OpenStreetMap Project

I wrangled data from the Mission District of San Francisco. I made a custom extract on Mapzen at https://mapzen.com/data/metro-extracts/your-extracts/dfe5d8113149 and downloaded the "OSM XML" file. The borders for my selected region are not exact so there is some data from neighboring districts as well. The goal of this project was to iteratively parse, audit and clean the data, read the cleaned data into csv files, and then import those csv files into a SQL database. The resultant database was queried to gain insight into the OpenStreetMap data.

The OSM file was made up of "node" and "way" tags that contained nested tags. Below is a small section of the raw OSM data from a sample file I created.

```
<node changeset="37490520" id="61674029" lat="37.7721197" lon="-122.
4370577" timestamp="2016-02-27T22:21:55Z" uid="371121" user="AndrewS
now" version="9">
    <tag k="name" v="The Page" />
    <tag k="amenity" v="pub" />
    <tag k="addr:city" v="San Francisco" />
    <tag k="wheelchair" v="yes" />
    <tag k="addr:street" v="Divisadero Street" />
    <tag k="addr:housenumber" v="298" />
</node>
```

Problems Encountered

After examining the csv files written from data.py I noticed the following problems:

- Overabbreviated street names ("Mision st.")
- Inconsistent Postal Codes
- Special Characters in Street Names

Writing the csv Files

I used the data.py script I completed in an Udacity quiz to read the xml tags and organize them by 'nodes' and 'ways.' The original osm file was written into 5 csv files:

The attributes of the 'node' tags were written to a file called nodes.csv. These attributes were the id, lat and lon coordinates, timestamp, user id and changeset for a particular node.

The attributes of the tags nested within a node were written to a file called nodes_tags.csv. These attributes were the id, key, value, and type for that particular node. Some of these keys were: "name", "amenity", "addr:street" and the corresponding values were: "The Page", "pub", "Divisadero Street."

The attributes of the 'way' tags were placed in a file called ways.csv and held the same categories as

the nodes.csv file.

The 'way' tags contained two sub tags 'nd' and 'tag.' The attributes in the 'nd' tags were written to a file called ways_nodes.csv and contained the tag's id, the id of the node it corresponds to, and the position of the tag. The attributes in the 'tag' tags were written to a file called ways_tags.csv and contained the id, key, value, and type for that tag.

Auditing Street Names

I began by auditing the data to look for problems. I used three functions to view all of the street names so I could see which naming conventions were used such as "ln" or "lane". The first function checks if the element that has been passed to it is a street name.

```
def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")
```

The audit_street_type() function takes in a dictionary "street_types" and a street name. It uses a regular expression to check if the street type is in a list of expected street types. If the street type is not in the list, it is added to the dictionary "street_types" where the key is a particular street type and the value is a set of all the names that have that type.

```
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)

expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Plac
e", "Square", "Lane", "Road",
            "Trail", "Parkway", "Commons"]

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_types[street_type].add(street_name)
```

The audit() function iteratively parses the tag elements nested within the "node" or "way" tags, and calls the is_street_name() and audit_street_type() functions. This prints a dictionary of street types and the set of names for each type. I used the resultant street types dictionary to decide which street names I wanted to fix.

```
def audit(osmfile):
    street_types = defaultdict(set)
    for event, elem in ET.iterparse(osmfile, 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'])
```

```
    print street_types
    return street_types
```

Fixing Street Names

After auditing, I created the mapping dictionary to be used when updating street names.

```
mapping = {"St" : "Street", "St." : "Street"}
```

The update_name() function takes in a name and mapping dictionary. Then it uses the same regular expression from earlier to check if the street name has same type as any of the keys in the mapping dictionary. If there is a match, the name is updated to have the type "Street."

```
def update_name(name, mapping):

    # YOUR CODE HERE
    m = street_type_re.search(name)
    if m:
        street_type = m.group()
        if street_type in mapping.keys():

            # Had to use re.sub to replace last word in street names
            name = re.sub(street_type, mapping[street_type], name)

    return name
```

I called the is_street_name() and update_name() functions from the shape_element() function in data.py. This was done for both 'node' and 'way' tags. Then I opened the csv files that were created in data.py to check if the street names had been fixed.

```
# These are the street names that I want to call update_name on
                if is_street_name(sub):

                    # Loop through st_types dictionary and update st
reet names
                    for st_type, ways in st_types.iteritems():
                        for name in ways:
                            if name == sub.attrib['v']:
                                sub.attrib['v'] = update_name(name,
mapping)
```

Auditing and Fixing Postal Codes

I wrote a similar function to check if a tag element is a postal code. I printed all of the postal codes and saw that some of them had the format 'CA:XXXXX' instead of the desired 'XXXXX' format.

```
def is_postcode(elem):
    return (elem.attrib['k'] == "addr:postcode")
```

I called is_postcode() from inside my shape_element() function. I split the attribute on ":" into two values and kept the second which contained only the numbers in the postal code.

```
# Change postcodes with format CA:94112 to 94112
                elif is_postcode(sub):
                    if len(sub.attrib['v'].split(":")) == 2:
                        sub.attrib['v'] = sub.attrib['v'].split(":")
[1]
```

Removing Special Characters from Street Names

When I first audited the street names I noticed that some of them written in this format 'Bartlett Street #203'. Inside my shape_element() function, I edited my 'Clean Street Names' section to check if the street name contains a '#', split on white spaces and delete the last element of the list containing the '#' character. The other elements in the list were recombined and update_name() was called on them. If the '#' character was not present then update_name() was immediately called.

```
if is_street_name(sub):

                    for st_type, ways in st_types.iteritems():
                        for name in ways:
                            if name == sub.attrib['v']:



                                if symbol in sub.attrib['v']:
                                    words_in_name = sub.attrib['v'].
split(" ")

                                    del words_in_name[-1]
                                    sub.attrib['v'] = words_in_name[
0] + " " + words_in_name[1]
                                    sub.attrib['v'] = update_name(su
b.attrib['v'], mapping)

                                else:
                                    sub.attrib['v'] = update_name(na
me, mapping)
```

Creating a SQL Database

I created my sql database "mission_district.db" from the command prompt:

```
cd \
cd sqlite_windows
sqlite3 mission_district
```

I put the csv files in the sqlite_windows folder and used the following commands to import them:

```
sqlite> CREATE TABLE TableName(column1 datatype, ...);
sqlite> .mode csv
.import file.csv TableName
```

File Sizes

```
mission_district.osm.............115 MB
mission_district.db..............66.5 MB
nodes.csv........................46 MB
nodes_tags.csv...................1.43 MB
ways.csv.........................3.89 MB
ways_nodes.csv...................16.8 MB
ways_tags.csv....................5.10 MB
```

Data Overview

Find the number of chosen business i.e. "cafe"

I used UNION to combine the results of two columns where each column was a subquery of amenities from the Nodes_Tags and Ways_Tags tables.

```
QUERY = """
SELECT COUNT(DISTINCT Both.value)
from (SELECT Nodes_Tags.value
from Nodes_Tags
where Nodes_Tags.id in (SELECT id from Nodes_Tags where value = 'caf
e') and Nodes_Tags.key = 'name'
UNION
SELECT Ways_Tags.value
from Ways_Tags
where Ways_Tags.id in (SELECT id from Ways_Tags where value = 'cafe'
) and Ways_Tags.key = 'name')
as Both;"""
```

140

Number of Ways

```
QUERY = """
SELECT COUNT(DISTINCT Ways.id)
from Ways;"""
```

60051

Number of Nodes

```
QUERY = """
SELECT COUNT(DISTINCT Nodes.id)
from Nodes;"""
```

569337

Number of Unique Users

```
QUERY = """
SELECT COUNT(DISTINCT total.uid)
from
(SELECT Nodes.uid
from Nodes
UNION SELECT Ways.uid
from Ways) as total;"""
```

544

Top 10 Cuisines

I excluded 'coffee_shop', 'ice_cream' and 'juice' so I could limit my search to savory food.

```
QUERY = """
SELECT both.value, COUNT(both.value)
from (SELECT *
from Nodes_Tags
where Nodes_Tags.key = 'cuisine'
UNION
SELECT *
from Ways_Tags
where Ways_Tags.key = 'cuisine') as both
GROUP BY both.value
HAVING COUNT(both.value) > 1
AND both.value != 'coffee_shop'
AND both.value != 'ice_cream'
AND both.value != 'juice'
ORDER BY COUNT(both.value)
DESC
limit 10;"""
```

```
          0   1
0   mexican  48
1     pizza  31
2  japanese  26
3  american  25
4      thai  24
```

```
5   chinese  19
6    burger  16
7   italian  15
8  sandwich  15
9    indian  11
```

Top 10 Amenities

```
QUERY = """
SELECT both.value, COUNT(both.value)
from (SELECT *
from Nodes_Tags
where Nodes_Tags.key = 'amenity'
UNION
SELECT *
from Ways_Tags
where Ways_Tags.key = 'amenity') as both
GROUP BY both.value
HAVING COUNT(both.value) > 1
ORDER BY COUNT(both.value)
DESC
limit 10;"""
```

```
                  0    1
0        restaurant  454
1           parking  171
2              cafe  162
3          post_box  142
4             bench  134
5   bicycle_parking  119
6            school   98
7  place_of_worship   95
8               bar   85
9       car_sharing   74
```

First 10 Mexican Restaurants added

```
QUERY = """
SELECT DISTINCT Both.value, Both.timestamp
from (SELECT Nodes_Tags.value, Nodes.timestamp
from Nodes_Tags
JOIN Nodes
ON Nodes_Tags.id = Nodes.id
where Nodes_Tags.id in (SELECT id from Nodes_Tags where value = 'mex
ican') and Nodes_Tags.key = 'name'
UNION
```

```
SELECT Ways_Tags.value, Ways.timestamp
from Ways_Tags
JOIN Ways
ON Ways_Tags.id = Ways.id
where Ways_Tags.id in (SELECT id from Ways_Tags where value = 'mexic
an') and Ways_Tags.key = 'name')
as Both
order by both.timestamp asc
limit 10;"""
```

```
0     Green Chile Kitchen  2012-03-04T01:42:38Z\r
1                 Papalote  2012-06-07T18:06:52Z\r
2  Papalote Mexican Grill  2012-09-24T04:22:46Z\r
3      La Taqueria Menudo  2012-12-07T09:52:10Z\r
4                  El Faro  2012-12-18T20:09:56Z\r
5                Padrecito  2013-04-22T16:42:58Z\r
6              La Tortilla  2013-05-27T00:45:09Z\r
7    Zapata Mexican Grill  2013-05-27T01:31:14Z\r
8           Puerto Allegre  2013-12-19T08:14:37Z\r
9                Gallardos  2013-12-25T19:32:34Z\r
```

Additional Ideas

Problems with chain businesses

Chain businesses such as Martha & Bros. and Peet's Coffee used multiple naming conventions. Having these businesses in the listed in the same format would make queries involving them more accurate. This would be helpful when looking up unique business names for cafes.

```
QUERY = """
SELECT DISTINCT Both.value
from (SELECT Nodes_Tags.value
from Nodes_Tags
where Nodes_Tags.id in (SELECT id from Nodes_Tags where value = 'caf
e') and Nodes_Tags.key = 'name'
UNION
SELECT Ways_Tags.value
from Ways_Tags
where Ways_Tags.id in (SELECT id from Ways_Tags where value = 'cafe'
) and Ways_Tags.key = 'name')
as Both;"""
```

```
73                Martha & Bros. Coffee
74        Martha & Bros. Coffee Company
75          Martha and Bros. Coffee Co.
```

```
90                  Peet's Coffe& Tea
91                      Peet's Coffee
92                Peet's Coffee & Tea
93                       Peets Coffee
```

A list of expected business names for known chain businesses could be created and a regular expression could be used to match businesses names to the names in the expected list. This would be similar to auditing the street names. An improvement on this approach would be to audit all of the restaurants for multiple locations with different names. This solution would be more general but also more difficult because the auditing script would need to identify which groups of listings should have the same name. This would probably involve regular expressions based on matching the first two or three elements in each name.

Conclusion

After auditing and cleaning the OpenStreetMap data for the mission district and neighboring areas, it seems that the data in for San Francisco may have been an underwhelming choice. Because of the large tech culture in San Francisco, it is likely that many people have cleaned elements regarding street names and postal codes. I was able to correct some street names and postal codes. When querying the database, I was surprised to see that parking was the second most common amenity. It was interesting that there are more places of worship than bars in my chosen area. I live in the mission district and do not see many places of worship in general.

In [ ]: