P3 Wrangle OpenStreetMap Data

February 27, 2017

1 Intro

1.1 Map Area

Las Vegas (NV), United States - https://www.openstreetmap.org/relation/170117

I will be performing data wrangling on Las Vegas OpenStreetMap data. I chose this specific region because it is my current home city and I am interested in cleaning and improving the data for it.

2 Problems Encountered in the Map

I used the code in *P3 Code* jupyter notebook to audit and, clean and prepare the data.

Below are the problems that I noticed after I initially explored the data:

- Some addresses contain abbriviated street types in street (*W. Sahara Ave*) and use 'S' or 'S.' for 'South' (*Las Vegas Blvd S*)
- Street names containing unit/suite number which is supposed to be in `note` tag not in `addr:street` (*Howard Hughes Pkwy #790*)
- Several postal codes are in the wrong format. They start with state abbreviation NV instead of just being a number (*NV 89117 should be just 89117*)
- Way nodes have data uploaded from **Topologically Integrated Geographic Encoding and Referencing system (TIGER)** :

  ```
  <tag k="tiger:name_base" v="Sleek"/>
  ```

  According to the OpenStreetMap documentation:

  > "It is unlikely that the TIGER data ever will be imported again. Enough editing has occurred since the original upload of the TIGER 2005 data (which was not uploaded until 2007) that it will be difficult to determine if differences between future TIGER and OSM are because of good corrections made by OSM editors or from bad TIGER data. With the US mapping community growing strongly now, it isn't as much of a concern. Do not worry about getting your work overwritten by new TIGER data. Go map!"

Before converting the data from osm to csv and loading it to the database I will perform some data cleaning and organization.

2.1 Fixing Street Names

The code snippet below is from the `get_tags()` function. This code first checks if the tag is of type "address" and then if the value of the tag is in the `st_types` list. Then it runs functions to remove abbreviations and numbers from the street names.

```
if tag["key"] == "street":
            for key,value in st_types.iteritems():
                if i.attrib["v"] in value:
                    #Fix street type if it is in the `st_types` list we found before
                    tag["value"] = update_name(i.attrib["v"] , mapping)
                    #Remove building number from street address
                    tag["value"] = remove_bld_num(tag["value"], mapping)
```

2.1.1 Abbreviations

To replace abbreviated street names I used the function and mapping below. In the result of this function a name like *"3547 S Maryland Pkwy"* will be converted to *"3547 South Maryland Parkway"*

```
mapping = { "St": "Street", "St.": "Street", "Ave": "Avenue", "AVE": "Avenue",
            "Ave.": "Avenue", "ave": "Avenue",  "Blvd": "Boulevard",
            "Blvd.": "Boulevard", "blvd": "Boulevard", "blvd.": "Boulevard",
            "Rd": "Road", "Rd.": "Road", "Rd5": "Road", "Dr" : "Drive",
            "Dr." : "Drive", "Pkwy" : "Parkway", "Cir" : "Circle", "Ln" : "Lane",
            "Ln." : "Lane","S": "South","S.": "South", "N": "North", "N.": "North",
            "W": "West", "W.": "West", "E": "East","E.": "East"}

def update_name(name, mapping):
    new_name = []
    for i in name.split(" "):
        if i in mapping.keys():
            i = mapping[i]
        new_name.append(i)
    return " ".join(new_name).replace(",","")
```

2.1.2 Numbers After Street Names

In order to fix street names with suite/unit number I used the function below. It will convert an address like *"2230 Corporate Circle Suite 250"* in to *"2230 Corporate Circle"*

```
bld_num = ["Suite", "Ste", "#", "STE"]

def remove_bld_num(name, mapping):
    for i in name.split(" "):
        if any(bld in i for bld in bld_num):
            return name.split(i)[0].strip()
    return name
```

There are not very many abbreviated street names or street names containing suite/unit number. In terms of street addresses the data I have received from OpenStreetMap was pretty clean already.

2.2 Fixing postal codes

To fix postal codes I first check if the postal code is in the wrong format (i.e. does not start with '89'). The below code snippet is from the `get_tags()` function.

```
if tag["key"] == "postcode":
     if not i.attrib['v'].startswith("89"):
        tag["value"] = fix_zip(i.attrib['v'])
```

The above code calls `fix_zip()` function in the case of the invalid zipcode:

```
def fix_zip(name):
    return name[-5:]
```

In the result of this code invalid zip code like *"NV 89119"* will be converted to *"89119"*

There were not many invalid postal codes in the whole dataset (only about 20). Most likely these were caused simply by an error.

3 Overview of the data

3.1 FIle sizes

The sizes of the files used in the project are following:

- las-vegas_nevada.osm - 205.6MB
- nodes.csv - 78.2MB
- nodes_tags.csv - 2.2MB
- ways.csv - 5.8MB
- ways_tags.csv - 17.8MB
- ways_nodes.cv - 27.9MB

(**Note**: I am using SQL Server database instead of SQLite in my project.)
Size of the SQL tables containg the data (using sql query: `EXEC sp_spaceused N'tablename'`):

- nodes table - 44.8MB
- nodes_tags table - 3MB
- ways table - 3.6MB
- ways_tags table - 21.9MB
- ways_nodes table - 65.7MB

3.2 Overview Statistics

3.2.1 Number of unique users

788 unique users

```
select count(distinct u.uid)
    from (select uid from nodes union select uid from ways) as u
```

3.2.2 Number of Nodes and Ways

524,287 nodes and **101,159** ways

```
select count(id)
from nodes

select count(id)
from ways
```

3.2.3 Number of chosen type of nodes

Number of nodes with cuisine value 'burger': 86

```
select count(distinct id) from nodes_tags where value like 'burger'
```

3.2.4 Additional statistics

Top most used 3 key/value combinations from node_tags:

key	value	num
highway	crossing	7036
highway	turning_circle	6979
power	tower	6753

```
   select top 3 key, value, count(*) num   from nodes_tags  group by key, value
order by num desc
```

4 Other ideas about the dataset

4.1 Ideas for additional improvements, their benefits or problems

Upon further investigation of top used key/value pairs in nodes_tags I found out that `state/'state name'` pair is presented more than once. It will be right to think that the dataset should only have 1 such pair rnamely **state/NV**. However, below is result and the the query that show all the state values present and their count

```
select key, value, count(*) num
from nodes_tags
where key='state'
```

key	value	num
state	NV	491
state	AZ	40
state	Nevada	4

We see that there are 4 nodes with state 'Nevada' as opposed to 491 nodes with state 'NV' which makes me come to the conclusion that the `NV` is the default format for the state. Moreover, there are 40 nodes with state 'AZ', which seems very odd to me since we are looking at the data for the city in Nevada not Arizona.

If converting `state/Nevada` to `state/NV` seems pretty much obvious in benefiting the data cleaning and uniformity, then `state/AZ` may cause a problem and needs further investigation.

In the query below I select all names for the nodes that have state 'AZ' to see what map point they represent.

```
select n.id, nt.key, nt.value
FROM nodes n
left join nodes_tags nt on n.id=nt.id
where n.id in (select id from nodes_tags where key='state' and value ='AZ' )
and nt.key='name'
order by n.id desc
```

As a result of this query I got all the names for map points with 'AZ' as a state. The names share something in common: almost all of them are either 'Bay', 'Cove' or 'Mine', natural points in other words. However, there is one exception: one point is an airport (Temple Bar Airport). As one possible explanation for nature points: these map points may be on the border of NV and AZ states. But can an airport be on the border of 2 states?

5 References

http://wiki.openstreetmap.org/wiki/TIGER