OpenStreetMap Data Case Study

Map area

San Francisco, CA, Unite States http://metro.teczno.com/#san-francisco (http://metro.teczno.com/#san-francisco) http://www.openstreetmap.org/relation/111968 (http://www.openstreetmap.org/relation/111968)

This map is the map of one of my favorite city , so I'm more interested to see what database querying reveals, and I'd like an opportunity to contribute to its improvement on OpenStreetMap.org.

Problems Encountered in the Map

After downloading and unziping the whole file of the San Francisco city, I found that the whole isn't too big to test and run(around 300 MB). So, I just explored, test and aduit the whole osm file of San Francisco. I noticed three main problems with the data, I will address them in the following order:

- Inconsistent street type (eg: 'Avenue', 'Ave', 'Ave.') and typos(eg: 'Boulavard', 'Boulvard')
- Inconsistent postal codes (eg: '94002','94002-3585','CA 94544','CA:94103') and totally wrong postal codes('515')
- Inconsistent city names (eg: 'Berkeley','Berkeley, CA') , unnecessary dents (eg:'Fremont '), inconsistent capitalization(eg: 'san francisco', 'San Francisco'), typos(eg: "San Fransisco")

Inconsistent street type

According to the data, there are some traditional names of street which are widely used within San Francisco, therefore I updated the expected street name list which used in course's quiz, as this:

```
expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "Lane", "Road",
            "Trail", "Parkway", "Commons", "Way", "Highway", "Path", "Terrace", "Alley", "Center",
            "Circle", "Plaza", "Real"]
```

Then I audit the street type of the data agasin the expected list, recording the wrong street types and names. After I inspected the whole wrong street types, I made the decision which wrong types are just amendable abbreviations or typos, others are just total wrong information. I also updata the mapping list which was used to correct these amenbable types in the later part.

```
street_mapping = { "St": "Street",
                   "St.": "Street",
                   "Steet": "Street",
                   "st": "Street",
                   "street": "Street",
                   "Ave": "Avenue",
                   "Ave.": "Avenue",
                   "ave": "Avenue",
                   "avenue": "Avenue",
                   "Rd.": "Road",
                   "Rd": "Road",
                   "Blvd": "Boulevard",
                   "Blvd,": "Boulevard",
                   "Blvd.": "Boulevard",
                   "Boulavard": "Boulevard",
                   "Boulvard": "Boulevard",
                   "Dr": "Drive",
                   "Dr.": "Drive",
                   "Pl": "Plaza",
                   "Plz": "Plaza",
                   "square": "Square"
                   }
```

In the data wrangling part, I programmatically correct those amendable data into right format using:

```
if k == "addr:street":
    m = street_type_re.search(tag.attrib['v'])
    if m:
        street_type = m.group()
        if street_type in error_street_type:
            if street_type in street_mapping:
                tag.attrib['v']=tag.attrib['v'].replace(street_type,street_mapping[
street_type])
    else:
        continue
        node_tags['key']=k.split(':')[1]
        node_tags['type']=k.split(':')[0]
        node_tags['id']=element.attrib['id']
        node_tags['value']=tag.attrib['v']
```

Postal codes

The types of postal codes are much less than street types, so I didn't have a expected list to inspect the postal codes data. I just looked through all the types of postal codes, and made the audit decision, and created the mapping list for correcting postal codes.

```
postcode_mapping={"CA 94030": "94030",
                  "CA 94133": "94133",
                  "CA 94544": "94544",
                  "CA 94103": "94103"
                   }
```

In this list, I just unified the postal codes without the specification of 'CA'. Meanwhile, I also made a total wrong postal code list:

```
error_postcode={'1087', '515'}
```

Finally, I just ignored this format '94002-3585', and recognized it as a normal right format of postal code.

In the data wrangling part, I programmatically correct those amendable data into right format using:

```
if k == "addr:postcode":
    if tag.attrib['v'] in error_postcode:
        continue
    else:
        if tag.attrib['v'] in postcode_mapping:
            tag.attrib['v']=postcode_mapping[tag.attrib['v']]
        node_tags['key']=k.split(':')[1]
        node_tags['type']=k.split(':')[0]
        node_tags['id']=element.attrib['id']
        node_tags['value']=tag.attrib['v']
        tags.append(node_tags)
```

City names

Similarly, I also loojed through all the city names of the data, found out those totally wrong city names and those just in wrong format, and created the mapping list for correcting wrong formatted city name.

```
cityname_mapping={"Berkeley, CA": "Berkeley",
                  "Fremont ": "Fremont",
                  "Oakland, CA": "Oakland"
                  "Oakland, Ca": "Oakland"
                  "San Francisco, CA": "San Francisco"
                  "San Francisco, CA 94102": "San Francisco"
                   }
```

I deleted all the suffixs of city names, made them unified. And decided throwed out those totally wrong city names:

```
error_cityname={'155', '157'}
```

In the data wrangling part, I programmatically correct those amendable data into right format using:

```
if k == "addr:city":
    if tag.attrib['v'] in error_cityname:
        continue
    else:
        if tag.attrib['v'] in cityname_mapping:
            tag.attrib['v']=cityname_mapping[tag.attrib['v']]
        node_tags['key']=k.split(':')[1]
        node_tags['type']=k.split(':')[0]
        node_tags['id']=element.attrib['id']
        node_tags['value']=tag.attrib['v']
        tags.append(node_tags)
```

Building database and import data

Import csv files into different tables

Using import nodes and nodes' tags csv files as examples:

```
sqlite> .mode csv
sqlite> .import "/Users/yangrenqin/udacity/P3/nodes.csv" nodes
```

```
sqlite> .mode csv
sqlite> .import "/Users/yangrenqin/udacity/P3/nodes_tags.csv" nodes_tags
```

Using sql query to verify data after audit and update

Verify street types and names

```
import sqlite3
import pandas as pd
db = sqlite3.connect('sanfrancisco.db')
c=db.cursor()
query="SELECT tags.value, COUNT(*) as count\
 FROM (SELECT * FROM nodes_tags\
 UNION ALL\
 SELECT * FROM ways_tags) tags\
 WHERE tags.key='street'\
 GROUP BY tags.value\
```

```
 ORDER BY count DESC;"
c.execute(query)
rows=pd.DataFrame(c.fetchall(),columns=['Street','count'])
db.close()
```

Here are the top twenty results, beginning with the highest count:

Indext	Street	Count
0	El Camino Real	386
1	Jefferson Avenue	324
2	Roosevelt Avenue	270
3	Hudson Street	240
4	Woodside Road	223
5	Hamilton Avenue	220
6	Madison Avenue	206
7	Vera Avenue	201
8	Redwood Avenue	194
9	Kentfield Avenue	192
10	Martin Luther King Jr Way	177
11	King Street	175
12	Hoover Street	170
13	Oak Avenue	168
14	Valota Road	166
15	Hopkins Avenue	163
16	Brewster Avenue	162
17	University Avenue	157
18	Fulton Street	156
19	Bay Road	154

And last twenty results:

Index	Street	Count
1359	West MacArthur Boulevard	1
1360	West Parnassus Court	1
1361	West Ranger Avenue	1
1362	Westmoor Avenue	1
1363	Whittle Avenue	1
1364	William Saroyan Place	1
1365	Williams Street	1
1366	Willie Mays Plaza	1
1367	Wood Street	1
1368	Woodminster Lane	1
1369	Yacht Road	1
1370	Yosemite Avenue	1
1371	Youngs Valley Road	1
1372	Zoo Avenue	1
1373	central Avenue	1
1374	market Street	1
1375	pine Street	1
1376	shattuck Avenue	1
1377	townsend Street	1
1378	ygnacio Valley Road	1

Verify postal codes

```
db = sqlite3.connect('sanfrancisco.db')
c=db.cursor()
query="SELECT tags.value, COUNT(*) as count\
 FROM (SELECT * FROM nodes_tags\
 UNION ALL\
```

```
 SELECT * FROM ways_tags) tags\
 WHERE tags.key='postcode'\
 GROUP BY tags.value\
 ORDER BY count DESC;"
c.execute(query)
rows=pd.DataFrame(c.fetchall(),columns=['Postal code','count'])
db.close()
```

Here are the top and last ten results, beginning with the highest count:

Index	Postcode	count
0	94063	358
1	94587	250
2	94109	200
3	94103	192
4	94061	161
5	94114	129
6	94113	111
7	94110	65
8	94102	63
9	94107	63

Last ten results:

Index	Postcode	count
120	94549-5506	1
121	94552	1
122	94563	1
123	94606-3636	1
124	94612-2202	1
125	94621	1
126	94708	1

Index	Postcode	count
127	94720	1
128	94720-1076	1
129	95498	1

Verify city names

```
db = sqlite3.connect('sanfrancisco.db')
c=db.cursor()
query="SELECT tags.value, COUNT(*) as count\
 FROM (SELECT * FROM nodes_tags\
 UNION ALL\
 SELECT * FROM ways_tags) tags\
 WHERE tags.key='city'\
 GROUP BY tags.value\
 ORDER BY count DESC;"
c.execute(query)
rows=pd.DataFrame(c.fetchall(),columns=['City','count'])
db.close()
```

Since there are just around 40 cities, I just print out all the results:

Index	City	Count
0	Redwood City	23533
1	Berkeley	3358
2	Palo Alto	1651
3	San Francisco	1379
4	Union City	252
5	Burlingame	158
6	Oakland	143
7	San Mateo	38
8	Alameda	24
9	Walnut Creek	18
10	Albany	16

Index	City	Count
11	Hayward	15
12	Atherton	13
13	San Carlos	12
14	Fremont	11
15	Emeryville	8
16	Mill Valley	6
17	San Leandro	6
18	Belmont	4
19	Pacifica	4
20	Castro Valley	3
21	Daly City	3
22	Lafayette	3
23	Menlo Park	3
24	Piedmont	3
25	Richmond	3
26	Sausalito	3
27	Brisbane	2
28	Foster City	2
29	Half Moon Bay	2
30	Newark	2
31	San Bruno	2
32	East Palo Alto	1
33	El Cerrito	1
34	Greenbrae	1
35	Kensington	1

Index	City	Count
36	Kentfield	1
37	Marin City	1
38	Montara	1
39	Moraga	1
40	Orinda	1
41	Pleasant Hill	1
42	South San Francisco	1
43	Tiburon	1

According to the results I got from the database, I believe I've successfully audit, clean and correct some amendable data within the three aspects I've mentioned. The data from those three fields are proved to be internally consistent and verifiable, clean of typos and wrong informations, and contains all correct amendable data.

Data overview

This section contains basic statistics about the dataset, the sql queries used to gather them.

File sizes

```
san-francisco.osm ......... 326 MB
sanfrancisco.db ........... 211 MB
nodes.csv ................. 117 MB
nodes_tags.csv ............ 4.6 MB
ways.csv ..................... 9 MB
ways_tags.csv .............39.4 MB
ways_nodes.cv ............ 15.2 MB
```

Number of nodes

```
db = sqlite3.connect('sanfrancisco.db')
c=db.cursor()
query="SELECT COUNT(*) FROM nodes;"
c.execute(query)
result=c.fetchall()
db.close()
```

1410191

Number of ways

```
query="SELECT COUNT(*) FROM ways;"
```

(remaining parts are the same with above) 154315

Number of unique users

```
query="SELECT COUNT(DISTINCT(e.uid))\
 FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) as e;"
```

1289

Number of nodes with cafe

```
query="SELECT COUNT(*) as num FROM nodes_tags\
 WHERE value='cafe';"
```

541

Top 10 contributing users

```
query="SELECT e.user, COUNT(*) as num\
 FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) AS e\
 GROUP BY e.user\
 ORDER BY num DESC\
 LIMIT 10;"
```

User	Count
'oldtopos'	334076
'KindredCoda'	144613
'osmmaker'	140043
'DanHomerick'	119462
'nmixter'	77785
'woodpeck_fixbot'	46008
'StellanL'	43335
'oba510'	38785
'dchiles'	38472

User	Count
'Speight'	30346

Additional ideas

- For the three fields I've inspected, audited and cleaned: Within these fields, there are still at least few aspects that could take further explore, audit and clean. Like, I only audited and cleaned the street names by whether their street types are in correct format. However, there are still some street names, expect their street type which at the last part, are in wrong format or just totally wrong data. And, for the postal code, I also only check and unified their format, it's still possible for some postal codes lie out of the postal code range of San Francisco. Therefore, I just suggest to further audit and clean these fields against more detailed schema.
- For the fields I didn't wrangle and clean: Since I only explored `k:"addr:xxx"` part of data, including `"addr: street"`,`"addr: postcode"` and `"addr: city"`. There are many fields worthy to take inspect, audit and clean. Among many of them, I propose to explore `"user"` data. When I looked through the osm.file or csv.file of the data, I noticed that there are many potential wrong formats or typos or even totally wrong informations in the `"user"` attribute within many nodes or ways or tags block.Therefore, explore and clean for these data are definitely challenge but worth doing, since their represents customers' information, it's very important to show enough respect to customers. However, the potential problems within this exploration are the task load. Since every node, node's tag, way and way's tag have this attribute, the collection of data and running audit and cleaning program could be very time-consuming, and it would be very frustrating to debug this program.
- For GPS input: I noticed that GPS data usually represented by the Street names in second level `“k”` tags pulled from Tiger GPS data and divided into segments. As I looked through and inspected this part of data, I found that there is significant less typos, wrong street types or format problems. I supposed this is caused by GPS data processor, which is proved to be a potentially better way to collect map data. Therefore, I suggest that along with a precise GPS data processor in place and a pretty robust data processor similar or better than `'Data Wrangling part.py'`, which particularly working with auditing and cleaning the data collected by GPS. Although, Collecting data through GPS with its cleaning method would possibly input a great amount of cleaned data to OpenStreetMap.org, the challenges and problems would occur during developing the audit and clean method for GPS data are still uncertain.

Additional Data Exploration

Top 10 appearing amenities

```
query="SELECT value, COUNT(*) as num\
```

```
FROM nodes_tags\
WHERE key='amenity'\
GROUP BY value\
ORDER BY num DESC\
LIMIT 10;"
```

Amenity	Count
restaurant	1685
place_of_worship	903
school	805
fire_hydrant	698
post_box	588
cafe	540
bench	439
fast_food	354
drinking_water	333
bicycle_parking	272

Ten most popular cuisines

```
query="SELECT nodes_tags.value,COUNT(*) as num\
FROM nodes_tags,\
(SELECT DISTINCT(id) FROM nodes_tags WHERE value='restaurant') AS i\
WHERE nodes_tags.id=i.id AND nodes_tags.key='cuisine'\
GROUP BY nodes_tags.value\
ORDER BY num desc\
LIMIT 10;"
```

Cuisine	Count
mexican	122
italian	86
pizza	84
chinese	76
thai	68

Cuisine	Count
japanese	65
american	53
burger	47
indian	46
sandwich	44

Most popular cafe

```
query="SELECT nodes_tags.value,COUNT(*) as num\
 FROM nodes_tags,\
 (SELECT DISTINCT(id) FROM nodes_tags WHERE value='cafe') AS i\
 WHERE nodes_tags.id=i.id AND nodes_tags.key='name'\
 GROUP BY nodes_tags.value\
 ORDER BY num desc\
 LIMIT 10;"
```

Cafe	Count
Starbucks	68

(Not surprise ..)

Conclusion

Although I believe this dataset has been well cleaned for the purposes of this exercise within the three fields I've explored and cleaned, it's obvious that this dataset for San Francisco area is still incomplete and needs to be audit and clean much further. For these I didn't explored part, I am interested in explore `"user"` part and GPS data which would be possible to input a great amount of cleaned data to OpenStreetMap.org. I hope my work could make a little difference on the improve the OpenStreetMap.