Project3_final

February 3, 2016

1 Open Street Map, Data Wrangling with MongoDb

1.1 1- Problems encountered in the map

1.1.1 Through visually inspecting a small sample size(1/10) data set for the city of Toronto, Canada and also running some Python code,I noticed the same issue that was addressed in the Lesson 6 exercises, namely different ways that street types appeared. In the process, I added more keys and values to the mapping dictionary. {"Dr":"Drive", "Cresent":"Crescent"}

1.1.2 The name of cities and towns were started with their title, such as Town of Withby or City of Oshawa. I think the title should be taken out, for the sake of more efficient parsing and processing, although there is a distinction betweent towns and cities. Therefore, I took out the when forming my JSON document.

1.1.3 I also noticed the map includes towns such as Whitby and King City and cities such as Oshawa that are not part of Toronto. They are not even part of Greater Toronto / Metropolitan Toronto. But they are close to Greater Toronto. One way of getting only toronto would be to programatically finding addresses that are in this cities or town in tags that are part of element way ("k" : "addr:city") and recursively taking all the nodes with the same id included in the element way out and finally taking the element itself out by JSON. There are other means as well, such as through exporting the area within specific lats and longs that encompass the City of Toronto itself.

1.1.4 Bellow is some of the functions and codes I added to exercise 6 solutions to clean and extract data.

```
In [ ]: #example: change "city of Burlington" to "Burlington"
        def edit_city_name(city_name):
            if (city_name.startswith("City of") or city_name.startswith("city of")\
                or city_name.startswith("Town of") or city_name.startswith("town of")):
                return city_name.split()[2]
            else:
                return city_name
        """
        .
        .
        .
        """

        #this is part of shape_element funtion that includes using the above function and other changes
        #before saving it to the JSON file
                for tag in element.iter("tag"):
                    attrib_val = tag.attrib['k']
                    if( not (problemchars.search(attrib_val)) and attrib_val.count(":") < 2):
```

```
                    if(attrib_val.startswith("addr:")):
                        if('address' not in node):
                            node['address'] = {}
                        if attrib_val == "addr:street":
                            node['address'][attrib_val[5:]] = change_street_type(tag.attrib['v'])
                        if attrib_val == "addr:city":
                            node['address'][attrib_val[5:]] = edit_city_name(tag.attrib['v'])
                            #print "node::: ", node
                    elif(attrib_val.find(":") != -1):
                        node[attrib_val] = tag.attrib['v']
                    else:
                        if('other_tags' not in node):
                            node['other_tags'] = {}
                        node['other_tags'][attrib_val] = tag.attrib['v']
            return node
```

1.1.5 Here is one of the codes that I used to audit the original file:

```
In [ ]: #this is a piece of code that will put all the values of tags of the same element type in a col
        #and see abnormalities or problems.
        def compare_tags(filename):
            tags = {}
            count = 0
            for event, elem in ET.iterparse(filename, events = ("start",)):
                if elem.tag not in tags:
                    tags[elem.tag] = {}
                for attribute, val in elem.attrib.iteritems():
                    if attribute not in tags[elem.tag]:
                        tags[elem.tag][attribute] = [val]
                    else:
                        tags[elem.tag][attribute].append(val)
                for child in elem.getchildren():
                    if child.tag not in tags[elem.tag] and child.tag != elem.tag:
                        tags[elem.tag][child.tag] = {}
                    for attribute, val in child.attrib.iteritems():
                        if attribute not in tags[elem.tag][child.tag]:
                            tags[elem.tag][child.tag][attribute] = [val]
                        else:
                            tags[elem.tag][child.tag][attribute].append(val)
            return tags
        def test():
            print "hello"
            tags = compare_tags('toronto_sample.osm')
            pprint.pprint(tags)

        if __name__ == "__main__":
            test()
```

1.2 2- Overview of the data and data exploration using MongoDb

1.2.1 Here is some statistic of data using MongoDb:

```
In [ ]: file sizes:
        toronto_sample.osm          108,229KB
```

```
toronto_sample.osm.json    112,439KB

#number of nodes
db.toronto.find({"type":"node"}).count()
461837

#number of ways
db.toronto.find({"type":"way"}).count()
65399

#number of unique users
len(db.toronto.distinct("created.user"))
1026

#Top 10 sources used for data
aggreg = db.toronto.aggregate([{"$group":{"_id":"$other_tags.source", "count":{"$sum":1}}},
                                                    {"$sort":{"count":-1}}, {"$limit":10}])
for doc in aggreg:
    print doc

{u'count': 441069, u'_id': None}
{u'count': 45408, u'_id': u'CanVec 6.0 - NRCan'}
{u'count': 12756, u'_id': u'NRCan-CanVec-7.0'}
{u'count': 7228, u'_id': u'Bing'}
{u'count': 4746, u'_id': u'Geobase_Import_2009'}
{u'count': 3866, u'_id': u'NRCan-CanVec-10.0'}
{u'count': 3207, u'_id': u'CanVec_Import_2009'}
{u'count': 2454, u'_id': u'yahoo'}
{u'count': 1816, u'_id': u'Yahoo'}
{u'count': 1704, u'_id': u'NRCan-CanVec-8.0'}

#Top 10 users/contributors
aggreg = db.toronto.aggregate([{"$project":{"user":"$created.user","source":"$other_tags.source
                                {"$group":{"_id":"$user", "count":{"$sum":1}}},
                                {"$project":{"count":"$count","user":"$created.user","source":"$
                                 {"$sort":{"count":-1}}, {"$limit":10}])
for doc in aggreg:
    print doc
{u'count': 328597, u'_id': u'andrewpmk'}
{u'count': 48356, u'_id': u'MikeyCarter'}
{u'count': 40013, u'_id': u'Kevo'}
{u'count': 16291, u'_id': u'Victor Bielawski'}
{u'count': 13367, u'_id': u'Bootprint'}
{u'count': 8348, u'_id': u'geobase_stevens'}
{u'count': 7626, u'_id': u'rw__'}
{u'count': 4592, u'_id': u'Gerit Wagner'}
{u'count': 3760, u'_id': u'brandoncote'}
{u'count': 3485, u'_id': u'andrewpmk_imports'}

#Top 10 leisure places in Toronto
leisures = db.toronto.aggregate([{"$match":{"other_tags.leisure":{"$exists":1}}},
{"$group":{"_id":"$other_tags.leisure", "count":{"$sum":1}}},
{"$sort":{"count":-1}}, {"$limit":10}])
```

```
for leisure in leisures:
    print leisure
{u'count': 553, u'_id': u'pitch'}
{u'count': 419, u'_id': u'park'}
{u'count': 409, u'_id': u'playground'}
{u'count': 67, u'_id': u'swimming_pool'}
{u'count': 51, u'_id': u'garden'}
{u'count': 49, u'_id': u'track'}
{u'count': 31, u'_id': u'sports_centre'}
{u'count': 20, u'_id': u'golf_course'}
{u'count': 10, u'_id': u'ice_rink'}
{u'count': 9, u'_id': u'picnic_table'}

#Calculating the ratio of roads that have designated Bicycle lane. As I expected very small num
all_roads = db.toronto.aggregate([{"$match":{"other_tags.surface":"asphalt"}},
                                  {"$group":{"_id":"$other_tags.surface", "count":{"$sum":1}}}])
bicycle_roads = db.toronto.aggregate([{"$match":{"other_tags.bicycle":"yes"}},
                                  {"$group":{"_id":"$other_tags.bicycle", "count":{"$sum":1}}}])
print float(list(bicycle_roads)[0]['count'])/float(list(all_roads)[0]['count'])
0.0361610188765

#Top 10 sport aminities. I thought there would be many more hokey rinks.
sports = db.toronto.aggregate([{"$match":{"other_tags.sport":{"$exists":"true"}}},
                                  {"$group":{"_id":"$other_tags.sport", "count":{"$sum":1}}},{"$s
for sport in sports:
    print sport
{u'count': 173, u'_id': u'baseball'}
{u'count': 171, u'_id': u'soccer'}
{u'count': 99, u'_id': u'tennis'}
{u'count': 44, u'_id': u'basketball'}
{u'count': 30, u'_id': u'multi'}
{u'count': 26, u'_id': u'golf'}
{u'count': 21, u'_id': u'swimming'}
{u'count': 6, u'_id': u'hockey'}
{u'count': 5, u'_id': u'cricket'}
{u'count': 4, u'_id': u'bowls'}

#top 10 religions based on number of place of worship
religions = db.toronto.aggregate([{"$match":{"other_tags.amenity":{"$exists":1}, "other_tags.am
{"$group":{"_id":"$other_tags.religion", "count":{"$sum":1}}},
{"$sort":{"count":-1}}, {"$limit":10}])
for religion in religions:
    print religion
{u'count': 146, u'_id': u'christian'}
{u'count': 24, u'_id': None}
{u'count': 8, u'_id': u'jewish'}
{u'count': 4, u'_id': u'muslim'}
{u'count': 2, u'_id': u'buddhist'}
{u'count': 1, u'_id': u'hindu'}
{u'count': 1, u'_id': u'sikh'}
```

1.3 Other ideas about dataset:

1.3.1 **I think for the sake of quick parsing and usability in different applications including large scale machine learning applications, it might be a good idea to add an attribute to nodes identifying the type. As it stands, nodes don't give any information on what is their nature, a lamp post, a traffic light or any other spot in the city. This could be done through adding a tag for the purpose.**

1.3.2 **There are tags such as <"k":"building", "v":"yes">. Can't it be changed to <"type":"building">? this will be a lot faster and streight forward for parsing, as we will have to only look for the type rather that different variations of "k".**

1.3.3 **In some tags(amenity in this case) such as place of worship, the field is blank(religion in this case). This also can be edited as it will be important in accurate data collection. Place of worship is give just as an example.**

1.4 In conclusion:

1.4.1 **Through visual inspection of dataset and auditing with the use of Python codes, the data set for Toronto seems to include cities nearby that are not even part of metropolitan Toronto. The solution might be to extract the data by use of lat an long or to parse data and eliminate elements and tags that are related to areas that fall outside. It seems to be a good idea to include an attribute in node element to indicate the nature of the node, at least for the sake of parsing and extracting large amount of information, with higher speed. For some purposes, the title "city" or "town" should be taken out so that the data can be searched by the name of the city or town without the title. Majority of people don't know the title or are not aware of it. The other solution is adding an attribute or a tag for that purpose. There were some cleaning up required to correct the abbreviated names of ways such as changing Rd to Road and some misspelled ones such as cresent to crescent. In short for the sake of extracting data specially for large scale applications the Toronto data set needs some editing and corrections.**

In [ ]: