Lesson 6: Case Study – Openstreetmap Data

Iterative Parsing

```
def count_tags(filename):
    tree = ET.parse(filename)
    root = tree.getroot()
    tags = {}
    iter_ = tree.getiterator()
    for elem in iter_:
        if elem.tag in tags:
            tags[elem.tag] += 1
        else:
            tags[elem.tag] = 1
    return tags
```

Tag Types

```
def key_type(element, keys):
    if element.tag == "tag":
        if lower.search(element.attrib['k']):
            keys['lower'] +=1
        elif lower_colon.search(element.attrib['k']):
            keys['lower_colon'] +=1
        elif problemchars.search(element.attrib['k']):
            keys['problemchars'] +=1
        else:
            keys['other'] +=1
    return keys
```

Exploring Users

```
def process_map(filename):
    users = set()
    for _, element in ET.iterparse(filename):
        if element.tag == "node" or element.tag == "way" or element.tag == "relation":
            if element.attrib['user'] not in users:
                users.add(element.attrib['user'])
    return users
```

Improving Street Names

```
mapping = { "St": "Street", "St.": "Street", "Ave":"Avenue", "Rd.":"Road"}
def update_name(name, mapping):
    fix = name.split()[-1]
    name = name.replace(fix, mapping[fix])
    return name
```

Preparing for Database

```
def shape_element(element):
    node = {}
    if element.tag == "node" or element.tag == "way" :
        node['id'], node['type'] = element.attrib['id'], element.tag
        try:
            node['visible'] =  element.attrib['visible']
        except KeyError:
            node['visible'] = 'false'
        node['created'] = {"version":element.attrib['version'], "changeset":element.attrib['changeset'],
"timestamp":element.attrib['timestamp'], "user":element.attrib['user'], "uid":element.attrib['uid']}
        try:
            node['pos'] = [float(element.attrib['lat']), float(element.attrib['lon'])]
        except KeyError:
            pass
        address = {}
        for tag in element.iter("tag"):
            if problemchars.search(tag.attrib['k']):
                pass
            elif tag.attrib['k'][0:5] == 'addr:':
                if tag.attrib['k'].count(':') <= 1:
                    if 'housenumber' in tag.attrib['k']:
                        address['housenumber'] = tag.attrib['v']
                    if 'postcode' in tag.attrib['k']:
                        address['postcode'] = tag.attrib['v']
                    if 'street' in tag.attrib['k']:
                        address['street'] = tag.attrib['v']
            else:
                node[tag.attrib['k']] = tag.attrib['v']
        if address:
            node['address'] = address

        nodes = []
        for nodess in element.iter("nd"):
            nodes.append(nodess.attrib['ref'])
        if nodes:
            node['node_refs'] = nodes
        return node
    else:
        return None
```

Final Project Code

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Code to parse Openstreetmaps (osm) data, to analyse, clean and convert to json for
uploading to mongodb for further analysis
"""

import xml.etree.cElementTree as ET
from collections import defaultdict
from pymongo import MongoClient
import re
import pprint
import codecs
import json
import pprint


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

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

expected = ['Street', 'Avenue', 'Boulevard', 'Drive', 'Road', 'Court', 'Place',
'Circuit', 'Lane', 'Parade', 'Crescent', 'Highway', 'Way', 'Close']

mapping = { "St": "Street", "St.": "Street","st": "Street", "street": "Street",
"Ave":"Avenue", "Av.":"Avenue", "Rd":"Road", "Rd.":"Road", "road":"Road",
"Hwy":"Highway", "place":"Place"}

# module to add street_type to a street_types list if not found in 'expected' list
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)

# module to list the types of tags count the instances of each type from the raw OSM
file
def audit():
    osm_file = open('sydney_australia.osm', 'r')

    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 tag.attrib['k'] == "addr:street":
                    audit_street_type(street_types, tag.attrib['v'])
    pprint.pprint(dict(street_types))

###### module to return a better name if in the mapping list, so fixed before going
into MongoDB ########
def update_name(name, mapping):
    fix = name.split()[-1]
    try:
        name = name.replace(fix, mapping[fix])
    except KeyError:
        pass
    return name

# module to fix a better name if in the mapping list
def better_name():
    for st_types, ways in street_types.iteritems():
        for name in ways:
            better_name = update_name(name, mapping)
            if better_name == name:
                print 'not fixed: ', name
            else:
                print name, "=>", better_name

# module to list the types of tags count the instances of each type from the raw OSM
file
def count_tags(filename):
    tree = ET.parse(filename)
    root = tree.getroot()
    tags = {}
    iter_ = tree.getiterator()
    for elem in iter_:
        if elem.tag in tags:
            tags[elem.tag] += 1
        else:
            tags[elem.tag] = 1
    pprint.pprint(tags)

# module to convert osm from xml format to JSON for importing into MongoDB
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

# module to pull out the desired info from the xml elements and return JSON
def shape_element(element):
    node = {}
    # filter for nodes or ways
    if element.tag == "node" or element.tag == "way" :
        node['id'], node['type'] = element.attrib['id'], element.tag
        try:
            node['visible'] =  element.attrib['visible']
        except KeyError:
            node['visible'] = 'false'
        node['created'] = {"version":element.attrib['version'],
"changeset":element.attrib['changeset'], "timestamp":element.attrib['timestamp'],
"user":element.attrib['user'], "uid":element.attrib['uid']}
        try:
            node['pos'] = [float(element.attrib['lat']),
float(element.attrib['lon'])]
        except KeyError:
            pass
        address = {}
        for tag in element.iter("tag"):
            # ignore if tag contains problem characters
            if problemchars.search(tag.attrib['k']):
                pass
            elif tag.attrib['k'][0:5] == 'addr:':
                if tag.attrib['k'].count(':') <= 1:
                    if 'housenumber' in tag.attrib['k']:
                        address['housenumber'] = tag.attrib['v']
                    if 'postcode' in tag.attrib['k']:
                        address['postcode'] = tag.attrib['v']
                    if 'street' in tag.attrib['k']:

                        ######### THIS IS WHERE THE STREET NAME IS FIXED BEFORE IT IS
ADDED TO MONGODB ###############
                        better_name = update_name(tag.attrib['v'], mapping)

#####################################################################################
#########

                        if better_name == tag.attrib['v']:
                            pass
                        else:
                            print tag.attrib['v'], "=>", better_name

                        address['street'] = better_name
            else:
                node[tag.attrib['k']] = tag.attrib['v']

        if address:
            node['address'] = address

        nodes = []
        for nodess in element.iter("nd"):
            nodes.append(nodess.attrib['ref'])
        if nodes:
            node['node_refs'] = nodes
        return node
    else:
        return None

# module to add json data in MongoDB
def add_data(db):
    with open('sydney_australia.osm.json', 'r') as f:
        for line in f:
            db.openmaps.insert(json.loads(line))
    f.close
```

```
# module to make queries on the MongoDB and also fix some of the data
def query_db(db):
    # count of nodes
    total = db.openmaps.find({'type':'node'}).count()
    print 'Total nodes = ', total

    # count of ways
    total = db.openmaps.find({'type':'way'}).count()
    print 'Total ways = ', total

    # count of nodes and ways
    total = db.openmaps.find().count()
    print 'Total nodes and ways = ', total

    # find the users with most edits
    pipeline = [{'$group': {'_id':'$created.user', 'count':{'$sum':1}}}, {'$sort':
{'count':-1}}, {"$limit":5}]
    result = db.openmaps.aggregate(pipeline)
    print 'Prolific users:'
    for i in result:
        pprint.pprint(i)

    # find the users with most edits as a % of total
    pipeline = [{'$group': {'_id':'$created.user', 'count':{'$sum':1}}}, {'$project':
{'_id':'$_id', 'percent':{'$divide':['$count', total]}}}, {'$sort': {'percent':-1}},
{"$limit":5}]
    result = db.openmaps.aggregate(pipeline)
    print '% edits:'
    for i in result:
        pprint.pprint(i)

    # sort postcodes by _id to check for high/low values to make sense
    pipeline = [{'$match':{'address.postcode':{'$exists':1}}},
{'$group':{'_id':'$address.postcode', 'count':{'$sum':1}}}, {'$sort': {'_id':-1}},
{'$limit':5}]
    result = db.openmaps.aggregate(pipeline)
    print 'top postcodes:'
    for i in result:
        pprint.pprint(i)

    # print count of types fo amenities
    print len(db.openmaps.distinct('amenity'))

    # prin tcount of distinct users that contributed
    print len(db.openmaps.distinct('created.user'))

    pipeline = [{'$match':{'amenity':{'$exists':1}}}]

    # amenities with a count of 1
    pipeline = [{'$match':{'amenity':{'$exists':1}}}, {'$group':{'_id':'$amenity',
'count':{'$sum':1}}}, {'$sort': {'count':1}}, {'$match':{'count':1}}]
    result = db.openmaps.aggregate(pipeline)
    for i in result:
        pprint.pprint(i)
    # total of number of amenities in the data set
    pipeline = [{'$match':{'amenity':{'$exists':1}}}, {'$group':{'_id':'$amenity',
'count':{'$sum':1}}}, {'$group':{'_id':'total amenity', 'count':{'$sum':'$count'}}}]
    result = db.openmaps.aggregate(pipeline)
    for i in result:
        pprint.pprint(i)

    # top aamenities
    pipeline = [{'$match':{'amenity':{'$exists':1}}}, {'$group':{'_id':'$amenity',
'count':{'$sum':1}}}, {'$sort': {'count':-1}}, {'$limit':10}]
    result = db.openmaps.aggregate(pipeline)
    print 'top amenities:'
    for i in result:
        pprint.pprint(i)

    # top postcode for schools
    pipeline = [{'$match':{'amenity':{'$exists':1}, 'amenity':'school'}},
{'$group':{'_id':'$address.postcode', 'count':{'$sum':1}}}, {'$sort': {'count':-1}},
{'$limit':10}]
    result = db.openmaps.aggregate(pipeline)
    print 'pubs:'
    for i in result:
        pprint.pprint(i)

    # top postcode for pubs
    pipeline = [{'$match':{'amenity':{'$exists':1}, 'amenity':'pub'}},
{'$group':{'_id':'$address.postcode', 'count':{'$sum':1}}}, {'$sort': {'count':-1}},
{'$limit':10}]
    result = db.openmaps.aggregate(pipeline)
    print 'pubs:'
    for i in result:
        pprint.pprint(i)

# module to fix the postcodes and amenities in MongoDB
def fix_db(db):
    # fix postcodes with NSW in them
    result = db.openmaps.find({"address.postcode": {"$regex": "NSW "}})
    for i in result:
        pprint.pprint(i['address']['postcode'][-4:])
        i['address']['postcode'] = i['address']['postcode'][-4:]
        db.openmaps.save(i)

    # fix amenities typo 'scol' to 'school'
    result = db.openmaps.find({"amenity": {"$regex": "scol"}})
    for i in result:
        pprint.pprint(i)
        i['amenity'] = 'school'
        db.openmaps.save(i)

    # fix amenities typo '+' to ' '
    result = db.openmaps.find({"amenity": {"$regex": "+"}})
    for i in result:
        pprint.pprint(i)
        i['amenity'] = i['amenity'].replace('+', ' ')
        db.openmaps.save(i)

def get_db():
    from pymongo import MongoClient
    client = MongoClient('localhost:27017')
    db = client['openmaps']
    return db

if __name__ == "__main__":
    # list the types of tags count the instances of each type
    count_tags('sydney_australia.osm')

    # print a list of street types not in the 'expected' types list
    audit()

    # module to fix street_type name
    better_name()

    # module to convert OSM xml file to JSON for MongoDB
    process_map('sydney_australia.osm', False)

    # modules to create MondoDB
    db = get_db()
    add_data(db)

    # module to query MongoDB and make changes to fix data
    query_db(db)
    fix_db(db)
```