Prj 3: Wrangle OpenStreetMap Data

San Diego, CA, United States:

- https://www.openstreetmap.org/export#map=8/32.292/-115.790
- https://mapzen.com/data/metro-extracts/metro/san-diego_california/

San Diego is my favourate city in the United States so I'd like an opportunity to contribute to its improvement on OpenStreetMap.org.

Problems encountered in the map

In this case, I used the audit.py file audit three keys, 'city', 'street', 'postcode' using a sample data. There are several problems of the data: 1. Thec OSM file contains cities other than San Diego, such as 'Bonita', 'Santee', 'Lemon Grove',etc. All of them are cities around San Diego. This auditing demonstrates that the metro extract not only includes the city itself, but also inclue surrounding cities in the sprawl. Since our focus in the city of San Diego, we filtered out the records outside San Diego using get_tages function which will be explained later. 2. Some of the street names are over abbreviated, such as the following words are included in a street name. * 'Av', 'Ave', 'Ave.' * 'Blvd','Blvd.' * 'Dr','Dr.' * 'Ln','Wy','Rd','St' 3. Inconsistent postal codes appeared in the data, such as "CA 91914", "91911-6124", "91902". Most of the post codes are numbers but some starts with 'CA'.

After the auditing process, I used the data.py file to clean the data as well as create .csv to be inserted into the SQL database. Some of the new function are created to clean the data.

```
def get_tags(element,idd): # filter out records not in San Diego
    tags = []
    for tag in element.iter('tag'):
        tag_dic = {}
        k = tag.attrib['k']
        v = tag.attrib['v']
        if PROBLEMCHARS.search(k):
            continue
        elif LOWER_COLON.search(k):
            tag_dic['type'] = k[:k.find(':')]
            tag_dic['key']  = k[k.find(':')+1:]
        else:
            tag_dic['type'] = 'regular'
            tag_dic['key']  = k

        if tag_dic['key'] == "city":  # check city value
            if not v in ["San Diego","san diego"]:
                continue

        tag_dic['id'] = idd
        tag_dic['value'] = update_value(k,v)
        tags.append(tag_dic)
    return tags
```

```
mapping = { "Av": "Avenue",'Ave': 'Avenue','Ave.': 'Avenue',\
'Blvd': 'Boulevard','Blvd.': 'Boulevard',\
'Dr': 'Drive','Dr.':'Drive','Ln': 'Lane','Rd':'Road',\
'Wy':'Way','St':'Street'}

def update_value(k,v): # update postcode and street values
    if k == "addr:postcode":
        if re.match(r'^\d{5}$', v): # chekc whether it is a standard zip code
            return v
        elif  re.match(r'^(\d{5})-\d{4}$', v): # check whether it is a nine digit
zip code
            return re.findall(r'^(\d{5})-\d{4}$', v)[0]
        elif v.startswith('CA '): # check whether it starts with 'CA'
            v = v[3:]
            return v
        return v
    if k == "addr:street":
        last_word = v.split()[-1] # check the last words of street name
        if last_word in mapping.keys():
            v = v.replace(last_word,mapping[last_word])
            return v
        return v
    return v
```

After this cleaning, the unique cities in the documents are [(u'San Diego',), (u'san diego',)]. Also, all of the post code are 5 digit numbers. The over-abbreviated street names also replaced with complete names.

Import .csv files to sqlite database

```
import sqlite3
import xml.etree.cElementTree as ET
import pprint
from collections import defaultdict
import os
```

The .csv files are created using data.py file. After preparing the data to be inserted into a SQL database, I used the DB-API to import the .csv files to the database san_diego.

```
def create_or_open_db(filename):
    file_exists = os.path.isfile(filename)
    conn = sqlite3.connect(filename)
    if file_exists:
        print ''' "{}" database successfully opened '''.format(filename)
    else:
        print ''' "{}" database successfully created '''.format(filename)
    return conn

conn = create_or_open_db('san_diego.sqlite')
cur = conn.cursor()
```

```
"san_diego.sqlite" database successfully created
```

I have created five tables in the database: Nodes, Node_tags, Ways, Ways_tages, Way_nodes. Here, I only presented the code for creating and inserting lines into Nodes as others follow the similar process.

```
conn = sqlite3.connect("san_diego.sqlite")
cur = conn.cursor()
QUERY = '''create table if not exists Nodes(
id INTEGER PRIMARY KEY,
lat float, lon float,
user TEXT, uid INTEGER,
version TEXT,
changeset INTEGER,
timestamp TEXT);'''
cur.execute(QUERY)

with open('nodes.csv','rb') as fin:
    # csv.DictReader uses first line in file for column headings by default
    dr = csv.DictReader(fin) # comma is default delimiter
    to_db = [(i['id'],
i['lat'],i['lon'],i['user'],i['uid'],i['version'],i['changeset'],i['timestamp'])
for i in dr]

conn = sqlite3.connect("san_diego.sqlite")
conn.text_factory = str
cur = conn.cursor()
cur.executemany("INSERT INTO Nodes (id,
lat,lon,user,uid,version,changeset,timestamp) VALUES (?, ?,?,?,?,?,?,?);", to_db)
conn.commit()
conn.close()
```

Overview of data

The size of the file are shown below.

```
$ ls -l san-diego_california.osm
```

- san-diego_california.osm : 307.601339 MB
- nodes.csv : 83.918036 MB
- nodes_tags.csv : 87.046370 MB
- ways.csv : 5.442807 MB
- ways_nodes.csv : 19.579910 MB
- ways_tage.csv : 20.394770 MB

Number of nodes:

```
conn = sqlite3.connect("san_diego.sqlite")
cur = conn.cursor()
QUERY = '''SELECT COUNT(*) FROM Nodes;'''
cur.execute(QUERY)
cur.fetchall()
```

[(1027374,)]

Number of ways:

```
QUERY = '''SELECT COUNT(*) FROM Ways;'''
cur.execute(QUERY)
cur.fetchall()
```

[(92011,)] Similarly, we got: * number of node tags: 1967445 * number of way tags: 535951 * number of way nodes: 816038

Number of unique users:

```
QUERY = '''SELECT COUNT(DISTINCT(N_W.uid))
FROM (SELECT uid FROM Nodes UNION ALL SELECT uid FROM Ways) N_W;'''
cur.execute(QUERY)
cur.fetchall()
```

[(1006,)]

Average post per user:

```
QUERY = '''SELECT AVG(num)
FROM (SELECT COUNT(*) as num
FROM (SELECT user FROM Nodes UNION ALL SELECT user FROM Ways) N_W
GROUP BY N_W.user);'''
cur.execute(QUERY)
cur.fetchall()
```

[(1110.500992063492,)]

Which users contribute the most:

```
QUERY = '''SELECT N_W.user, COUNT(*) as num
FROM (SELECT user FROM Nodes UNION ALL SELECT user FROM Ways) N_W
GROUP BY N_W.user
ORDER BY num DESC
LIMIT 5;'''
cur.execute(QUERY)
cur.fetchall()
```

The five top contributing uses are:

```
[(u'n76', 335123),
 (u'Adam Geitgey', 161979),
 (u'Sat', 128138),
 (u'woodpeck_fixbot', 91196),
 (u'Zian Choy', 16965)]
```

It is superising that there users are putting so much effort into it!

More problems encounted

- misplace of street and post code

Before exploring more in this database, I found a problem in a document who has a street name '92131', so I checked its other attributes in the tables.

```
QUERY = '''SELECT *
FROM  Ways_tags
WHERE id = (SELECT id FROM  Ways_tags
WHERE key='street' and value = '92131');'''
cur.execute(QUERY)
u = cur.fetchall()
```

Therefore, the street and post code are misplaced in this document.

```
[(132979649, u'building', u'yes', u'regular'),
 (132979649, u'street', u'92131', u'addr'),
 (132979649, u'postcode', u'Scripps Ranch Blvd.', u'addr'),
 (132979649, u'housenumber', u'10006', u'addr')]
```

I then used the update method in sqlite to swap the values of postcode and street.

```
QUERY1 = '''update Ways_tags
set value = '92131' WHERE key ='postcode';'''
cur.execute(QUERY1)
QUERY2 = '''update Ways_tags
set value = 'Scripps Ranch Blvd.' WHERE key='street';'''
cur.execute(QUERY2)
conn.commit()
```

- Other cities are included

Even through we filterd the documents with city other than San Diego, there are still lots of records that are in other cities. But this would not be a big problem as there are only 733 records in this category.

```
QUERY = '''SELECT value, count(*)
FROM  Node_tags
WHERE key = 'location' and value not like '%San Diego%'
group by value
order by count(*)
desc;'''
cur.execute(QUERY)
KEY = cur.fetchall()
```

```
(u'Orange, Coronado', 4),
 (u'H, Chula Vista', 3),
 (u'Palm And Palm, Imperial Beach', 3),
 (u'Palomar, Chula Vista', 3),
 (u'Brandywine, Chula Vista', 2),
 (u'Broadway And Palomar, Chula Vista', 2),
 (u'Campo And Kenwood, Unincorporated', 2),
 (u'Chula Vista', 2),
 (u'H And Tierra Del Rey, Chula Vista', 2),
 (u'Madison, El Cajon', 2), ...
```

- Unexpected key attributes

Some of the key attribute of the documents are unexpected and hard to interpret, such as 'fixme', 'de','el','he','hu','mn','ja','lt',etc. even when looking at the value associates with it. I looked at the values of the document whoes key contains 'fixme'.

```
QUERY = '''SELECT value
FROM  Node_tags
WHERE key like '%fixme%';'''
cur.execute(QUERY)
KEY = cur.fetchall()
```

Some values are as follows. It seems like these records are done by in person visit, rather than srippting information from other sources.

```
[(u'ref? Caltrans list is missing an exit',),
 (u'needs better tags for supermarket + pharmacy',),
 (u'map this nice little park',),
 (u'map this area',),
 (u'this location looks very wrong (based on bing)',),
 (u'location looks wrong (ref: bing maps)',),
 (u'this location is wrong. was in the middle of elem school',),
 (u"this doesn't look like the right loction (bing)",),
 (u'map this nice little park',),
 (u'check religion',),
 (u'Move to new address',),
 (u'location is wrong, it seems',),
 (u'location looks wrong (ref: bing maps)',),
```

After check the geolocations of some of these documents, I found that these documents are hard to be fixed because we don't know what are the most important information the person wanted to convey and how would these user want we to fix it. For example, the information associated with 'location looks wrong (ref: bing maps)' are shown below. Another document posted by the same user 'Sat' says 'this location looks very wrong (based on bing)'. The name attribute associated with that post is 'The Church of Jesus Christ of Latter-day Saints'. However, it is hard to know how to fix this document with the data available.

```
(358845344,  32.8313175,   -117.2161553,   u'Sat',   48060,   u'3',   14801803,
u'2013-01-27T01:32:35Z'),
(358860822,   32.8286761,   -117.2157428,   u'Sat',   48060,   u'3',   14801520,
u'2013-01-27T00:50:37Z')]
```

Additional ideas

I am interested in what are the sources of the data.

```
QUERY = '''SELECT value, count(*)
FROM  Node_tags
where key = "source"
group by value
order by count(*)
desc
limit 10;'''
cur.execute(QUERY)
u = cur.fetchall()
```

We see that the SanGIS website is the dominate data source.

```
[(u'SanGIS Addresses Public Domain (http://www.sangis.org/)', 342435),
 (u'SanDag Public Transit (http://www.sandag.org/)', 2161),
 (u'Bing', 675),
 (u'SanGIS Business Sites Public Domain (http://www.sangis.org/)', 651),
 (u'Yahoo', 640),
 (u'Caltrans', 316),
 (u'USGS Geonames', 177),
 (u'bing_imagery', 170),
 (u'survey', 106),
 (u'Direct observation / signage', 43)]
```

The top ten poplular amenities are:

```
QUERY = '''SELECT N_W.value, COUNT(*) as num
FROM  (SELECT * FROM Node_tags UNION ALL
      SELECT * FROM Ways_tags) N_W
WHERE key='amenity'
```

```
GROUP BY value
ORDER BY num DESC
LIMIT 10;'''
cur.execute(QUERY)
u = cur.fetchall()
```

```
[(u'parking', 2022),
 (u'place_of_worship', 948),
 (u'school', 595),
 (u'fast_food', 577),
 (u'restaurant', 549),
 (u'bar', 270),
 (u'cafe', 174),
 (u'fuel', 139),
 (u'toilets', 117),
 (u'bank', 116)]
```

The 10 different most popular shops are:

```
[(u'convenience', 167),
 (u'supermarket', 113),
 (u'clothes', 87),
 (u'hairdresser', 53),
 (u'car_repair', 32),
 (u'department_store', 30),
 (u'bicycle', 29),
 (u'mobile_phone', 26),
 (u'alcohol', 22),
 (u'beauty', 21)]
```

What do people like to eat in San Diego:

```
QUERY = '''SELECT value, count(*)
FROM  Node_tags
WHERE key = 'cuisine'
group by value
order by count(*)
desc;'''
cur.execute(QUERY)
KEY = cur.fetchall()
```

Interestingly, burger is the No.1 popular cuisine and even Maxican food is not as popular as burgers.

```
[(u'burger', 178),
 (u'mexican', 116),
 (u'sandwich', 92),
 (u'pizza', 72),
 (u'coffee_shop', 41),
 (u'american', 34),
 (u'italian', 26),
 (u'chinese', 22),
 (u'chicken', 20),
 (u'thai', 18)]
```

Religion sorted by count

```
QUERY = '''SELECT value, count(*)
FROM  Node_tags
WHERE key = 'religion'
group by value
order by count(*)
desc;'''
cur.execute(QUERY)
KEY = cur.fetchall()
```

No wonder christian is the biggest religion.

```
[(u'christian', 847),
 (u'jewish', 8),
 (u'muslim', 6),
 (u'buddhist', 4),
 (u'hindu', 4),
 (u'ascended_master_teachings', 1),
 (u'bahai', 1),
 (u'scientologist', 1),
 (u'taoist', 1)]
```

Conclusion

In this exercise, we found that the nature of OSM -volunteered geographic information- makes the map data contains errors and incomplete. We have implemented data auditing and cleaning using data.py and audit.py data processor as well as builting a database to store the information. Despite its incompleteness and errorous records, we found the map is a very useful resource to gain insights for the area of interest, such as the geolocation of places, what are the popular/unpopular food/shop/religion in San Diego, etc.

To implement future improvement, we propose to conduct cross validation of OSM with other data sources, such as google map, USGS website, etc through APIs. More specificly, each document should have validated source and should be processed using automatic data processor before uploading to the OSM server. This may sound difficult as the data are gained by volunteers, people might not be willing to spend time waiting on data validation. But it is possible to achieve due to the large number of active users. Also, we need more robust GPS data processors to get the data cleaned. In this way, the maps on openstreetmap.org will be more informative and useful.