Data Wrangling

November 8, 2020

1 Data Wrangling With MongoDB Project

Project 2 from Udacity Data Analyst Nano Degree program by Jay Zmudka

---

1.0.1 Overview of the data

Openstreetmap is an opensource mapping system on the internet that is freely available at http://openstreetmap.com. The data is stored in an XML format. There are several ways to download map data for a given area. I chose to download map data for downtown Detroit, MI because it is near me and I'm familiar with the streets and buildings. I played with the website until I got a file with a size greater than 50MB. The bounds are:

minlat="42.3097000" minlon="-83.1011000" maxlat="42.3626000" maxlon="-82.9663000"

I used the overpass-api to download the osm data. https://www.openstreetmap.org/export#map=13/42.3362/-83.0336

The data file is around 70 MB and over the recommended maximum size file for Github, so I zipped it to get it down to 6 MB. I also zoomed in further to get a smaller sample.osm file that is 4 mb in size.

The code below loads the needed libraries for the project and gives a function to open the zip file with the data. To choose the sample or full file, uncomment the one you want to use.

[1]:
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import xml.etree.ElementTree as ET
from pprint import pprint as pp
import re
import codecs
import json
from zipfile import ZipFile
import pymongo

samplefile = 'sample.osm'
zipfile = 'openstreetmapdata.zip'
filename = 'openstreetmapdata'
outfile = 'openstreetmapdata.json'
```

```
# uncomment the data you want to use
usefile = samplefile # smaller sample file
#usefile = openzipfile(zipfile, filename) #Full size file 70mb

# extracts osm file from zip archive
def openzipfile(zipf, filename):
    with ZipFile(zipf) as osmzip:
        xfile = osmzip.open(filename)
        return xfile
```

1.0.2 Problems encountered in the map

The openstreet data is stuctured in an XML format. It uses 3 map elements. * Node: A point with coordinates and data about that location * Way: Defines linear features like streets and boundaries * Relation: Explains how other elements go together

There are also tags with other information with K and V values that are Key:Value pairs.

Here is a sample:

```
<node id="5959554765" visible="true" version="3" changeset="63267190" timestamp="2018-10-06T2
<tag k="addr:city" v="Detroit"/>
<tag k="addr:housenumber" v="400"/>
<tag k="addr:postcode" v="48243-1312"/>
<tag k="addr:state" v="MI"/>
<tag k="addr:street" v="Renaissance Drive West"/>
<tag k="name" v="Detroit Marriott at the Renaissance Center"/>
<tag k="operator" v="Marriott"/>
<tag k="phone" v="+1 313-568-8000"/>
<tag k="tourism" v="hotel"/>
</node>
```

I wrote some auditing fuctions to find probems with the data I took the following steps:

- Checked for abbreviated street names
- Checked for invalid URL's
- Checked for redundant TIGER data

I wrote the following functions

[2]:
```
#auditing functions

ROOTTAGS = [ 'node', 'way', 'relation'] #the three map elements

# Lists the total tags in .osm file
def list_totals(usefile):
    tagslist ={}
```

```
    total = 0
    for _, element in ET.iterparse(usefile):
        total += 1
        if element.tag not in tagslist:
            tagslist[element.tag] = 1
        else:
            tagslist[element.tag] = tagslist[element.tag] + 1
    print 'Counting all the tags: \n'
    print total, 'tags in total \n'
    pp(tagslist)

# Lists websites with bad URL's
def list_websites(usefile):
    websites = []
    for _, element in ET.iterparse(usefile):
        if len(element.findall('tag')) > 0:
            for tag in element.findall('tag'):
                if tag.attrib['k'] == 'website':
                    websites.append(tag.attrib['v'])
    print 'List of websites'
    pp(websites)
    badurls = []
    for w in websites:
        if not w.startswith('http'):
            badurls.append(w)
    print "\nThese websites are not proper URL's"
    pp(badurls)

# Lists some ways with redundant tiger tags
def find_tiger(usefile):
    tigertags = 0
    print 'Samples of ways containing redundant tiger street data:'
    for _, element in ET.iterparse(usefile):
        if element.tag == 'way':
            if len(element.findall('tag')) > 0:
                for tag in element.findall('tag'):
                    if tag.attrib['k'].startswith('tiger'):
                        print ET.tostring(element)
                        tigertags += 1
                        break
                pass
        if tigertags > 1:
            break

#compiles and prints list of street suffixes
def street_endings(usefile):
    street_endings = set()
```

```
    for _, element in ET.iterparse(usefile):
        if element.tag == 'way':
            if len(element.findall('tag')) > 0:
                for tag in element.findall('tag'):
                    if tag.attrib['k'] == 'name':
                        street_endings.add(tag.attrib['v'].split()[-1])
    print street_endings
```

Now let's audit the data. How many total tags are in the file?

[3]:
```
list_totals(usefile)
```

```
Counting all the tags:

74700 tags in total

{'bounds': 1,
 'member': 45199,
 'nd': 10927,
 'node': 7577,
 'osm': 1,
 'relation': 211,
 'tag': 9422,
 'way': 1362}
```

The class material used street name abbreviations as an example of something to fix. I will make a list of them below.

[4]:
```
street_endings(usefile)
```

```
set(['Substation', 'Boulevard', 'Group', '600', 'West', 'Pyramid', 'Martius',
'#8735', 'Woodward', '300', 'Fountain', 'Authority', 'Lot', 'East', '100',
'Row', 'Suites', 'Parc', 'Deli', 'Amphitheatre', "Steven's", 'Park', 'Free',
'Promenade', 'Parking', 'Building)', 'Protection', 'Riverfront', '400',
'Esplanade', 'AT&T', 'Law', 'Hall', 'Bank', 'Rooms', 'Building', '200',
'School', 'Theater', 'Center', 'Tunnel', '(1905)', 'Plaza', 'Drive', 'Detroit',
'Headquarters', 'Freeway', 'Garage', 'Square', 'Place', 'Monument', 'Tower',
'Tavern', 'Patio', '(1928)', 'Apartments', "DuMouchelle's", 'House',
'Scientology', 'Deck', 'Alley', 'Qube', 'Street', '500', 'QLINE', 'Church',
'Riverwalk', 'Mover', 'Cafe', 'Avenue', 'Campus', '(1915)'])
```

There aren't any abbreviations here. That isn't a problem that needs to be fixed in this data set. Here I check for tags with bad website URL's

[5]:
```
list_websites(usefile)
```

```
List of websites
['http://www.bangkokcrossingthaifood.com/',
 'https://www.starbucks.com/store/12550/us/marriott-rencen-detroit/renaissance-
center-detroit-mi-48243/renaissance',
```

```
 'https://citymarketdetroit.com/',
 'www.eatdimestore.com',
 'https://www.mckinsey.com/midwest/detroit',
 'https://entangleagency.com',
 'http://cadillacsquarediner.com',
 'https://salondetroit.com',
 'https://www.newcadillacsquare.com',
 'parcderoit.com',
 'http://www.sspeterandpauljesuit.org/',
 'http://www.christcd.org/',
 'http://www.portdetroit.com',
 'https://www.bcbsm.com/',
 'https://www.tcfcenterdetroit.com/']

These websites are not proper URL's
['www.eatdimestore.com', 'parcderoit.com']
```

I noticed some redundant street information in the file. It is from The Topologically Integrated Geographic Encoding and Referencing system (TIGER) data, produced by the US Census Bureau. It was imported into OSM in 2006 and 2007. This is extraneous data because it duplicates the address information already there. Here is how I checked for it:

[6]:
```
find_tiger(usefile)
```

```
Samples of ways containing redundant tiger street data:
<way changeset="70380121" id="8732681" timestamp="2019-05-18T07:03:36Z"
uid="1240864" user="Howpper" version="20" visible="true">
  <nd ref="62601111" />
  <nd ref="1254758160" />
  <tag k="highway" v="primary" />
  <tag k="lanes" v="4" />
  <tag k="maxspeed" v="30 mph" />
  <tag k="name" v="West Jefferson Avenue" />
  <tag k="oneway" v="yes" />
  <tag k="ref" v="M 10" />
  <tag k="tiger:cfcc" v="A45" />
  <tag k="tiger:county" v="Wayne, MI" />
  <tag k="tiger:name_base" v="Jefferson" />
  <tag k="tiger:name_direction_prefix" v="W" />
  <tag k="tiger:name_type" v="Ave" />
 </way>

<way changeset="76521942" id="8734148" timestamp="2019-11-02T08:31:11Z"
uid="6656962" user="StudentinGear" version="31" visible="true">
  <nd ref="62618538" />
  <nd ref="1254758157" />
  <nd ref="6938840835" />
  <nd ref="5552762753" />
  <nd ref="62618540" />
```

```
 <nd ref="62618542" />
 <nd ref="4181389363" />
 <tag k="cycleway:right" v="no" />
 <tag k="highway" v="primary" />
 <tag k="lanes" v="4" />
 <tag k="maxspeed" v="30 mph" />
 <tag k="name" v="East Jefferson Avenue" />
 <tag k="oneway" v="yes" />
 <tag k="ref" v="M 10" />
 <tag k="tiger:cfcc" v="A41" />
 <tag k="tiger:county" v="Wayne, MI" />
 <tag k="tiger:name_base" v="Jefferson" />
 <tag k="tiger:name_direction_prefix" v="E" />
 <tag k="tiger:name_type" v="Ave" />
</way>
```

1.0.3 Cleaning the data

Here are a couple functions I wrote. One fixes bad URL's and the other removes the tiger tags

[7]:
```
# cleaning functions

#increments tigertag counter
tigertags_count = 0 # counter for tigertags

def increment_tigertags():
    global tigertags_count
    tigertags_count += 1

# Sets to be used for the cleaning and insertion of data

CREATED = [ 'version', 'changeset', 'timestamp', 'user', 'uid']
problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')

# Function to find and fix bad url's
def fix_urls(element):
    if element.tag == 'node' or element.tag == 'way':
        if len(element.findall('tag')) > 0:
                for tag in element.findall('tag'):
                    if tag.attrib['k'] == 'website':
                        url = tag.attrib['v']
                        url = url.lower().strip()
                        if not url.startswith('http'):
                            print 'old url', url
                            url = 'http://' + url
                            print 'fixed url', url
                        tag.attrib['v'] = url
```

```
    return element

# Function to remove tiger: tags
def remove_tigertags(element):
    if element.tag == 'way':
        tt = False
        if len(element.findall('tag')) > 0:
                for tag in element.findall('tag'):
                    if tag.attrib['k'].startswith('tiger'):
                        element.remove(tag)
                        increment_tigertags()
    return element
```

This function cleans up an element and returns a dictionary document ready to be inserted into a Json file. I use the following structure for my JSON data:

```
{'address': {'city': 'Detroit',
         'country': 'US',
         'housenumber': '500',
         'postcode': '48226',
         'state': 'MI',
         'street': 'Woodward Avenue'},
 'amenity': 'bank',
 'created': {'changeset': '79123908',
         'timestamp': '2020-01-02T17:32:29Z',
         'uid': '1836535',
         'user': 'GITNE',
         'version': '2'},
 'id': '5965241106',
 'name': 'Flagstar Bank',
 'pos': [42.3299882, -83.0451545],
 'type': 'node',
 'visible': 'true'}
```

[8]:
```
#Function that takes a Elementtree element and returns cleaned up json ready␣
 ↪dictionary
#This function is modified from 12: Quiz Preparing for Database MongDB in the␣
 ↪Udacity material

def shape_element(element):
    node = {}
    if element.tag in ROOTTAGS:

        fix_urls(element) #Fixes bad URL's
        remove_tigertags(element)#Removes tiger tags

        node['type'] = element.tag
        node['id'] = element.attrib['id']
```

```
        if 'visible' in element.attrib:
            node['visible'] = element.attrib['visible']
        node['created'] = {}

        for x in CREATED: #Rolls up created info tags
            node['created'][x] = element.attrib[x]

        if 'lat' in element.attrib: #adds Long and Lat tuple to document
            node['pos'] = (float(element.attrib['lat']), float(element.
↪attrib['lon']))

        # Rolls up address tags and adds other tags to document
        if len(element.findall('tag')) > 0:
            node['address'] = {}
            for tag in element.findall('tag'):

                if problemchars.search(tag.attrib['k']): #checks for bad␣
↪characters
                    pass
                elif tag.attrib['k'] == 'type':
                    node['relation_type'] = tag.attrib['v']
                elif 'addr:' in tag.attrib['k']:
                    if len(tag.attrib['k'].split(':')) > 2:
                        pass
                    else:
                        addc = tag.attrib['k'].split(':')[1]
                        node['address'][addc] = tag.attrib['v']
                else:
                    node[tag.attrib['k']] = tag.attrib['v']

        #makes list of node_refs tags
        if len(element.findall('nd')) > 0:
            node['node_refs'] = []
            for nd in element.findall('nd'):
                node['node_refs'].append(nd.attrib['ref'])

        # removes empty address tags
        if 'address' in node:
                if len(node['address']) == 0:
                    del node['address']

        return node
    else:
        return None
```

Here we iterate through the OSM file, calling the shape_element function, and output the cleaned data to a Json file

[9]:
```
# read data into elementtree objects
# calls shape_element fuction
# writes out json file

data = []
with codecs.open(outfile, "w") as fo:
    for _, element in ET.iterparse(usefile):
        piece = shape_element(element)
        if piece != None:
            data.append(piece)
            fo.write(json.dumps(piece) + "\n")
    print 'Removed', tigertags_count, 'tiger tags'
    print 'Output saved to file:', outfile
```

```
old url www.eatdimestore.com
fixed url http://www.eatdimestore.com
old url parcderoit.com
fixed url http://parcderoit.com
Removed 987 tiger tags
Output saved to file: openstreetmapdata.json
```

1.0.4 Exploring the data with MongoDB

This inserts our JSON file data into MongoDB

[10]:
```
# insert into MongoDB database
connection = pymongo.MongoClient('localhost', 27017)
if 'openstreetmap' in connection.list_database_names():
    db = connection['openstreetmap']
    db.main.drop()
    print('Deleted existing database')

print 'Making new database openstreetmap'
db = connection['openstreetmap']

with open(outfile) as f:
    data = [json.loads(line) for line in f]
    db.main.insert_many(data)
```

```
Deleted existing database
Making new database openstreetmap
```

Now that we have a mongoDB database. Lets display some statistics.

[11]:
```
# explore the db with PyMongo
#find size stats of database
```

```
# database stats
stats = 'dbstats', db.command('dbstats')
pp(stats)
#size of file in KB
print db.command('dbstats')['storageSize'] / 1024, 'KB file'

# how many nodes, ways, and relations
print
print db.main.find().count(), 'total documents\n'
print db.main.count_documents({'type':'node'}), 'nodes'
print db.main.count_documents({'type':'way'}), 'ways'
print db.main.count_documents({'type':'relation'}), 'relations\n'
```

```
('dbstats',
 {u'avgObjSize': 273.03551912568304,
  u'collections': 1,
  u'dataSize': 2498275.0,
  u'db': u'openstreetmap',
  u'fsTotalSize': 484272263168.0,
  u'fsUsedSize': 134843179008.0,
  u'indexSize': 4096.0,
  u'indexes': 1,
  u'numExtents': 0,
  u'objects': 9150,
  u'ok': 1.0,
  u'storageSize': 4096.0,
  u'views': 0})
4.0 KB file

9150 total documents

7577 nodes
1362 ways
211 relations


C:\Users\Owner\anaconda3\envs\py2\lib\site-packages\ipykernel_launcher.py:12:
DeprecationWarning: count is deprecated. Use Collection.count_documents instead.
  if sys.path[0] == '':
```

Lets find out how many unique users contributed to the OSM data. Who are the top 5 contributers?

[12]:
```
#how many unique users
print len(db.main.distinct('created.user')), 'unique users\n'


#Which 5 users contributed the most to the data?
```

```
result = db.main.aggregate([{'$group' : { '_id' : '$created.user',
                                          'count' : {'$sum': 1}}},
                            {'$sort' : {'count' : -1 }},
                            {'$limit' : 5}])
print 'Top 5 contributers'
for doc in result:
    print doc
```

```
130 unique users

Top 5 contributers
{u'count': 3036, u'_id': u'StudentinGear'}
{u'count': 1078, u'_id': u'lmum'}
{u'count': 1015, u'_id': u'MichaelGSmith'}
{u'count': 542, u'_id': u'marianp_telenav'}
{u'count': 480, u'_id': u'jmcbroom'}
```

How many bus stops are in the data?

[13]:
```
# number of busstops

print db.main.count_documents({'highway':'bus_stop'}), 'bus stops'
```

```
31 bus stops
```

Which Node is the most referenced?

[14]:
```
# which node is referenced the most?

result = db.main.aggregate([ {'$match' : {'type' : {'$in': ['way',␣
 ↪'relation']}}},
                            {'$unwind': '$node_refs'},
                            {'$group': {'_id' : '$node_refs',
                                        'count' :{'$sum': 1}}},
                            {'$sort' : { 'count' : -1 }},
                            {'$limit' : 1 }])
print 'Here is the most referenced node:\n'
for doc in result:
    id = doc['_id']
    for i in db.main.find({'id' : id}):
        pp(i)
        print 'Referenced in', doc['count'], 'records'
        coords = i['pos']
```

```
Here is the most referenced node:

{u'_id': ObjectId('5fa898750e1a32c0ed82ab90'),
 u'created': {u'changeset': u'65652458',
              u'timestamp': u'2018-12-20T21:09:21Z',
              u'uid': u'6656962',
```

```
              u'user': u'StudentinGear',
              u'version': u'2'},
 u'id': u'1236512572',
 u'pos': [42.3320589, -83.0430423],
 u'type': u'node',
 u'visible': u'true'}
Referenced in 7 records
```

One of the cool features of mongodb is Geospacial queries. I'm going to try one. What are the 5 closest restaurants to that node?

[15]:
```
# 5 closest retaurants to that node

#build index on id, geo2d index on pos field
db.main.create_index([('pos', pymongo.GEO2D)])

closest_restaurants = db.main.find({'pos' : {'$near':  coords}, 'amenity' :␣
 ↪'restaurant'}).limit(5)
print '\nFive nearest restaurants \n'
for i in closest_restaurants:
    print i['name']
    if 'address' in i:
        print i['address']['housenumber'], i['address']['street']
    else:
        print 'No address given'
    if 'cuisine' in i:
        print 'Cuisine', i['cuisine']
    print '----------------'
```

```
Five nearest restaurants

Cadillac Square Diner
111 Cadillac Square
Cuisine american
----------------
IHOP
No address given
Cuisine breakfast;pancake
----------------
Lunch Time Global
660 Woodward Avenue
Cuisine soup
----------------
Applebee's
No address given
Cuisine american
----------------
```

```
Central Kitchen + Bar
660 Woodward Avenue
Cuisine american
----------------
```

1.0.5 Other ideas about the dataset

Openstreetmap is a great resource if you want a bare bones mapping system available without a cost. The popular Pokemon Go app, for instance, runs on Openstreetmap data. There is a lot of incomplete data about amenities such as restaurants and stores. Entries lack addresses, phone numbers, and websites. I also noticed that most nodes have no description at all. All of this is fixable under their current system with better quality contributions.

Here is a sample of restaurants and stores missing address data.

[16]:
```
# give us 3 restaurants that are missing address data
sample_restaurants = db.main.find({'type' : 'node', 'amenity':'restaurant',␣
 ↪'address': {'$exists' : False}}).limit(3)

print 'Restaurants'
for doc in sample_restaurants:
    pp(doc)
    print
```

```
Restaurants
{u'_id': ObjectId('5fa898750e1a32c0ed82ae3d'),
 u'amenity': u'restaurant',
 u'created': {u'changeset': u'29172136',
              u'timestamp': u'2015-03-01T08:55:28Z',
              u'uid': u'2441939',
              u'user': u'thevirginian',
              u'version': u'3'},
 u'cuisine': u'thai',
 u'id': u'2390730308',
 u'name': u'Bangkok Crossing',
 u'pos': [42.3306851, -83.0456979],
 u'type': u'node',
 u'visible': u'true',
 u'website': u'http://www.bangkokcrossingthaifood.com/'}

{u'_id': ObjectId('5fa898750e1a32c0ed82ae4d'),
 u'amenity': u'restaurant',
 u'created': {u'changeset': u'45788919',
              u'timestamp': u'2017-02-03T20:06:34Z',
              u'uid': u'1482360',
              u'user': u'jmcbroom',
              u'version': u'2'},
 u'cuisine': u'tex-mex',
```

```
 u'id': u'2558966844',
 u'name': u'Calexico',
 u'pos': [42.3329263, -83.0474543],
 u'type': u'node',
 u'visible': u'true'}

{u'_id': ObjectId('5fa898750e1a32c0ed82b3b9'),
 u'amenity': u'restaurant',
 u'created': {u'changeset': u'45791955',
              u'timestamp': u'2017-02-03T22:03:46Z',
              u'uid': u'1482360',
              u'user': u'jmcbroom',
              u'version': u'1'},
 u'cuisine': u'steak_house',
 u'id': u'4661181827',
 u'name': u'Texas De Brazil',
 u'pos': [42.3324368, -83.0471077],
 u'type': u'node',
 u'visible': u'true'}
```

[17]:
```
# give us 3 shops that are missing address data
sample_stores = db.main.find({'shop' : {'$exists' : True}, 'address' :␣
 ↪{'$exists' : False}}).limit(3)

print 'Shops'
for doc in sample_stores:
    pp(doc)
    print
```

```
Shops
{u'_id': ObjectId('5fa898750e1a32c0ed82b9c3'),
 u'created': {u'changeset': u'63243814',
              u'timestamp': u'2018-10-05T21:25:32Z',
              u'uid': u'1723055',
              u'user': u'Johnny Mapperseed',
              u'version': u'1'},
 u'id': u'5959960423',
 u'name': u'Pure Detroit',
 u'pos': [42.3266229, -83.048085],
 u'shop': u'clothes',
 u'type': u'node',
 u'visible': u'true'}

{u'_id': ObjectId('5fa898750e1a32c0ed82b9cc'),
 u'brand': u'Under Armour',
 u'brand:wikidata': u'Q2031485',
 u'brand:wikipedia': u'en:Under Armour',
```

```
 u'clothes': u'men;women',
 u'created': {u'changeset': u'72497367',
              u'timestamp': u'2019-07-22T03:59:06Z',
              u'uid': u'115918',
              u'user': u'Timothy Smith',
              u'version': u'2'},
 u'id': u'5960160426',
 u'name': u'Under Armour',
 u'pos': [42.3329852, -83.0481265],
 u'shop': u'clothes',
 u'type': u'node',
 u'visible': u'true'}

{u'_id': ObjectId('5fa898750e1a32c0ed82b9f7'),
 u'created': {u'changeset': u'63293170',
              u'timestamp': u'2018-10-08T01:43:00Z',
              u'uid': u'1723055',
              u'user': u'Johnny Mapperseed',
              u'version': u'1'},
 u'id': u'5965241107',
 u'name': u'BESA',
 u'pos': [42.3304601, -83.0455006],
 u'shop': u'yes',
 u'type': u'node',
 u'visible': u'true'}
```

Most of the map data in OSM is contributed by volunteers. As is, I would have to cross reference some address database to find a restaurant near me. They could contact businesses and encourage them to make their own entries and keep them up to date. Maybe the local chambr of commerce could be involved. They could also find corporate sponsors that have a stake in having up to date map data, like auto manufacturers or ride share companies, to pay to have their employees contribute to the map data. Opensteetmap has a lot of potential because it is opensource and anybody can contribute. In the future, I would like to see elevation tags added to the node data so that it can be use for topographical mapping.

1.0.6 websites I used for reference

https://wiki.openstreetmap.org/wiki/Elements
https://www.tutorialspoint.com/how-to-count-the-number-of-documents-in-a-mongodb-collection
https://docs.mongodb.com/manual/reference/command/dbStats/#dbStats.dataSize
https://docs.mongodb.com/manual/core/2dsphere/
https://docs.mongodb.com/manual/reference/operator/aggregation/
https://www.geeksforgeeks.org/drop-collection-if-already-exists-in-mongodb-using-python/
https://docs.python.org/2/library/xml.etree.elementtree.html
https://docs.python.org/3/library/json.html

[ ]: