OpenStreetMap Data Case Study

Map Area

San Jose, CA, United States

- Map Area In OpenStreetMap
- Documents From Mapzen(OSM file for this project)

This map is of where I live now. I like Bay Area so much and so I moved to Mountain View right after my graduation in Cleveland, OH, but I still support Cavaliers! I am eager to get familiar to where I newly move in and I am so grad I can take this chance to contribute to the improvements.

Problems Encountered in the Map

After downloading the dataset, I took a small sample size of the San Jose area and ran it against *audit.py*, I found four problems in the data.

- Street names: overabbreviated and misspelled street names and mixed street names with units
  - Blake Ave Ave to Avenue)
  - Los Gatos Boulvevard (Boulvevard to Boulevard)
  - West Evelyn Avenue Suite #114 (Suite #114 should be removed)
- City names: misspelled city names, inconsistent case sensitivity and inconsistent format
  - Los Gato (Los Gato to Los Gatos)
  - Santa clara, SUnnyvale (clara to Clara, SUnnyvale to Sunnyvale)
  - San Jos\xe9 (should be San Jose)
- Phone number: inconsistent formats
  - Various formats:(408) 238-2086,(1)(408) 766-7000,(408)-252-5299,+ 408 980 6400, +1 408 2626949, etc..
  - All these different formats should be unified
- Postal code: inconsistent formats
  - Various formats: CA 95116,95134-1358,'95014-2143;95014-2144'u'94087\u200e',
  - The data has various formats of postal codes, which should be unified.

Before updating all the data and fixing problems above, I ran *audit.py* to read over the entire dataset and find out all different types of expressions and formats. And so I can create different mappings and methods of compilations to cover all different formats.

Street Names

To correct street names, I iterated each word in an address and correct it to its corresponding mapping in *audit.py* using the following function:

```
def update_street_name(name):
    name = name.title()
    if "," in name:
```

```
        name = name.split(',')[0] # remove city and states
    name_array = name.split(' ')
    for i in range(len(name_array)):
        if name_array[i] in street_mapping:
            if name_array[i].lower() not in ['suite','ste','ste.']: #letter after
suites represent its corresponding building, not an abbreviation
                name_array[i] = street_mapping[name_array[i]]
            else:
                name_array[i] = street_mapping[name_array[i]]
                break
    return ' '.join(name_array)
```

This updated all street names, such as Great America Pkwy ste 201 to Great America Parkway Suite 201

City names

Since San Jose has a certain number of counties, after setting each county name in a common format, like case sensitivity, I only have couple city names out of expectations through the whole dataset and so I will have these names corrected by mapping.

Phone number

I set up a format that all phone numbers should follow, which is like "+1 408 200 4868". Since some of the phone numbers had letters representing numbers, I replaced letters back to numbers. After making sure all numbers are digits, I cleaned phone numbers by removing all symbols, including spaces, not digits, and I would have numbers with consecutive 10 digits or 11 digits, which I edited then to the format for phone numbers.

```
def update_phone_num(num):
    m = phone_type_re.match(num)
    keypad =
{'2':'ABCabc','3':'DEFdef','4':'GHIghi','5':'JKLjkl','6':'MNOmno','7':'PQRSpqrs','8
':'TUVtuv','9':'WXYZwxyz'}
    if m is None:
        if re.search(r'[a-zA-Z]',num) is not None: #transfer letters to digits
            for key in keypad:
                num = re.sub(r'['+keypad[key]+']',key,num)
        if '-' in  num:   # Remove all possible punctuations
            num = re.sub('-','',num)
        if '.' in num:
            num = re.sub('\.','',num)
        if '(' in num or ')' in num:
            num = re.sub('[()]','',num)
        num = re.sub('\s+','',num)
        if re.search(r'\d{11}',num) is not None:
            num = re.search(r'\d{11}',num).group()
            num = num[:1] + ' ' + num[1:4] + ' ' + num[4:7] + ' ' +num[7:]
        if re.search(r'\d{10}',num) is not None:
            num = re.search(r'\d{10}',num).group()
            num = num[:3] + ' ' + num[3:6] + ' ' +num[6:]
        if re.match(r'\d{3}\s\d{3}\s\d{4}', num):
```

```
            num = '+1 ' +num
        if re.match(r'1\s\d{3}\s\d{3}\s\d{4}',num):
            num = '+'+ num
        if re.match(r'\+1\d{10}$',num) is not None:
            num = num[:2] + ' ' + num[2:5] + ' ' + num[5:8] + ' ' +num[8:]
    return num
```

Postal code

For the part of postal code, I only kept the postal codes with 5 digits, instead of the one with 9 digits. Since not all postal codes have 9 digits, in order to keep a common format, I removed the last 4 digits if they have 9.

For data cannot be updated

After updating all the data, I audited it again and found some data cannot be updated. I found "+1","2924779" in phone number and 'CUPERTINO' in postal code. Data like this might not be able to be updated unless it has more information. However, I will still keep it and move it to the database since at least, we can still dig some information from them, which, though, are not perfect.

Overview of the data

This section contains basic statistics about the dataset, the SQL queries used to gather them, and some additional ideas about the data in context.

File sizes

```
san-jose_california.osm ......... 343 MB
mydb.db ......................... 204 MB
nodes.csv ....................... 133 MB
nodes_tags.csv .................. 2.82 MB
ways.csv ........................ 13.0 MB
ways_tags.csv ................... 20.3 MB
ways_nodes.cv ................... 45.7 MB
```

Number of unique users

```
SELECT COUNT(DISTINCT(e.uid)) AS Count_of_users
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e;')
```

```
Count_of_users
1334
```

Number of nodes

```
SELECT COUNT(*) as Number_of_Nodes FROM nodes
```

```
Number_of_Nodes
1657147
 ```
Number of ways
```

SELECT COUNT(*) as Number_of_Ways FROM ways; ```

```
Number_of_Ways
226154
```

Top 10 users of contribution

```
SELECT e.user as User, COUNT(*) as Number_of_Contributions
        FROM (SELECT user FROM Nodes UNION ALL SELECT user FROM Ways) e
        GROUP BY e.user
        ORDER BY Number_of_Contributions DESC
        LIMIT 10;
```

```
User           Number_of_Contributions
andygol         295728
nmixter         285524
mk408           148037
Bike Mapper     90980
samely          81414
RichRico        75663
dannykath       73014
karitotp        62436
MustangBuyer    51667
Minh Nguyen     43793
```

Number of one-time contributors

```
SELECT count(*) as first_time_contribution_user
FROM    (select e.user,count(*) as count
        From (select user from nodes union all select user from ways) e
        group by e.user
        having count = 1);
```

```
first_time_contribution_user
289
```

Other ideas about the dataset

I found the some types of data had various formats. I think that users could have suggestions while they are input data. For example, we can have various ways to type in a phone number, but one of
```

them could be suggested and so to reduce the number of formats. San Jose is a city easy to find a parking spot, but the dataset seems to not have that many parking lot. I found a total of 469 nodes with parking information and that still have a spaces to improve. While users type in information of parking, they could have a checkbox to check if they have parking lot, if so they can detail what kind of parking it has. So we have a key named parking with a value yes or no.

```
SELECT nodes_tags.key, nodes_tags.value, count(*) as num
FROM nodes_tags
where value like '%parking%'
GROUP BY nodes_tags.value
ORDER BY num DESC
limit 40
```

```
key                                 value                               num
amenity                              bicycle_parking                      186
amenity                              parking_space                        90
amenity                              parking_entrance                     86
amenity                              parking                              83
name                                 Apartment Visitor Parking            10
vending                              parking_tickets                      4
name                                 Adobe Employee Parking               1
name                                 Construction Parking Only            1
name                                 ES03 parking                         1
name                                 Earl Carmichael Park Parking         1
fixme                                Entrance to underground parking. Needs to be
mapped. 1
name                                 Parking spot 1                       1
name                                 San Jose Parking                     1
name                                 VIP Bike Parking                     1
name                                 VIP Parking                          1
service                              parking_aisle                        1
```

Conclusion

The data from OpenStreetMap is raw, but it contains a lot of information, though with various formats. We can still find distributions of amenities, parking information, contributions from users, etc. However, we can still see the dataset does not have additional information about amenities, tourist attractions and other popular places. As I said before, we can add restrictions or suggestions for users while they are editing the dataset, which can help them reduce the chance to have typo or keep the format of input the same. We may need to add more tags options, like a link to wiki for more historic information about a popular place. Generally speaking, the dataset has what a map should have in base, but more information or options could be added in to each node to make it more friendly to users.