Wrangle OpenStreetMap Data

Map Area

St. Petersburg, Florida, United States: https://www.openstreetmap.org/relation/118894 (https://www.openstreetmap.org/relation/118894)

Background: As of 2020, St. Petersburg is the 5th most populous city in Florida with a population over 250,000.

The reason why I chose this city is because I have lived there for 14 years and I am interested to find some interesting facts about the city.

Problems Encountered in the Map

Inconsistent Street Names

After auditing the data, I found that there are inconsistencies in terms of the street names of addresses. For example, there are different kinds of street like "St", "street", "ST", etc. To fix this, I included all those different kinds of street in the mapping variable to convert them into the correct format "Street." I also did the same thing for roads, avenues, boulevards, centers, etc. The function "update_street_name" updates the incorrect street name into the correct format.

In [1]:

```python
# ================================================== #
#     Helper Functions for Auditing Street Types     #
# ================================================== #

#Function to use to see if the input string is not in the expected list.
def audit_street_type(street_types, street_name): #case study function
    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)

#Function to use to look at the k attribute only if it equals to "addr:street."
def is_street_name(ele):
    return (ele.attrib['k'] == "addr:street")

#Function to use to look at all the street types, which helps us fill-up the
#"mapping" dictionary.
def audit(osmfile): #case study function
    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'])
            elem.clear()
    osm_file.close()
    return street_types

# ================================================== #
#        Functions for Updating Street Names         #
# ================================================== #

#If street name has a string that is contained in the mapping variable,
#then the change will be made.
#To format the street name, functions update_street_name and audit_street_name_tag will be
#used in the shape_element function.
def update_street_name(nm, mapping):
    for key, val in mapping.iteritems():
        if key in nm:
            return nm.replace(key, val)
    return nm

def audit_street_name_tag(ele):
    strtname = ele.get('v')
    f = street_type_re.search(strtname)
    if f:
        strtname_better = update_street_name(strtname,mapping)
        return strtname_better
    return strtname
```

Inconsistent Postal Codes

There are postal codes in the data that do not follow the standard 5 digit display. For example, there are postal codes that include the state's abbreviation like "FL 33761" or that have more than 5 digits like "33701-1007." To have consistent queries, we have to make sure that all postal codes are in the standard 5 digit display. The function "update_postcode" updates the inconsistent postal code into the correct format.

In [ ]:
```
# ================================================== #
#      Helper Functions for Auditing Postal Codes    #
# ================================================== #

#Function to use to see if the input string is not in the expected list.
def audit_zip_codes(zip_types, zip_nm, reg_exp, zip_expected):
    f = reg_exp.search(zip_nm)
    if f:
        zip_type = f.group()
        if zip_type not in zip_expected:
             zip_types[zip_type].add(zip_nm)

#Function to use to look at the k attribute only if it equals to "addr:postcode."
def is_zip_name(ele):
    return (ele.attrib['k'] == "addr:postcode")

#Function to use to look at all the postal codes, which helps us fill-up
#the "mapping" dictionary.
def audit(filenm, reg_exp):
    for evt, ele in ET.iterparse(filenm, events = ("start",)):
        if ele.tag == "way" or ele.tag == "node":
            for tag in ele.iter("tag"):
                if is_zip_name(tag):
                    audit_zip_codes(zip_types, tag.attrib['v'], reg_exp, zip_expected)
    pprint.pprint(dict(zip_types))

# ================================================== #
#         Functions for Updating Zip Codes           #
# ================================================== #

#Making all postal codes a 5 digit number. This will fix any postal code
#that have more than 5 digits and begins with a state abbreviation like "FL" for Florida.
#To format the postal codes, functions update_postcode and audit_postcode_tag will be
#used in the shape_element function
def update_postcode(nm):
    if "-" in nm:
        nm = nm.split("-")[0].strip()
    elif "FL" in nm:
        nm = nm.split("FL ")[1].strip('FL ')
    elif len(str(nm)) > 5:
        nm = nm[0:5]
    elif nm.isdigit() == False:
         nm = 00000
    return nm

def audit_postcode_tag(ele, reg_exp=re.compile(r'\b\S+\.?$', re.IGNORECASE)):
    postal_cd = ele.get('v')
    f = reg_exp.search(postal_cd)
    if f:
        postal_cd_better = update_postcode(postal_cd)
        return postal_cd_better
    return postal_cd
```

Data Overview and Additional Ideas

File Sizes

```
petersburg.osm ---------> 81.3 MB
petersburg.db ----------> 44.1 MB
nodes.csv --------------> 30.4 MB
nodes_tags.csv --------->  0.9 MB
ways.csv --------------->  4.6 MB
ways_tags.csv ---------->  5.5 MB
ways_nodes.csv ---------> 10.1 MB
```

Number of unique users

In [ ]:
```
#number of unique users
cur.execute("SELECT COUNT(DISTINCT(e.uid))FROM
            (SELECT uid FROM Nodes UNION ALL SELECT uid FROM Ways) as e;")
res = cur.fetchall()

pprint.pprint(res)
```

```
[(618,)]
```

Number of nodes

In [ ]:
```
#number of nodes
cur.execute("SELECT count(*) FROM nodes;")
res = cur.fetchall()

pprint.pprint(res)
```

```
[(329119,)]
```

Number of ways

In [ ]:
```
#number of ways
cur.execute("SELECT count(*) FROM ways;")
res = cur.fetchall()

pprint.pprint(res)
```

```
[(64945,)]
```

Popular religions

In [ ]:
```
#Most popular religions
cur.execute("select value, count(*) as num from
            (select key,value from nodes_tags UNION ALL select key,value from ways_tags)
            as e where key='religion' group by value order by num desc;")
res = cur.fetchall()

pprint.pprint(res)
```

```
[(u'christian', 62),
(u'unitarian_universalist', 1),
(u'bahai', 1)]
```

Top 10 contributing users

In [ ]:
```
#Top 10 contributing users
cur.execute("select e.user, count(*) as num from
            (select user from nodes UNION ALL select user from ways)
            as e group by user order by num desc limit 10;")
res = cur.fetchall()
pprint.pprint(res)
```

```
[(u'Andrew Matheny_import', 207238),
(u'Omnific', 30300),
(u'coleman', 27748),
(u'jharpster-import', 20567),
(u'jharpster', 19511),
(u'Karthoo_import', 11890),
(u'OSMWeekly', 8921),
(u'TheDude05', 8230),
(u'GeoKitten_import', 5987),
(u'ninja_import', 4818)]
```

Total Users

In [ ]:
```
#Total users
cur.execute("select count(e.user) from
            (select user from nodes UNION ALL select user from ways) as e;")
res = cur.fetchall()

pprint.pprint(res)
```

```
[(394064,)]
```

Additional Ideas

Problem: There is no doubt that the data in the OpenStreetMap is very inconsistent. Before cleaning the data, we saw that there were different abbreviations for “street” such as “st.”, “Street”, “ST”, etc. The same can be said with postal codes, as there were postal codes that have letters or more than the standard 5-digit display. We needed to use our update_street_name and update_postcode functions to get rid of such inconsistencies in the data. It is clear that there is no standard formatting in place, at least for the postal codes and street names, on the website.

Solution: There should be some kind of a validation error when a user fails to input the data correctly (for formatting) on the front-end. This would eliminate the need of having to clean the already submitted data. This would also save time because we can just go straight to the data analyzation part.

Issue: Since each country has their own formatting standards for street names, postal codes, and abbreviations, implementing a validation error may take a lot of work. Validation errors would have to be region specific and for this reason, the effort and the cost of creating and maintaining this improvement could be so overwhelming that it may require a dedicated team to handle this improvement.

Conclusion

It is clear that the St. Petersburg OpenStreetMap dataset is riddled with inconsistencies. The data entry by the contributing users certainly played a part in it. This is why I suggest the OpenStreetMap to use validation errors in its data entries to avoid inconsistent data.

While we did not completely clean all of the data, it was cleaned enough for the purposes of this project. We imported the dataset into our database and we were able to find some interesting facts about the city’s data by using SQL queries.

```
In [ ]:
```