OpenStreetMap Case Study - San Jose, CA

Introduction to OpenStreetMap

According to the organization's *(openstreetmap.org)* website, OpenStreetMap is built by a community of mappers that contribute and maintain data about roads, trails, cafés, railway stations, and much more, all over the world. It is free to use under an open license.

My chose map area : San Jose, California, United States
https://www.openstreetmap.org/relation/112143
Data is downloaded as an OSM XML file from: https://mapzen.com/data/metro-extracts/metro/san-jose_california/

I am interested in exploring this area, because it is very close to where I live and I am familiar with street names and amenities around San Jose. I would also like to complete this project by providing insights to further improve the consistency and validity of data.

Problems encountered in map - Data audit

After running auditing.py to check for the correct address formatting, the main areas for improvement are:

Unusual / Unrecognizable Street Names

- Abbreviated street names ('Wolfe Rd', 'Berryessa Rd')
- Missing the street type from the end. ('North 23rd', 'South 25th')
- House number, or apt number entered into the wrong space when data was filled out. ('Southeast 132nd Street #1', 'Northwest Byron Street #100')
- Invalid values. ('yes', '?', etc.)
- Apartment complex's name is entered instead of valid street address. ('Portofino', 'Seville', etc.)

Abbreviated street names are cleaned systematically with iterative parsing in auditing.py in order to make the data formatting more consistent:

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

```
osmfile = "san_jose_california.osm"

#unusual street types
street_type_re = re.compile(r'\S+\.?$', re.IGNORECASE)
street_types = defaultdict(set)

#expected values based on current usps database
expected_street = ["Close", "Cove", "Commons", "Bend", "Grove", "Green", "Glen",
            "Gardens", "Heights", "Highway", "Hills", "Landing", "Mall",
            "Meadows", "Park", "Parkway", "Plaza", "Point", "Ridge", "Row",
            "Run", "Spur", "Square", "Station", "Terrace", "Trail", "View",
            "Vista", "Walk", "Wall", "Alley", "Center", "Crescent", "Way",
            "Road", "Street", "Avenue", "Boulevard", "Drive", "Court",
            "Place", "Alley", "Circle", "Estates", "Lane", "Loop", "Expressway"]

#corrections
mapping = { "street": "Street",
            "st": "Street",
            "boulevard" : "Boulevard",
            "avenue": "Avenue",
            "ave": "Avenue",
            "av.": "Avenue",
            "Ter": "Terrace",
            "Steet": "Street",
            "St.": "Street",
            "St": "Street",
            "street": "Street",
            "Pkwy": "Parkway",
            "ST": "Street",
            "Rd.": "Road",
            "Rd": "Road",
            "ROAD": "Road",
            "RD": "Road",
            "Pl": "Plaza",
            "PL": "Plaza",
            "Ln.": "Lane",
            "Hwy": "Highway",
            "Dr.": "Drive",
```

```
            "Dr": "Drive",
            "Ct": "Court",
             "CT": "Court",
            "court": "Court",
            "Blvd.": "Boulevard",
            "Blvd": "Boulevard",
            "Boulvevard": "Boulevard",
            "Ave.": "Avenue",
            "Ave": "Avenue",
            "Av.": "Avenue",
            "AVENUE": "Avenue",
            "AVE": "Avenue",
            "ave": "Avenue",
            "Cir": "Circle",
            "Sq": "Square"}


#checks if values are in the expected_street names list
#and adds them to the street types dict if not
def audit_street_type(street_types, street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type not in expected_street:
            street_types[street_type].add(street_name)

#helper function used in audit(osmfile), to use correct element
def is_street_name(tag):
    return (tag.attrib['k'] == "addr:street")

#checks steet types for node and way top level tags
#prints out the unusual street types in a dictionary format - if needed remove #
#returns the unusual street types
def audit(osmfile):
    osm_file = open(osmfile, "r")
    street_types = defaultdict(set)
    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 is_street_name(tag):
                    audit_street_type(street_types, tag.attrib['v'])
    #pprint.pprint(dict(street_types))
    osm_file.close()
    return street_types


#Cleaning the data by replacing the unusual street types with the corrections in mappi
ng
#prints out the cleaned data
def fix_street(osmfile):
    st_types = audit(osmfile)
    for st_type, ways in st_types.iteritems():
        for name in ways:
            if st_type in mapping:
                better_name = name.replace(st_type, mapping[st_type])
                print name, "=>", better_name


fix_street(osmfile)
```

```
Wolfe Rd => Wolfe Road
Mt Hamilton Rd => Mt Hamilton Road
Berryessa Rd => Berryessa Road
Saratoga Los Gatos Rd => Saratoga Los Gatos Road
Quimby Rd => Quimby Road
San Antonio Valley Rd => San Antonio Valley Road
Homestead Rd => Homestead Road
Mt. Hamilton Rd => Mt. Hamilton Road
Silver Creek Valley Rd => Silver Creek Valley Road
wilcox ave => wilcox Avenue
Cortona court => Cortona Court
Monterey Hwy => Monterey Highway
Fountain Oaks Dr => Fountain Oaks Drive
Minto Dr => Minto Drive
1350 S Park Victoria Dr => 1350 S Park Victoria Drive
Linwood Dr => Linwood Drive
1490 S Park Victoria Dr => 1490 S Park Victoria Drive
Samaritan Dr => Samaritan Drive
Evergreen Village Sq => Evergreen Village Square
N 5th St => N 5th Street
Monroe St => Monroe Street
Casa Verde St => Casa Verde Street
Celadon Cir => Celadon Circle
Los Gatos Boulvevard => Los Gatos Boulevard
N 1st street => N 1st Street
Los Gatos Blvd => Los Gatos Boulevard
Mission College Blvd => Mission College Boulevard
Stevens Creek Blvd => Stevens Creek Boulevard
Santa Teresa Blvd => Santa Teresa Boulevard
Palm Valley Blvd => Palm Valley Boulevard
N McCarthy Blvd => N McCarthy Boulevard
Cherry Ave => Cherry Avenue
```

```
Saratoga Ave => Saratoga Avenue
Greenbriar Ave => Greenbriar Avenue
Blake Ave => Blake Avenue
Foxworthy Ave => Foxworthy Avenue
N Blaney Ave => N Blaney Avenue
Meridian Ave => Meridian Avenue
Westfield Ave => Westfield Avenue
The Alameda Ave => The Alameda Avenue
Seaboard Ave => Seaboard Avenue
Walsh Ave => Walsh Avenue
E Duane Ave => E Duane Avenue
W Washington Ave => W Washington Avenue
1425 E Dunne Ave => 1425 E Dunne Avenue
Cabrillo Ave => Cabrillo Avenue
Hollenbeck Ave => Hollenbeck Avenue
Los Gatos Blvd. => Los Gatos Boulevard
Perivale Ct => Perivale Court
```

City Names in the OSM file

```
#checks what cities are present in this OSM file

def audit_city(osmfile):

    osm_file = open(osmfile, "r")

    cities = set()

    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" and tag.attrib['v'] != "San Jose" an
d tag.attrib['v'] != "san Jose" and tag.attrib['v'] != "San jose" and tag.attrib['v']
!= "san jose":

                    cities.add(tag.attrib['v'])

    print "Cities in the OSM file:", cities


audit_city(osmfile)
```

```
Cities in the OSM file: set(['cupertino', 'Sunnyvale, CA', 'Santa Clara', 'Moffett Fie
ld', 'Felton', 'campbell', 'Los Gato', 'Milpitas', 'Mountain View', 'Fremont', 'Campbe
lll', 'Coyote', 'SUnnyvale', u'San Jos\xe9', 'Saratoga', 'Los Gatos, CA', 'Sunnyvale',
'Alviso', 'Mt Hamilton', 'Santa clara', 'Cupertino', 'los gatos', 'santa clara', 'sant
a Clara', 'Morgan Hill', 'Los Gatos', 'sunnyvale', 'Campbell', 'Redwood Estates'])
```

Incorrect Postal Zipcodes

Next, we take a look at the postal codes in the OSM file. To account for the inconsistency and filter out the invalid zipcodes, iterative parsing of elements is being used in auditing.py:

```
expected_zipcode = ["94089", "95002", "95008", "95013", "95014", "95032",
```

```
                    "95035", "95037", "95050", "95054", "95070", "95110",
                    "95111", "95112", "95113", "95116", "95117", "95118",
                    "95119", "95120", "95121", "95122", "95123", "95124",
                    "95125", "95126", "95127", "95128", "95129", "95130",
                    "95131", "95132", "95133", "95134", "95135", "95136",
                    "95138", "95139", "95140", "95148"]

invalid_zipcodes = defaultdict(set)

def audit_zipcode(invalid_zipcodes, zipcode):
    digits = zipcode[0:6]
    if digits not in expected_zipcode:
        invalid_zipcodes[digits].add(zipcode)

def is_zipcode(tag):
    return (tag.attrib['k'] == "addr:postcode")

def audit_zip(osmfile):
    osm_file = open(osmfile, "r")
    invalid_zipcodes = defaultdict(set)
    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 is_zipcode(tag):
                    audit_zipcode(invalid_zipcodes, tag.attrib['v'])
    #pprint.pprint(dict(invalid_zipcodes))
    osm_file.close()
    return invalid_zipcodes
```

Based on the audit above, there are zipcodes that have a formatting of xxxxx-xxxx (10 characters). To make the data more consistent only the first 5 numbers will be retained. Also, we have an invalid zipcode (951251) which has a typo and cannot be corrected with certainty. In this case and when the corrections cannot be completed with certainty, we will replace the incorrect value with "None". Also, some zipcodes start with 'CA'. These zipcodes will be cleaned by removing the letters.
Some examples, in which the data was cleaned by the previous outline:

1. CA 95116 => 95116
2. set(['95191'])
3. 951251 => None
4. 95112-5005 => 95112

Tags with Problematic Character

```
lower = re.compile(r'^([a-z]|_)*$')
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')

# check the "k" value for each "<tag>"
def key_type(element, keys):
    if element.tag == "tag":
        for tag in element.iter('tag'):
            if lower.search(tag.attrib['k']):
                keys['lower'] += 1
            elif lower_colon.search(tag.attrib['k']):
                keys['lower_colon'] += 1
            elif problemchars.search(tag.attrib['k']):
                print tag.attrib
                keys['problemchars'] += 1
            else:
                keys['other'] += 1
    return keys

def process_map(osmfile):
    keys = {"lower": 0,
            "lower_colon": 0,
            "problemchars": 0,
            "other": 0}
    for _, element in ET.iterparse(osmfile):
        keys = key_type(element, keys)
    print keys

process_map(osmfile)
```

```
{'k': 'service area', 'v': '20 miles'}
{'problemchars': 1, 'lower': 482276, 'other': 22335, 'lower_colon': 231485}
```

As we can see, there is only one problematic character in the OSM file.

The above characteristics can be described as follows:

1. "lower" : 482276, tags containing only lowercase letters and are valid
2. "lower_colon" : 231485, valid tags with a colon in their names

3. "problemchars" : 1, tags with problematic characters
4. "other" : 22335, other tags that do not fall into the other three categories

Overview of the data

File size and type

After running auditing.py and the unusual street names are removed, I am going to transform the values first into their respective CSV files based on the tags (transforming.py) and then into a SQL database (insert_into_database.py).

The details of the files are the following:
san_jose_california.osm: 380,743,544 bytes (380.7 MB)
san_jose_california.db: 266,190,848 bytes (266.2 MB)
nodes_tags.csv: 3,109,747 bytes (3.1 MB)
nodes.csv: 146,479,470 bytes (146.5 MB)
ways_nodes.csv: 48,711,778 bytes (48.7 MB)
ways_tags.csv: 22,170,698 bytes (22.2 MB)
ways.csv: 14,018,735 bytes (14 MB)

Overall, the OSM XML file consists of elements:
- nodes: points in space, along with at least one id number and a coordinate
- ways: ordered lists of nodes that define a polyline, such as linear features and area boundaries
- relations: which may or may not be used used to explain how other elements work together

Tag Types

To get a general idea about the number of times certain tags can be encountered, iterative parsing was used on the map file. The function below returns a dictionary with the tag name as the key and the number of times this tag is found in the OSM file as the value.

```
import pprint


filename = 'san_jose_california.osm'


def count_tags(filename):
    data = {}
    for event, element in ET.iterparse(filename):
        if element.tag in data.keys():
```

```
            data[element.tag] += 1
        else:
            data[element.tag] = 1
    pprint.pprint(data)

count_tags(filename)
```

```
{'bounds': 1,
 'member': 20014,
 'nd': 2045744,
 'node': 1752581,
 'osm': 1,
 'relation': 2083,
 'tag': 736097,
 'way': 235451}
```

Number of Nodes

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

1752581

Number of Ways

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

235451

Number of Unique Users

```
def process_map(filename):
    users = set()
    for _, element in ET.iterparse(filename):
        if element.get("uid"):
            users.add(element.attrib["uid"])
    print len(users)

process_map(filename)
```

```
1407
```

There are 1407 unique users in the area.

Data Exploration

Cities in the OSM file

```
sqlite> 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;
```

```
Sunnyvale|3421
San Jose|1042
Morgan Hill|397
Santa Clara|323
Saratoga|233
San José|174
Los Gatos|138
Milpitas|105
Campbell|76
Cupertino|61
Alviso|11
Mountain View|7
san jose|6
Campbelll|3
sunnyvale|3
san Jose|2
santa Clara|2
Coyote|1
Felton|1
Fremont|1
Los Gato|1
Los Gatos, CA|1
Moffett Field|1
Mt Hamilton|1
Redwood Estates|1
```

```
SUnnyvale|1
San jose|1
Santa clara|1
Sunnyvale, CA|1
campbell|1
cupertino|1
los gatos|1
santa clara|1
```

Based on the findings, most of the addresses are from Sunnyvale. This confirms my suspicion that it would be more appropriate to name the map area as San Jose and its victinity.

Top 10 Cuisines

```
sqlite> SELECT value, COUNT(*) as count
        FROM nodes_tags
        WHERE nodes_tags.key = 'cuisine'
        GROUP BY value
        ORDER BY count DESC;
```

Below are the results:

```
vietnamese|118
mexican|114
sandwich|96
pizza|91
chinese|88
coffee_shop|76
japanese|47
indian|46
burger|42
american|39
```

After running this query, I have noticed that on the bottom of this list there are some cuisines misspelled or containing typos, which makes the results a little bit skewed. There are a couple of results where the cuisine is not clearly determined and contains a list of the type of foods that can be bought at the place.

Number of Schools

```
sqlite> SELECT COUNT(*) AS num
        FROM nodes_tags
        WHERE nodes_tags.value = 'school';
```

This query revealed that there are 149 schools in the area.

Top 10 Amenities

```
sqlite> SELECT value, count(*) as num FROM nodes_tags
        WHERE key='amenity'
        GROUP BY value
        ORDER BY num DESC;
```

```
restaurant|884
fast_food|422
bench|313
cafe|257
bicycle_parking|202
place_of_worship|171
toilets|160
school|143
bank|130
parking_space|128
```

As we can see, restaurant is the top amenity, followed by fast food and bench. Bench in this context, I assume, refers to parks and BBQ area.

Findings and Ideas about the Dataset

The dataset was audited and the abbreviated street names were cleaned as applicable, however, the consistency of the dataset could be further improved and a periodic audits would also be beneficial. It is interesting to see how many misspellings and typos exist and probably affect the outcome of some research that may be done on this dataset. The map can be beneficial for small-scale applications, such as finding amenities, however, there is a lot to be done to improve its accuracy if the intention is to grow.

It is certainly a challenge to ensure that all the data being entered into the map area is valid, does not have any typos, and are entered into the correct place. Human error cannot be eliminated completely, however, certain measures could help. For example, preventing multiple entry of data into one field at a time can be achieved with restricting characters such as colon (:), semicolon (;), or comma (,) where it is applicable. Although, the wiki.openstreetmap.org website does offer some guidance for editing, after querying some fields I still have the sense that this is not being monitored closely. Also, at some fields, such as amenities; multiple choice selections or a list of options would help to categorize data.

References

1. OpenStreetMap website, https://www.openstreetmap.org/about.
2. OSM file formats, OpenStreetMap Wikia website http://wiki.openstreetmap.org/wiki/OSM_file_formats
3. Udacity's Data Analyst Nanodegree track - Data Wrangling https://classroom.udacity.com/nanodegrees/nd002/parts/0021345404/modules/316820862075461/lessons/5436095827/concepts/8063289030923
4. USPS street names and suffixes, http://pe.usps.gov/text/pub28/28apc_002.htm
5. Zip code data. http://www.city-data.com/zipmaps/San-Jose-California.html