OpenStreetMap Data Case Study

Ken Musante, DAND, October 2017

I. Problems encountered:

   a. Needed Geographic Area which was at least 50 mb. Consequently, I selected the SF Bay Area even though I am not that familiar with the area.

   b. Did not fully understand what I needed to do to complete problem set: This involved multiple reads and reviews of the Forums as well as the 1 on 1 help

   c. 1 on 1 help was a new concept: It took me a while to understand how to get information to experts on the 1 on 1 desk

   d. Size of the file was massive. I tried several times to manipulate the original OSM file and was very frustrated and wasted a lot of time. I appreciate the 'iterparse' command more than you know.

   e. Understanding how/why to apply the Lessons to the data. Some of the lessons were not communicated in a way that allowed me to easily understand how to apply it to a new file. This took a lot of additional work and time.

   f. Unable to use Jupyter Notebook to manipulate the csv files. I tried and failed several times before understanding that an easier way was to just manipulate them directly in sqlite3.

   g. Did not grasp concept that I needed to convert and clean the file in the same program. For my first 3 submissions I did not have the cleansing functions inside my createcsvfile9.py

II. OverView of Data

   a. Files created in Jupyter Notebook are attached along with the coding for the steps described below. I was very pleased to get additional practice in Jupyter Notebook. I found it easy and helpful to use that tool to document work and pleased I could apply prior lessons to this project. While I wont repeat all of the code found in the Jupyter Notebook, here are the steps I undertook.

      i. First step was to create several different sample file sizes for easy manipulation. This was helpful as I could see file in Sublime

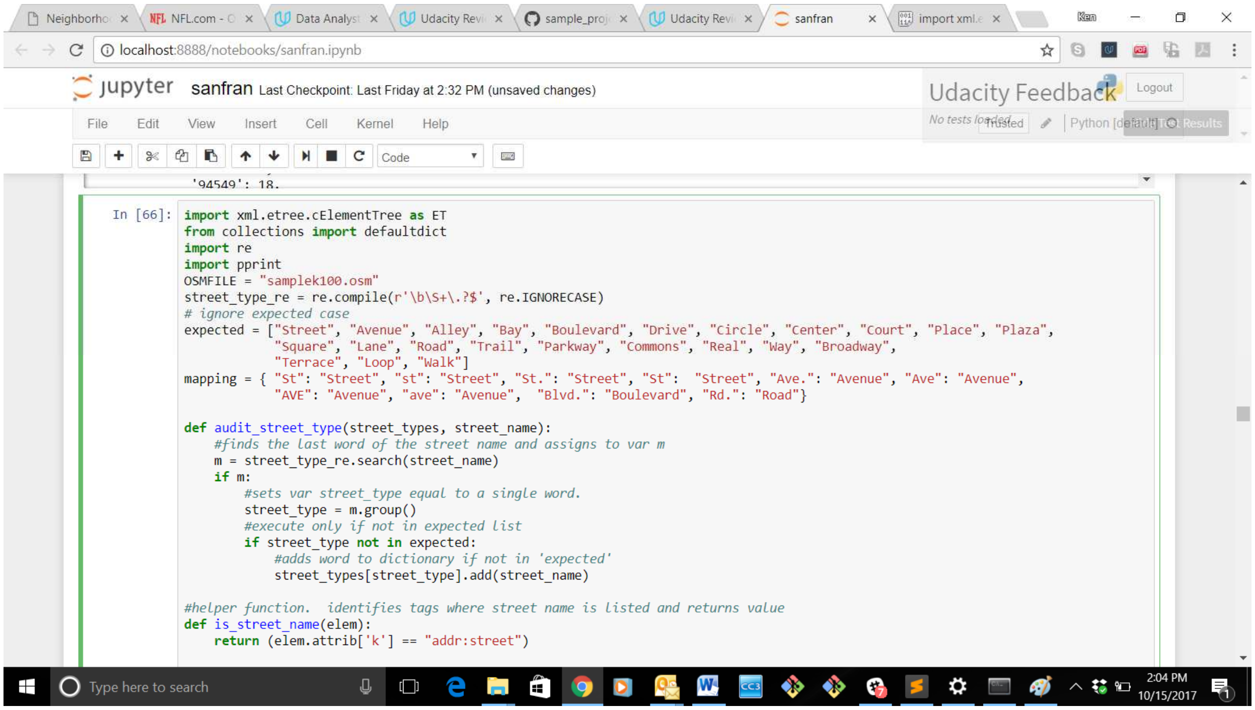
      ii. My next step was to count tags and put in Dictionary so I could understand file sizes

      iii. Next I queried specific fields so I could get an idea of the users(dictionary), location names, restaurant types(dictionary), web URLS and cities (dictionary)

      iv. I then queried and updated a dictionary for the street names such that the last word in the street names was consistent (as we did in the class lesson). Many of the street names had non standard endings so this program was very helpful/useful in correcting the inconsistent street names.

The ‘expected’ names were ignored and left alone while the ‘mapping’ was used to make the street names consistent.

The complete code is included in my Github under the name fixstname7.py. The ‘expected’ and ‘mapping variables are printed from a screenshot:


localhost:8888/notebooks/sanfran.ipynb
jupyter sanfran Last Checkpoint: Last Friday at 2:32 PM (unsaved changes)
Udacity Feedback
Logout
File
Edit
View
Insert
Cell
Kernel
Help
Code
Type here to search
2:04 PM
10/15/2017

```
'94549': 18,
```

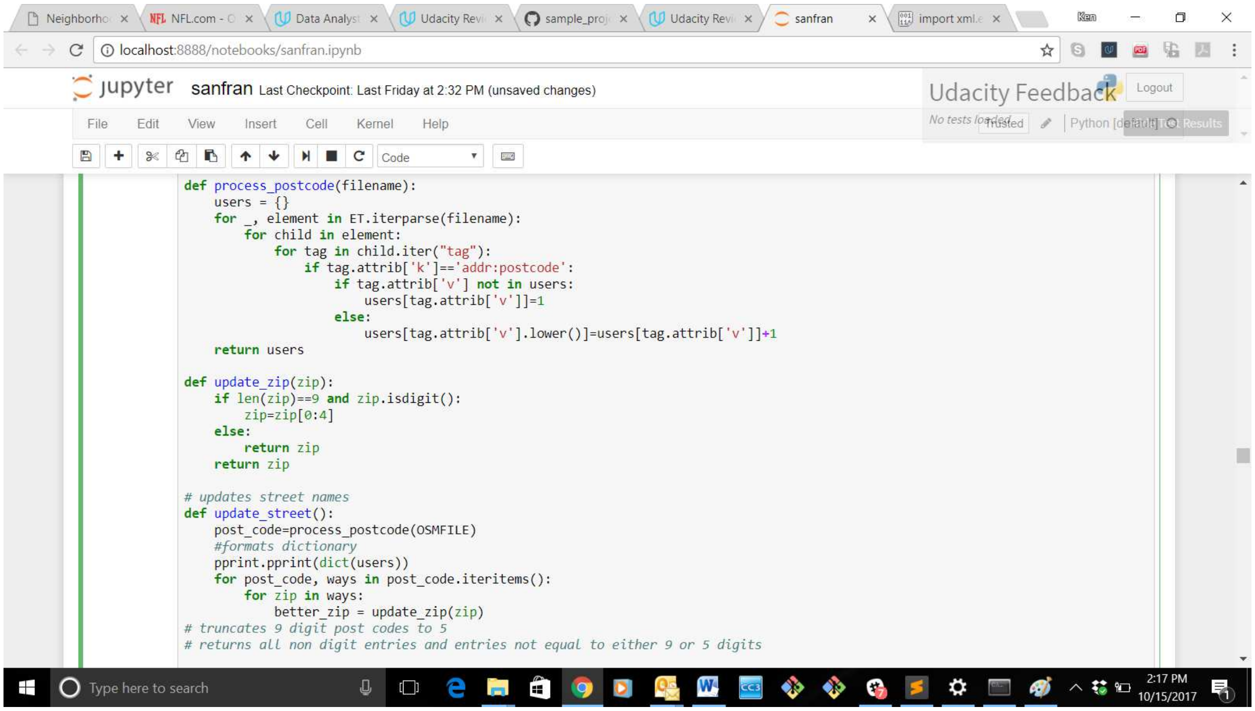
In [66]:

```
import xml.etree.cElementTree as ET
from collections import defaultdict
import re
import pprint
OSMFILE = "samplek100.osm"
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)
# ignore expected case
expected = ["Street", "Avenue", "Alley", "Bay", "Boulevard", "Drive", "Circle", "Center", "Court", "Place", "Plaza",
            "Square", "Lane", "Road", "Trail", "Parkway", "Commons", "Real", "Way", "Broadway",
            "Terrace", "Loop", "Walk"]
mapping = { "St": "Street", "st": "Street", "St.": "Street", "St":  "Street", "Ave.": "Avenue", "Ave": "Avenue",
            "AVE": "Avenue", "ave": "Avenue",  "Blvd.": "Boulevard", "Rd.": "Road"}

def audit_street_type(street_types, street_name):
    #finds the last word of the street name and assigns to var m
    m = street_type_re.search(street_name)
    if m:
        #sets var street_type equal to a single word.
        street_type = m.group()
        #execute only if not in expected list
        if street_type not in expected:
            #adds word to dictionary if not in 'expected'
            street_types[street_type].add(street_name)

#helper function.  identifies tags where street name is listed and returns value
def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")
```

v. I next looked at postal codes. Some were 9 digits and some had hyphens and some had alpha characters like this:

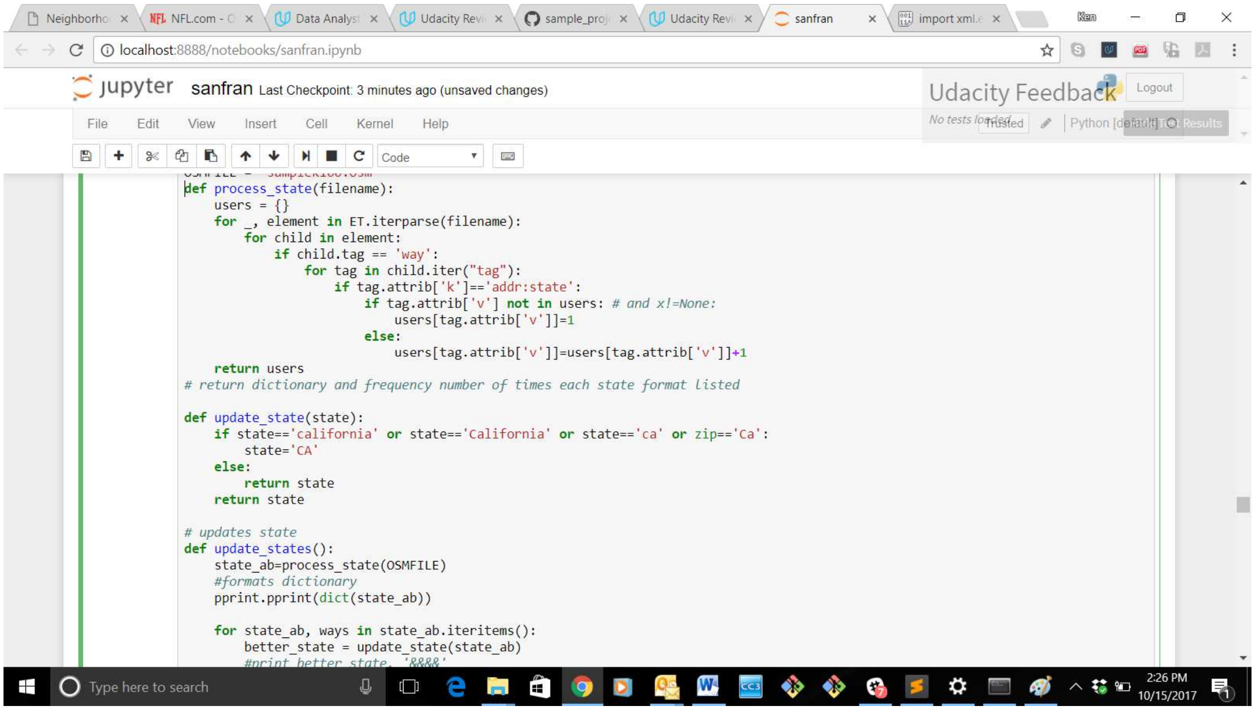
1. '944023025'
2. '94118-1316'
3. ‘CA’
4. ‘ca’

I wrote the following code to address this (the complete code is in my Git hub under the name fixpostalcode8.py). This function truncates postal code s to 5 digits if they are 9 and outsorts entries which are not numeric.



```python
def process_postcode(filename):
    users = {}
    for _, element in ET.iterparse(filename):
        for child in element:
            for tag in child.iter("tag"):
                if tag.attrib['k']=='addr:postcode':
                    if tag.attrib['v'] not in users:
                        users[tag.attrib['v']]=1
                    else:
                        users[tag.attrib['v'].lower()]=users[tag.attrib['v']]+1
    return users

def update_zip(zip):
    if len(zip)==9 and zip.isdigit():
        zip=zip[0:4]
    else:
        return zip
    return zip

# updates street names
def update_street():
    post_code=process_postcode(OSMFILE)
    #formats dictionary
    pprint.pprint(dict(users))
    for post_code, ways in post_code.iteritems():
        for zip in ways:
            better_zip = update_zip(zip)
# truncates 9 digit post codes to 5
# returns all non digit entries and entries not equal to either 9 or 5 digits
```

vi. I also looked at state names. Although all entries were close, they were not consistent. With entries of ‘CA’, ‘ca’, ‘Ca’, ‘california’ and ‘California’. The following code makes the above entries consistent at ‘CA’ (the complete code is in my Github as fixstate9.py). The heart of the code is below:



```python
def process_state(filename):
    users = {}
    for _, element in ET.iterparse(filename):
        for child in element:
            if child.tag == 'way':
                for tag in child.iter("tag"):
                    if tag.attrib['k']=='addr:state':
                        if tag.attrib['v'] not in users: # and x!=None:
                            users[tag.attrib['v']]=1
                        else:
                            users[tag.attrib['v']]=users[tag.attrib['v']]+1
    return users
# return dictionary and frequency number of times each state format listed

def update_state(state):
    if state=='california' or state=='California' or state=='ca' or zip=='Ca':
        state='CA'
    else:
        return state
    return state

# updates state
def update_states():
    state_ab=process_state(OSMFILE)
    #formats dictionary
    pprint.pprint(dict(state_ab))

    for state_ab, ways in state_ab.iteritems():
        better_state = update_state(state_ab)
        #print better_state, '&&&&'
```

vii. I then moved the above three cleansing routines into my createcsvfile9.py files so that I could cleanse the data and parse into 5 csv files:

1. NODES_PATH = "nodes.csv"
2. NODE_TAGS_PATH = "nodes_tags.csv"
3. WAYS_PATH = "ways.csv"
4. WAY_NODES_PATH = "ways_nodes.csv"
5. WAY_TAGS_PATH = "ways_tags.csv"

viii. I used the Jupyter Notebook to view the csv files

b. I attempted to use Jupyter Notebook to set up the Tables. I failed and found it much easier to set them up in sqlite3. Running queries directly in sqlite3 was easy. Here are my queries and response:

i. Size of the file: I'll provide a couple of responses to ensure I understand your query-

1. San Francisco.osm-1.38 GB
2. San Francisco Data Base-10.1 MB
3. Samplek100.osm-13.6 MB
4. Nodes.csv-5.5 MB
5. Nodes_tags.csv-98 KB
6. Ways.csv-97 KB
7. Ways_nodes.csv-361 KB
8. Ways_tags.csv-252 KB

This could also mean the number of rows. I found 2 ways to do this. Both solutions resulted in 66,368 rows:

SOLUTION #1:
"SELECT count(*) as count FROM nodes ORDER BY count desc limit 1;"
Answer: 66,368

SOLUTION #2:
"SELECT count(*) as count from nodes;"
Answer: 66,368

ii. Number of unique 'users':
"SELECT count(uid) as count from nodes GROUP BY uid ORDER BY count desc LIMIT 1;"

Answer: 13,156

iii. Number of nodes and ways:

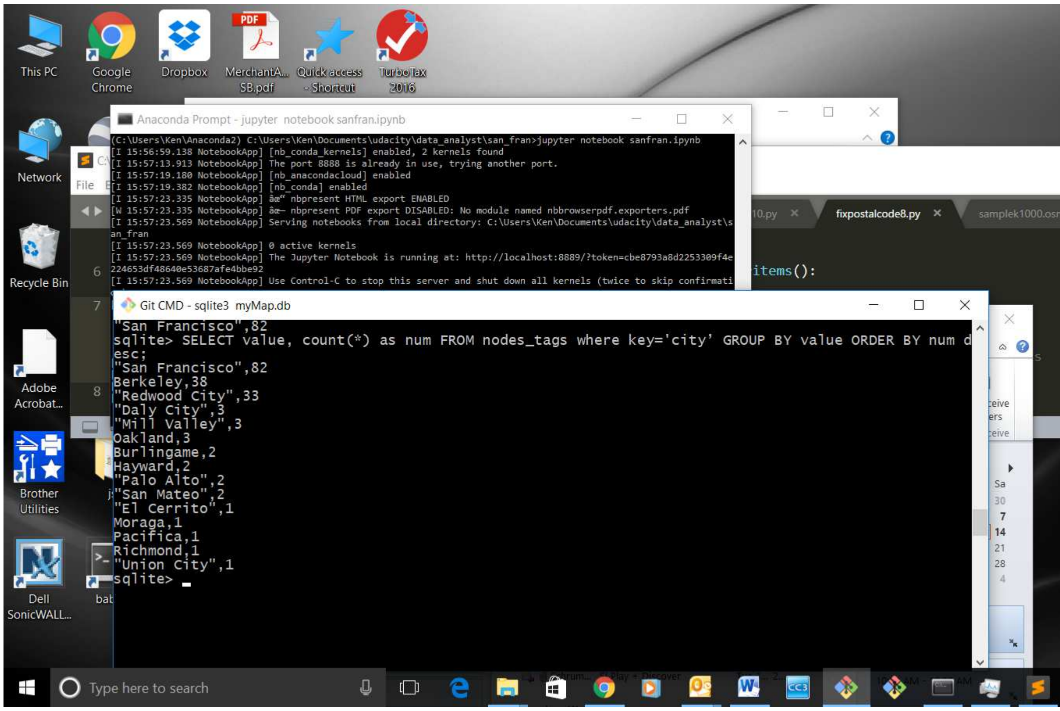
1. “SELECT count(id) as count FROM nodes ORDER BY count desc limit 1;”
   Answer: 66,368 node ids

2. “SELECT count(id) as count FROM ways ORDER BY count desc limit 1;”
   Answer: 1,406 way ids

iv. Cities and frequency of appearance:

“SELECT value, count(*) as num FROM nodes_tags WHERE key=’city’ GROUP BY value ORDER BY num desc;”

Answer:



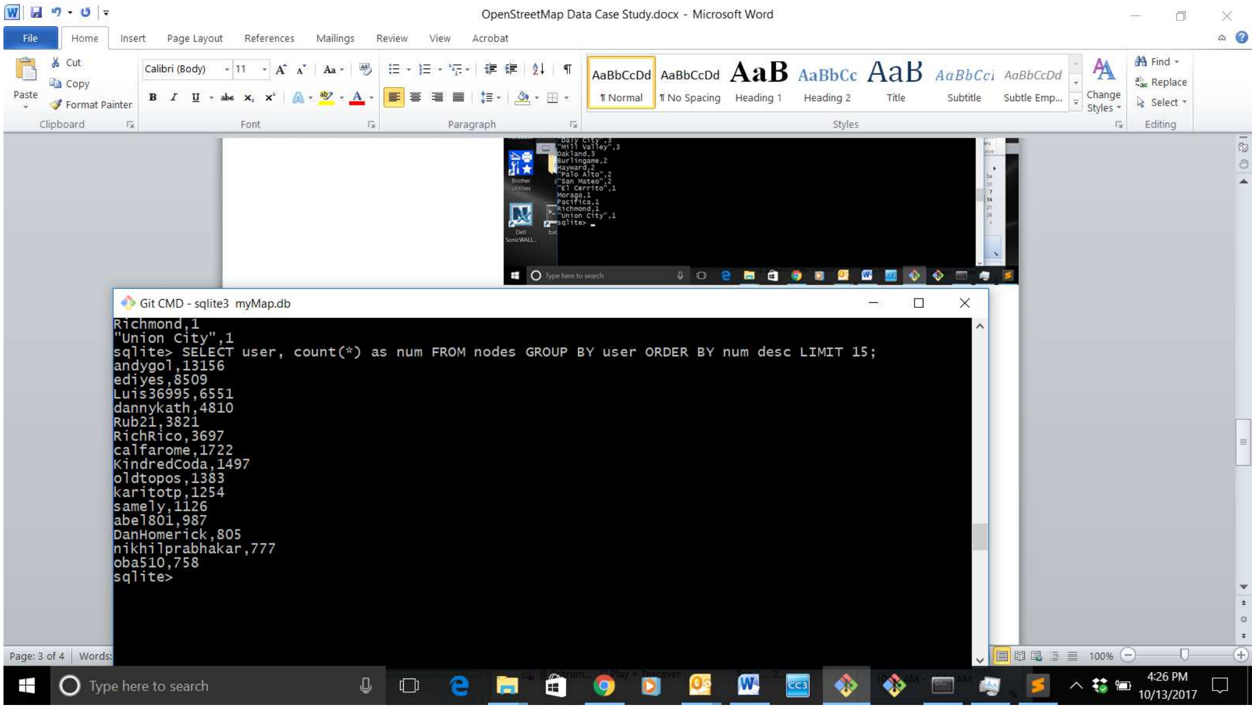
```
(C:\Users\Ken\Anaconda2) C:\Users\Ken\Documents\udacity\data_analyst\san_fran>jupyter notebook sanfran.ipynb
[I 15:56:59.138 NotebookApp] [nb_conda_kernels] enabled, 2 kernels found
[I 15:57:13.913 NotebookApp] The port 8888 is already in use, trying another port.
[I 15:57:19.180 NotebookApp] [nb_anacondacloud] enabled
[I 15:57:19.382 NotebookApp] [nb_conda] enabled
[I 15:57:23.335 NotebookApp] âœ“ nbpresent HTML export ENABLED
[W 15:57:23.335 NotebookApp] âœ— nbpresent PDF export DISABLED: No module named nbbrowserpdf.exporters.pdf
[I 15:57:23.569 NotebookApp] Serving notebooks from local directory: C:\Users\Ken\Documents\udacity\data_analyst\s
an_fran
[I 15:57:23.569 NotebookApp] 0 active kernels
[I 15:57:23.569 NotebookApp] The Jupyter Notebook is running at: http://localhost:8889/?token=cbe8793a8d2253309f4e
224653df48640e53687afe4bbe92
[I 15:57:23.569 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmati
```

```
"San Francisco",82
sqlite> SELECT value, count(*) as num FROM nodes_tags where key='city' GROUP BY value ORDER BY num d
esc;
"San Francisco",82
Berkeley,38
"Redwood City",33
"Daly City",3
"Mill Valley",3
Oakland,3
Burlingame,2
Hayward,2
"Palo Alto",2
"San Mateo",2
"El Cerrito",1
Moraga,1
Pacifica,1
Richmond,1
"Union City",1
sqlite>
```

v. Most prolific ‘user’s

“SELECT user, count(*) as num FROM nodes GROUP BY user ORDER BY num desc LIMIT 15;”

Answer:



```
"Daly City",3
"Mill Valley",3
Oakland,3
Burlingame,2
Hayward,2
"Palo Alto",2
"San Mateo",2
"El Cerrito",1
Moraga,1
Pacifica,1
Richmond,1
"Union City",1
sqlite>
```

```
Richmond,1
"Union City",1
sqlite> SELECT user, count(*) as num FROM nodes GROUP BY user ORDER BY num desc LIMIT 15;
andygol,13156
ediyes,8509
Luis36995,6551
dannykath,4810
Rub21,3821
RichRico,3697
calfarome,1722
KindredCoda,1497
oldtopos,1383
karitotp,1254
samely,1126
abel801,987
DanHomerick,805
nikhilprabhakar,777
oba510,758
sqlite>
```

vi. We had only 1 Taco Bell and 0 Mcdonalds and 1 Starbucks. The solution below indicates there were 2 when I asked for Taco Bell, McDonalds or Starbucks and our sample had 31 restaurants.

“SELECT key, count(*) as num FROM nodes_tags WHERE value=’McDonalds’ or value=’Taco Bell’;”

Answer: name|1

vii. “SELECT key, count(*) as num FROM nodes_tags WHERE value=’McDonalds’ or value=’Taco Bell’ or value=’Starbucks’;”

Answer: name|2

viii. “SELECT key, count (*) as num from nodes_tags WHERE value=’restaurant’;”

Answer: amenity|31

ix. And the Starbucks entry was made by Seandebasti

“SELECT nodes.id, nodes.user FROM nodes join nodes_tags on nodes_tags.id=nodes.id WHERE value=’Starbucks’;”

Answer: 4075218516|Seandebasti

III. Additional ideas

One of the issues with the OpenStreetMapping is that the data is all input by individuals. It would be ideal if the data could also be populated by other data sources such as Yelp to get the ratings on certain restaurants or a traffic application which could provide periods where the area might be congested. Other ideas could be to include statistical information on such things as earthquakes or weather.

Doing so would provide additional information for entities seeking to obtain a location for a movie shoot, new headquarters or anticipating when it is best to schedule shipments and how they might be delayed. If you think about a specific business to business company, it would be ideal to know such information about an area before you did a lot of in person visits to determine whether to open a new office there.

This would create an additional query for OpenStreetMaps. Instead of relying on users inputting information, OpenStreetMaps would then need to be querying these databases on some periodic basis. Some of this information could be at odds with user input and it would create additional complexities for OpenStreetMaps when these other databases changed their formats or way in which data was made available or even in their pricing plans.

Obviously one of my issues was the enormity of the data. Adding additional data sets not only complicates the data but also makes the file sizes all the larger. This could make it even harder to wrangle in order to obtain useful data.

IV. Conclusion

a. The biggest issue in working with the OpenStreets database is that it was so large it was difficult to work with unless I took a smaller sample. While this was helpful, I wish I had a more powerful PC to be able to run queries on the entire dataset. Certainly working with the data once it was in the CSV files was quick and easy to manipulate. It also might have been useful to divide up the csv files by City so even though I could not look at an entire data set I could look at an entire city.