jose','san Jose':'San
              'Campbelll':'Campbell', 'SUnnyvale':'Sunnyvale', u'San Jos\xe9
              'san jose':'San Jose', 'Los Gatos, CA':'Los Gatos',
              'Mt Hamilton':'Mount Hamilton', 'Santa clara':'Santa Clara',
              'los gatos':'Los Gatos', 'santa clara':'Santa Clara', 'santa C
              'sunnyvale':'Sunnyvale'}

def update_city(name, mapping):
    if name in mapping:
        return mapping[name]
    else:
        return name
```

Auditing Street Names

I made a following list of expected street types:

In [8]:
```
expected_types = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place"
            "Road", "Trail", "Parkway", "Commons", "Alley", "Expressway"]
```

Then I wrote a function to audit all the street names against this expected names; if the audited name is not in the Expected names, I will add it to the 'unexpected_types' which is a dictionary. Its keys are the unexpected street types, and its values are Set of street names with that unexpected types. Here is part of the function and the resulted dictionary:

In [ ]:
```
if elem.tag == "node" or elem.tag == "way":
    for tag in elem.iter("tag"):
        if tag.attrib['k'] == "addr:street":
            st_name = tag.attrib['v']
            st_type = street_type(st_name)
            if st_type not in expected_types:
                unexpected_types[st_type].add(st_name)
```

In [ ]:
```
'Alameda': set(['The Alameda']),
 'Ave': set(['1425 E Dunne Ave',
             'Blake Ave',
             'Cabrillo Ave',
             'Cherry Ave',
             'E Duane Ave',
             'Foxworthy Ave',
             'Greenbriar Ave',
             'Hillsdale Ave',
             'Hollenbeck Ave',
             'Meridian Ave',
             'N Blaney Ave',
             'Saratoga Ave',
             'Seaboard Ave',
             'The Alameda Ave',
             'W Washington Ave',
             'Walsh Ave',
             'Westfield Ave']),
 'Barcelona': set(['Calle de Barcelona']),
 'Bascom': set(['S. Bascom']),
 'Bellomy': set(['Bellomy']),
 'Blvd': set(['Los Gatos Blvd',
              'McCarthy Blvd',
              'Mission College Blvd',
              'N McCarthy Blvd',
              'Palm Valley Blvd',
              'Santa Teresa Blvd',
              'Stevens Creek Blvd']),
 'Blvd.': set(['Los Gatos Blvd.']),
 'Boulvevard': set(['Los Gatos Boulvevard'])
```

After running this function for the first time, I found out that there are many streets names with 'Way', 'Circle', 'Terrace' and 'Expressway' as their types in the OSM. All of these types are legitimate and my 'expected_list' does not include them. So, I added them to the 'expected_types' list:

In [11]:
```
expected_types = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place"
                  "Lane", "Road", "Trail", "Parkway", "Highway", "Commons",
                  "Expressway", "Way", "Circle", "Terrace"]
```

The other major problems with street types are:

- Abbreviations: 'Blvd' for 'Boulevard' or 'Rd' for 'Road'
- Lower case types: “street” for “Street” or “court” for “Court”
- Typos: “Boulvevard” for Boulevard
- Missing types in which the type of the street is not mentioned in the street name: 'S. Bascom'

- extra address information added after the street type: “Stewart Drive Suite #1” or “West Evelyn Avenue Suite #114”

To deal with first three problems, I corrected the unexpected street types by using the following mapping dictionary and 'update_street_type' function:

In [12]:
```
street_mapping = { "St": "Street", "St.": "Street", "street": "Street", "Blv
                   "Blvd": "Boulevard", "Rd.": "Road", "Rd": "Road", "Ct.": '
                   "Ct": "Court", "Ave": "Avenue", "Ave,": "Avenue", "ave": '
                   "Cir" : "Circle", "Dr" : "Drive", "Ct": "Court", "ct": "Co
                   "court": "Court", "Rd": "Road", "Hwy": "Highway", "Ln": "I

def update_street_type(name, mapping):
    st_type = street_type(name)
    if st_type in street_mapping:
        modified_name = street_type_re.sub(mapping[st_type], name)
    else:
        modified_name = name

    return modified_name
```

Auditing Postal Codes

In United States, zip codes are five-digit code plus four additional digits to identify a geographic segment within the five-digit delivery area. Most of the time, this second part is not mentioned in postal codes, but if it is, it would be separated with a dash '-' from the first part.

The first three digits of the five-digit code for San Jose area are one of these: '940', '945', '950', '951'.

Using regular expressions, I wrote a function 'is_only_digit' to check whether or not a postal code contains only digits and dash. I, also wrote the 'out_of_range' function to check if the first three digits of postal code are in the right range. Part of 'audit_postcodes' function code and its output result is presented here:

In [13]:
```
if elem.tag == "node" or elem.tag == "way":
    for tag in elem.iter("tag"):
        if tag.attrib['k'] == "addr:postcode":
            postcode = tag.attrib['v']
            if not is_only_digit(postcode):
                problem_postcode.add(postcode)
            elif out_of_range(postcode):
                problem_postcode.add(postcode)
```

```
set(['CUPERTINO', 'CA 95054', 'CA 94035', '95014-2143;95014-2144', 'CA 95
110', 'CA 95116', 'CA 94085', '95914', 'CA 94086', u'94087\u200e'])
```

Fortunately there are quite a few postal codes with invalid format or value:

- One out range value: '95914'
- One with a city name instead of postal code: 'CUPERTINO'
- Five postal codes that start with 'CA': 'CA 95054'
- Two with invalid format: '95014-2143;95014-2144', u'94087\u200e'

For the first two cases I am curious if I can find some more information about them, like address or house number; then, using the address information, I could probably find the right postal codes in the Google Map! I wrote a function named 'find_postcode_addr' which finds house number and street address of a given postal code, if available.

Fortunately, the addresses are available: '20632 Cleo Avenue' & '10330 North Wolfe Road'; after checking them on Google Map, I found that the postal code for both of them is: '95014'

To deal with the postal codes that start with 'CA', I used Regular Expression to simply ommit the 'CA' part from the postal code. I also used Regular Expression in the 'five_digit_postcode' function to modify the extended postal codes to a simple five-digit postal code.

Finally, I integrated all above modifications in the 'update_postcode' function to correct the format and value of postal codes.

In [27]:
```
CA_INCLUDED = re.compile(r'CA ')
TWO_PART_POSTCODE = re.compile(r"(.+?)-(.+)")

def five_digit_postcode(postcode):
    m = TWO_PART_POSTCODE.match(postcode)
    if m:
        return m.group(1)
    else:
        return postcode
```

Building The SQLite Database

In this section I will parse the elements in the OSM XML file and transform them from document format to tabular format; this will make it possible to write them into .csv files. These csv files can be easily imported to a SQL database as tables

Tabular format

I will use the 'shape_element' function to transform the OSM XML elements to the tabular format based on the schema that is provided in source files by the name of 'schema.py'. This schema is designed to match with the Table Schema (https://gist.github.com/swwelch/f1144229848b407e0a5d13fcb7fbbd6f) in the SQL database

'shape_element' function takes an element as input. If the element top level tag is "node" the returned dictionary should have the format {"node": .., "node_tags": ...} If the element top level tag is "way" the dictionary should have the format {"way": ..., "way_tags": ..., "way_nodes": ...}

Before writing an element into this format, 'shape_element' would clean the input data if it is necessary. You can find the source code of 'shape_element' in xml_to_csv.py

Writing the tranformed data into CSV files

'process_map' function Iteratively process each XML element and write it to the corresponding csv file through the following process: first, 'get_element' function will parse through the OSM file and hands over elements to 'shape_element' function which shapes each element to tabular format. Then, 'validate_element' function will use Cerberus Validator to check if the tabular format is consistent with the Schema. Finally, an extended version of csv.DictWriter is used to write the validated data into the corresponding csv file (You can find the source code of these functions in xml_to_csv.py) Here is part of the 'process_map' function:

In [24]:
```
for element in get_element(file_in, tags=('node', 'way')):
    el = shape_element(element)
    if el:
        if validate is True:
            validate_element(el, validator)
        if element.tag == 'node':
            nodes_writer.writerow(el['node'])
            node_tags_writer.writerows(el['node_tags'])
        elif element.tag == 'way':
            ways_writer.writerow(el['way'])
            way_nodes_writer.writerows(el['way_nodes'])
            way_tags_writer.writerows(el['way_tags'])
```

Since the cerberus.validator is very slow, first I used the smaller version(4.6 MB) of OSM file, to check if the generated tabular data is valid. After that, I turned off the validator and apply the 'process_map' function to the main OSM file (402.5 MB)

Building the Database

First I created the 'osm_sanjose_db' SqLite database with 5 tables based on this Table Schema (https://gist.github.com/swwelch/f1144229848b407e0a5d13fcb7fbbd6f).

Finally, using the following commands, I import each of the 5 CSV files into the corresponding table in the database:

```
sqlite> .mode csv
sqlite> .import name.csv table_name
```

CSV header issue!

When I first tried to import the nodes.csv file into the nodes table, I got a "type mismatch" error. After doing more investigation on this error, I realized that the problem is with CSV header line (field names). So, I made the csv files another time but this time without writing the header into them. The problem solved!

Overview of the Data

In this section I use some SQL queries to find some basic statistics about the dataset

File Sizes

- san-jose_california.osm : 402.5 MB
- sanjose_osm.db.db : 213.2 MB
- nodes.csv : 155.7 MB
- nodes_tags.csv : 3.2 MB
- ways.csv : 14.7 MB
- ways_tags.csv : 23.1 MB
- ways_nodes.cv : 51.5 MB

Number of nodes

In [78]:
```
QUERY = "SELECT COUNT(*) FROM nodes;"
c.execute(QUERY)
rows = c.fetchall()
print rows[0][0]
```

```
1850355
```

Number of tags

In [38]:
```
QUERY = "SELECT COUNT(*) FROM ways;"
c.execute(QUERY)
rows = c.fetchall()
print rows[0][0]
```

```
244809
```

Number of unique users

In [117]:
```
QUERY = """SELECT COUNT(DISTINCT(e.uid)) as unique_users
FROM (SELECT uid FROM nodes UNION SELECT uid FROM ways) e;"""
c.execute(QUERY)
rows = c.fetchall()
print rows[0][0]
```

```
1472
```

Top contributors

In [93]:
```
QUERY = """SELECT e.user, COUNT(*) as num
FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
GROUP BY e.user
ORDER BY num DESC
LIMIT 10;"""
c.execute(QUERY)
rows = c.fetchall()
print rows
```

```
[(u'andygol', 295776), (u'nmixter', 282921), (u'mk408', 135059), (u'Bike
Mapper', 94277), (u'samely', 80771), (u'RichRico', 75726), (u'dannykath',
73961), (u'MustangBuyer', 64756), (u'3vivekb_sjsidewalks_import', 64320),
(u'karitotp', 62180)]
```

Number of users contributing less than 10 times

> Almost half of the users contribute less than 10 times

In [112]:
```
QUERY = """SELECT COUNT(*) as num FROM
    (SELECT e.user, COUNT(*) as num
     FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
     GROUP BY e.user
     HAVING num < 10);"""
c.execute(QUERY)
rows = c.fetchall()
print rows[0][0]
```

```
700
```

Top ten amenities

In [58]:
```
QUERY = """SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key = 'amenity'
GROUP BY value
ORDER BY num DESC
LIMIT 10;"""
c.execute(QUERY)
rows = c.fetchall()
print rows
```

```
[(u'restaurant', 891), (u'fast_food', 428), (u'bench', 318), (u'cafe', 26
2), (u'bicycle_parking', 208), (u'place_of_worship', 170), (u'toilets', 1
61), (u'school', 141), (u'bank', 129), (u'parking_space', 128)]
```

> Total number of amenities is 4459 and bout 37 percent of them are food places (restaurant, fast-food and cafe)

In [101]:
```
QUERY = """SELECT COUNT(*) as num
FROM nodes_tags
WHERE key = 'amenity';"""
c.execute(QUERY)
rows = c.fetchall()
print rows[0][0]
```

```
4459
```

All Religions

In [119]:
```
QUERY = """SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key='religion'
GROUP BY value
ORDER BY num DESC;"""
c.execute(QUERY)
rows = c.fetchall()
print rows
```

```
[(u'christian', 176), (u'jewish', 6), (u'buddhist', 3), (u'muslim', 2),
(u'caodaism', 1), (u'hindu', 1), (u'rosicrucian', 1), (u'scientologist',
1), (u'shinto', 1), (u'sikh', 1), (u'unitarian_universalist', 1), (u'zoro
astrian', 1)]
```

Top Cuisines and Restaurants

In the first query I search for the top 25 cuisines in the San Jose area. Vietnamese and Mexican cuisines are by far more frequent than the others. (Personally I thought number of Mexican food places should be way more than Vietnamese places!)

In [103]:
```
QUERY = """SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key='cuisine'
GROUP BY value
ORDER BY num DESC
LIMIT 25;"""
c.execute(QUERY)
rows = c.fetchall()
print rows
```

```
[(u'vietnamese', 121), (u'mexican', 116), (u'sandwich', 97), (u'pizza', 9
2), (u'chinese', 88), (u'coffee_shop', 78), (u'japanese', 47), (u'india
n', 45), (u'burger', 43), (u'american', 38), (u'italian', 35), (u'thai',
31), (u'ice_cream', 28), (u'sushi', 24), (u'mediterranean', 17), (u'chick
en', 16), (u'korean', 16), (u'asian', 11), (u'seafood', 11), (u'juice',
9), (u'barbecue', 8), (u'greek', 8), (u'bagel', 7), (u'steak_house', 7),
(u'regional', 6)]
```

Then I wrote a query to only find most frequent restaurants. This new ranking is obviously different from the previous one: Vietnamese restaurant is still on top, Chinese is second (it had the fifth place in the previous ranking) and Mexican is third. While more than 76 percent of the Chinese food places are restaurants, only 54 percent of Mexican food places are restaurants; as a result, Mexicans have a lower ranking when it comes to number of restaurants!

In [120]:
```
QUERY = """SELECT value, COUNT(*) as num
FROM nodes_tags
JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='restaurant') i
    ON nodes_tags.id=i.id
WHERE key='cuisine'
GROUP BY value
ORDER BY num DESC
LIMIT 25;"""
c.execute(QUERY)
rows = c.fetchall()
print rows
```

```
[(u'vietnamese', 81), (u'chinese', 67), (u'mexican', 63), (u'pizza', 58),
(u'japanese', 44), (u'indian', 32), (u'italian', 31), (u'american', 28),
(u'thai', 28), (u'sushi', 23), (u'sandwich', 18), (u'korean', 14), (u'med
iterranean', 14), (u'burger', 12), (u'seafood', 9), (u'barbecue', 7),
(u'steak_house', 7), (u'greek', 6), (u'ice_cream', 6), (u'regional', 5),
(u'asian', 4), (u'breakfast', 4), (u'chicken', 3), (u'french', 3), (u'int
ernational', 3)]
```

Other Ideas About the Datasets

Additional Data Exploration

Top Cities

> In this section I first find the ranking of cities by the frequency that they appear on OSM of San Jose area. The numbers are hugely skewed. While Sunnyvale frequency is 3440, the second position in ranking is city of San Jose with frequency of just 1232! (76 percent of all city names are these two)

In [2]:
```
QUERY = """SELECT value, COUNT(*) as num
FROM (select * from nodes_tags union select * from ways_tags)
WHERE key = 'city'
GROUP BY value
ORDER BY num DESC;"""
c.execute(QUERY)
rows = c.fetchall()
print rows
```

```
[(u'Sunnyvale', 3440), (u'San Jose', 1232), (u'Morgan Hill', 403), (u'San
ta Clara', 338), (u'Saratoga', 233), (u'Los Gatos', 149), (u'Milpitas', 1
12), (u'Campbell', 96), (u'Cupertino', 65), (u'Alviso', 11), (u'San Jos\x
e9', 11), (u'Mountain View', 7), (u'Coyote', 1), (u'Felton', 1), (u'Fremo
nt', 1), (u'Moffett Field', 1), (u'Mount Hamilton', 1), (u'New Almaden',
1), (u'Redwood Estates', 1)]
```

In [123]:
```
QUERY = """SELECT COUNT(*) as num
FROM (SELECT * FROM nodes_tags UNION SELECT * FROM ways_tags)
WHERE key = 'city';"""
c.execute(QUERY)
rows = c.fetchall()
print rows[0][0]
```

```
6104
```

Top Postal Codes

There are 13144 postal codes in OSM dataset(Number of all tags with 'postcode' as their key)

The frequencies for different postal codes are hugely skewed. Among 56 different postal codes in OSM of San Jose area, the '95014' postal code is repeated 9924 times out of total number of 13144. It means more than 75 percent of all the existing postal codes belong to a single area!

'95014' area code is just a small part of the San Jose area. It means that the contributions to the other parts of the San Jose area in OSM data is very limited and the OSM data for these areas is very incomplete. This is consistent with what we previously saw about the frequency of Top Cities.

In [125]:
```
QUERY = """SELECT COUNT(*) as num
FROM (SELECT * FROM nodes_tags UNION SELECT * FROM ways_tags)
WHERE key = 'postcode';"""
c.execute(QUERY)
rows = c.fetchall()
print rows[0][0]
```

```
13144
```

In [126]:
```
QUERY = """select value, count(*) as num
from (select * from nodes_tags union select * from ways_tags)
where key = 'postcode'
group by value
order by num desc
;"""
c.execute(QUERY)
rows = c.fetchall()
print rows
print len(rows)
```

```
[(u'95014', 9924), (u'95037', 420), (u'95070', 239), (u'94087', 211),
(u'94086', 203), (u'95051', 178), (u'95129', 169), (u'95127', 119), (u'95
054', 114), (u'95035', 109), (u'95125', 103), (u'95135', 102), (u'95008',
95), (u'95126', 95), (u'95030', 90), (u'95128', 90), (u'95134', 88), (u'9
5050', 85), (u'95112', 75), (u'94085', 74), (u'95113', 67), (u'95123', 5
7), (u'95110', 56), (u'94089', 50), (u'95032', 47), (u'95118', 40), (u'95
140', 38), (u'95002', 25), (u'95131', 25), (u'95124', 17), (u'95119', 1
6), (u'95120', 16), (u'95116', 12), (u'95136', 11), (u'95117', 10), (u'95
133', 10), (u'95130', 8), (u'95121', 7), (u'95122', 7), (u'95138', 7),
(u'95111', 6), (u'95132', 6), (u'95033', 5), (u'95148', 4), (u'94088',
2), (u'94538', 2), (u'94024', 1), (u'94035', 1), (u'94084', 1), (u'9480
7', 1), (u'95013', 1), (u'95052', 1), (u'95086', 1), (u'951251', 1), (u'9
5141', 1), (u'95191', 1)]
56
```

Suggestions

The results of the past two explorations clearly show that the data on some areas in compare to others is very incomplete. Since OSM has a tradition of making as few rules as possible, we should find a way to encourage the contributors to pay more attention to less complete areas;

There are some Monitoring Tools (https://wiki.openstreetmap.org/wiki/Quality_assurance#Monitoring_tools) already available in OSM which help to monitor the local community; like “Find Nearby Contributors” or “Find local changset discussions” in which we can encourage others to contribute more to some specific areas. We can also do so by attending local community meetups, post to a regional mailing list or forum section, or message some or all of the local contributors directly.

There are some cons and pros for this way of encouraging local contributors:

Benefits: It is using the tools that are already available in OSM; also, we are not making any new rules or limitations.

Problems: Since this method is totally dependent on how other local contributors react to our requests, it could be a very slow process and the results are not guaranteed

Conclusion

There are many inconsistencies in address names in San Jose OSM data, especially when it comes to street names! but the good news is with a good data processor a significatn portion of these inconsistencies could be cleaned.

Another major problem is that the data is obviously incomplete. As I discussed in the previous section, about 75 percent of all available postal codes in OSM data belongs to a small part of the San Jose area; or 76 percent of all city names are either 'Sunnyvale' or 'San Jose'. It means, most of the users contributed just to a very few parts of the San Jose area.