OpenStreetMap Sample Project

Data Wrangling with MongoDB

Map Area: Charlotte, NC, United States

https://www.openstreetmap.org/relation/177415 (https://www.openstreetmap.org/relation/177415)

http://metro.teczno.com/#charlotte (http://metro.teczno.com/#charlotte)

1. Problems Encountered in the Map ¶

ANS :

After initially downloading a small sample size of the Charlotte area and running it against a provisional check_data.py file, I noticed three main problems with the data, which I v discuss in the following order:

Over-abbreviated street names (“West Sugar Creek Rd.”,"East Jefferson Street Ste C").

To resolve this we need to map the abbreviated keywords with full one. (def process_address_street_name)

Inconsistent postal codes (“NC28209”, “28105-4837”, “28226”).

To resolve this we need to remove the unwanted prefix and unwanted suffix. (def process_address_post_code)

“Incorrect” postal codes (Charlotte area zip codes all begin with “282” however a large portion of all documented zip codes were outside this region.)

To resolve this we need ignore zipcode does not start with 282. (def should_ignore_addresss)

The records that has be clean and ignore is:

The records be cleaned street name is 8 records.

The records be cleaned postcode is : 5 records.

The records be ignore because postcode does not start with 282 : 186 records.

In [3]:

```
import pprint
import re
import codecs
import json
from collections import defaultdict

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)

CREATED = ["version", "changeset", "timestamp", "user", "uid"]
def shape_element(element, record_data={'clean_street': [], 'clean_postcode': [], 'ignore_postcode': []}):
    # In particular the following things should be done:
    # - you should process only 2 types of top level tags: "node" and "way"
    def process_normal_attr(target_element, _node):
        _node['type'] = target_element.tag
        for k in target_element.attrib:
            if k not in CREATED and k not in ['lat', 'lon']:
                _node[k] = target_element.attrib[k]

    # - 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"

    def process_created(target_element, _node):
        create_dict = {}
        for create_key in CREATED:
            if create_key in target_element.attrib:
                create_dict[create_key] = target_element.attrib[create_key]
        if len(create_dict) > 0:
            _node['created'] = create_dict
        return create_dict

    #     - attributes for latitude and longitude should be added to a "pos" array,
    #       for use in geospacial indexing. Make sure the values inside "pos" array are floats
    #       and not strings.
```

```
def process_geo(target_element, _node):
    pos = []
    for pos_key in ['lat', 'lon']:
        if pos_key in target_element.attrib:
            pos.append(float(target_element.attrib[pos_key]))
    if len(pos) == 2:
        _node["pos"] = pos
    return pos

# - if second level tag "k" value contains problematic characters, it should be ignored
# - if there is a second ":" that separates the type/direction of a street,
#   the tag should be ignored,

def should_ignore_tag(target_element):
    return problemchars.match(target_element.attrib['k']) or "street:" in target_element.attrib['k']

def should_ignore_addresss(addresss):
    postcode = addresss.get("postcode", None)
    return postcode is not None and not postcode.startswith("282")

# - if second level tag "k" value starts with "addr:", it should be added to a dictionary "address"

def is_address_tag(target_element):
    return target_element.attrib['k'].startswith("addr:")

street_mapping = {"S": "South",
                  "Ste": "Suite",
                  "St.": "Street",
                  "St": "Street",
                  "Rd": "Road",
                  "Rd.": "Road"
                  }

def process_address_street_name(street_name):

    for abbv in ["Ste", "St.", "St", "Rd", "Rd.", "S"]:
        if abbv + " " in street_name or street_name.endswith(abbv):
            record_data['clean_street'].append(street_name)
            street_name = street_name.replace(abbv, street_mapping[abbv])
    return street_name

def process_address_post_code(post_code):

    if re.compile(r'^[a-zA-Z]{2}[0-9]{5}$', re.IGNORECASE).search(post_code):
        record_data['clean_postcode'].append(post_code)
        return post_code[2:]
    if len(post_code) > 5:
        record_data['clean_postcode'].append(post_code)
    return post_code[:5]

def process_address_tag(target_element, address={}):
    k = target_element.attrib['k'].replace("addr:", "")
    if k == 'street':
        address[k] = process_address_street_name(target_element.attrib['v'])
    elif k == 'postcode':
        address[k] = process_address_post_code(target_element.attrib['v'])
    else:
        address[k] = target_element.attrib['v']
    return address

# - if second level tag "k" value does not start with "addr:", but contains ":", you can process it
#   same as any other tag.

def process_way_sub_element(way_element, _node={}):
    node_refs = []
    for nd in way_element.iter("nd"):
        node_refs.append(nd.attrib['ref'])
    _node["node_refs"] = node_refs

node = {}
if element.tag == "node" or element.tag == "way":
    process_normal_attr(element, node)
    process_created(element, node)
    process_geo(element, node)

    address = {}
    for tag in element.iter("tag"):
        if not should_ignore_tag(tag):
            if is_address_tag(tag):
                process_address_tag(tag, address=address)
                if should_ignore_addresss(address):
                    record_data['ignore_postcode'].append(address)
                    return None
            else:
                node[tag.attrib['k']] = tag.attrib['v']
    if len(address) > 0:
        node['address'] = address
    if element.tag == "way":
        process_way_sub_element(element, node)
    return node
else:
    return None
```

In [4]:

```
def process_map(file_in, pretty=False):
    import xml.etree.cElementTree as ET
    # You do not need to change this file
    file_out = "{0}.json".format(file_in)
    data = []
    record_data={'clean_street': [], 'clean_postcode': [], 'ignore_postcode': []}
    with codecs.open(file_out, "w") as fo:
        for _, element in ET.iterparse(file_in):
            el = shape_element(element,record_data)
            if el:
                data.append(el)
                if pretty:
                    fo.write(json.dumps(el, indent=2) + "\n")
                else:
                    fo.write(json.dumps(el) + "\n")

    print 'The records be cleaned street name is : ',str(len(record_data['clean_street'])) ,' records.'
    print 'The records be cleaned postcode is : ',str(len(record_data['clean_postcode'])) ,' records.'
    print 'The records be ignore because postcode does not start with 282 : ',str(len(record_data['ignore_postcode'])) ,
records.'
    from pymongo import MongoClient
    client = MongoClient("mongodb://localhost:27017")
    db = client.examples
    db.char.insert(data)
    return data

OSMFILE = 'charlotte.osm'
data = process_map(OSMFILE, True)
```

```
The records be cleaned street name is :  8  records.
The records be cleaned postcode is :  5  records.
The records be ignore because postcode does not start with 282 :  186  records.
```

Sort postcodes by count, descending

In [5]:

```
from pymongo import MongoClient
client = MongoClient("mongodb://localhost:27017")
db = client.examples
pipeline = [
            {"$match": {"address.postcode": {"$exists":1}}},
            {"$group":{"_id":"$address.postcode","count":{"$sum":1}}},
            {"$sort": {"count":-1}}
]
result = [doc for doc in db.char.aggregate(pipeline)]
import pprint
pprint.pprint(result[0])
pprint.pprint(result[-1])
```

```
{u'_id': u'29732', u'count': 313}
{u'_id': u'28097', u'count': 3}
```

Sort cities by count, descending

In [69]:

```
pipeline =  [{"$match":{"address.city":{"$exists":1}}},
 {"$group":{"_id":"$address.city", "count":{"$sum":1}}},
 {"$sort":{"count":-1}}]
result = [doc for doc in db.char.aggregate(pipeline)]
pprint.pprint(result)
```

```
[{u'_id': u'Rock Hill', u'count': 337},
 {u'_id': u'Pineville', u'count': 81},
 {u'_id': u'Charlotte', u'count': 80},
 {u'_id': u'York', u'count': 72},
 {u'_id': u'Matthews', u'count': 30},
 {u'_id': u'Concord', u'count': 12},
 {u'_id': u'Lake Wylie', u'count': 6},
 {u'_id': u'Locust', u'count': 3},
 {u'_id': u'Monroe', u'count': 3},
 {u'_id': u'Fort Mill, SC', u'count': 3},
 {u'_id': u'Belmont, NC', u'count': 3},
 {u'_id': u'Rock Hill, SC', u'count': 3}]
```

There are the data not belong to Charlotte city.

2. Data Overview

File sizes

```
In [48]: suffixes = ['B', 'KB', 'MB', 'GB', 'TB', 'PB']
         def humansize(nbytes):
             if nbytes == 0: return '0 B'
             i = 0
             while nbytes >= 1024 and i < len(suffixes)-1:
                 nbytes /= 1024.
                 i += 1
             f = ('%.2f' % nbytes).rstrip('0').rstrip('.')
             return '%s %s' % (f, suffixes[i])

         print 'charlotte.osm : '+humansize(os.path.getsize('charlotte.osm'))
         print 'charlotte.osm.json : '+humansize(os.path.getsize('charlotte.osm.json'))

         charlotte.osm : 294.21 MB
         charlotte.osm.json : 398.77 MB
```

Number of documents

```
In [19]: db.char.find().count()

Out[19]: 1571411
```

Number of nodes

```
In [20]: db.char.find({"type":"node"}).count()

Out[20]: 1486064
```

Number of ways

```
In [21]: db.char.find({"type":"way"}).count()

Out[21]: 85347
```

Number of unique users

```
In [22]: len(db.char.distinct("created.user"))

Out[22]: 337
```

Top 1 contributing user

```
In [28]: qry = db.char.aggregate([{"$group":{"_id":"$created.user", "count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":1}])

         result = [doc for doc in qry]
         result[0]

Out[28]: {u'_id': u'jumbanho', u'count': 831567}
```

Number of users appearing only once (having 1 post)

```
In [31]: qry = db.char.aggregate([{"$group":{"_id":"$created.user", "count":{"$sum":1}}},
                             {"$group":{"_id":"$count", "num_users":{"$sum":1}}},
                             {"$sort":{"_id":1}}, {"$limit":1}])

         result = [doc for doc in qry]
         result[0]

Out[31]: {u'_id': 1, u'num_users': 56}
```

number of chosen type of nodes

In [58]:

```
#db.char.distinct("amenity")
qry = db.char.aggregate([{"$match":{"amenity":{"$exists":1}}},
                        {"$group":{"_id":"$amenity", "count":{"$sum":1}}}, ])
result = [doc for doc in qry]
for node_info in result:
    print "%s : %s"%(node_info['_id'],node_info['count'])
```

```
university : 2
arts_centre : 1
marketplace : 1
toilets : 7
college : 1
nightclub : 4
pool : 1
food_court : 1
swimming_pool : 6
drinking_water : 1
community_centre : 1
veterinary : 1
closed : 1
taxi : 2
parking_entrance : 9
bank : 16
atm : 4
pub : 3
bicycle_parking : 2
convenience : 3
doctors : 1
shelter : 15
post_office : 12
assisted_living : 1
cinema : 7
library : 33
place_of_worship : 592
bar : 4
grave_yard : 82
police : 7
theatre : 7
kindergarten : 2
public_building : 2
bus_station : 1
telephone : 4
fast_food : 72
car_wash : 11
dentist : 2
fire_station : 52
townhall : 8
parking : 347
restaurant : 124
car_rental : 1
prison : 2
hospital : 22
bench : 31
post_box : 3
pharmacy : 22
waste_basket : 4
fountain : 12
cafe : 9
fuel : 39
courthouse : 1
school : 422
```

3. Additional Ideas

ANS :

To increase the number of data.

We can motivate contributor to provide more data by give them a credit on website.

To impute the missing values.

we can find the some data by other one attribute for example we can find the city name from postcode. If we use google api we can also find the postcode by lat,lng.

To increase correctness of data.

We can cross-validate the data with other such as google api .

ref : http://stackoverflow.com/questions/9689692/google-reverse-geocoding-how-to-snap-to-nearest-full-postcode (http://stackoverflow.com/questions/9689692/google-rever
geocoding-how-to-snap-to-nearest-full-postcode)

ref : https://developers.google.com/maps/documentation/geocoding/intro (https://developers.google.com/maps/documentation/geocoding/intro)

Top 10 appearing amenities

In [62]:

```
qry = db.char.aggregate([{"$match":{"amenity":{"$exists":1}}}, {"$group":{"_id":"$amenity",
"count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":10}])

result = [doc for doc in qry]
pprint.pprint(result)
```

```
[{u'_id': u'place_of_worship', u'count': 592},
 {u'_id': u'school', u'count': 422},
 {u'_id': u'parking', u'count': 347},
 {u'_id': u'restaurant', u'count': 124},
 {u'_id': u'grave_yard', u'count': 82},
 {u'_id': u'fast_food', u'count': 72},
 {u'_id': u'fire_station', u'count': 52},
 {u'_id': u'fuel', u'count': 39},
 {u'_id': u'library', u'count': 33},
 {u'_id': u'bench', u'count': 31}]
```

Biggest religion (no surprise here)

In [65]:

```
qry = db.char.aggregate([{"$match":{"amenity":{"$exists":1}, "amenity":"place_of_worship"}},

{"$group":{"_id":"$religion", "count":{"$sum":1}}},

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

result = [doc for doc in qry]
pprint.pprint(result)
```

```
[{u'_id': u'christian', u'count': 582}]
```

Most popular cuisines

In [67]:

```
qry = db.char.aggregate([{"$match":{"amenity":{"$exists":1}, "amenity":"restaurant"}},
                         {"$group":{"_id":"$cuisine", "count":{"$sum":1}}},
                         {"$sort":{"count":-1}}, {"$limit":2}])

result = [doc for doc in qry]
pprint.pprint(result)
```

```
[{u'_id': None, u'count': 65}, {u'_id': u'pizza', u'count': 10}]
```

Some of attributes are boolean