Wrangle OpenStreetMap Data

October 18, 2015

- Author: Ren Zhang
- email: zhang ren@bentley.edu

0.1 References

- I am using the OpenStreetMap data of Boston, MA area downloaded from mapzen, the date I downloaded the data set is sept. 18 2015.
- The data file is in XML format, and here is a description of the OpenStreetMap XML format.

0.2 Problems Encountered in the Map

1. There are some incorrect street names and also some abbreviated street thpes
2. There are a few incorrect zip codes as well

0.2.1 Street Names

Some street names are abbreviated, such as "Adams St", I updated these abbreviations to better names. Some street names are incorrect, such as "Boylston Street, 5th Floor", I also corrected them as well. See the code and results part for details.

0.2.2 Zip Codes

The zip code of Boston area should starts with *02*. I noticed there are a couple zip codes not belonging to Boston area, such as zip code start with *012* but rather are zip codes of places near Boston. Luckly these incorrect cases are very rare. See the code and results part for details.

0.3 Data Overview

Here we provide some overview of the dataset

0.3.1 Data file size

- boston_massachusetts.osm(The original downloaded OpenStreetMap in xml format): 401MB
- boston_massachusetts.osm.json(The processed OpenStreetMap in json format): 622MB

0.3.2 Summary statistics of the dataset

- Number of documents: 2180736
- Number of unique users: 1007
- Number of nodes: 1886049
- Number of ways: 294201

0.3.3 Some other interesting findings

- Number of methods be used to create data entry: 26

The queries used to generate these results are in the code.

0.4 Other ideas about the datasets

I looked at users contributed to boston OpenStreetMap most frequently. The top three users are: + crschmidt, who contributed more than 56.44% + jremillard-massgis, with 20.12% contributions + Ocean-Vortex, with only 4.13% constibutions but still rank the third

After a bit googling, I think I find the crschmidt who ranked the first here.

I also looked at the top cuisines: 1. pizza 2. american 3. chinese 4. italian 5. mexican 6. indian 7. thai 8. sandwich 9. japanese 10. asian

Boston is famous for the many universities in the city, We also took a look at that as well. The results are also included in the code section

0.5 Code and results

```
In [1]: # load libraries
        import os
        import xml.etree.cElementTree as cET
        from collections import defaultdict
        import pprint
        import re
        import codecs
        import json
        import string
        from pymongo import MongoClient
```

```
In [2]: # set up map file path
        filename = "boston_massachusetts.osm" # osm filename
        path = "d:\GithubRepos\Udacity\P3" # directory contain the osm file
        bostonOSM = os.path.join(path, filename)

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

        # initial version of expected street names
        expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "Lane", "Road
```

0.5.1 Audit the Street Names

Look at the street names, print out all the street names that is with a unexpected street type

```
In [3]: def audit_street_type(street_types, street_name):
            # add unexpected street name to a list
            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):
            # determine whether a element is a street name
            return (elem.attrib['k'] == "addr:street")

        def audit_street(osmfile):
            # iter through all street name tag under node or way and audit the street name value
            osm_file = open(osmfile, "r")
            street_types = defaultdict(set)
            for event, elem in cET.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'])
            return street_types

        st_types = audit_street(bostonOSM)
        # print out unexpected street names
        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']),
 '6': set(['South Station, near Track 6']),
 '846028': set(['PO Box 846028']),
 'Artery': set(['Southern Artery']),
 'Ave': set(['360 Huntington Ave',
             '738 Commonwealth Ave',
             'Blue Hill Ave',
             'Boston Ave',
             'College Ave',
             'Commonwealth Ave',
             'Concord Ave',
             'Francesca Ave',
             'Harrison Ave',
             'Highland Ave',
             'Huntington Ave',
             'Josephine Ave',
             'Lexington Ave',
             'Massachusetts Ave',
             'Morrison Ave',
             'Mystic Ave',
             'Somerville Ave',
             "St. Paul's Ave",
             'Washington Ave',
             'Western Ave',
             'Willow Ave']),
 'Ave.': set(['Brighton Ave.',
              'Huntington Ave.',
              'Massachusetts Ave.',
              'Somerville Ave.',
              'Spaulding Ave.']),
 'Boylston': set(['Boylston']),
 'Broadway': set(['Broadway', 'West Broadway']),
```

```
'Brook': set(['Furnace Brook']),
'Cambrdige': set(['Cambrdige']),
'Center': set(['Cambridge Center']),
'Circle': set(['Edgewood Circle', 'Stein Circle']),
'Corner': set(['Webster Street, Coolidge Corner']),
'Ct': set(['Kelley Ct']),
'Elm': set(['Elm']),
'Federal': set(['Federal']),
'Fellsway': set(['Fellsway']),
'Fenway': set(['Fenway']),
'Floor': set(['Boylston Street, 5th Floor']),
'HIghway': set(['American Legion HIghway']),
'Hall': set(['Faneuil Hall']),
'Hampshire': set(['Hampshire']),
'Highway': set(['American Legion Highway',
                'Cummins Highway',
                "Monsignor O'Brien Highway",
                'Providence Highway',
                'Santilli Highway']),
'Holland': set(['Holland']),
'Hwy': set(["Monsignor O'Brien Hwy"]),
'LEVEL': set(['LOMASNEY WAY, ROOF LEVEL']),
'Lafayette': set(['Avenue De Lafayette']),
'Mall': set(['Cummington Mall']),
'Newbury': set(['Newbury']),
'Park': set(['Austin Park',
             'Batterymarch Park',
             'Canal Park',
             'Exeter Park',
             'Giles Park',
             'Malden Street Park',
             'Monument Park']),
'Pkwy': set(['Birmingham Pkwy']),
'Pl': set(['Longfellow Pl']),
'Rd': set(['Abby Rd',
           'Aberdeen Rd',
           'Bristol Rd',
           'Goodnough Rd',
           'Oakland Rd',
           'Soldiers Field Rd',
           'Squanto Rd']),
'Row': set(['Assembly Row', 'East India Row', 'Professors Row']),
'ST': set(['Newton ST']),
'Sedgwick': set(['Sedgwick']),
'South': set(['Charles Street South']),
'Sq.': set(['1 Kendall Sq.']),
'St': set(['1629 Cambridge St',
           '644 Beacon St',
           'Adams St',
           'Antwerp St',
           'Arsenal St',
           'Athol St',
           'Bagnal St',
           'Beacon St',
```

```
            'Brentwood St',
            'Broad St',
            'Cambridge St',
            'Congress St',
            'Court St',
            'Cummington St',
            'Dane St',
            'Duval St',
            'E 4th St',
            'Elm St',
            'Everett St',
            'George St',
            'Grove St',
            'Hampshire St',
            'Holton St',
            'Kirkland St',
            'Leighton St',
            'Litchfield St',
            'Lothrop St',
            'Mackin St',
            'Maverick St',
            'Medford St',
            'Merrill St',
            'Newbury St',
            'Norfolk St',
            'Portsmouth St',
            'Richardson St',
            'Salem St',
            'Sea St',
            'South Waverly St',
            'Stewart St',
            'Summer St',
            'Ware St',
            'Waverly St',
            'Winter St']),
 'St,': set(['Walnut St,']),
 'St.': set(['Albion St.',
             'Boylston St.',
             'Brookline St.',
             'Centre St.',
             'Elm St.',
             'Main St.',
             'Marshall St.',
             'Maverick St.',
             'Pearl St.',
             'Prospect St.',
             "Saint Mary's St.",
             'Stuart St.',
             'Tremont St.']),
 'Street.': set(['Hancock Street.']),
 'Terrace': set(['Alberta Terrace', 'Norfolk Terrace', 'Westbourne Terrace']),
 'Turnpike': set(['Boston Providence Turnpike']),
 'Way': set(['Artisan Way',
             'Courthouse Way',
```

```
            'David G Mugar Way',
            'Davidson Way',
            'Evans Way',
            'Harry Agganis Way',
            'Ross Way',
            'Yawkey Way']),
 'Wharf': set(['Long Wharf', 'Rowes Wharf']),
 'Windsor': set(['Windsor']),
 'Winsor': set(['Winsor']),
 'ave': set(['Massachusetts ave']),
 'floor': set(['First Street, 18th floor', 'Sidney Street, 2nd floor']),
 'place': set(['argus place']),
 'rd.': set(['Corey rd.']),
 'st': set(['Church st']),
 'street': set(['Boston street'])}
```

0.5.2 Street Name Correction Strategy

Based on the auditing results, I came up with the following mapping dictionary, which addressed the abbrivations and the incorrect names.

```
In [4]: # creating a dictionary for correcting street names
        mapping = { "Ct": "Court",
                    "St": "Street",
                    "st": "Street",
                    "St.": "Street",
                    "St,": "Street",
                    "ST": "Street",
                    "street": "Street",
                    "Street.": "Street",
                    "Ave": "Avenue",
                    "Ave.": "Avenue",
                    "ave": "Avenue",
                    "Rd.": "Road",
                    "rd.": "Road",
                    "Rd": "Road",
                    "Hwy": "Highway",
                    "HIghway": "Highway",
                    "Pkwy": "Parkway",
                    "Pl": "Place",
                    "place": "Place",
                    "Sedgwick": "Sedgwick Street",
                    "Sq.": "Square",
                    "Newbury": "Newbury Street",
                    "Boylston": "Boylston Street",
                    "Brook": "Brook Parkway",
                    "Cambrdige": "Cambrdige Center",
                    "Elm": "Elm Street",
                    "Webster Street, Coolidge Corner": "Webster Street",
                    "Faneuil Hall": "Faneuil Hall Market Street",
                    "Furnace Brook": "Furnace Brook Parkway",
                    "Federal": "Federal Street",
                    "South Station, near Track 6": "South Station, Summer Street",
                    "PO Box 846028": "846028 Surface Road",
                    "First Street, Suite 303": "First Street",
```

```
                    "Kendall Square - 3": "Kendall Square",
                    "Franklin Street, Suite 1702": "Franklin Street",
                    "First Street, Suite 1100": "First Street",
                    "Windsor": "Windsor Stearns Hill Road",
                    "Winsor": "Winsor Village Pilgrim Road",
                    "First Street, 18th floor": "First Street",
                    "Sidney Street, 2nd floor": "Sidney Street",
                    "Boston Providence Turnpike": "Boston Providence Highway",
                    "LOMASNEY WAY, ROOF LEVEL": "Lomasney Way",
                    "Holland": "Holland Albany Street",
                    "Hampshire": "Hampshire Street",
                    "Boylston Street, 5th Floor": "Boylston Street",
                    "Fenway": "Fenway Yawkey Way",
                    "Charles Street South": "Charles Street"}

        # function that corrects incorrect street names
        def update_name(name, mapping):
            for key in mapping:
                if key in name:
                    name = string.replace(name,key,mapping[key])
            return name
```

0.5.3 Audit the Zip Codes

The zip codes in boston should start with *02*, audit incorrect zip codes in the data set

```
In [11]: 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 cET.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')
                             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(bostonOSM)
         for zipcode in zipcodes:
             print zipcode, zipcodes[zipcode]
```

```
01240 1
Mass Ave 1
MA 5
01250 1
01821 1
MA 02116 4
01944 1
01125 1
01238 1
MA 02186 1
```

The mistake zip codes are not that many in our dataset, and some of them are zip code of places close to Boston in Massachusetts.

0.5.4 Process the OpenStreetMap XML file

The function to process the XML file, make ready for insert into MongoDB

In [13]:
```
CREATED = [ "version", "changeset", "timestamp", "user", "uid"]

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

            # we only process the node and way tags
            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"]

                # the key-value pairs with attributes in the CREATED list are added under key "created
                for elem in CREATED:
                    if elem in element.attrib:
                        node["created"][elem]=element.attrib[elem]

                # attributes for latitude and longitude are added to a "pos" array
                # include latitude value
                if "lat" in element.attrib:
                    node["pos"].append(float(element.attrib["lat"]))
                # include longitude value
                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']

                        # handling the street attribute, update incorrect names using the strategy dev
                        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

        # process the xml openstreetmap file, write a json out file and return a list of dictionaries
        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 cET.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
```

In [14]:
```
# process the file
data = process_map(bostonOSM, True)
```

0.5.5 Insert the data into local MongoDB Database

In [ ]:
```
client = MongoClient()
db = client.BostonOSM
collection = db.bostonMAP
collection.insert(data)
```

In [16]:
```
collection
```

Out[16]:
```
Collection(Database(MongoClient('localhost', 27017), u'BostonOSM'), u'bostonMAP')
```

0.5.6 Summary Statistics of Data

size of the original xml file

In [23]:
```
os.path.getsize(bostonOSM)/1024/1024
```

Out[23]:
```
401L
```

size of the processed json file

In [24]:
```
os.path.getsize(os.path.join(path, "boston_massachusetts.osm.json"))/1024/1024
```

Out[24]:
```
622L
```

The Number of documents

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

```
Out[17]: 2180736
```

The Number of Unique users

```
In [53]: # Number of unique users
         len(collection.group(["created.uid"], {}, {"count":0}, "function(o, p){p.count++}"))
```

```
Out[53]: 1007
```

The Number of Nodes

```
In [51]: # Number of nodes
         collection.find({"type":"node"}).count()
```

```
Out[51]: 1886049
```

The Number of Ways

```
In [52]: # Number of ways
         collection.find({"type":"way"}).count()
```

```
Out[52]: 294201
```

The Number of Methods Used to Create Data Entry

```
In [31]: pipeline = [{"$group":{"_id": "$created_by",
                                "count": {"$sum": 1}}}]
         result = collection.aggregate(pipeline)
         print(len(result['result']))
```

```
26
```

0.5.7 Some more exploration on the data set

Top three users with most contributions

```
In [32]: pipeline = [{"$group":{"_id": "$created.user",
                                "count": {"$sum": 1}}},
                     {"$sort": {"count": -1}},
                     {"$limit": 3}]
         result = collection.aggregate(pipeline)
         result['result']
```

```
Out[32]: [{u'_id': u'crschmidt', u'count': 1230914},
          {u'_id': u'jremillard-massgis', u'count': 438724},
          {u'_id': u'OceanVortex', u'count': 90091}]
```

Proportion of the top user contributions

```
In [33]: pipeline = [{"$group":{"_id": "$created.user",
                                "count": {"$sum": 1}}},
                     {"$project": {"proportion": {"$divide" :["$count",collection.find().count()]}}},
                     {"$sort": {"proportion": -1}},
                     {"$limit": 3}]
         result = collection.aggregate(pipeline)
         result['result']
```

```
Out[33]: [{u'_id': u'crschmidt', u'proportion': 0.5644488833127899},
          {u'_id': u'jremillard-massgis', u'proportion': 0.20118161941656396},
          {u'_id': u'OceanVortex', u'proportion': 0.041312199184128665}]
```

Most popular cuisines

```
In [36]: pipeline = [{"$match":{"amenity":{"$exists":1}, "amenity":"restaurant", "cuisine":{"$exists":1
                     {"$group":{"_id":"$cuisine", "count":{"$sum":1}}},
                     {"$sort":{"count":-1}},
                     {"$limit":10}]
         result = collection.aggregate(pipeline)
         result['result']
```

```
Out[36]: [{u'_id': u'pizza', u'count': 36},
          {u'_id': u'american', u'count': 33},
          {u'_id': u'chinese', u'count': 31},
          {u'_id': u'italian', u'count': 26},
          {u'_id': u'mexican', u'count': 26},
          {u'_id': u'indian', u'count': 20},
          {u'_id': u'thai', u'count': 18},
          {u'_id': u'sandwich', u'count': 12},
          {u'_id': u'japanese', u'count': 12},
          {u'_id': u'asian', u'count': 11}]
```

Universities

```
In [41]: pipeline = [{"$match":{"amenity":{"$exists":1}, "amenity": "university", "name":{"$exists":1}}
                     {"$group":{"_id":"$name", "count":{"$sum":1}}},
                     {"$sort":{"count":-1}}]
         result = collection.aggregate(pipeline)
         result['result']
```

```
Out[41]: [{u'_id': u'Boston University', u'count': 41},
          {u'_id': u'Massachusetts Institute of Technology', u'count': 10},
          {u'_id': u'Suffolk University', u'count': 8},
          {u'_id': u'Harvard University', u'count': 6},
          {u'_id': u'Boston University Medical Campus', u'count': 3},
          {u'_id': u'University of Massachusetts Boston', u'count': 3},
          {u'_id': u'Harvard Medical School', u'count': 2},
          {u'_id': u'Northeastern University', u'count': 2},
          {u'_id': u'Benjamin Franklin Institute of Technology', u'count': 2},
          {u'_id': u'University Hall', u'count': 2},
          {u'_id': u'Littauer Center', u'count': 2},
          {u'_id': u'Boston College', u'count': 2},
          {u'_id': u'Harvard School of Public Health', u'count': 1},
          {u'_id': u'Boston College - Newton Campus', u'count': 1},
          {u'_id': u'Harvard University Northwest Building', u'count': 1},
          {u'_id': u'Boston College (Brighton Campus)', u'count': 1},
          {u'_id': u'David Rubenstein Building', u'count': 1},
          {u'_id': u'White Hall', u'count': 1},
          {u'_id': u'Mower Hall', u'count': 1},
          {u'_id': u'Andover Newton Theological School', u'count': 1},
          {u'_id': u'Westmorely Hall', u'count': 1},
          {u'_id': u'Robinson Hall', u'count': 1},
          {u'_id': u'Eliot House', u'count': 1},
```

```
{u'_id': u'Conant Hall', u'count': 1},
{u'_id': u'Engineering Science Lab', u'count': 1},
{u'_id': u'Malkin Athletic Center (MAC)', u'count': 1},
{u'_id': u'Hastings Building', u'count': 1},
{u'_id': u'Harvard Science Center', u'count': 1},
{u'_id': u'Tufts University Dowling Hall', u'count': 1},
{u'_id': u'Blackstone South', u'count': 1},
{u'_id': u'Vanserg Building', u'count': 1},
{u'_id': u'Knowles Center', u'count': 1},
{u'_id': u'Claverly Hall', u'count': 1},
{u'_id': u'Belfer Building', u'count': 1},
{u'_id': u'Gutman Building', u'count': 1},
{u'_id': u'Larsen Building', u'count': 1},
{u'_id': u'Winthrop House', u'count': 1},
{u'_id': u'CGIS Knafel', u'count': 1},
{u'_id': u'Radcliffe University', u'count': 1},
{u'_id': u'Tufts Dental School', u'count': 1},
{u'_id': u'Emerson Hall', u'count': 1},
{u'_id': u'38 Oxford', u'count': 1},
{u'_id': u'Buckingham House', u'count': 1},
{u'_id': u'Byerly Hall', u'count': 1},
{u'_id': u'Mount Ida College', u'count': 1},
{u'_id': u'Harvard School of Dental Medicine', u'count': 1},
{u'_id': u'Harvard Hall', u'count': 1},
{u'_id': u'Sever Hall', u'count': 1},
{u'_id': u'Smith Campus Center', u'count': 1},
{u'_id': u'Taubman Building', u'count': 1},
{u'_id': u'Biological Laboratory', u'count': 1},
{u'_id': u'Memorial Hall', u'count': 1},
{u'_id': u'Hollis Building;Hollis Hall', u'count': 1},
{u'_id': u'Radcliffe Institute at Harvard University', u'count': 1},
{u'_id': u'Harvard University Extension School', u'count': 1},
{u'_id': u'Harvard Business School', u'count': 1},
{u'_id': u'Putnam Building', u'count': 1},
{u'_id': u'Boston University School of Medicine', u'count': 1},
{u'_id': u'Cambridge Innovation Center', u'count': 1},
{u'_id': u'Myles Standish Hall', u'count': 1},
{u'_id': u'Divinity Hall', u'count': 1},
{u'_id': u'Farlow', u'count': 1},
{u'_id': u'John F Kennedy School of Government', u'count': 1},
{u'_id': u'Lowell Lecture Hall', u'count': 1},
{u'_id': u'Rubinstein Hall', u'count': 1},
{u'_id': u'CGIS South', u'count': 1},
{u'_id': u'Radcliffe Institute for Advanced Studies', u'count': 1},
{u'_id': u'Radcliffe Gym', u'count': 1},
{u'_id': u'Picower Institute for Learning and Memory (MIT)', u'count': 1},
{u'_id': u'Randolph Hall', u'count': 1},
{u'_id': u'McGovern Institute for Brain Research (MIT)', u'count': 1},
{u'_id': u'Lehman Dudley House', u'count': 1},
{u'_id': u'Soldiers Field Park Apartments', u'count': 1},
{u'_id': u'Longfellow Building', u'count': 1},
{u'_id': u'Agassiz House', u'count': 1},
{u'_id': u'University Herbaria', u'count': 1}]
```

```
In [42]: print(len(result['result']))
```

```
76
```