Lesson 4 Wrangle OpenStreetMap Data project

For this project, I have selected Fullerton, CA, USA (https://www.openstreetmap.org/relation/2312655) as my location and will use SQL as the database.   I chose Fullerton, because this is where I grew up. While I don't live there any longer, my parents still live there. The data file downloaded open the OpenStreetMap website called 'fullerton_ca.osm' file is almost 300 MB. I will, thus, use a smaller file to first test and clean data before moving onto the actual file. I used the code provided in the 'Project Details' section to create a smaller sample file.

These are the steps I took.

1. Clean and observe the smaller dataset which took every 50-th top elements
2. Clean and observe the smaller dataset which took every 10-th top elements
3. Clean and observe the full dataset
4. Create csv files, create sql tables, and insert csv files into sql
5. Run SQL queries to provide an overview of the dataset
6. Other ideas about the dataset

At the end of the each of the steps details below, I have added a list of some function calls that I have made.

1. Cleaning the smaller dataset by cutting the file by taking every 50-th top level element

I first went through the smaller dataset by cutting the file by taking every 50-th top level element. Going through the smaller dataset helps to understand it quicker than from looking at the massive dataset initially. I checked to see how the data are organized by opening it in WordPad. The format looks to be the same as what we had observed in the lessons, with 'node' and 'way' tags.

In python, I made the "get_tags(SAMPLE_FILE_50)" function call, which returns the dictionary of tag and number of times it appears. This resulted in the following:

dictionary of tag and counts

```
dictionary of tag and counts
{'member': 144,
 'nd': 28295,
 'node': 24189,
 'osm': 1,
 'relation': 23,
 'tag': 16660,
 'way': 2700}
```

The tag names look familiar from the lessons. The 'relation' and 'member' tags will be ignored since the project does not require creating tables for them.

Next, I printed the keys to the 'tag' node to see what data are captured. The 'tag' node had much more keys than what I had expected. There were especially a lot of 'tiger:key' keys. At least what I found out was that these are Topologically Integrated Geographic Encoding and Referencing (TIGER) data (https://en.wikipedia.org/wiki/Topologically_Integrated_Geographic_Encoding_and_Referencing). I will assume these data to be correct.

Checking for errors

I first checked the latitude and longitude values of the 'node'. Since the latitudes and longitudes of places in a city should be very close, these data should be in a very tight range. Latitude and longitude are in a very small range as expected and there seems to be no non-float value. Minimum and maximum for latitude are 33.7 and 34.0, while minimum and maximum for longitude are -118.1 and -117.6, respectively.

Next, I checked if all the house numbers are integers and whether the zipcodes are 5 digit integers. While all the house numbers were integer, there were one data in 'addr:postcode' of ['CA 90638'], which I made a note to fix later.

I then made to check some string data. Checking 'addr:state', 'addr:city' and 'addr:street' as well as street type in 'addr:street' showed that they almost all clean. One street address started with 'E.', which I want to change to 'East'.

FUNCTION CALLS MADE:

```
SAMPLE_FILE_50 = "fullerton_sample_k_50.osm"
cut_sample_file(50, SAMPLE_FILE_50)
get_tags(SAMPLE_FILE_50)
get_tag_keys(SAMPLE_FILE_50)
check_lat_lon(SAMPLE_FILE_50)
check_tag_attrib_is_int(SAMPLE_FILE_50, 'addr:housenumber')
check_tag_attrib_is_int(SAMPLE_FILE_50, 'addr:postcode')
check_tag_attrib_is_str(SAMPLE_FILE_50, 'addr:state')
check_tag_attrib_is_str(SAMPLE_FILE_50, 'addr:city')
check_tag_attrib_is_str(SAMPLE_FILE_50, 'addr:street')
get_street_types(SAMPLE_FILE_50)
```

2. Re-running the checks by using dataset which took every 10-th top level element

I created a new data file to be every 10-th top level element of the original data, and went through the same error-checking process as I in #1 using the new file. There were more errors to fix, of which I made a list.

- addr:postcode: 'CA 90638' -> '90638'
- addr:postcode: 'Disneyland' -> '92802'
- addr:state: 'ca' -> 'CA'
- addr:city: 'la habra' -> 'La Habra'
- addr:street: starts with 'E ' or 'E. ' -> starts with 'East ' and similary for West, South and North
- addr:street: ends with 'Ave' -> ends with 'Avenue'
- addr:street: ends with 'Blv', 'Blvd' or 'Blvd.' -> ends with 'Boulevard'
- addr:street: ends with 'Dive' -> ends with 'Drive'
- addr:street: ends with 'Wy' -> ends with 'Way'
- addr:street: 'E La Palma Ave #G' -> 'East La Palma Avenue'
- addr:street: 'stephens' and ' stephens'-> 'North Euclid Street'

For 'stephens' and ' stephens' (one has an extra space in the beginning), I checked the latitude and longitude of where they appeared on the OpenStreetMap. Both time, it brought me to 'North Euclid Street'. Here is the link to one of them http://www.openstreetmap.org/search?query=33.8743353%2C-117.9420547#map=18/33.87434/-117.94206 .

I have also checked many more fields using the 'check_tag_attrib_is_str' function and passing in the name as a parameter, but there were no other errors to correct.

Before moving onto the full dataset, I created a function to make the above edits to the data. There were several ways to do this. One way was to edit by rule-based. For example for the 'addr:postcode' of 'CA 90638', if there is a space in the string, I can try to split by a space. Then I can check if the last value in the list is an integer and take that value in that case. However, I decided to create the mapping dictionary of errors and fixes, and to use that to edit. This was because there were only few instances of total correction. I tested the edit logics first before I moved onto the full dataset.

FUNCTION CALLS MADE:

```
SAMPLE_FILE_10 = "fullerton_sample_k_10.osm"
cut_sample_file(10, SAMPLE_FILE_10)
get_tags(SAMPLE_FILE_10)
get_tag_keys(SAMPLE_FILE_10)
check_lat_lon(SAMPLE_FILE_10)
check_tag_attrib_is_int(SAMPLE_FILE_10, 'addr:housenumber')
check_tag_attrib_is_int(SAMPLE_FILE_10, 'addr:postcode')
check_tag_attrib_is_str(SAMPLE_FILE_10, 'addr:state')
check_tag_attrib_is_str(SAMPLE_FILE_10, 'addr:city')
check_tag_attrib_is_str(SAMPLE_FILE_10, 'addr:street')
```

```
get_street_types(SAMPLE_FILE_10)
edit_tag_error_value(SAMPLE_FILE_10)
plus many other check_tag_attrib_is_str(SAMPLE_FILE_10, 'key')
```

3. Clean and observe the full data set

I followed the same procedures as #2 above to observe and edit the full dataset. After finding more errors, I updated the error maps to include them, as well as any one-off errors. For example, 'addr:housenumber' with 'Radiator Springs Racers Show Building' value should be removed from the 'way' node, as this is a ride in Disneyland. And this required a separate function, because it was not editing the 'value' in a node but was removing the node.

Once I was satisfied with the error edits, I created a new csv file with the correction. I went through the error check once again on the cleaned csv file to make sure I did not miss anything.

FUNCTION CALLS MADE:

```
<CHECKING THE FULL FILE>
get_tags(OSM_FILE)
get_tag_keys(OSM_FILE)
check_lat_lon(OSM_FILE)
check_tag_attrib_is_int(OSM_FILE, 'addr:housenumber')
check_tag_attrib_is_int(OSM_FILE, 'addr:postcode')
check_tag_attrib_is_str(OSM_FILE, 'addr:state')
check_tag_attrib_is_str(OSM_FILE, 'addr:city')
check_tag_attrib_is_str(OSM_FILE, 'addr:street')
get_street_types(OSM_FILE)
edit_tag_error_value(OSM_FILE)

<CREATING A NEW CLEAN FILE>
OSM_FILE_CLEAN = "fullerton_ca_clean.osm"
edit_tag_error_value(OSM_FILE, OSM_FILE_CLEAN)

<CHECKING THE NEW CLEAN FILE>
check_tag_attrib_is_int(OSM_FILE_CLEAN, 'addr:housenumber')
check_tag_attrib_is_int(OSM_FILE_CLEAN, 'addr:postcode')
check_tag_attrib_is_str(OSM_FILE_CLEAN, 'addr:state')
check_tag_attrib_is_str(OSM_FILE_CLEAN, 'addr:city')
check_tag_attrib_is_str(OSM_FILE_CLEAN, 'addr:street')
```

```
get_street_types(OSM_FILE_CLEAN)
```

4. Create csv files, create sql tables, and insert csv files into sql

Data looks clean and we are ready to upload into sql. I created the csv files, utilizing the code used from the lessons. As I was uploading the csv files into sql later on, there were error message of datatype mismatch. After doing some investigations, I found out these were due to the header row. So I did not include the header rows on the csv files and made some small changes.

I then printed out the first five rows of the csv files and counted the number of rows so I can check versus the uploaded sql tables later.

I then created tables in sql by running the following (https://gist.github.com/swwelch/f1144229848b407e0a5d13fcb7fbbd6f). This was followed by uploading the data in to the tables.

```
sqlite> .mode csv
sqlite> .import nodes.csv nodes
sqlite> .import nodes_tags.csv nodes_tags
sqlite> .import ways.csv ways
sqlite> .import ways_nodes.csv ways_nodes
sqlite> .import ways_tags.csv ways_tags
```

FUNCTION CALLS MADE:

```
OSM_FILE_CLEAN = "fullerton_ca_clean.osm"
create_clean_csv(OSM_FILE_CLEAN)
peek_csv('nodes.csv', NODE_FIELDS)
peek_csv('nodes_tags.csv', NODE_TAGS_FIELDS)
peek_csv('ways_nodes.csv', WAY_TAGS_FIELDS)
peek_csv('ways_tags.csv', WAY_NODES_FIELDS)
```

5. Run SQL queries to provide an overview of the dataset

I first checked the number of rows for each tables to see if they match the counts from the csv file. And they matched!

```
sqlite> select count(*) from nodes;
1209409
sqlite> select count(*) from nodes_tags;
38927
sqlite> select count(*) from ways;
135004
```

```
sqlite> select count(*) from ways_nodes;
1415520
sqlite> select count(*) from ways_tags;
792347
```

I then decided to run some SQL queries to observe the data. I first wanted to see who the biggest users were in Nodes and Ways as well as for both combined.

```
-----Biggest users for Nodes table-----
select user, count(user) as num from nodes group by uid order by num desc
limit 10;
your output:
   (u'calfarome_labuilding', 241788)
   (u'Brian@Brea', 161545)
   (u'RichRico_labuildings', 145295)
   (u'piligab_labuildings', 112562)
   (u'Jothirnadh_labuildings', 95288)
   (u'yurasi_import', 87260)
   (u'manoharuss_imports', 51091)
   (u'dannykath_labuildings', 48031)
   (u'Aaron Lidman', 32136)
   (u'saikabhi_LA_imports', 24381)

-----Biggest users for Ways table-----
select user, count(user) as num from ways group by uid order by num desc
limit 10;
your output:
   (u'Brian@Brea', 21046)
   (u'calfarome_labuilding', 20702)
   (u'RichRico_labuildings', 14481)
   (u'piligab_labuildings', 9854)
   (u'yurasi_import', 7697)
   (u'Jothirnadh_labuildings', 6870)
   (u'Aaron Lidman', 6092)
   (u'manoharuss_imports', 4818)
   (u'dannykath_labuildings', 4302)
   (u'karitotp', 3848)

-----Biggest users for both Nodes and Ways tables-----
select user, sum(n) as tot  from ( select user, count(*) n from nodes group
by user UNION ALL select user, count(*) from ways group by user ) x group by
user order by tot desc limit 10;
your output:
   (u'calfarome_labuilding', 262490)
   (u'Brian@Brea', 182591)
   (u'RichRico_labuildings', 159776)
   (u'piligab_labuildings', 122416)
   (u'Jothirnadh_labuildings', 102158)
   (u'yurasi_import', 94957)
   (u'manoharuss_imports', 55909)
   (u'dannykath_labuildings', 52333)
   (u'Aaron Lidman', 38228)
   (u'saikabhi_LA_imports', 26104)
```

Since the Nodes table has much more rows, top 10 users for both tables is the same as top 10 users in Nodes table. Top 9 users in the Ways table are also the top 9 in Nodes table, although the ranking is slightly different. The 10th spot in Ways table of 'karitotp' were only different name to appear from the Nodes table, replacing the 'saikabhi_LA_imports' at the 10th spot.

While I searched for the Fullerton, California in the OpenStreetMap, it looks like data point from other cities were also included. I counted city by number of times it appeared in Nodes table.

```
-----Cities in Nodes and Ways tags-----
select value, sum(n) as tot  from ( select value, count(*) n from
nodes_tags where key = "city" group by value UNION ALL select value,
count(*) from ways_tags where key = "city" group by value ) x group by
value order by tot desc;
your output:
   (u'Anaheim', 145)
   (u'Fullerton', 66)
   (u'La Palma', 58)
   (u'Brea', 44)
   (u'Buena Park', 37)
   (u'Cypress', 30)
   (u'Orange', 23)
   (u'Whittier', 18)
   (u'La Habra', 15)
   (u'Placentia', 11)
   (u'La Mirada', 7)
   (u'Diamond Bar', 5)
   (u'Stanton', 4)
   (u'Cerritos', 3)
   (u'Rowland Heights', 3)
   (u'Yorba Linda', 3)
   (u'La Habra Heights', 2)
   (u'Santa Fe Springs', 2)
   (u'Garden Grove', 1)
   (u'Villa Park', 1)
```

It is interesting there were more tags for Anaheim than Fullerton.

I also wanted to see what were the top 10 'amenity' in Nodes_tags table as well as count of 'amenity' types.

```
-----Amenities sorted by count in Nodes_tags-----
select value, count(value) as num from nodes_tags where key = "amenity"
group by value order by num desc limit 10;
your output:
   (u'fountain', 570)
   (u'bench', 230)
   (u'place_of_worship', 158)
   (u'drinking_water', 149)
   (u'restaurant', 148)
   (u'school', 128)
   (u'fast_food', 127)
   (u'toilets', 107)
   (u'parking', 67)
   (u'cafe', 57)

-----Number of unique Amenities-----
select count(distinct value) from nodes_tags where key = "amenity" ;
your output:
   (75,)
```

There are 75 unique values in Nodes_tags with key of 'amenity'. The top three are fountain, bench, and place_of_worship.

6. Other ideas about the dataset

As I was going through the data, the number of unique 'tag' nodes was quite daunting at over 200. If I was the user creating a new data, I would think it would take a quite some time to figure out which 'tag' nodes to use. It would be helpful to create a dictionary describing different 'tag' names.

Also, there were some inconsistencies and redundancies. For example, while most of address related data were stored in 'addr:<field>' node, there were separate nodes for county. It would be more consistent if 'addr:county' node was used instead. Additionally, county and state, appear in 'addr:<field>', 'county:<field>', 'gnis:<field>', and 'isin:<field>'. Eliminating duplicity would be very helpful.

If there were a standard for 'tag' nodes, it would benefit users inputting the data as well as users using the data. However, this could be prove to be a big project with a lot of coordinations. In addition, a lot of clean-up of current massive data may be necessary.