Data Analyst Nanodegree

Project 3 : Wrangle OpenStreet Map Data ¶

Shreyas Ramnath, 25th July 2017

Map Location: San Jose, California, United States (https://s3.amazonaws.com/metro-extracts.mapzen.com/san-jose_california.osm.bz2)

Goal: Auditing & Cleaning Open Street Map Dataset, Converting from the XML to JSON format, Analyzing insight within the data.

Bibliography:

Udacity "Data Wrangling with MongoDB" (https://www.udacity.com/course/data-wrangling-with-mongodb--ud032)

CCEO Street Abbreviations Guide (http://www.cceo.org/addressing/documents/StreetAbbreviationsGuide.pdf)

MongoDB Importing XML to JSON (https://docs.mongodb.org/manual/reference/program/mongoimport/)

1. Data Auditing

In [1]:

```
import xml.etree.cElementTree as ET
import re
import codecs
import json
import collections
import pymongo
import pprint
```

In [2]:

```
import os
datadirectory = "/home/shreyas/Downloads"
datafile = "sj.osm"
calculated_data = os.path.join(datadirectory, datafile)
```

> Used cElementTree module to parse the dataset and did a count of the unique types of elements to ascertain entire structure to the dataset at hand.

```
In [3]: def count_numberOf_tags(filename):
                tags = {}
                for event, elem in ET.iterparse(filename):
                    if elem.tag in tags:
                        tags[elem.tag] += 1
                    else:
                        tags[elem.tag] = 1
                return tags
        calculated_tags = count_numberOf_tags(calculated_data)
        pprint.pprint(calculated_tags)
```

```
{'bounds': 1,
 'member': 18335,
 'nd': 1965861,
 'node': 1680038,
 'osm': 1,
 'relation': 1760,
 'tag': 705767,
 'way': 229482}
```

Using the functions typeOfkey and procMap we check the "k" value for each "" ,to see if they can be valid keys. As in the course quiz earlier, one would like to change the data model and expand the "addr:street" type of keys to a dictionary like this: **{"address": {"street": "Some value"}}** So, we have to see if we have such tags, and if we have any tags with problematic characters.

For the function 'key_type', we have a count of each of three tag categories in a dictionary: "lowerNcolon", for otherwise valid tags with a colon in their names, "faultyChar", for tags with problematic characters "lower", for tags that contain only lowercase letters and are valid,

In [4]:

```
import re

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

def typeOfkey(element, keys):
    if element.tag == "tag":
        for tag in element.iter('tag'):
            k = tag.get('k')
            if lower.search(k):
                keys['lower'] += 1
            elif lowerNcolon.search(k):
                keys['lowerNcolon'] += 1
            elif faultyChar.search(k):
                keys['faultyChar'] += 1
            else:
                keys['other'] += 1
    return keys


def procMap(filename):
    keys = {"lower": 0, "lowerNcolon": 0, "faultyChar": 0, "other": 0}
    for _, element in ET.iterparse(filename):
        keys = typeOfkey(element, keys)

    return keys

calculated_keys = procMap(calculated_data)
pprint.pprint(calculated_keys)
```

```
{'faultyChar': 1, 'lower': 459154, 'lowerNcolon': 224641, 'other': 21971}
```

How many unique users have contributed to the map in San Jose area? 1361 unique users have already worked on this map area !

In [5]:

```
#people invovlved in the map editing.
def procMap(filename):
    users = set()
    for _, element in ET.iterparse(filename):
        for e in element:
            if 'uid' in e.attrib:
                users.add(e.attrib['uid'])
    return users
users = procMap(calculated_data)
len(users)
```

Out[5]:

```
1361
```

2. Existing Problems

2.1 Street name full form

> The main problem in this dataset comes from the street name inconsistencies. We build regular expressions matching last element in the string i.e, where street type is based. Then a list of mapping that need'nt be cleaned is computed.

In [6]:

```
from collections import defaultdict

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

expected = ["Avenue", "Boulevard", "Commons", "Court", "Drive",
            "Lane","Parkway","Place", "Road", "Square", "Street",
            "Trail"]

mapping = {'Ave'  : 'Avenue',
           'Blvd' : 'Boulevard',
           'Dr'   : 'Drive',
           'Ln'   : 'Lane',
           'Pkwy' : 'Parkway',
           'Rd'   : 'Road',
           'Rd.'   : 'Road',
           'St'   : 'Street',
           'street' :'Street',
           'Ct'   : 'Court',
           'Cir'  : 'Circle',
           'Cr'   : 'Court',
           'ave'  : 'Avenue',
           'Hwg'  : 'Highway',
           'Hwy'  : 'Highway',
           'Sq'   : 'Square'}
```

> - auditStreet searches the input string for the regular expression for matching. If there is a match and it is not within the "expected" list, the match is added as a key and the string is added to the set.
> - isstreetName looks at the attribute k if k="addre:street"
> - datAudit returns the list that matches the previous two functions. With the list of all the abbreviated street types we can understand and fill the "mapping" dict as a pre-process to convert these street names into correct form. **For presentation purpose I have shown only a sample, by removing for loops and the counter you can get the entire output**.

In [7]:

```
def auditStreet(street_types, street_name):
    m = streeTypeRegex.search(street_name)
    if m:
        street_type = m.group()
        if street_type not in expected:
            street_types[street_type].add(street_name)

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

def datAudit(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 isstreetName(tag):
                    auditStreet(street_types, tag.attrib['v'])

    return street_types

calStreetType = datAudit(calculated_data)
calStType = dict(calStreetType)
counter = 0;
for i in calStType:
    print i , calStType[i]
    counter = counter + 1
    if counter == 4:
        break;
```

```
Walk set(['Paseo de San Antonio Walk'])
Rd set(['Wolfe Rd', 'Mt Hamilton Rd', 'Berryessa Rd', 'Saratoga Los Gatos R
d', 'Quimby Rd', 'San Antonio Valley Rd', 'Homestead Rd', 'Mt. Hamilton Rd',
'Silver Creek Valley Rd'])
7.1 set(['Hwy 17 PM 7.1'])
Hill set(['Blossom Hill'])
```

newName takes the old name and updates them. **Remove the last break for the entire output !**

In [8]:

```
def newName(name, mapping, regex):
    m = regex.search(name)
    if m:
        street_type = m.group()
        if street_type in mapping:
            name = re.sub(regex, mapping[street_type], name)

    return name

for street_type, ways in calStreetType.iteritems():
    for name in ways:
        better_name = newName(name, mapping, streeTypeRegex)
        print name, "=>", better_name
    break
```

```
Martin Avenue #6 => Martin Avenue #6
Pruneridge Ave #6 => Pruneridge Ave #6
```

2.2 Incorrect Zip Codes

> Most of the zip codes are correct, but there are still many zip codes with incorrect 5 digit formats in the data. **Remove the last break for the entire output !**

In [9]:

```
from collections import defaultdict

def audit_zipcode(invalid_zipcodes, zipcode):
    twoDigits = zipcode[0:2]

    if not twoDigits.isdigit():
        invalid_zipcodes[twoDigits].add(zipcode)

    elif twoDigits != 95:
        invalid_zipcodes[twoDigits].add(zipcode)

def is_zipcode(elem):
    return (elem.attrib['k'] == "addr:postcode")

def audit_zip(osmfile):
    osm_file = open(osmfile, "r")
    invalid_zipcodes = 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_zipcode(tag):
                    audit_zipcode(invalid_zipcodes,tag.attrib['v'])

    return invalid_zipcodes

calculated_zipcode = audit_zip(calculated_data)
calZipCode = dict(calculated_zipcode)
counter = 0;
for i in calZipCode:
    counter = counter + 1
    if counter == 2:
        break
    print i , calZipCode[i]
```

```
CA set(['CA 95110', 'CA 94035', 'CA 94086', 'CA 95054', 'CA 95116'])
```

The output of the cleaned zip codes is below. There are the formatting of 5 digits, 4 digits and 5 digits which are valid.**Remove the last break for the entire output !**

In [10]:

```
def update_name(zipcode):
    testNum = re.findall('[a-zA-Z]*', zipcode)
    if testNum:
        testNum = testNum[0]
    testNum.strip()
    if testNum == "CA":
        convertedZipcode = (re.findall(r'\d+', zipcode))
        if convertedZipcode:
            if convertedZipcode.__len__() == 2:
                return (re.findall(r'\d+', zipcode))[0] + "-" +(re.findall(r
'\d+', zipcode))[1]
            else:
                return (re.findall(r'\d+', zipcode))[0]

for street_type, ways in calculated_zipcode.iteritems():
    for name in ways:
        better_name = update_name(name)
        print name, "=>", better_name
    break
```

```
CA 95110 => 95110
CA 94035 => 94035
CA 94086 => 94086
CA 95054 => 95054
CA 95116 => 95116
```

Steps to transform the data from XML to JSON

- Process only 2 types of top level tags: "node" and "way
- All attributes of "node" and "way" should be turned into regular key/value pairs, except: attributes in the CREATED array should be added under a key "created", attributes for latitude and longitude should be added to a "pos" array, for use in geo-spatial indexing. Make sure the values inside "pos" array are floats and not strings.
- If second level tag "k" value contains problematic characters, it should be ignored
- If second level tag "k" value starts with "addr:", it should be added to a dictionary "address"
- If second level tag "k" value does not start with "addr:", but contains ":", you can process it same as any other tag.
- If there is a second ":" that separates the type/direction of a street, the tag should be ignored
- After all the cleaning and data transformation is done, we use procMap and convert the file from XML into JSON format
- **Remove the semicolon at the end to view the contents of the JSON file !**

In [11]:

```
import re
import codecs
import json

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

```
CREATED = [ "version", "changeset", "timestamp", "user", "uid"]


def shape_element(element):
    node = {}
    if element.tag == "node" or element.tag == "way" :

        node['type'] = element.tag
        # initialize empty address set
        address = {}
        # parsing through each of the attributes
        for a in element.attrib:
            if a in CREATED:
                if 'created' not in node:
                    node['created'] = {}
                node['created'][a] = element.get(a)
            elif a in ['lat', 'lon']:
                continue
            else:
                node[a] = element.get(a)
        # populating the position by latitute and longitude
        if 'lat' in element.attrib and 'lon' in element.attrib:
            node['pos'] = [float(element.get('lat')), float(element.get('lon'
))]

        # parsing second-level tags for nodes
        for e in element:
            # parsing second-level tags for ways and populating `node_refs`
            if e.tag == 'nd':
                if 'node_refs' not in node:
                    node['node_refs'] = []
                if 'ref' in e.attrib:
                    node['node_refs'].append(e.get('ref'))

            # Ignore non-tag elements and elements which are without `k` or `v
`

            if e.tag != 'tag' or 'k' not in e.attrib or 'v' not in e.attrib:
                continue
            key = e.get('k')
            val = e.get('v')

            # skipping faulty characters
            if problemchars.search(key):
                continue

            # parsing addresses of k-v pairs
            elif address_regex.search(key):
                key = key.replace('addr:', '')
                address[key] = val

            # catching all if everything else falls through
            else:
                node[key] = val
        # compiling the address
        if len(address) > 0:
            node['address'] = {}
            street_full = None
            street_dict = {}
            street_format = ['prefix', 'name', 'type']
            # parsing through address objects
            for key in address:
                val = address[key]
                if street_regex.search(key):
                    if key == 'street':
                        street_full = val
                    elif 'street:' in key:
                        street_dict[key.replace('street:', '')] = val
                else:
                    node['address'][key] = val
            # assigning street_full or fallback to compile street dict
            if street_full:
                node['address']['street'] = street_full
            elif len(street_dict) > 0:
                node['address']['street'] = ' '.join([street_dict[key] for key
 in street_format])
        return node
    else:
        return None


def process_map(file_in, pretty = False):
    file_out = "{0}.json".format(file_in)
    data = []
    with codecs.open(file_out, "w") as fo:
        for _, element in ET.iterparse(file_in):
            el = shape_element(element)
            if el:
                data.append(el)
                if pretty:
                    fo.write(json.dumps(el, indent=2)+"\n")
                else:
                    fo.write(json.dumps(el) + "\n")
    return data
process_map(calculated_data);
```

3. Data Wrangling with MongoDB

In [12]:

```
import signal
import subprocess
pro = subprocess.Popen('mongod', preexec_fn = os.setsid)
```

In [13]:

```
from pymongo import MongoClient

db_name = 'openstreetmap'

# Connecting to MongoDB
client = MongoClient('localhost:27017')
db = client[db_name]
```

In [14]:
```
# Building the mongoimport command
collection = calculated_data[:calculated_data.find('.')]
json_file = calculated_data + '.json'

mongoimport_cmd = 'mongoimport -h 127.0.0.1:27017 ' + \
                  '--db ' + db_name + \
                  ' --collection ' + collection + \
                  ' --file ' + json_file

# Before importing, drop the collection , if it is already running
if collection in db.collection_names():
    print 'Dropping collection: ' + collection
    db[collection].drop()

# Executing the command
print 'Executing: ' + mongoimport_cmd
subprocess.call(mongoimport_cmd.split())
```

```
Dropping collection: /home/shreyas/Downloads/sj
Executing: mongoimport -h 127.0.0.1:27017 --db openstreetmap --collection /ho
me/shreyas/Downloads/sj --file /home/shreyas/Downloads/sj.osm.json
```

Out[14]:
```
0
```

In [15]:
```
sanjose_california = db[collection]
```

Contrast of the two osm and json files to get an idea

In [16]:
```
import os
print 'OSM file {} GB'.format(os.path.getsize(calculated_data)/1.0e9) #Convers
ion from bytes to Gigabytes
print 'JSON file {} GB'.format(os.path.getsize(calculated_data + ".json")/1.0e
9) #Conversion from bytes to Gigabytes
```

```
OSM file 0.365134512 GB
JSON file 0.420199428 GB
```

What are the total number of documents ?

In [17]:
```
sanjose_california.find().count()
```

Out[17]:
```
1909520
```

What are the total number of unique users who have contributed to the San Jose map?

In [18]:
```
len(sanjose_california.distinct('created.user'))
```

Out[18]:
```
1355
```

What is the total number of nodes and ways in the map data?

```
In [19]: print "Nodes:",sanjose_california.find({'type':'node'}).count()
         print "Ways:",sanjose_california.find({'type':'way'}).count()
```

```
Nodes: 1680030
Ways: 229454
```

Who are the top three contributors to the map,? Their name, number of contributions and their unique id

```
In [20]: result = sanjose_california.aggregate( [
                                                { "$group" : {"_id" : "$created.user",

                                                "count" : { "$sum" : 1} } },
                                                { "$sort" : {"count" : -1} },
                                                { "$limit" : 3 } ] )

         print(list(result))
```

```
[{u'count': 295630, u'_id': u'andygol'}, {u'count': 285192, u'_id': u'nmixte
r'}, {u'count': 147442, u'_id': u'mk408'}]
```

4. Diving Deeper into MongoDB

What are the top five amenities in San Jose Area?

```
In [21]: amenity = sanjose_california.aggregate([{'$match': {'amenity': {'$exists': 1
         }}}, \
                                        {'$group': {'_id': '$amenity', \
                                                    'count': {'$sum': 1}}}, \
                                        {'$sort': {'count': -1}}, \
                                        {'$limit': 5}])
         print(list(amenity))
```

```
[{u'count': 2067, u'_id': u'parking'}, {u'count': 1024, u'_id': u'restauran
t'}, {u'count': 532, u'_id': u'fast_food'}, {u'count': 531, u'_id': u'schoo
l'}, {u'count': 353, u'_id': u'place_of_worship'}]
```

What the are the top three popular cuisines in San Jose?

In [22]:

```
cuisine = sanjose_california.aggregate([{"$match":{"amenity":{"$exists":1},
                                 "amenity":"restaurant",}},
                      {"$group":{"_id":{"Food":"$cuisine"},
                                 "count":{"$sum":1}}},
                      {"$project":{"_id":0,
                                  "Food":"$_id.Food",
                                  "Count":"$count"}},
                      {"$sort":{"Count":-1}},
                      {"$limit":3}])
print(list(cuisine))
```

```
[{u'Food': None, u'Count': 288}, {u'Food': u'mexican', u'Count': 90}, {u'Foo
d': u'vietnamese', u'Count': 77}]
```

What are the 5 most popular postal codes in San Jose?

In [23]:

```
postcode = sanjose_california.aggregate( [
    { "$match" : { "address.postcode" : { "$exists" : 1} } },
    { "$group" : { "_id" : "$address.postcode", "count" : { "$sum" : 1} } },
    { "$sort" : { "count" : -1}},
      {"$limit":5}] )
print(list(postcode))
```

```
[{u'count': 347, u'_id': u'95014'}, {u'count': 239, u'_id': u'95070'}, {u'cou
nt': 209, u'_id': u'94087'}, {u'count': 196, u'_id': u'94086'}, {u'count': 17
4, u'_id': u'95051'}]
```

Are there users who have only one post till date?

In [24]:

```
users = sanjose_california.aggregate( [
    { "$group" : {"_id" : "$created.user",
                "count" : { "$sum" : 1} } },
    { "$group" : {"_id" : "$count",
                "num_users": { "$sum" : 1} } },
    { "$sort" : {"_id" : 1} },
    { "$limit" : 1} ] )
print(list(users))
```

```
[{u'num_users': 289, u'_id': 1}]
```

If there are buildings, How many of each type are there?