OpenStreetMap Data Case Study

Jungmin Park (Jungmin.j.park@gmail.com)

1. Map Area

Boston, United States (https://mapzen.com/data/metro-extracts/metro/boston_massachusetts/)
The OSM XMS file is 414 MB.
I chose this area since I want to visit this area in a few months for a tour.

2. Parsing the OSM file.

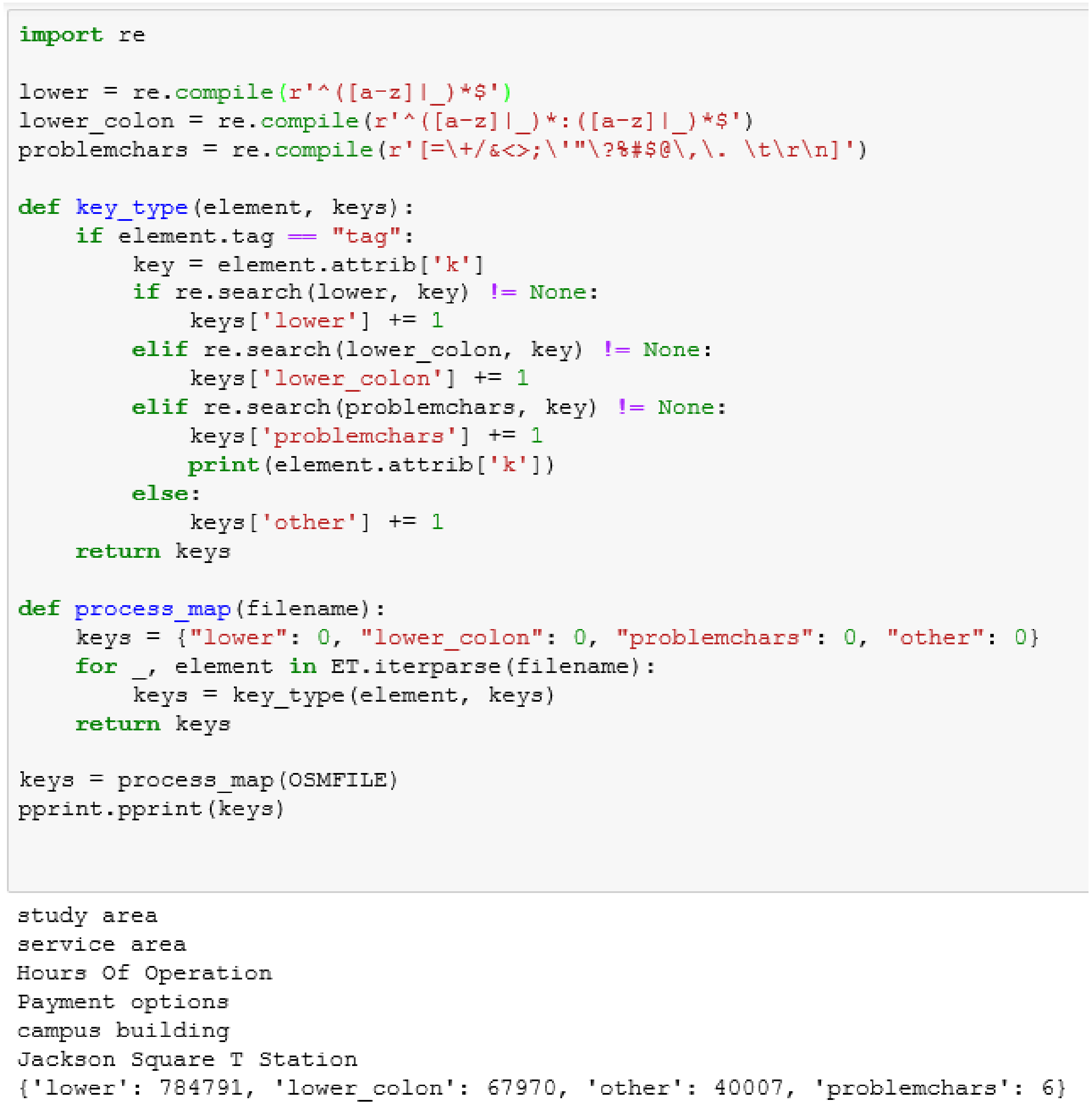
As you can see from the following result, I counted each number of the tag.
'bounds': 1, 'member': 10763, 'nd': 2327487, 'node': 1932903, 'osm': 1, 'relation': 1241, 'tag': 892774, 'way': 308653

```
import os
import pprint
import xml.etree.ElementTree as ET


filename = "boston_massachusetts.osm" # osm filename
path = "C:\Users\Jungmin\Downloads" # directory contain the osm file
OSMFILE = os.path.join(path, filename)

# iterative parsing
def count_tags(filename):
    tags = {}
    for event, elem in ET.iterparse(filename):
        if elem.tag not in tags:
            tags[elem.tag] = 1
        else:
            tags[elem.tag] += 1
    return tags


tags = count_tags(OSMFILE)
pprint.pprint(tags)
```

```
{'bounds': 1,
 'member': 10763,
 'nd': 2327487,
 'node': 1932903,
 'osm': 1,
 'relation': 1241,
 'tag': 892774,
 'way': 308653}
```

3. Exploring the Data

I checked the "k" value for each "tag" and saw if there were any potential problems.
As you can see from the following result, problem characters are not really problems.



```python
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":
        key = element.attrib['k']
        if re.search(lower, key) != None:
            keys['lower'] += 1
        elif re.search(lower_colon, key) != None:
            keys['lower_colon'] += 1
        elif re.search(problemchars, key) != None:
            keys['problemchars'] += 1
            print(element.attrib['k'])
        else:
            keys['other'] += 1
    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

keys = process_map(OSMFILE)
pprint.pprint(keys)
```

```
study area
service area
Hours Of Operation
Payment options
campus building
Jackson Square T Station
{'lower': 784791, 'lower_colon': 67970, 'other': 40007, 'problemchars': 6}
```

4. Sample Data and Problems Encountered in the Map

- Unnecessary abbreviated street names such as 'Brentwood St'
- Incorrect postal codes (Boston area zip codes all begin with “02” however some portion of zip codes were outside this region. The length of zip codes should be 5.)

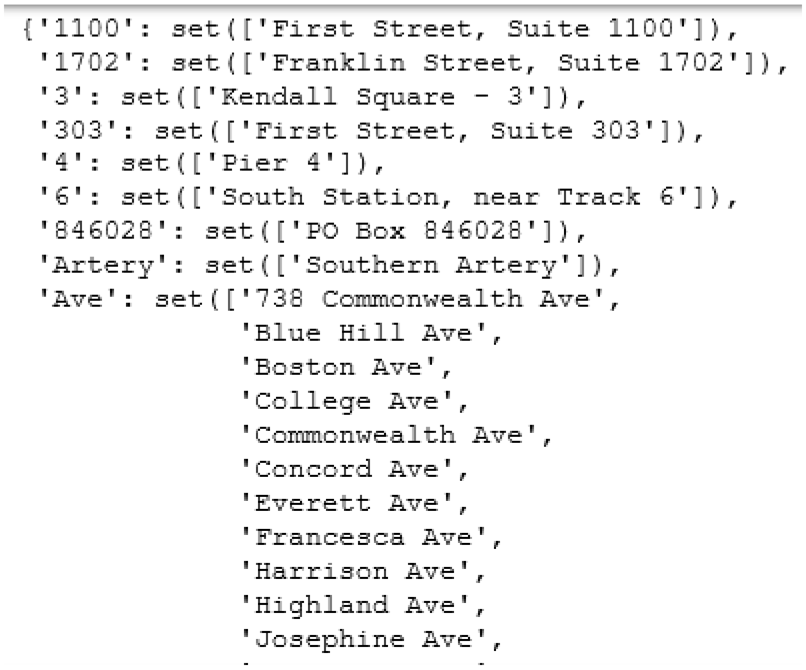
```
from collections import defaultdict

TAG_KEYS = ['addr:street', 'addr:postcode']
item_limit = 100

def examine_tags(osmfile, item_limit):
    print "Examining tag keys: {}".format(TAG_KEYS)
    osm_file = open(osmfile, "r")

    # initialize data with default set data structure
    data = defaultdict(set)

    # iterate through elements
    for _, elem in ET.iterparse(osm_file, events=("start",)):
        # check if the element is a node or way
        if elem.tag == "node" or elem.tag == "way":
            # iterate through children matching `tag`
            for tag in elem.iter("tag"):
                # skip if does not contain key-value pair
                if 'k' not in tag.attrib or 'v' not in tag.attrib:
                    continue
                key = tag.get('k')
                val = tag.get('v')
                # add to set if in tag keys of interest and is below the item limit
                if key in TAG_KEYS and len(data[key]) < item_limit:
                    data[key].add(val)
    return data

tag_data = dict(examine_tags(OSMFILE, item_limit))
pprint.pprint(tag_data)
```

```
Examining tag keys: ['addr:street', 'addr:postcode']
{'addr:postcode': set(['01125',
                       '01238',
                       '01240',
                       '01854',
                       '01944',
                       '02026',
                       '02026-5036',
                       '02108',
                       '02109',
                       '02110',
                       '02110-1301',
                       '02111',
                       '02113',
                       '02114',
                       '02114-3203',
                       '02115',
                       '02116',
                       '02118',
'addr:street': set(['1629 Cambridge St',
                    '61 Union Square',
                    'Agawam Road',
                    'Albion Street',
                    'Antwerp St',
                    'Arlington Street',
                    'Ashburton Place',
                    'Ashley Street',
                    'Athol St',
                    'Avenue Louis Pasteur',
                    'Bagnal St',
                    'Beacon Street',
                    'Belmont Street',
                    'Berkeley Street',
                    'Blossom Street',
                    'Boston street',
                    'Bow Street',
                    'Brattle Street',
                    'Brentwood St',
```

5. Auditing Street Type

I looked at the street names, and printed out all the street names that is with a unexpected street type.

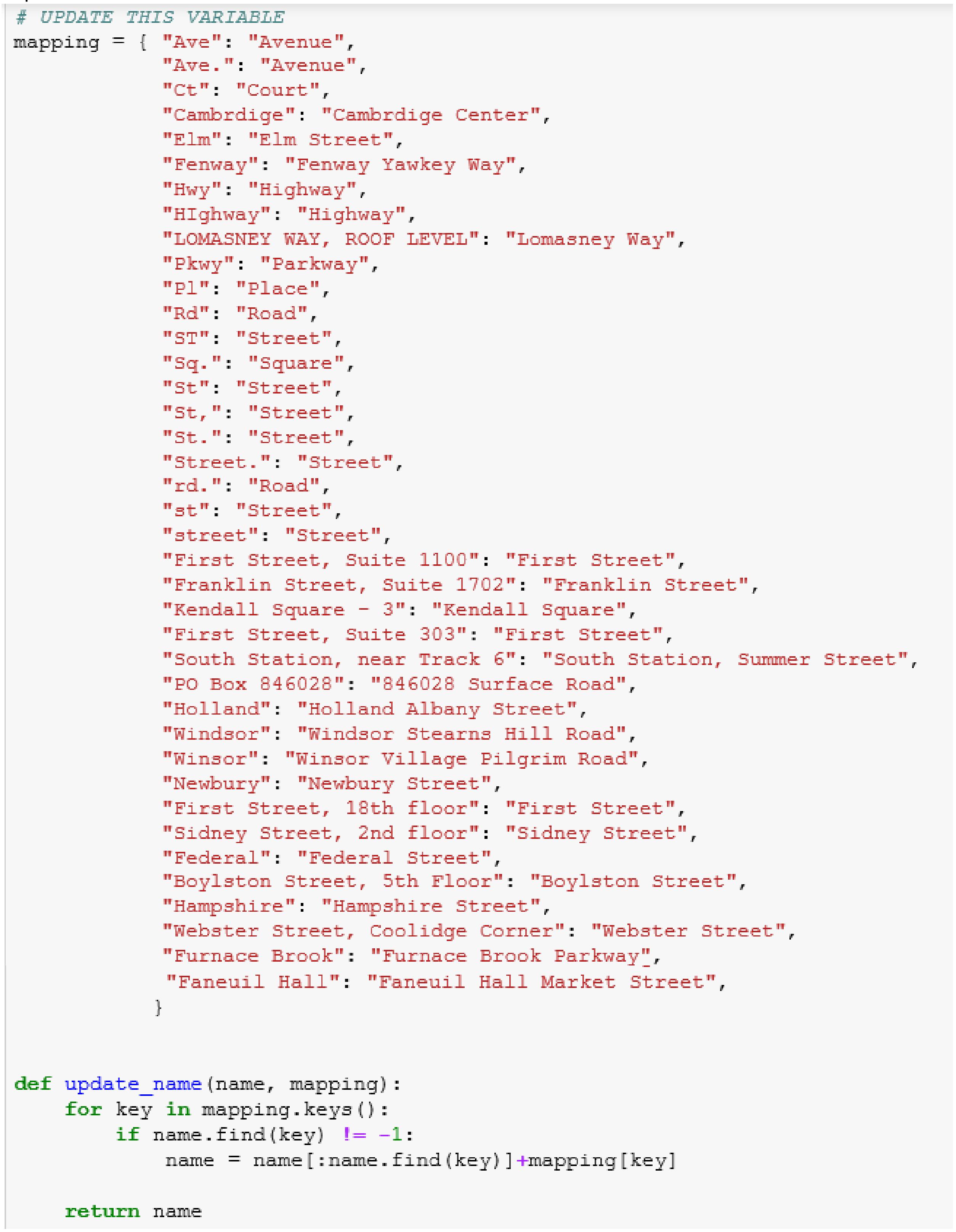
```
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)

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

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

st_types = audit(OSMFILE)
pprint.pprint(dict(st_types))
```



```
{'1100': set(['First Street, Suite 1100']),
 '1702': set(['Franklin Street, Suite 1702']),
 '3': set(['Kendall Square - 3']),
 '303': set(['First Street, Suite 303']),
 '4': set(['Pier 4']),
 '6': set(['South Station, near Track 6']),
 '846028': set(['PO Box 846028']),
 'Artery': set(['Southern Artery']),
 'Ave': set(['738 Commonwealth Ave',
             'Blue Hill Ave',
             'Boston Ave',
             'College Ave',
             'Commonwealth Ave',
             'Concord Ave',
             'Everett Ave',
             'Francesca Ave',
             'Harrison Ave',
             'Highland Ave',
             'Josephine Ave',
```

6. Updating Street Name

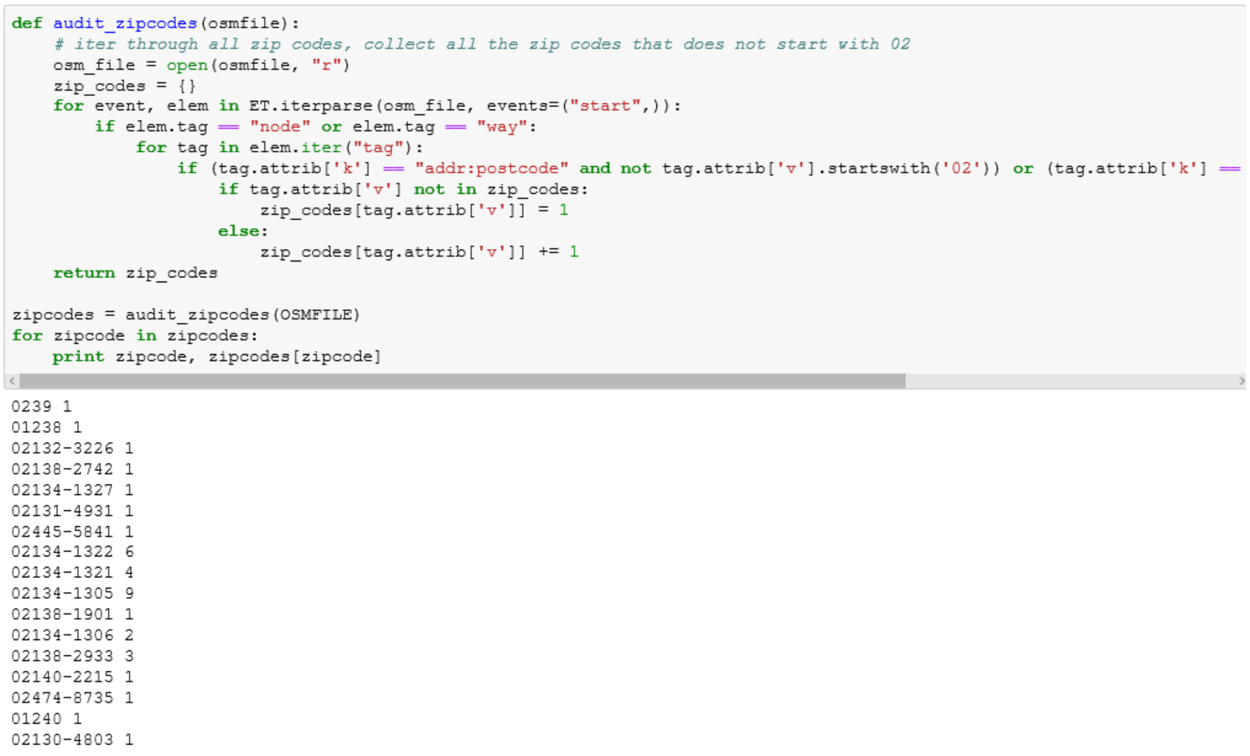
I updated street names.



```
# UPDATE THIS VARIABLE
mapping = { "Ave": "Avenue",
            "Ave.": "Avenue",
            "Ct": "Court",
            "Cambrdige": "Cambrdige Center",
            "Elm": "Elm Street",
            "Fenway": "Fenway Yawkey Way",
            "Hwy": "Highway",
            "HIghway": "Highway",
            "LOMASNEY WAY, ROOF LEVEL": "Lomasney Way",
            "Pkwy": "Parkway",
            "Pl": "Place",
            "Rd": "Road",
            "ST": "Street",
            "Sq.": "Square",
            "St": "Street",
            "St,": "Street",
            "St.": "Street",
            "Street.": "Street",
            "rd.": "Road",
            "st": "Street",
            "street": "Street",
            "First Street, Suite 1100": "First Street",
            "Franklin Street, Suite 1702": "Franklin Street",
            "Kendall Square - 3": "Kendall Square",
            "First Street, Suite 303": "First Street",
            "South Station, near Track 6": "South Station, Summer Street",
            "PO Box 846028": "846028 Surface Road",
            "Holland": "Holland Albany Street",
            "Windsor": "Windsor Stearns Hill Road",
            "Winsor": "Winsor Village Pilgrim Road",
            "Newbury": "Newbury Street",
            "First Street, 18th floor": "First Street",
            "Sidney Street, 2nd floor": "Sidney Street",
            "Federal": "Federal Street",
            "Boylston Street, 5th Floor": "Boylston Street",
            "Hampshire": "Hampshire Street",
            "Webster Street, Coolidge Corner": "Webster Street",
            "Furnace Brook": "Furnace Brook Parkway",
             "Faneuil Hall": "Faneuil Hall Market Street",
           }


def update_name(name, mapping):
    for key in mapping.keys():
        if name.find(key) != -1:
            name = name[:name.find(key)]+mapping[key]

    return name
```

7. Auditing Zip Codes

I looked at the Zip codes, and printed out all the Zip codes that are with an unexpected Zip code type.



```
def audit_zipcodes(osmfile):
    # iter through all zip codes, collect all the zip codes that does not start with 02
    osm_file = open(osmfile, "r")
    zip_codes = {}
    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:postcode" and not tag.attrib['v'].startswith('02')) or (tag.attrib['k'] ==
                    if tag.attrib['v'] not in zip_codes:
                        zip_codes[tag.attrib['v']] = 1
                    else:
                        zip_codes[tag.attrib['v']] += 1
    return zip_codes

zipcodes = audit_zipcodes(OSMFILE)
for zipcode in zipcodes:
    print zipcode, zipcodes[zipcode]
```

```
0239 1
01238 1
02132-3226 1
02138-2742 1
02134-1327 1
02131-4931 1
02445-5841 1
02134-1322 6
02134-1321 4
02134-1305 9
02138-1901 1
02134-1306 2
02138-2933 3
02140-2215 1
02474-8735 1
01240 1
02130-4803 1
```

8. Updating Zip Codes

The mistake zip codes not starting with "02" are only 10 in our dataset, so I changed those zip codes into '00000' to distinguish from other zip codes when I need to analyze dataset. Also, I changed the zip codes whose length were over 5.

```
# UPDATE THIS VARIABLE
mapping_zip = { "0239": "00000",
                "01238": "00000",
                "02132-3226": "02132",
                "02138-2742": "02138",
                "02134-1327": "02134",
                "02131-4931": "02131",
                "02445-5841": "02445",
                "02134-1322": "02134",
                "02134-1321": "02134",
                "02134-1305": "02134",
                "02138-1901": "02138",
                "02134-1306": "02134",
                "02138-2933": "02138",
                "02140-2215": "02140",
                "02474-8735": "02474",
                "01240": "00000",
                "02130-4803": "02130",
                "02114-3203": "02114",
                "02134-1316": "02134",
                "02134-1318": "02134",
                "MA 02118": "02118",
                "MA 02116": "02118",
                "01944": "00000",
                "01125": "00000",
                "02138-2736": "02138",
                "02134-1433": "02134",
                "Mass Ave": "00000",
                "02110-1301": "02110",
                "MA 02186": "02186",
                "02138-2903": "02138",
                "02134-1409": "02134",
                "02138-3003": "02138",
                "02138-2735": "02138",
                "02132-1239": "02132",
                "01250": "00000",
                "02134-1420": "02134",
                "02134-1307": "02134",
                "02138-2701": "02138",
                "02445-7638": "02445",
                "02134-1319": "02134",
                "02138-2762": "02138",
                "02138-2763": "02138",
                "02138-2706": "02138",
                "02026-5036": "02026",
                "02138-2724": "02138",
               }

def update_zipcode(zipcode, mapping_zip):
    for key in mapping_zip.keys():
        if zipcode.find(key) != -1:
            zipcode = zipcode[:zipcode.find(key)]+mapping_zip[key]
    return zipcode
```

9. Inserting into database

I tried to insert data into database and also updated what I want to correct such as street type and zip codes.

```
import codecs
import json
from pymongo import MongoClient


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

def shape_element(element):
    node = {}
    node["created"]={}
    node["address"]={}
    node["pos"]=[]
    refs=[]

    if element.tag == "node" or element.tag == "way" :
        if "id" in element.attrib:
            node["id"]=element.attrib["id"]
        node["type"]=element.tag

        if "visible" in element.attrib.keys():
            node["visible"]=element.attrib["visible"]

        for elem in CREATED:
            if elem in element.attrib:
                node["created"][elem]=element.attrib[elem]
        if "lat" in element.attrib:
            node["pos"].append(float(element.attrib["lat"]))
        if "lon" in element.attrib:
            node["pos"].append(float(element.attrib["lon"]))

        for tag in element.iter("tag"):
            if not(problemchars.search(tag.attrib['k'])):
                if tag.attrib['k'] == "addr:housenumber":
                    node["address"]["housenumber"]=tag.attrib['v']
                if tag.attrib['k'] == "addr:postcode":
                    node["address"]["postcode"]=tag.attrib['v']
                    node["address"]["postcode"] = update_zipcode(node["address"]["postcode"], mapping_zip)
                if tag.attrib['k'] == "addr:street":
                    node["address"]["street"]=tag.attrib['v']
                    node["address"]["street"] = update_name(node["address"]["street"], mapping)
                if tag.attrib['k'].find("addr")==-1:
                    node[tag.attrib['k']]=tag.attrib['v']
        for nd in element.iter("nd"):
             refs.append(nd.attrib["ref"])
        if node["address"] =={}:
            node.pop("address", None)
        if refs != []:
           node["node_refs"]=refs
        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 the file
data = process_map(OSMFILE, True)
```

```
client = MongoClient()
db = client.Boston
collection = db.bostonMAP
collection.insert(data)
```

10. Overview statistics

```
# size of the original xml file
os.path.getsize(OSMFILE)/1024/1024
```

```
414L
```

```
# size of the processed json file
os.path.getsize(os.path.join(path, "boston_massachusetts.osm.json"))/1024/1024
```

```
640L
```

```
# Number of unique users
len(collection.group(["created.uid"], {}, {"count":0}, "function(o, p){p.count++}"))
```

```
1187
```

```
# Number of nodes
collection.find({"type":"node"}).count()
```

```
1932545
```

```
# Number of ways
collection.find({"type":"way"}).count()
```

```
308568
```

11. Additional exploration

Top three users contribute most of dataset. It is supposed that they were forced to insert the initial dataset. Especially, 'crschmidt' contributed approaximately 54% of map.

```
# Top three users with most contributions

pipeline = [{"$group":{"_id": "$created.user",
                       "count": {"$sum": 1}}},
            {"$sort": {"count": -1}},
            {"$limit": 3}]
result = collection.aggregate(pipeline)

for x in xrange(3):
    get_record = result.next()
    print get_record
```

```
{u'count': 1204304, u'_id': u'crschmidt'}
{u'count': 430624, u'_id': u'jremillard-massgis'}
{u'count': 92178, u'_id': u'OceanVortex'}
```

```
# Proportion of the top three users' contributions
pipeline = [{"$group":{"_id": "$created.user",
                       "count": {"$sum": 1}}},
            {"$project": {"proportion": {"$divide" :["$count",collection.find().count()]}}},
            {"$sort": {"proportion": -1}},
            {"$limit": 3}]

result = collection.aggregate(pipeline)

for x in xrange(3):
    get_record = result.next()
    print get_record
```

```
{u'_id': u'crschmidt', u'proportion': 0.5372625087216202}
{u'_id': u'jremillard-massgis', u'proportion': 0.19210940971360965}
{u'_id': u'OceanVortex', u'proportion': 0.041122327526057795}
```

12. Conclusion

Because the open street map dataset is a human edited dataset, there were some errors as we expected. I tried to update data but actually the decision to handle with wrong data has to be changed depending on how to analyze it next. The best way is finding the true data and updating the wrong data into the true data, but it is not easy and sometimes it is waste of time.
Also, I think if 'OpenStreetMap.org' offer the input screen to constrain what we can put the correct data such as 5 digit Zipcode, the dataset could be more precise.