OpenStreetMap Data Case Study

Map Area

St. Louis, MO, United States

- https://www.openstreetmap.org/export#map=11/38.6533/-90.2441

This map is the place I was when I first got in United States, I studied here and got my Master degree here. I live here for two years. This is a chance to know more about this place, see what are the things I've already know and what are the things I didn't know about.

Problems Encountered in the Map

After unzip the osm file, I found that it is really large, and maybe the size is the first problem I have to deal with. Other than this, when I process the data, I find other problems too:

- No 'user' attribute in Node
- Not sure about the abbreviation in 'name_type' value in ways_tags table.
- ID is too large for *int* type in SQL
- Postcode looks weird

No 'user' attribute in Node

While I'm working on converting the XML file to CSV format, it gave me the error about 'no user in node' or 'no uid in node'. Then I had a look at the XML data, there are indeed some node do not have a 'user' or 'uid' attribute, and this cannot fit the rule that these are required attribute of a Node data type. So what I did is simply ignore these data in the shape_element function:

```
def shape_element(element, node_attr_fields=NODE_FIELDS, way_attr_fields=WAY_FIELDS,
                  problem_chars=PROBLEMCHARS, default_tag_type='regular'):
    """Clean and shape node or way XML element to Python dict"""

    node_attribs = {}
    way_attribs = {}
    way_nodes = []
    tags = []  # Handle secondary tags the same way for both node and way elements

    if element.tag == 'node':
        node_id = element.attrib['id']
        # turns out some of the entries don't have a 'user' attribute
        # while 'user' is a required attribute
        # so dump those entries withou an 'user' attribute
        if 'user' not in element.attrib:
            return
        for each_node_field in node_attr_fields:
            # copy the attrbutes to the dictionary
            node_attribs[each_node_field] = element.attrib[each_node_field]
        # for each 'tag' under this 'node'
        for each_tag in element:
            if each_tag.tag == 'tag':
                temp_dict = {}
                temp_dict['id'] = node_id
                temp_dict['value'] = each_tag.attrib['v']
                if re.search(LOWER_COLON, each_tag.attrib['k']) != None:
                    # extract those keywords out of the attributes if there are
                    # ':' in that attrbutes using regular expression
                    temp_dict['type'] = each_tag.attrib['k'][ :
each_tag.attrib['k'].index(':')]
                    temp_dict['key'] =
each_tag.attrib['k'][each_tag.attrib['k'].index(':') + 1:]
                else:
                    # if there is no ':' in attributes
                    # simply copy the value
                    temp_dict['type'] = 'regular'
                    temp_dict['key'] = each_tag.attrib['k']
                tags.append(temp_dict)

    # similar process as 'node'
    elif element.tag == 'way':
        way_id = element.attrib['id']
        for each_way_field in way_attr_fields:
            way_attribs[each_way_field] = element.attrib[each_way_field]
            ii_nd = 0
        for each_tag in element:
            if each_tag.tag == 'nd':
                temp_dict = {}
                temp_dict['id'] = way_id
                temp_dict['node_id'] = each_tag.attrib['ref']
                temp_dict['position'] = ii_nd
                ii_nd += 1
                way_nodes.append(temp_dict)
            if each_tag.tag == 'tag':
                temp_dict = {}
                temp_dict['id'] = way_id
                temp_dict['value'] = each_tag.attrib['v']
                if re.search(LOWER_COLON, each_tag.attrib['k']) != None:
                    temp_dict['type'] = each_tag.attrib['k'][ :
each_tag.attrib['k'].index(':')]
                    temp_dict['key'] =
each_tag.attrib['k'][each_tag.attrib['k'].index(':') + 1:]
                else:
                    temp_dict['type'] = 'regular'
                    temp_dict['key'] = each_tag.attrib['k']
                tags.append(temp_dict)

    if element.tag == 'node':
        return {'node': node_attribs, 'node_tags': tags}
    elif element.tag == 'way':
        return {'way': way_attribs, 'way_nodes': way_nodes, 'way_tags': tags}
```

Here I say if 'user' not in element.attrib, then return nothing and continue. Also, if there is not a user there, no uid as well.

'name_type' value in ways_tags table

The 'name_type' value in ways_tags table is an abbreviation about the type of the way, so not sure whether are they all good and match to the name. So wrote a query to check them:

```
SELECT              WT.VALUE AS `NAME`, AB.VALUE AS ABBR
FROM                WAYS_TAGS WT,
                    (
                        SELECT                          *
                        FROM                            WAYS_TAGS WT
                        WHERE                           WT.KEY = 'NAME_TYPE'
                    ) AB
WHERE               WT.ID = AB.ID AND WT.KEY = 'NAME'
;
```

Here are the top ten results, beginning with the highest count:

```
NAME                               ABBR
Market Street                      St
De Baliviere Avenue                Ave
Kay Court                          Ct
Hayes Lane                         Ln
St Ferdinand Avenue                Ave
Sublette Avenue                    Ave
Sublette Avenue                    Ave
Sublette Avenue                    Ave
Odell Street                       St
Odell Street                       St
Odell Street                       St
Odell Street                       St
Odell Street                       St
Lake Avenue                        Ave
Lake Avenue                        Ave
Grandview Place                    Pl
Grandview Place                    Pl
Locke Avenue                       Ave
South 6th Street                   St
```

This abbreviation is not good, So I wrote piece of Python code to clean them

```
import pandas as pd
# read the csv file
ways_tags_df = pd.read_csv('ways_tags.csv')
# get the unique id list for cleansing
ids = list(ways_tags_df['id'].unique())

# for each id in the whole csv file
for each_id in ids:
    # get all content under this id
    temp_frame_each_id = ways_tags_df.query(''' id == %d ''' % each_id)
    # if there is a 'name' key under this id
    if 'name' in list(temp_frame_each_id['key']) and 'name_type' in
```

```
list(temp_frame_each_id['key']):
        # get the last word of the value whose key is 'name'
        # from observing, that would be the full type name of this way, like
'Street', 'Avenue'
        # instead of 'st', 'ave'
        way_type_full = temp_frame_each_id.query(''' key == 'name'
''')['value'].values[0].split()[-1]
        # get the index of the 'name_type' which is under the same id
        name_type_index = temp_frame_each_id[temp_frame_each_id['key'] ==
'name_type'].index[0]
        # update the value of it using the index we get
        ways_tags_df.loc[name_type_index, 'value'] = way_type_full
```

I did this cleansing process using Pandas, which is a very good package in python. After this, the 'name_type' will have the full name of the type instead of the abbreviation. Here's the result

```
NAME                      ABBR
Market Street             Street
De Baliviere Avenue       Avenue
Kay Court                 Court
Hayes Lane                Lane
St Ferdinand Avenue       Avenue
Sublette Avenue           Avenue
Sublette Avenue           Avenue
Sublette Avenue           Avenue
Odell Street              Street
Odell Street              Street
Odell Street              Street
Odell Street              Street
Odell Street              Street
Lake Avenue               Avenue
Lake Avenue               Avenue
Grandview Place           Place
Grandview Place           Place
Locke Avenue              Avenue
South 6th Street          Street
```

Now they're much better.

ID is too large for int

ID value is too large for int data type, when I use the wizard to import the csv file, it keep telling me that the value is out of range. So I dropped the table and import again giving the ID 'BIGINT' data type, then it's all good.

Postcode looks weird

I checked with google about the postcode in St. Louis county, they all start with '63'. However I noticed there are lots of postcodes start with '62', so I wrote a query to check how many weird postcodes start with '62'

```
SELECT                          WT.VALUE, COUNT(WT.VALUE) AS NUM_OF_POST
FROM                            WAYS_TAGS WT
WHERE                           WT.KEY LIKE '%zip%'
GROUP BY                        WT.VALUE
ORDER BY                        NUM_OF_POST DESC
;
```

These postcodes which show up the most are all postcode from St. Louis, MO, looking good:

```
VALUE   NUM_OF_POST
63376   2606
63021   2150
63017   1987
63031   1957
63026   1956
63301   1944
63123   1862
63366   1776
63122   1774
63303   1703
```

Many of these starts with '62' are the postcode from Illinois, since St. Louis include the west part lies in Missouri and the east part lies in Illinois. So maybe this is not a problem.

```
VALUE   NUM_OF_POST
62269   1071
63114   1070
63050   1045
63034   992
63368   982
63051   976
62221   956
63049   936
63042   897
62223   880
63379   835
63043   817
63121   805
62220   804
62249   796
62208   769
62035   768
```

So then I count the postcodes start with '62' and selected them out to have a look.

```
SELECT                          WT.VALUE, COUNT(WT.VALUE) AS NUM_OF_POST
FROM                            WAYS_TAGS WT
WHERE                           WT.KEY LIKE '%zip%' AND WT.VALUE LIKE '62%'
GROUP BY                        WT.VALUE
ORDER BY                        WT.VALUE
```

```
;
```

```
VALUE   NUM_OF_POST
62001   158
62002   1540
62009   87
62010   447
62012   428
62013   2
62014   149
62018   135
62021   50
62022   79
62023   3
62024   406
62025   1494
62026   1
62028   93
62031   94
62033   278
62034   661
62035   768
```

From the above 20 lines we can see that there are a lot of them start with 62, which means that they're all sort of related to St. Louis while belongs to Illinois. That's interesting.

Data Overview and Additional Ideas

This section contains basic statistics about the dataset.

File sizes

```
saint-louis_missouri.osm ......... 419.7 MB
nodes.csv ........................ 158.4 MB
nodes_tags.csv ................... 3.5 MB
ways.csv ......................... 11.5 MB
ways_tags.csv .................... 35.5 MB
ways_nodes.cv .................... 52 MB
```

Number of nodes

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

1915416

Number of ways

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

193566

Number of unique users

```
SELECT          COUNT(DISTINCT(e.uid))
FROM            (
                  SELECT    uid
                  FROM      nodes
                  UNION ALL
                  SELECT    uid
                  FROM      ways
                ) e
;
```

1419 unique users

Top 10 contributing users

```
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;
```

The above code will cause an error saying ' lost connection with SQL server ' Below is the statistics from first 50k rows of the result

```
user                    num
woodpeck_fixbot         32792
Mark Sims               3696
Millbrooky              2742
maxerickson             2004
eric22                  1722
g246020                 790
frajer                  725
wegavision              567
Drew G                  548
CmdrThor                364
```

Additional Ideas

Improvements for encouraging users to contribute

From the above top 10 contributers we can see that the fixbot contribute much more than a normal user. So maybe for user, it's too tedious for them to type something into some text box on webpage.

Solution: Base on this, I'm thinking about a more interesting way of interacting with the system. Maybe an app that can automaticaly get the geographical info and only let the user input the more general stuff.
Also, we can contact the restaurants to provide coupons for the users who contribute a lot on food informations.
Or we can have an app which allows user to post pictures, if the user post a picture and type in the info for a place, then he or she will have a sort of points. When the points get to a certain level or number, they would have a virtual badge for this, or get some reward, like a coupon.
Problems: If there no some sort of reward, people might not have a great passion about this. Naturally, if we provide the rewards, cost would go up for sure. So we have to come up with a solution that either we can benifit from this, or let the stores or restaurants provide rewards.

About the source

We can see Bing is the main source of this, maybe they are collaborating?
Solution: On this, maybe we can find more sources to provide a much more accurate or comprehensive info. For example, can we include Google Map as well, maybe have some APIs for users to extract info from Google Map or Yelp.
If we have more than one sources on the same node, we can choose the most accurate one, or the most popular one, and also show the others as references.
Problems: We might encounter the problem that there are many redundant data, and those will take up our storage space quickly, or take up the computing resources quickly. Also, we might have trouble cleaning all these data to make it more efficient for user to use.

Pokemon

Solution and idea: What if we collab with Pokemon GO or that kind of AR games. Let users go to some place, and put some rare Pokemon there to encourage users go there and sign in with our app and then seduce them to type in some info we need :P. Or say while they are busy using their camera to catch Pokemon, we use their camera to scan for keywords, like street name and store name. In this way, users don't have to type anything and they can contribute while having fun, isn't it great.
Problems: Developing cost might be high, and there is a strong probability that the scanned keywords need to be further confirmed or validated.

Additional Data Exploration

Top 10 sources

```
SELECT        NT.VALUE, COUNT( NT.VALUE ) AS NUM_OF_SRC
FROM          NODES_TAGS NT
WHERE         NT.KEY LIKE 'source'
GROUP BY     NT.VALUE
ORDER BY     NUM_OF_SRC DESC
LIMIT 10
;
```

```
VALUE                  NUM_OF_SRC
Bing ....................... 800
USGS Geonames .............. 320
county_import_v0.1 ......... 186
Yahoo ...................... 78
survey ..................... 31
local_knowledge ............ 27
ourairports.com ............ 15
Local Knowledge ............ 9
Self ....................... 3
United States Census Bureau . 2
```

Top 10 appearing amenities

```
SELECT                          NT.VALUE, COUNT(*) AS NUM_OF_AMENTITIES
FROM                            NODES_TAGS NT
WHERE                           NT.KEY = 'amenity'
GROUP BY                        NT.VALUE
ORDER BY                        NUM_OF_AMENTITIES DESC
LIMIT                           10
;
```

```
VALUE                  NUM_OF_AMENTITIES
place_of_worship       1426
school                 842
grave_yard             513
restaurant             106
post_office            96
fire_station           82
fast_food              74
parking                45
fuel                   40
townhall               39
```

Religions

```
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
;
```

```
value                            num
christian                        1353
jewish                           3
```

```
unitarian_universalist           2
buddhist                         2
eckankar                         1
muslim                           1
```

Popular cuisines

```
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;
```

```
value                   num
american ............... 17
sandwich ............... 10
italian ................ 9
mexican ................ 6
pizza .................. 5
japanese ............... 3
thai ................... 3
burger ................. 2
vietnamese ............. 2
american;tex-mex ....... 1
dessert ................ 1
ice_cream .............. 1
italian-american ....... 1
coffee_shop  ........... 1
greek .................. 1
chicken ................ 1
indian ................. 1
steak_house  ........... 1
international .......... 1
sushi .................. 1
```

Conclusion

After this exploration of the dataset of St. Louis, the data for this area is definitely incomplete. However, surprisingly, the data is good enough, and there are not a lot of things to clean.
For me, there are so many very good Chinese food are not listed. And there are still some data are not good, such as some nodes don't have a 'user' or 'uid' attribute. Due to the limitedness of my computer, it's pretty hard for me to process all the data in some query.