Data Analyst Nanodegree – DAND P3 Project: OpenStreetMap Data Case Study Report

By Teresa Aysan – Submitted April 29, 2017

Map Area

Nashville, TN, United States.

I selected Zen Nashville Raw OpenStreet DataSet. As directed in the instruction, the map information text file explains why. Note that all the results below are based on the full nashville_tennessee.osm file, not the sample_100.osm file that was submitted in the zip file.

1. Problems Encountered in the Map

I started by cleaning the street types as we had been taught in the course. I thought I would also clean other data elements. I ended up learning how to thoroughly clean the street names, not just the street types.

Lessons learned from cleaning street names:

- Many of the addr:street names were corrupted and needed to be cleaned elsewhere in the street name, not just the street type. These are the steps that led me there:
- Step 1: Is the last word in street name an expected street type? If yes, no update required.
- Step 2: Is the last word in the type_mapping dictionary? If yes, update.
- Investigate the output. Some street types and words inside the street name also need cleaning.
- Step 3: If "no" to both steps 1 and 2 then loop through all the words in the street name and update based on the "not_found_mapping" dictionary which is a dictionary that I created to map all kinds words / abbreviations to the correct word.

```
words = [x.strip() for x in name.split()]
for w in range(len(words)):
  if words[w] in not_found_mapping:
    words[w] = not_found_mapping[words[w]]
  name = " ".join(words)
```

- Step 4: While doing that, I realized that in some cases where the new type contained the same order of letters as the old type, that I ended up with an oddball name. Ave to Avenue; St to Street or str to Street. But not Ln to Lane for example. To fix these, I learned how to use the boundaries argument so that the whole word would be considered during the cleaning and not just part of the word. For example, St = Street...
    - Without boundaries, Stuart would change to Streetuart,
    - With boundaries, it stays as Stuart.

```
boundaries = re.compile(r'\b'+ m.group() + r'\b')
name = re.sub(boundaries, type_mapping[m.group()], name)
```

- Step 5: For oddballs, I learned to use the replace command:

`name = name.replace (C1TY AVENUE','City Avenue')` as an example – note the number 1 in C1TY).

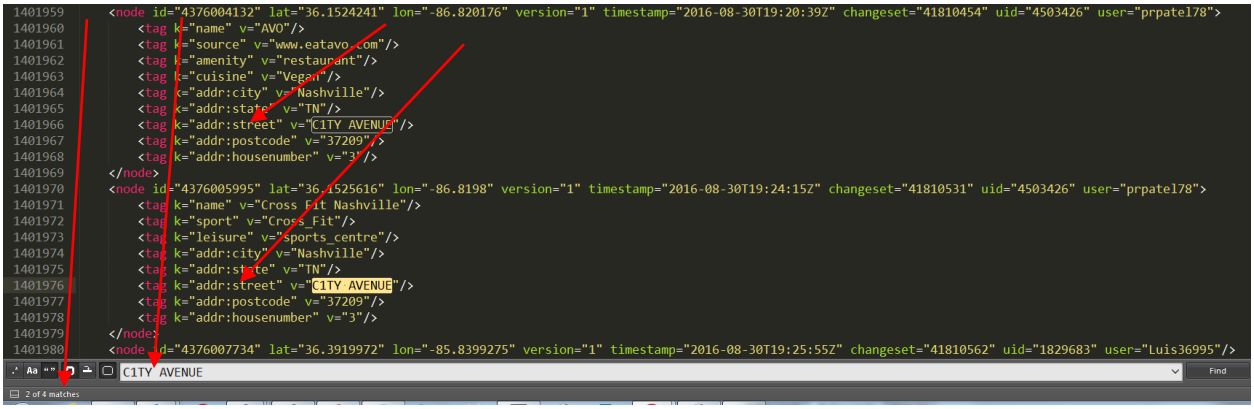
    - I learned how to print the path and file name that was executing.
    - I learned how to put my mapping dictionaries into their own .py file and then call that file / those dictionaries into my audit / cleaning / updating and parsing code so that I didn't have to update multiple versions of the dictionaries in each code file as I discover new types or new words to add to the dictionaries.

```
from myDictionaries import  not_found_mapping, type_mapping, expected
```

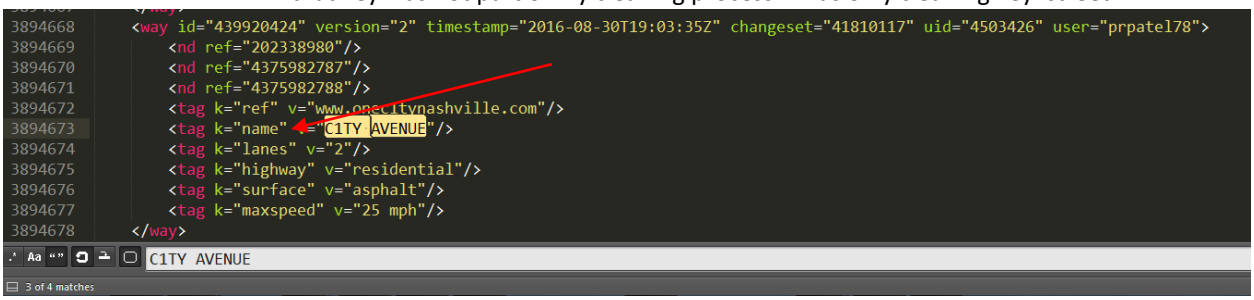
    - Sample before and after updates through the data.py code to show that my code works:
        - This one did not need cleaning:
            - child.attrib[v] before update Main Street
            - child.attrib[v] after update Main Street
        - This one did, for the same street but a different record:
            - child.attrib[v] before update Main St

- child.attrib[v] after update Main Street

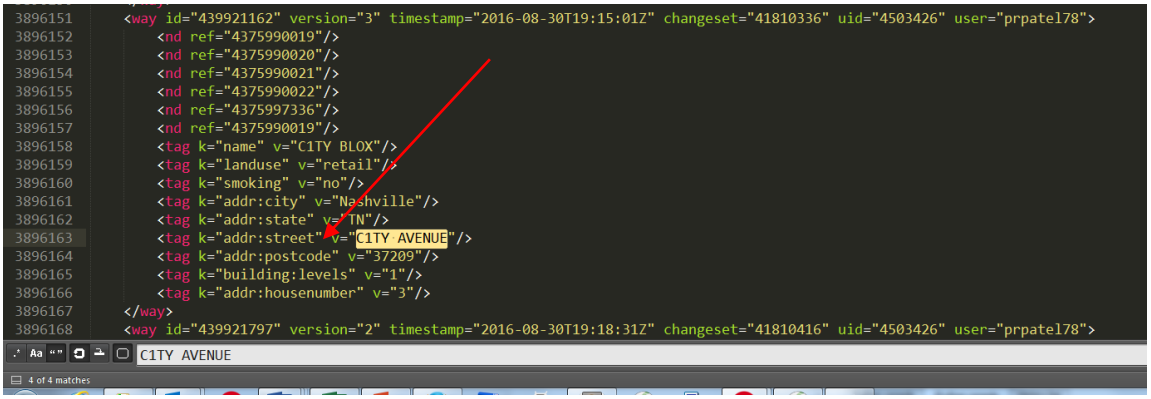
- This one needed cleaning in many ways: change C1TY to City (note the #1), AVENUE to Avenue:
  - child.attrib[v] before update C1TY AVENUE
  - child.attrib[v] after update City Avenue
- Search for “C1TY AVENUE” in the raw text file of data:
  - Note that there are 4 matches. Two in nodes_tags and 2 in ways_tags.
  - In the nodes_tags, both got cleaned because they are key ‘street’ (I will tell you below how to check that they are cleaned)



- This one in ways_tags did not get cleaned because it is the value for key ‘name’ – and that key was not part of my cleaning process. I was only cleaning key ‘street’



- This one in ways_tags got cleaned because the key is ‘street’



- Evidence that the data is cleaned: When you run my code “P3 SQL Queries.py”, select option “0. Cleaning data proof”.
- Which is a query that looks for ‘C1TY AVENUE’ and ‘City Avenue’ in both nodes_tags and ways_tags. Look at the output and you will see that the data is cleaned.
- For query: `SELECT * FROM nodes_tags  WHERE value = 'C1TY AVENUE' ;`

- Output is: Number of rows: 0 There are no records found.

- For query: `SELECT * FROM nodes_tags WHERE value = 'City Avenue' ;`
  - Output is: Number of rows: 2

```
        id    key       value  type
0  4376004132  street  City Avenue  addr
1  4376005995  street  City Avenue  addr
```

- For query: `SELECT * FROM ways_tags WHERE value = 'C1TY AVENUE' ;`
  - Output is: Number of rows: 1

```
        id  key      value     type
0  439920424  name  C1TY AVENUE  regular
```

- For query: `SELECT * FROM ways_tags WHERE value = 'City Avenue' ;`
  - Output is: Number of rows: 1

```
        id    key       value  type
0  439921162  street  City Avenue  addr
```

- I learned that even though we think that we have cleaned all the data of a certain type, a Data Analyst has to be careful to select all the keys that need cleaning.

2. Overview of the Data

- mapparser code outputs the following statistics which shows how many nodes, tags, and ways to get the feeling on how much of which data you can expect to have in the map.

```
{'bounds': 1,
 'member': 16862,
 'nd': 1510808,
 'node': 1325949,
 'osm': 1,
 'relation': 1904,
 'tag': 924006,
 'way': 136702}
```

- My code "P3 SQL Queries.py", has a lot of queries. For your convenience I have divided them into groups. You can select the group that you want when you run the code. We talked about group 0. above.
- Group "1. File sizes." This will output the sizes of the files in a certain path. It will ask you for your path. This is the output when I select "Teresa's path" locally – you cannot do this because I could not submit these large files:

```
FILE SIZES:
a.  myNashvilleOSMdb.db.....: 162M
b.  nashville_tennessee.osm.....: 283M
c.  nodes.csv.....: 106M
d.  nodes_tags.csv.....: 4M
e.  ways.csv.....: 7M
f.  ways_nodes.csv.....: 35M
g.  ways_tags.csv.....: 26M
```

- I learned how to use 'global' to be able to use a global variable inside a definition. (e.g. global RESTART)
- I learned that this error *"ValueError: Length mismatch: Expected axis has 0 elements, new values have 8 elements"* means that my query is returning no results. To prevent that in the future, I put an `if len(rows) == 0:` condition to bypass the error.
- I learned how to put my queries in a list, so that I would not be repeating 'execute' and 'print' code for every query. I also learned how to use dataframes to have nicer looking output:

```
queries_list = ([SELECT * FROM nodes_tags  WHERE value = 'C1TY AVENUE' ; , SELECT * FROM nodes_tags  WHERE value = 'City Avenue' ])
for query in queries_list:
    print "\nFor query: ", query, "\nOutput is:"
    cur.execute(query)
    # Printing with panda dataframe
    names = [description[0] for description in cur.description]
    rows = cur.fetchall()
    if len(rows) == 0:
       print "\nThere are no records found.\n"
    else:
       df = pd.DataFrame(rows)
       df.columns = names
       print df
```

- But option "2. Simple Query: Printing a list without using pandas dataframes(df)" shows that I also know how to print without df.
- User queries are in option "4. Using queries to understand the values of the tags." Uses UNION, HAVING:
    - Number of DISTINCT users across nodes and ways:

      SELECT COUNT(DISTINCT(subq1.uid)) FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) subq1;
      Output is: 1035
    - Number of users that contributed only once:

      SELECT COUNT(*) FROM (SELECT subq1.user, COUNT(*) as num FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) subq1 GROUP BY subq1.user HAVING num=1) subq2; **Output** is: 184
    - The top 10 user contributors & # of contributions:

      SELECT uid, user, COUNT(uid) AS uid_count FROM nodes GROUP BY uid ORDER BY COUNT(*) DESC LIMIT 10;

uid	user	uid_count
0	147510	woodpeck_fixbot 277898
1	1791301	Shawn Noble 149198
2	44793	st1974 88092
3	120146	TIGERcnl 52269
4	371121	AndrewSnow 51120
5	510836	Rub21 48617
6	1767688	StevenTN 27429
7	42429	42429 25545
8	153669	dchiles 24055
9	2400191	darksurge 23857
    - All users:

      SELECT uid, COUNT(*) AS uid_count FROM nodes GROUP BY uid ORDER BY COUNT(*) DESC #; **Output is a list. From that list:**
        - Total # of users in nodes (not combined with ways) is 994,
        - max contributions is 277898
        - min is 1.

      Option "5. Using queries and subqueries for **averages** ...":
      SELECT uid, avg(uid_count) AS average FROM (SELECT uid, COUNT(*) AS uid_count FROM nodes GROUP BY uid ORDER BY COUNT(*) DESC) as subquery;
      Output: **average** # of contributions per user is 1406

Wrestling with my overall objective.

I pretended that I worked for the municipality of Nashville. I wanted an overall query that would be important to the municipality of Nashville from a business perspective. I decided to research which "counties" ''Airport Road" goes through because I was planning a major road reconstruction and I wanted to engage the country governments for awareness and input.

Through investigation of the data by looking at the txt file and by doing queries, I discovered that "county" was a key in ways_tags type "tiger" and in relation type "gnis"; it was a value in relation keys "border_type" and "place"; and it was a word in some of the "name" keys. It took many many hours of queries and investigation to discover that. Then I was left with the problem of which one to use to meet my objective. I tried several queries of JOINing databases and complex WHERE and GROUP conditions, but I kept running into the problem of "no records" found because of the combination of street = "Airport Road" AND trying to get a list of the counties.

I discovered that the counties that I wanted were in ways_tags with key "name" and type "regular". But how do I do a combo query on the same file, selecting key = "street" with value "Airport Road" AND key = "county" with any value? It did not seem possible. I kept getting 0 records no matter what combination I tried.

Through building the complex queries, I finally figured out that I had to **JOIN** the ways_tags database **to itself** to get the list of counties that Airport Road goes through. That's why you will see so many queries in my "P3 SQL Queries.py" code. The final query that worked is in this selection of my queries code: "8. Queries to find statistics on which Counties Airport Road goes through." I listed the ways_tags records in the output and I counted them so I would have a DISTINCT list with the number of times that "Airport Road" goes through each county.

```
def airportCountiesQueries():
    print "\nQueries to find statistics on which Counties Airport Road goes through."
    queries_list = ([ ...
                '''SELECT ways_tags.id, ways_tags.key, ways_tags.value, ways_tags.type, sq1.id, sq1.key, sq1.value, sq1.type
                FROM ways_tags JOIN (SELECT id, key, value, type FROM ways_tags
                WHERE value = 'Airport Road') as sq1 ON ways_tags.id = sq1.id WHERE ways_tags.key = 'county' ;''',
                '''SELECT ways_tags.id, ways_tags.key, ways_tags.value, sq1.id, sq1.value, COUNT('ways_tags.value') as num
                FROM ways_tags JOIN (SELECT id, key, value FROM ways_tags
                WHERE value = 'Airport Road') as sq1 ON ways_tags.id = sq1.id WHERE ways_tags.key = 'county'
                GROUP BY (ways_tags.value) ;'''
               ])
    for query in queries_list:
        print "\nFor query: ", query, "\nOutput is:"
        cur.execute(query)
        # Printing with panda dataframe
... same as above df code
```

```
The output for the full list is: Number of rows:  17
   ways_tags.id ways_tags.key ways_tags.value ways_tags.type sq1.id  sq1.key sq1.value sq1.type
0      6822024       county    Hickman, TN       tiger   6822024   name  Airport Road  regular
1      6822041       county    Hickman, TN       tiger   6822041   name  Airport Road  regular
2     19386922        county     Bedford, TN       tiger  19386922   name  Airport Road  regular
...
14    115931306        county      Sumner, TN       tiger 115931306   name  Airport Road  regular
15    266744444        county    Robertson, TN       tiger 266744444   name  Airport Road  regular
16    266744445        county    Robertson, TN       tiger 266744445   name  Airport Road  regular
```

```
The output for the summary list is: Number of rows:  6
  ways_tags.id ways_tags.key ways_tags.value    sq1.id    sq1.value  num
0     19386927       county     Bedford, TN   19386927  Airport Road   3
1      6822041       county     Hickman, TN    6822041  Airport Road   2
2     19540751       county      Macon, TN   19540751  Airport Road   1
3     19567576       county  Montgomery, TN   19567576  Airport Road   1
4    266744445        county   Robertson, TN 266744445  Airport Road   3
5    115931306        county      Sumner, TN 115931306  Airport Road   7
```

I learned that it was a limitation of SQL and I realized that it would get very onerous if I wanted to have the output of a lot of other keys' values, so I asked the mentor if there was a simpler way, but he assured me that my code is as efficient as can be: For records with a tag id 'y' and value 'x', what other tags in that same record have an id 'z' have a value '*'? My code has one subquery that accesses the id 'y' with value 'x', and a main query that determines the rows that have that id 'z' with value '*'. I have to have as many subqueries as WHERE conditions.

Additional Statistics

As an employee of the municipality of Nashville, I'm interested in data that can assist me with managing my civic operations.

- What are the major cities referenced in the Nashville database? The use of UPPER was necessary in order to properly group the cities.

```
SELECT value, COUNT(*) AS num FROM nodes_tags WHERE key='city' GROUP BY UPPER(value) ORDER BY num DESC LIMIT 10
```

Clarksville	632
Nashville	177
Murfreesboro	35
Franklin	31
Brentwood	20
Springfield	16
Nolensville	14
Spring Hill	10
Columbia	8
McMinnville	6

- How many entries are there for 'Airport Road' in nodes_tags that have a reference in ways_nodes? This required JOINing databases. There were 3: One “JOSM” and two “level_crossing”

```
SELECT key, id, value FROM nodes_tags JOIN (SELECT DISTINCT(node_id) FROM ways_nodes JOIN (SELECT DISTINCT(id) FROM ways_tags
WHERE value = 'Airport Road') as subq  ON ways_nodes.id = subq.id) as subq1 ON nodes_tags.id = subq1.node_id;
```

- What are the top 30 ways_tags keys? They include ‘highway’ as the highest repetition, ‘county’, ‘building’, ‘service’, ‘access’, etc.

```
SELECT key, COUNT(*) FROM ways_tags GROUP BY key ORDER BY COUNT(*) DESC LIMIT 30;
```

- What are the top 30 nodes_tags keys? I’m showing you the top 10 in the interest of space:

```
SELECT key, COUNT(*) FROM nodes_tags GROUP BY key ORDER BY COUNT(*) DESC LIMIT 30
```

	key	COUNT(*)
0	power	20818
1	name	12252
2	ele	10406
3	highway	9649
4	feature_id	8926
5	amenity	8616
6	created	8558
7	county_id	8407
8	state_id	8407
9	source	3222

3. Other ideas about the dataset

The database structure should be improved to allow users to do more complex queries, with multiple tags, as evidenced by my need to find the counties that Airport Road goes through.

Municipalities, governments, government agencies and general business **would benefit** by making it easier and less cumbersome to use the data to research with multiple conditions. For example, which companies are located on 2nd Avenue within the following quadrant of latitudes and longitudes, so that I can advise them that 2nd Avenue will be closed for repair? This is my query that investigated the quadrant of Nashville based on lat and lon > and < average. The complex query described is for future investigation.

```
'''SELECT id, lat, lon, user, uid, version, changeset, timestamp FROM nodes, (SELECT avg(lat) AS lat_avg FROM nodes) AS subq1, (SELECT avg(lon)
AS lon_avg FROM nodes) AS subq2 WHERE lat>lat_avg AND lon>lon_avg;'''
```

The potential problem is that there would need to be more relational sub databases, and it would require some thought and input by the users to decide logical divisions of the existing nodes and ways databases.