OpenStreetMap Data Case Study

Map Area

New Orleans, LA, United States

https://www.openstreetmap.org/relation/131885#map=8/30.233/-89.434
(https://www.openstreetmap.org/relation/131885#map=8/30.233/-89.434)

New Orleans is the city where I studied for my master degree, and where I met my best friend. So I'm interested to see what database querying reveals, and get to know this lovely city better in a different way.

Downloading a Sample Part of the Area

In [3]:

```
import xml.etree.ElementTree as ET
OSM_FILE = "new-orleans_louisiana.osm"
SAMPLE_FILE = "sample.osm"

k = 20

def get_element(osm_file, tags=('node', 'way', 'relation')):


    context = iter(ET.iterparse(osm_file, events=('start', 'end')))
    _, root = next(context)
    for event, elem in context:
        if event == 'end' and elem.tag in tags:
            yield elem
            root.clear()


with open(SAMPLE_FILE, 'wb') as output:
    output.write('<?xml version="1.0" encoding="UTF-8"?>\n')
    output.write('<osm>\n  ')

    # Write every kth top level element
    for i, element in enumerate(get_element(OSM_FILE)):
        if i % k == 0:
            output.write(ET.tostring(element, encoding='utf-8'))

    output.write('</osm>')
```

Exploring the Dataset

Iterative Parsing

Through the count_tags function, we could get a dictionary called "tag_freq". It shows tag names as the key and number of times they showed in the dataset as the value.

In [4]:

```
#count_tags function returns a dictionary with the
#tag name as the key and number of times this tag can be encountered in
#the map as value.

import pprint

def count_tags(filename):
    tag_freq={}

    for event, elem in ET.iterparse(filename):
        if elem.tag not in tag_freq:
            tag_freq[elem.tag]=1
        else:
            tag_freq[elem.tag]+=1

    return tag_freq
```

In [5]:

```
count_tags(SAMPLE_FILE)
```

Out[5]:

```
{'member': 2940,
 'nd': 354697,
 'node': 320614,
 'osm': 1,
 'relation': 298,
 'tag': 83259,
 'way': 18914}
```

From above result, we can know the number of tags in the SAMPLE_FILE. For example, the number of "node" is 320614, and the number of "way" is 18914.

Explore the dataset a bit more

I used Sublime Text to read our SMAPLE_FILE.

After I looked the dataset thoroughly, I found that **the street name values are stored at "tag" element's "v" attribute when attribute "k" equals "addr:street". And possible street types are: Place, Street, Lane, Heights, Drive, Avenue, Boulevard, Court, Road.** These would be very helpful when we try to edit and clean the street name values later.

Problems Encountered in the Map

I noticed 4 main problems with the data:

Problem 1

Street names in the second level v needed to be improved, such as "Severn Ave" and "Youngswood Lp". It's better to improve these street names. For example, improve "Severn Ave" to "Severn Avenue", and improve "Youngswood Lp" to "Youngswood Loop".

Problem 2

There're some street names ended with direction, not street types, such as 'Diamondhead Drive West'and 'Bayou View East'.

Problem 3

Abscure street names in the dataset, such as "Longo","Wainwright".

Problem 4

Street name values are stored in attribute "v" while the second level "k" equals "addr:street" or "name".

I also checked postcode strings in the dataset under nodes tag and ways tag, and find out that they are pretty clean and organized.

Checking the "k" value

First, I checked the "k" value for each "tag" and see if there are any potential problems:

The 'key_type' function below counts each of four tag categories in a dictionary: "lower", for tags that contain only lowercase letters and are valid, "lower_colon", for otherwise valid tags with a colon in their names, "problemchars", for tags with problematic characters, and "other", for other tags that do not fall into the other three categories.

In [6]:

```
import re

lower = re.compile(r'^([a-z]|_)*$')
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')


def key_type(element, keys):
    if element.tag == "tag":

        k_value=element.get("k")

        l=re.search(lower,k_value)
        lc=re.search(lower_colon,k_value)
        p=re.search(problemchars,k_value)

        if l:
            keys['lower']+=1

        elif lc:
            keys['lower_colon']+=1

        elif p:
            keys['problemchars']+=1

        else:
            keys['other']+=1

        pass

    return keys



def process_map(filename):
    keys = {"lower": 0, "lower_colon": 0, "problemchars": 0, "other": 0}
    for _, element in ET.iterparse(filename):
        keys = key_type(element, keys)

    return keys
```

In [7]:

```
process_map(SAMPLE_FILE)
```

Out[7]:

```
{'lower': 32837, 'lower_colon': 34578, 'other': 15844, 'problemchars
': 0}
```

From above, we can see that there's 32837 k values that contain only lowercase letters and are valid, 34578 k values are valid tags with a colon in their names, 15844 other tags that do not fall into the other three categories and 0 problematic k values in the dataset.

Checking street names

In [22]:

```
from collections import defaultdict

street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)

#I firstly input normal street types like "Street","Avenue" and etc. into the ex
pected list. Then
#run the audit(SAMPLE_FILE) function to see the results. After that, I adjusted
the expected list
#manually by adding other street types into the list. Also, I updated the mappin
g dictionary after
#looking through following results.

expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Lane",
"Road",
            "Heights","Parkway","Terrace","Alley","Corner","Cove","Circle","High
way","Park",
            "Trace","Trail","View","Village","Randch","Way","Walk","Loop","Bayou
","Hollow","Hill","Ridge",
           "North","West","East"]


mapping = { "St": "Street",
            "St.": "Street",
            "Ave": "Avenue",
            "Rd.": "Road",
           "Pky":"Parkway",
           "Villa":"Village",
           "Lp":"Loop"
            }


def audit_street_type(street_types, street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
```

```
        if street_type not in expected:
            street_types[street_type].add(street_name)


def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")


def audit(osmfile):
    osm_file = open(osmfile, "r")
    street_types = defaultdict(set)
    for event, elem in ET.iterparse(osm_file, events=("start",)):

        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_street_name(tag):
                    audit_street_type(street_types, tag.attrib['v'])
    osm_file.close()

    return street_types


def update_name(name, mapping):

    m = street_type_re.search(name)
    #print m.group()
    if m.group() not in expected:
        if m.group() in mapping.keys():
            name = re.sub(m.group(), mapping[m.group()], name)

    return name

#audit(SAMPLE_FILE)


def test():
    st_types=audit(SAMPLE_FILE)
    for st_type, ways in st_types.iteritems():
        for name in ways:
            better_name = update_name(name, mapping)
            print name, "=>", better_name


if __name__ == '__main__':
    test()
```

```
Road 217 => Road 217
Highway 603 => Highway 603
Highway 604 => Highway 604
N Broad => N Broad
Broad => Broad
Willow St. => Willow Street
Mossy Oak => Mossy Oak
```

Live Oak => Live Oak
Mahalo Hui => Mahalo Hui
No Access => No Access
Jeff Davis => Jeff Davis
Road 390 => Road 390
Unknown => Unknown
Road 136 => Road 136
Youngswood Lp => Youngswood Loop
Mckinley => Mckinley
Mossy Oaks => Mossy Oaks
Bay Oaks => Bay Oaks
Grand Route St John => Grand Route St John
Rue Nichole => Rue Nichole
Demontluzin => Demontluzin
Old Highway 49 => Old Highway 49
Road 297 => Road 297
LA-23 => LA-23
Ala Moana => Ala Moana
Highway 43 => Highway 43
Bouslog => Bouslog
Ocean Wave => Ocean Wave
Road 346 => Road 346
Rue Delphine => Rue Delphine
Road 546 => Road 546
Esplanade => Esplanade
Dogwood => Dogwood
Rue Nadine => Rue Nadine
Gladstone => Gladstone
Cypress Knee => Cypress Knee
Hwy603 => Hwy603
Road 528 => Road 528
Road 361 => Road 361
Avenue A => Avenue A
Wainwright => Wainwright
Avenue B => Avenue B
Jordan Bluff => Jordan Bluff
Road 141 => Road 141
Romaneda => Romaneda
Ranch => Ranch
Road 306 => Road 306
Magazine St => Magazine Street
Canal St => Canal Street
Pine Ext => Pine Ext
Quail Creek => Quail Creek
Seabrook => Seabrook
Road 370 => Road 370
Devil'S Elbow => Devil'S Elbow
Highway 90 => Highway 90
Pinecrest => Pinecrest
Spanish Acres => Spanish Acres
Longfellow => Longfellow
Rue De Lasalle => Rue De Lasalle
Olivari => Olivari

```
Webb => Webb
Rue Le Ville => Rue Le Ville
General Taylor => General Taylor
Lanai Villa => Lanai Village
Rue Mignon => Rue Mignon
Terry Pky => Terry Parkway
S Massachusetts S => S Massachusetts S
Highway 53 => Highway 53
Rue Colette => Rue Colette
Rue Denise => Rue Denise
Cazabon Farm => Cazabon Farm
Road 273 => Road 273
Longo => Longo
Road 298 => Road 298
Road 530 => Road 530
N Claiborne Ave => N Claiborne Avenue
N Carrollton Ave => N Carrollton Avenue
Severn Ave => Severn Avenue
St. Peter => St. Peter
Road 418 => Road 418
Road 377 => Road 377
Gulfside => Gulfside
```

I adjusted the mapping dictionary to improve our street names. For example, I found "'Pky':{'Terry Pky'}", which 'Pky' was supposed to be the abbreviation of 'Parkway'. So I added "Pky" as the key, "Parkway" as the value into mapping dictionary.

In addition, I don't want to change the street names which have numbers at last becasue they're supposed to be highway. And this expression is actually right for highway names. For example, street names like "Highway 604","Road 310" and etc.

There's another type of street names I think we don't need to change, because its expression is pretty normal in our life. The type is ended with direction, such as 'North' and 'West'. Examples like 'Diamondhead Drive North', 'Seal Avenue North' and etc.

So I audited the street names which fell outside of our "expected" list. As above showed, for example, we've improved street name like "Decatur St => Decatur Street", "Youngswood Lp => Youngswood Loop","Severn Ave => Severn Avenue".

Prepare Dataset for SQL

The next step is to prepare the data to be inserted into a SQL database. I'm goint to parse the elements in the SAMPLE_FILE, to transform them into tabular format, and to finnaly write them to .csv files so we can use SQL database to analyze them.

In [16]:

```
import csv
import codecs

import cerberus

import schema
```

I named different .csv files for tables in SQL database later. And I imported schema and cerberus to prepare and validate for the following data transformation.

In [17]:

```
OSM_PATH = "sample.osm"

NODES_PATH = "nodes.csv"
NODE_TAGS_PATH = "nodes_tags.csv"
WAYS_PATH = "ways.csv"
WAY_NODES_PATH = "ways_nodes.csv"
WAY_TAGS_PATH = "ways_tags.csv"


SCHEMA = schema.schema
```

In [29]:

```
#-Shape each element into several data structures using a custom function

LOWER_COLON = re.compile(r'^([a-z]|_)+:([a-z]|_)+')
PROBLEMCHARS = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')

# Make sure the fields order in the csvs matches the column order in the sql tab
le schema
NODE_FIELDS = ['id', 'lat', 'lon', 'user', 'uid', 'version', 'changeset', 'times
tamp']
NODE_TAGS_FIELDS = ['id', 'key', 'value', 'type']
WAY_FIELDS = ['id', 'user', 'uid', 'version', 'changeset', 'timestamp']
WAY_TAGS_FIELDS = ['id', 'key', 'value', 'type']
WAY_NODES_FIELDS = ['id', 'node_id', 'position']

#The shape_element function transforms each element into the correct format.
#It takes as input an iterparse Element object and return a dictionary organizin
g
#all the information into the correct format.

#For example, for elements whose top level tag is "node", the "shape_element" fu
nction returns
#a dictionary whose format is {"node": .., "node_tags": ...}.
#To be specific, the "node" key holds attributes like "id","user","uid","version
","lat","lon","timestamp" and
#"changeset" for its top level "node". And "node_tags" key holds a list of dicti
```

```
onaries which includes

#attributes like "id","key","value" and "type" for its secondary tag.


def shape_element(element, node_attr_fields=NODE_FIELDS, way_attr_fields=WAY_FIE
LDS,
                  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 elemen
ts


    if element.tag == 'node':
        for attrib in element.attrib:
            if attrib in NODE_FIELDS:
                node_attribs[attrib]=element.attrib[attrib]

        for child in element:
            node_tag={}

            if LOWER_COLON.search(child.attrib['k']):
                if child.attrib['k'] == 'addr:street':
                    node_tag['value'] = update_name(child.attrib['v'], mapping)
                    node_tag['id'] = element.attrib['id']
                    node_tag['key'] = child.attrib['k'].split(':',1)[1]
                    node_tag['type'] = child.attrib['k'].split(':',1)[0]
                    tags.append(node_tag)
                else:
                    node_tag['id'] = element.attrib['id']
                    node_tag['key'] = child.attrib['k'].split(':',1)[1]
                    node_tag['type'] = child.attrib['k'].split(':',1)[0]
                    node_tag['value'] = child.attrib['v']
                    tags.append(node_tag)

            elif PROBLEMCHARS.search(child.attrib['k']):
                continue

            else:
                node_tag['id'] = element.attrib['id']
                node_tag['key'] = child.attrib['k']
                node_tag['type'] = 'regular'
                node_tag['value'] = child.attrib['v']
                tags.append(node_tag)

        return {'node': node_attribs, 'node_tags': tags}

    elif element.tag == 'way':
        for attrib in element.attrib:
            if attrib in WAY_FIELDS:
```

```
                way_attribs[attrib]=element.attrib[attrib]


        position=0
        for child in element:
            way_tag={}
            way_node={}

            if child.tag=='tag':
                if LOWER_COLON.search(child.attrib['k']):
                    if child.attrib['k'] == 'addr:street':
                        way_tag['value'] = update_name(child.attrib['v'], mappin
g)
                        way_tag['id'] = element.attrib['id']
                        way_tag['key'] = child.attrib['k'].split(':',1)[1]
                        way_tag['type'] = child.attrib['k'].split(':',1)[0]
                        tags.append(way_tag)
                    else:
                        way_tag['id'] = element.attrib['id']
                        way_tag['key'] = child.attrib['k'].split(':',1)[1]
                        way_tag['type'] = child.attrib['k'].split(':',1)[0]
                        way_tag['value'] = child.attrib['v']
                        tags.append(way_tag)

                elif PROBLEMCHARS.search(child.attrib['k']):
                    continue
                else:
                    way_tag['id'] = element.attrib['id']
                    way_tag['key'] = child.attrib['k']
                    way_tag['type'] = 'regular'
                    way_tag['value'] = child.attrib['v']
                    tags.append(way_tag)

            #print tags
            elif child.tag=='nd':
                way_node['id']=element.attrib['id']
                way_node['node_id']=child.attrib['ref']
                way_node['position']=position
                position+=1
                way_nodes.append(way_node)

        return {'way': way_attribs, 'way_nodes': way_nodes, 'way_tags': tags}
```

In [30]:

```
# ================================================== #
#               Helper Functions                     #
# ================================================== #

#- Use iterparse to iteratively step through each top level element in the XML
def get_element(osm_file, tags=('node', 'way', 'relation')):
    """Yield element if it is the right type of tag"""
```

```
    context = ET.iterparse(osm_file, events=('start', 'end'))

    _, root = next(context)
    for event, elem in context:
        if event == 'end' and elem.tag in tags:
            yield elem
            root.clear()

#- Utilize a schema and validation library to ensure the transformed data is in
the correct format
#Using the cerberus library can validate the output against this schema to ensur
e it is correct.

def validate_element(element, validator, schema=SCHEMA):
    """Raise ValidationError if element does not match schema"""
    if validator.validate(element, schema) is not True:
        field, errors = next(validator.errors.iteritems())
        message_string = "\nElement of type '{0}' has the following errors:\n{1}
"
        error_string = pprint.pformat(errors)

        raise Exception(message_string.format(field, error_string))

#- Write each data structure to the appropriate .csv files
class UnicodeDictWriter(csv.DictWriter, object):
    """Extend csv.DictWriter to handle Unicode input"""

    def writerow(self, row):
        super(UnicodeDictWriter, self).writerow({
            k: (v.encode('utf-8') if isinstance(v, unicode) else v) for k, v in
row.iteritems()
        })

    def writerows(self, rows):
        for row in rows:
            self.writerow(row)


# ================================================== #
#               Main Function                        #
# ================================================== #
def process_map(file_in, validate):
    """Iteratively process each XML element and write to csv(s)"""

    with codecs.open(NODES_PATH, 'w') as nodes_file, \
         codecs.open(NODE_TAGS_PATH, 'w') as nodes_tags_file, \
         codecs.open(WAYS_PATH, 'w') as ways_file, \
         codecs.open(WAY_NODES_PATH, 'w') as way_nodes_file, \
         codecs.open(WAY_TAGS_PATH, 'w') as way_tags_file:

        nodes_writer = UnicodeDictWriter(nodes_file, NODE_FIELDS)
        node_tags_writer = UnicodeDictWriter(nodes_tags_file, NODE_TAGS_FIELDS)
        ways_writer = UnicodeDictWriter(ways_file, WAY_FIELDS)
        way_nodes_writer = UnicodeDictWriter(way_nodes_file, WAY_NODES_FIELDS)
```

```
        way_tags_writer = UnicodeDictWriter(way_tags_file, WAY_TAGS_FIELDS)


        nodes_writer.writeheader()
        node_tags_writer.writeheader()
        ways_writer.writeheader()
        way_nodes_writer.writeheader()
        way_tags_writer.writeheader()

        validator = cerberus.Validator()

        for element in get_element(file_in, tags=('node', 'way')):
            el = shape_element(element)
            if el:
                if validate is True:
                    validate_element(el, validator)

                if element.tag == 'node':
                    nodes_writer.writerow(el['node'])
                    node_tags_writer.writerows(el['node_tags'])
                elif element.tag == 'way':
                    ways_writer.writerow(el['way'])
                    way_nodes_writer.writerows(el['way_nodes'])
                    way_tags_writer.writerows(el['way_tags'])


if __name__ == '__main__':
    process_map(OSM_PATH, validate=False)
```

Import csv files into SQL database

After audit and clean the dataset, I converted it from XML to CSV format, then imported the cleaned .csv files into a SQL database named "osm.db".

Statistical Overview of the Dataset

File sizes

In [35]:

```
import os

folder = '/Users/tangyiyi/Desktop/Data Analyst/Data Wrangling/Project'
folder_size = 0


for (path, dirs, files) in os.walk(folder):
    for file in files:
        if '.ipynb' not in file and '.py' not in file and '.DS_Store' not in fil
e and '.jpg' not in file and '.png' not in file:
            filename = os.path.join(path, file)
            folder_size = os.path.getsize(filename)
            print file, " = %0.1f MB" % (folder_size/(1024*1024.0))
```

```
new-orleans_louisiana.osm  = 1219.5 MB
nodes.csv  = 25.2 MB
nodes_tags.csv  = 0.3 MB
osm.db  = 31.9 MB
sample.osm  = 61.6 MB
ways.csv  = 1.1 MB
ways_nodes.csv  = 8.0 MB
ways_tags.csv  = 2.5 MB
```

Number of nodes and ways

In [38]:

```
import sqlite3

con = sqlite3.connect('osm.db')
cursor = con.cursor()
cursor.execute("SELECT count(*) FROM nodes;")

print(cursor.fetchall())
```

```
[(320614,)]
```

In [39]:

```
cursor.execute("SELECT count(*) FROM ways;")

print(cursor.fetchall())
```

```
[(18914,)]
```

As above, the number of nodes is 320614, and the number of ways is 18914.

Number of unique users

In [40]:

```
cursor.execute("SELECT count(distinct(u.uid)) FROM (Select uid FROM nodes UNION
ALL SELECT uid FROM ways) u;")

print(cursor.fetchall())
```

```
[(482,)]
```

I used "union all" function to combine nodes and ways tables accroding to their uid, and named the combiend table "u". Then I counted distinct uid numbers. So as above showed, there's 482 unique users in the dataset.

Number of cafes in the dataset & Who contributed to the cafe data

In [41]:

```
cursor.execute("SELECT count(*) FROM nodes_tags where value='cafe';")

print(cursor.fetchall())
```

```
[(7,)]
```

In [47]:

```
cursor.execute("SELECT nodes.user FROM nodes INNER JOIN nodes_tags ON nodes.id=n
odes_tags.id where nodes_tags.value='cafe';")

pprint.pprint(cursor.fetchall())
```

```
[(u'Matt Toups',),
 (u'wheelmap_visitor',),
 (u'wegavision',),
 (u'lokejul',),
 (u'lokejul',),
 (u'bhelx',),
 (u'anna2233',)]
```

There's 7 cafe in the dataset. And user named "Matt Toups", "wheelmap_visitor", "wegavision", "lokejul", "bhelx", "anna2233" contributed to the cafe data.

How many times did user "Matt Toups" contribute to this dataset

In [43]:

```
cursor.execute("SELECT count(*) FROM (SELECT user from nodes UNION ALL SELECT us
er from ways) u where user='Matt Toups';")

print(cursor.fetchall())
```

```
[(172024,)]
```

As results, user "Matt Toups"contributed 172024 times.

Other Ideas about the Datasets

Gamification

I found it interesting to review contributor statistical information in the dataset.

I found the contributions is pretty skewed:

1. Top contributor's contribution percentage: 50.67%
2. Top 3 contributors' contribution percentage: 80.06%
3. Top 10 contributors' contribution percentage: 94.26%

And I also found there's 119 contributors who just appeared one time. So all of these reminds me of "gamification" so that users could be motivated to contributes more and more for the OpenStreetMap. Gamification such as rewards, point scoring, and competitions showing on the leaderboard may all have positive influence on encouraging user participation in OpenStreetMap contributions.

Sum of contributions (posted by contributors)

In [72]:

```
cursor.execute("SELECT count(*) FROM (SELECT user from nodes UNION ALL SELECT us
er from ways) u;")

print(cursor.fetchall())
```

```
[(339528,)]
```

Sum of top 10 contributors' contributions

In [74]:

```
cursor.execute("SELECT sum(num) FROM (SELECT u.user, count(*) as num FROM (SELEC
T user FROM nodes UNION ALL SELECT user FROM ways) u group by u.user order by nu
m desc limit 10) e;")

pprint.pprint(cursor.fetchall())
```

```
[(320030,)]
```

Top 20 contributors

In [50]:

```
cursor.execute("SELECT u.user, count(*) as num FROM (SELECT user FROM nodes UNIO
N ALL SELECT user FROM ways) u group by u.user order by num desc limit 20;")

pprint.pprint(cursor.fetchall())
```

```
[(u'Matt Toups', 172024),
 (u'ELadner', 61360),
 (u'wvdp', 38444),
 (u'coleman_nolaimport', 17414),
 (u'ELadnerImp', 12556),
 (u'woodpeck_fixbot', 10518),
 (u'Matt Toups_nolaimport', 2665),
 (u'Minh Nguyen_nolaimport', 1920),
 (u'ceseifert_nolaimport', 1833),
 (u'Maarten Deen', 1296),
 (u'TIGERcnl', 1288),
 (u'Alecs01', 1174),
 (u'RichRico', 790),
 (u'Glassman', 769),
 (u'bot-mode', 743),
 (u'Skywave', 697),
 (u'42429', 584),
 (u'Minh Nguyen', 581),
 (u'maxerickson', 510),
 (u'DaveHansenTiger', 477)]
```

How many contributors only contribute one time

In [56]:

```
cursor.execute("SELECT count(*) FROM (SELECT u.user,count(*) as num FROM (SELECT
user FROM nodes UNION ALL SELECT user FROM ways) u group by u.user HAVING num=1)
i;")

pprint.pprint(cursor.fetchall())
```

```
[(119,)]
```

Additional data exploration using SQL

Top 5 amenities

In [75]:

```
cursor.execute("SELECT value, count(*) as num FROM nodes_tags WHERE key='amenity
' group by value order by num desc limit 5;")

pprint.pprint(cursor.fetchall())
```

```
[(u'place_of_worship', 48),
 (u'school', 48),
 (u'restaurant', 17),
 (u'kindergarten', 10),
 (u'grave_yard', 8)]
```

Count amenity bar

In [76]:

```
cursor.execute("SELECT count(*) FROM nodes_tags WHERE key='amenity'and value='ba
r';")
pprint.pprint(cursor.fetchall())
```

```
[(6,)]
```

Counclusion

After this review, I found the dataset is incomplete and cleaned up the street name data for this dataset, and then discussed about the contributor statistic related problem, such as initiating "gamification" to encourage user participation in contribution. Also, I noticed that lots of public domain data source (like tiger: Topologically Integrated Geographic Encoding and Referencing system) makes contributions into OpenStreetMap under user names like "Joel Carranza","chehrlic". With such data source or GPS working together, it will certainly help increasing the accurancy of the dataset input. However, it is possible that data from these source may have other potential problems that may need to cleaned or audied.

Reference

http://stackoverflow.com/questions/3095434/inserting-newlines-in-xml-file-generated-via-xml-etree-elementtree-in-python (http://stackoverflow.com/questions/3095434/inserting-newlines-in-xml-file-generated-via-xml-etree-elementtree-in-python) https://docs.python.org/2/library/xml.etree.elementtree.html (https://docs.python.org/2/library/xml.etree.elementtree.html) https://www.w3schools.com/sql/sql_quickref.asp (https://www.w3schools.com/sql/sql_quickref.asp) https://discussions.udacity.com/t/difficulty-with-inserting-csv-file-and-sql-query/186125/3 (https://discussions.udacity.com/t/difficulty-with-inserting-csv-file-and-sql-query/186125/3) https://www.daniweb.com/programming/software-development/code/216951/size-of-a-file-folder-directory-python (https://www.daniweb.com/programming/software-development/code/216951/size-of-a-file-folder-directory-python) https://blog.sqlauthority.com/2010/02/08/sql-server-find-the-size-of-database-file-find-the-size-of-log-file/ (https://blog.sqlauthority.com/2010/02/08/sql-server-find-the-size-of-database-file-find-the-size-of-log-file/) https://discussions.udacity.com/t/final-project-importing-cerberus-and-schema/177231/2 (https://discussions.udacity.com/t/final-project-importing-cerberus-and-schema/177231/2) https://gist.github.com/carlward/54ec1c91b62a5f911c42#file-sample_project-md (https://gist.github.com/carlward/54ec1c91b62a5f911c42#file-sample_project-md) https://discussions.udacity.com/t/cleaning-data-error-in-shape-element/208971/2 (https://discussions.udacity.com/t/cleaning-data-error-in-shape-element/208971/2) https://stackoverflow.com/questions/305378/list-of-tables-db-schema-dump-etc-using-the-python-sqlite3-api (https://stackoverflow.com/questions/305378/list-of-tables-db-schema-dump-etc-using-the-python-sqlite3-api)

In [ ]: