Data Wrangling Project on Open Street Map Data

Map Area

The selected area is present in Chennai, Tamil Nadu, India

https://mapzen.com/data/metro-extracts/your-extracts/0d12573bee46

This project involves wrangling of Open Street Map data which is in the XML format. I have chosen a part of Chennai because the city is familiar to me.

Problems encountered in the map

On looking through the data I noticed the following problems:

- Inconsistent street types (”Muthaiyah Street, Vellala Teynampet”,” Sivagnanam St.”)
- Inconsistent phone number formats (“+91 44 4264 3036”, ”044-42084422”)
- Invalid phone numbers (In India, landline phone numbers have 8 digits and mobile phone numbers have 10 digits. Toll free numbers may have 11 digits. There were some numbers which did not belong to any of these categories)

Inconsistent street names

Some of the entries had the complete address including state name and pincode mentioned in the ‘addr:street’ field. In these cases, the words before the first comma were taken as street names as it was appropriate for almost all entries. If the street name turned out to be a digit, the entry was cleared.

To tackle the problem of inconsistent street names the following code was used for updating them:

```
def update_name(name, mapping):
    m = street_type_re.search(name)
    if m is not None:
        if m.group() not in expected:
            if m.group() in mapping.keys():
                #replacing street names in 'name' with those in mapping
                name = re.sub(m.group(), mapping[m.group()], name)
            elif name.find(','):
                position=name.find(',')
                if position >0:         #If comma is present, position value is greater than 0
                    name=name[:position]  #Words before the comma are taken as street names
                    if name.isdigit(): #Street names are not represented by digits only
                        name=''
            else:
                name=name
    return name
```

This updated problematic address strings, such that “Muthaiyah Street, Vellala Teynampet” becomes “Muthaiyah Street” and “Sivagnanam St.” becomes “Sivagnanam Street”

Inconsistent and invalid phone numbers

The main inconsistencies found were the presence of white space and non-digit characters in between the numbers and presence of STD code and/or country code in some numbers.

The following update function for phone numbers involves removal of non-digit characters, country code and STD code in order to maintain a consistent format. Invalid phone numbers (in terms of number of digits) have also been removed.

```
def update_phones(phonenumber):
    # remove non-digit characters
    phonenumber = ''.join(ele for ele in phonenumber if ele.isdigit())

    if phonenumber.startswith('91'): #country code of India is 91
        phonenumber  = phonenumber[2:]
    if phonenumber.startswith('044'): #STD code for chennai is 044
        phonenumber  = phonenumber[3:]
    if phonenumber.startswith('044 '):
        phonenumber  = phonenumber[4:]

    #In India, landline phone numbers have 8 digits and mobile phone numbers have 10 digits.
    #Remove numbers of other lengths
    #Exception: toll-free numbers may start with 1800 and have 11 characters
    if len(phonenumber)==10 or len(phonenumber)==8 :
        phonenumber=phonenumber
    else:
        if phonenumber.startswith('1800') and len(phonenumber)==11:
            phonenumber=phonenumber
        else:
            phonenumber = ''
    return phonenumber
```

Data Overview and Additional Ideas

File sizes

Location.osm………	78.1 MB
Data.db………………	55.9 MB
nodes.csv…………….	29.5 MB
nodes_tags.csv…….	247 KB
ways.csv………………	4.88 MB
ways_tags.csv………	2.97 MB
ways_nodes.csv……	10.7 MB

Number of nodes

```
con = sqlite3.connect("data.db")
cur = con.cursor()

def number_of_nodes():
    result = cur.execute('SELECT COUNT(*) FROM nodes')
    return result.fetchone()[0]
print "Number of nodes: " , number_of_nodes()
```

```
Number of nodes:  365355
```

Number of ways

```
def number_of_ways():
    result = cur.execute('SELECT COUNT(*) FROM ways')
    return result.fetchone()[0]
print "Number of ways: " , number_of_ways()
```

```
Number of ways:  82551
```

Number of unique users

```
def number_of_unique_users():
    result = cur.execute('SELECT COUNT(DISTINCT(e.uid)) \
               FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e')
    return result.fetchone()[0]
print "Number of unique users: " , number_of_unique_users()
```

```
Number of unique users:  457
```

Top 10 contributing users

```
def top_contributing_users():
    users = []
    for row in cur.execute('SELECT e.user, COUNT(*) as num \
            FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e \
            GROUP BY e.user \
            ORDER BY num DESC \
            LIMIT 10'):
        users.append(row)
    return users
print "Top contributing users:\n" , pd.DataFrame(top_contributing_users())
```

```
Top contributing users:
              0      1
0      praveeng  55112
1   venkatkotha  32718
2      PlaneMad  32557
3   masthanvali  29483
4      pvprasad  23417
5       sramesh  22284
6     maheshrkm  21810
7  kranthikumar  21149
8     shalinins  20184
9  venugopal009  18703
```

The user with most contributions is ‘praveeng’.

Number of users appearing only once

```
def contributing_once():
    result = cur.execute('SELECT COUNT(*) \
        FROM\
        (SELECT e.user, COUNT(*) as num\
         FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e\
         GROUP BY e.user\
         HAVING num=1)')
    return result.fetchone()[0]
print contributing_once()
```

```
150
```

Additional Ideas

Suggestion

I have noticed that most of the problems are because of users entering data into the wrong field or not following a uniform format. About 33.8% of the users have contributed only once. First time users might not be aware of the format associated with different fields. So, my suggestion is to impose some restriction in the format while entering data. For example street name cannot be only digits and phone numbers should have specific number of digits. The data also needs to be checked for redundancy in key names. There need not be two keys for the same type of value for example, address:postcode and addr:postcode.

Anticipated problems

Creating a format for some of the fields like phone numbers and pincode are complicated because they might differ based on the location.

Additional Data Exploration

Common amenities

```
def common_amenities():
    amenitites = []
    for row in cur.execute('SELECT value, COUNT(*) as num \
            FROM nodes_tags  \
            WHERE key="amenity" \
            GROUP BY value \
            ORDER BY num DESC \
            LIMIT 10'):
        amenitites.append(row)
    return amenitites

print "Common ammenities:\n" , pd.DataFrame(common_amenities())
```

```
Common ammenities:
                 0    1
0       restaurant  141
1             bank   70
2              atm   64
3  place_of_worship   61
4          library   48
5         pharmacy   44
6             fuel   39
7         hospital   33
8           school   25
9        fast_food   21
```

Restaurant with a count of 141 is the most common amenity in this locality.

Biggest religion

```
def biggest_religion():
    for row in cur.execute('SELECT nodes_tags.value, COUNT(*) as num \
            FROM nodes_tags \
                JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value="place_of_worship") i \
                ON nodes_tags.id=i.id \
            WHERE nodes_tags.key="religion" \
            GROUP BY nodes_tags.value \
            ORDER BY num DESC \
            LIMIT 1;'):
        return row
print "Biggest religion: " , biggest_religion()
```

```
Biggest religion:  (u'hindu', 24)
```

Popular cuisine

```
def popular_cuisine():
    cuisine=[]
    for row in cur.execute('SELECT nodes_tags.value, COUNT(*) as num \
            FROM nodes_tags \
                JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value="restaurant") i \
                ON nodes_tags.id=i.id \
            WHERE nodes_tags.key="cuisine" \
            GROUP BY nodes_tags.value \
            ORDER BY num DESC \
            LIMIT 1'):
        cuisine.append(row)
    return cuisine
print "Popular cuisine: " , popular_cuisine()
```

```
Popular cuisine:  [(u'regional', 15)]
```

Conclusion

The selected portion of OSM data has been thoroughly audited in terms of street names and phone numbers. Exploration of the data has been carried out through SQL queries and some insightful results have been produced. Finally, suggestions have been made on how the data collection can be improved.

References

Udacity Forum
Stackoverflow
Regex101
https://gist.github.com/carlward/54ec1c91b62a5f911c42#file-sample_project-md