datawrangle_dallas

March 22, 2016

1 OpenStreetMap Sample Project

1.1 Data Wrangling with MongoDB

1.1.1 Karun Gahlawat

Dallas

https://s3.amazonaws.com/metro-extracts.mapzen.com/dallas_texas.osm.bz2

1.1.2 http://www.openstreetmap.org/#map=10/32.7740/-96.7902

DALLAS AREA EXPLORATION SECTION STARTS FINAL PROJECT CODE

1.2 Problems Encountered In Map

1.2.1 Data Size

The area picked is +500MB and is very slow to process basically anything on it. This dataset cannot be loaded / pushed into github and it took a lot of time to push it into mongodb. Especially, if inserting one by one. The idea behind inserting one by one was to take that opportunity to clean it as we go and then insert it. It appears though cleaning this data before inserting or after inserting is a better way.

1.2.2 Street Types Over Complicated for simple regular expressions

The street types in general are easily correctable. For e.g Road is types as Road and even if it is typed as Rd., it is a common replace string. Having said that (output.txt), 161 records out of 5558972 records have little ways to correct them.

In some cases, we have street names like **5229 alpha road dallas tx 75240** . This would cause the regular expression to take street type as **75240** due to the assumption that last street names would only include street names and not the whole address. In some other cases, **3740 N. Josey Lane, Suite 121** would appear. In this case **121** would appear as street type.

The solution to this would be to iterate through these records again and again bring them into a simpler form. This would require some manual input.

1.2.3 Postcodes are not really postcodes

this isn't really a postal code but a state. there is no simple soution to this it would seem. One way to fix it woud be to gather as much data as we can about address and cross reference it with google or like and get the right zipcode. That is a non trivial task.

1.2.4 Extra fields in some but not most records

Some extra fields are found in some records like * **tag k="building" v="office"/** * **tag k="building:levels" v="2"/** * **tag k="sport" v="swimming"/** * **tag k="amenity" v="parking"/**

these tags are not very common and would require to change the db schema a little bit to accomodate them. They seem like a nice addition to the **way** node that further describes some extra information about the way.

1.3 Overview of Data

Please see Last Section for Code Details that produced this data

- Total Records In DB: 5558972
- Data Size In DB: 1.3GB
- First Record In DB: {u'id': u'26450261', u'_id': ObjectId('56f1d4af222cc489da99b3f0'), u'type': u'node', u'pos': [-97.0027785, 32.9901295], u'created': {u'changeset': u'641383', u'version': u'4', u'user': u'brianboru', u'timestamp': u'2008-10-31T13:10:04Z', u'uid': u'9065'}}
- Top 5 Users
  - User 5 {u'count': 2254674, u'*id': u'woodpeck*fixbot'}
  - User 4 {u'count': 198416, u'_id': u'fmmute'}
  - User 3 {u'count': 176820, u'_id': u'TexasNHD'}
  - User 2 {u'count': 123490, u'_id': u'25or6to4'}
  - User 1 {u'count': 121506, u'_id': u'Chris Lawrence'}
- Top 5 Zip Codes
  - Zip Code 5 {u'count': 1211, u'_id': u'75104'}
  - Zip Code 4 {u'count': 629, u'_id': u'75093'}
  - Zip Code 3 {u'count': 343, u'_id': u'75070'}
  - Zip Code 2 {u'count': 227, u'_id': u'75051'}
  - Zip Code 1 {u'count': 181, u'_id': u'75069'}
  - address records: 119075
- Size of xml downloaded : + 500MB
- Size of json created: +250MB

1.4 Additional Ideas

- Needs more input for completeness. Fields like amenities, levels and others are important part of location information that could substantially increase the quality of this data.
- Most of the data input is by top 5 users. This data is relatively clean. For e.g. only 161 street types out of 5.5 Million records are un retrievable.
- Given more iterations, this data could be further processed and mapped to its quality.

1.5 Conclusion

- This is a *great* effort by Texans. With a little bit of information and processing, this massive data set could become a LOT useful!

1.6 Lesson 6 Problems Code

1.6.1 Count Tags

```
In [1]: #!/usr/bin/env python
        # -*- coding: utf-8 -*-
```

```
        """
        Your task is to use the iterative parsing to process the map file and
        find out not only what tags are there, but also how many, to get the
        feeling on how much of which data you can expect to have in the map.
        Fill out the count_tags function. It should return a dictionary with the
        tag name as the key and number of times this tag can be encountered in
        the map as value.

        Note that your code will be tested with a different data file than the 'example.osm'
        """
        import xml.etree.cElementTree as ET
        import pprint

        def count_tags(filename):
                # YOUR CODE HERE
            tags = {}
            context = ET.iterparse(filename, events=('start', 'end'))
            context = iter(context)
            for event, elem in context:
                if event == 'start':
                    if elem.tag in tags:
                        tags[elem.tag] += 1
                    else:
                        tags[elem.tag] = 1

            return tags

        def test():

            tags = count_tags('example.osm')
            pprint.pprint(tags)
            assert tags == {'bounds': 1,
                            'member': 3,
                            'nd': 4,
                            'node': 20,
                            'osm': 1,
                            'relation': 1,
                            'tag': 7,
                            'way': 1}



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

```
{'bounds': 1,
 'member': 3,
 'nd': 4,
 'node': 20,
 'osm': 1,
 'relation': 1,
 'tag': 7,
 'way': 1}
```

1.6.2 Example File for Lesson 6 Problems

In [ ]:
```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.3.3 (28791 thorn-03.openstreetmap.org)" copyright="OpenS
 <bounds minlat="41.9704500" minlon="-87.6928300" maxlat="41.9758200" maxlon="-87.6894800"/>
 <node id="261114295" visible="true" version="7" changeset="11129782" timestamp="2012-03-28T18:
 <node id="261114296" visible="true" version="6" changeset="8448766" timestamp="2011-06-15T17:0
 <node id="261114299" visible="true" version="5" changeset="8581395" timestamp="2011-06-29T14:1
 <node id="261146436" visible="true" version="5" changeset="8581395" timestamp="2011-06-29T14:1
 <node id="261147304" visible="true" version="7" changeset="8581395" timestamp="2011-06-29T14:1
 <node id="261224274" visible="true" version="5" changeset="8581395" timestamp="2011-06-29T14:1
 <node id="293816175" visible="true" version="47" changeset="8448766" timestamp="2011-06-15T16:5
 <node id="305896090" visible="true" version="37" changeset="15348240" timestamp="2013-03-13T07
 <node id="317636974" visible="true" version="12" changeset="15348240" timestamp="2013-03-13T08
 <node id="317636971" visible="true" version="13" changeset="15348240" timestamp="2013-03-13T08
 <node id="317637399" visible="true" version="2" changeset="14927972" timestamp="2013-02-05T22:4
 <node id="317637398" visible="true" version="2" changeset="14927972" timestamp="2013-02-05T22:4
 <node id="365214872" visible="true" version="3" changeset="8448766" timestamp="2011-06-15T17:0
 <node id="261299091" visible="true" version="6" changeset="8581395" timestamp="2011-06-29T14:1
 <node id="261114294" visible="true" version="6" changeset="8448766" timestamp="2011-06-15T17:0
 <node id="261210804" visible="true" version="4" changeset="3359748" timestamp="2009-12-13T00:3
 <node id="261221422" visible="true" version="7" changeset="8581395" timestamp="2011-06-29T14:1
 <node id="261221424" visible="true" version="7" changeset="8581395" timestamp="2011-06-29T14:1
  <tag k="highway" v="traffic_signals"/>
 </node>
 <node id="261198953" visible="true" version="6" changeset="8581395" timestamp="2011-06-29T14:1
 <node id="757860928" visible="true" version="2" changeset="5288876" timestamp="2010-07-22T16:1
  <tag k="amenity" v="fast_food"/>
  <tag k="cuisine" v="sausage"/>
  <tag k="name" v="Shelly's Tasty Freeze"/>
 </node>
  <way id="258219703" visible="true" version="1" changeset="20187382" timestamp="2014-01-25T02:
  <nd ref="2636086179"/>
  <nd ref="2636086178"/>
  <nd ref="2636086177"/>
  <nd ref="2636086176"/>
  <tag k="highway" v="service"/>
 </way>
 <relation id="1557627" visible="true" version="2" changeset="14326854" timestamp="2012-12-19T05
  <member type="node" ref="1258927212" role="via"/>
  <member type="way" ref="110160127" role="from"/>
  <member type="way" ref="34073105" role="to"/>
  <tag k="restriction" v="only_right_turn"/>
  <tag k="type" v="restriction"/>
 </relation>
</osm>
```

1.6.3 Audit Street Names

In [2]:
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import xml.etree.cElementTree as ET
import pprint
import re
"""
```

```
Your task is to explore the data a bit more.
Before you process the data and add it into MongoDB, you should check the "k"
value for each "<tag>" and see if they can be valid keys in MongoDB, as well as
see if there are any other potential problems.

We have provided you with 3 regular expressions to check for certain patterns
in the tags. As we saw in the quiz earlier, we would like to change the data
model and expand the "addr:street" type of keys to a dictionary like this:
{"address": {"street": "Some value"}}
So, we have to see if we have such tags, and if we have any tags with
problematic characters.

Please complete the function 'key_type', such that we have a count of each of
four tag categories in a dictionary:
  "lower", for tags that contain only lowercase letters and are valid,
  "lower_colon", for otherwise valid tags with a colon in their names,
  "problemchars", for tags with problematic characters, and
  "other", for other tags that do not fall into the other three categories.
See the 'process_map' and 'test' functions for examples of the expected format.
"""


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":
        # YOUR CODE HERE
        kvalue = element.attrib['k']
        print kvalue
        if lower.match(kvalue):
            print '\tlower'
            if 'lower' in keys:
                keys['lower'] += 1
            else:
                keys['lower'] = 1
        elif lower_colon.match(kvalue):
            print '\tlower_colon'
            if 'lower_colon' in keys:
                keys['lower_colon'] += 1
            else:
                keys['lower_colon'] = 1
        elif problemchars.match(kvalue):
            print '\tproblemchars'
            if 'problemchars' in keys:
                keys['problemchars'] += 1
            else:
                keys['problemchars'] = 1
        else:
            print '\tothers'
            if 'other' in keys:
                keys['other'] += 1
```

```
                else:
                    keys['other'] = 1

            #print keys
            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



        def test():
            # You can use another testfile 'map.osm' to look at your solution
            # Note that the assertion below will be incorrect then.
            # Note as well that the test function here is only used in the Test Run;
            # when you submit, your code will be checked against a different dataset.
            keys = process_map('example.osm')
            pprint.pprint(keys)
            assert keys == {'lower': 5, 'lower_colon': 0, 'other': 2, 'problemchars': 0}


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

```
highway
        lower
amenity
        lower
cuisine
        lower
name
        lower
highway
        lower
restriction
        lower
type
        lower
{'lower': 7, 'lower_colon': 0, 'other': 0, 'problemchars': 0}
```

```
        ---------------------------------------------------------------------------

        AssertionError                            Traceback (most recent call last)

        <ipython-input-2-4789581f47a6> in <module>()
         87
         88 if __name__ == "__main__":
    ---> 89     test()
```

```
        <ipython-input-2-4789581f47a6> in test()
         83     keys = process_map('example.osm')
         84     pprint.pprint(keys)
    ---> 85     assert keys == {'lower': 5, 'lower_colon': 0, 'other': 2, 'problemchars': 0}
         86
         87


        AssertionError:
```

1.6.4 Audit Users

```
In [3]: #!/usr/bin/env python
        # -*- coding: utf-8 -*-
        import xml.etree.cElementTree as ET
        import pprint
        import re
        """
        Your task is to explore the data a bit more.
        The first task is a fun one - find out how many unique users
        have contributed to the map in this particular area!

        The function process_map should return a set of unique user IDs ("uid")
        """

        def get_user(element):
            return


        def process_map(filename):
            users = set()
            for _, element in ET.iterparse(filename):
                if 'user' in element.attrib:
                    users.add(element.attrib['user'])

            return users


        def test():

            users = process_map('example.osm')
            pprint.pprint(users)
            assert len(users) == 6



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

```
set(['Umbugbene',
     'bbmiller',
     'fredr',
```

```
    'linuxUser16',
    'uboot',
    'woodpeck_fixbot'])
```

1.6.5 Audit and Fix Street Names

In [5]:
```
"""
Your task in this exercise has two steps:

- audit the OSMFILE and change the variable 'mapping' to reflect the changes needed to fix
    the unexpected street types to the appropriate ones in the expected list.
    You have to add mappings only for the actual problems you find in this OSMFILE,
    not a generalized solution, since that may and will depend on the particular area you are a
- write the update_name function, to actually fix the street name.
    The function takes a string with street name as an argument and should return the fixed nam
    We have provided a simple test so that you see what exactly is expected
"""
import xml.etree.cElementTree as ET
from collections import defaultdict
import re
import pprint

OSMFILE = "example.osm"
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)


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

# UPDATE THIS VARIABLE
mapping = { "St": "Street",
            "St.": "Street",
            "Rd.": "Road",
            "N.": "North",
            "Ave": "Avenue"
            }


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


    def update_name(name, mapping):

        # YOUR CODE HERE
        #print "name:=" , name
        m = street_type_re.search(name)
        if m:
            st = m.group()
            if st not in expected:
                name = re.sub(st, mapping[st], name)
        return name


    def test():
        st_types = audit(OSMFILE)
        assert len(st_types) == 3
        pprint.pprint(dict(st_types))

        for st_type, ways in st_types.iteritems():
            for name in ways:
                better_name = update_name(name, mapping)
                print name, "=>", better_name
                if name == "West Lexington St.":
                    assert better_name == "West Lexington Street"
                if name == "Baldwin Rd.":
                    assert better_name == "Baldwin Road"


    if __name__ == '__main__':
        test()


    ---------------------------------------------------------------------------

    AssertionError                            Traceback (most recent call last)

    <ipython-input-5-930afafd8298> in <module>()
     84
     85 if __name__ == '__main__':
---> 86     test()


    <ipython-input-5-930afafd8298> in test()
     70 def test():
     71     st_types = audit(OSMFILE)
---> 72     assert len(st_types) == 3
```

```
 73     pprint.pprint(dict(st_types))
 74
```

```
AssertionError:
```

1.6.6 Prepare for MongoDB using JSON Format

In [6]:
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import xml.etree.cElementTree as ET
import pprint
import re
import codecs
import json
"""
Your task is to wrangle the data and transform the shape of the data
into the model we mentioned earlier. The output should be a list of dictionaries
that look like this:

{
"id": "2406124091",
"type: "node",
"visible":"true",
"created": {
          "version":"2",
          "changeset":"17206049",
          "timestamp":"2013-08-03T16:43:42Z",
          "user":"linuxUser16",
          "uid":"1219059"
        },
"pos": [41.9757030, -87.6921867],
"address": {
          "housenumber": "5157",
          "postcode": "60625",
          "street": "North Lincoln Ave"
        },
"amenity": "restaurant",
"cuisine": "mexican",
"name": "La Cabana De Don Luis",
"phone": "1 (773)-271-5176"
}

You have to complete the function 'shape_element'.
We have provided a function that will parse the map file, and call the function with the elemen
as an argument. You should return a dictionary, containing the shaped data for that element.
We have also provided a way to save the data in a file, so that you could use
mongoimport later on to import the shaped data into MongoDB.

Note that in this exercise we do not use the 'update street name' procedures
you worked on in the previous exercise. If you are using this code in your final
project, you are strongly encouraged to use the code from previous exercise to
update the street names before you save them to JSON.
```

```
In particular the following things should be done:
- you should process only 2 types of top level tags: "node" and "way"
- 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"
    - 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.
- if the second level tag "k" value contains problematic characters, it should be ignored
- if the second level tag "k" value starts with "addr:", it should be added to a dictionary "ad
- if the second level tag "k" value does not start with "addr:", but contains ":", you can
  process it in a way that you feel is best. For example, you might split it into a two-level
  dictionary like with "addr:", or otherwise convert the ":" to create a valid key.
- if there is a second ":" that separates the type/direction of a street,
  the tag should be ignored, for example:

<tag k="addr:housenumber" v="5158"/>
<tag k="addr:street" v="North Lincoln Avenue"/>
<tag k="addr:street:name" v="Lincoln"/>
<tag k="addr:street:prefix" v="North"/>
<tag k="addr:street:type" v="Avenue"/>
<tag k="amenity" v="pharmacy"/>

  should be turned into:

{...
"address": {
    "housenumber": 5158,
    "street": "North Lincoln Avenue"
}
"amenity": "pharmacy",
...
}

- for "way" specifically:

  <nd ref="305896090"/>
  <nd ref="1719825889"/>

should be turned into
"node_refs": ["305896090", "1719825889"]
"""


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

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


def shape_element(element):
    node = {}
    if element.tag == "node" or element.tag == "way" :
```

```
        # YOUR CODE HERE
        if element.tag == "node":
            node['type'] = 'node'
        elif element.tag == "way":
            node['type'] = 'way'
        #"id": "2406124091",
#"type: "node",
#"visible":"true",
#"created": {
#          "version":"2",
#          "changeset":"17206049",
#          "timestamp":"2013-08-03T16:43:42Z",
#          "user":"linuxUser16",
#          "uid":"1219059"
#        },
#"pos": [41.9757030, -87.6921867],
#"address": {
#          "housenumber": "5157",
#          "postcode": "60625",
#          "street": "North Lincoln Ave"
#        },
#"amenity": "restaurant",
#"cuisine": "mexican",
#"name": "La Cabana De Don Luis",
#"phone": "1 (773)-271-5176"
#}
        atts = element.attrib
        address = {}
        pos = []
        created = {}
        node_refs = []
        node['address'] = address
        node['pos'] = pos
        node['created'] = created
        node['node_refs'] = node_refs
        ### parent level attributes
        for k,v in atts.iteritems():
            if k == 'id':
                node[k] = v
            elif k == 'visible':
                node[k] = v
            elif k == 'amenity':
                node[k] = v
            elif k == 'name':
                node[k] = v
            elif k == 'phone':
                node[k] = v
            elif k == 'version':
                created[k] = v
            elif k == 'changeset':
                created[k] = v
            elif k == 'timestamp':
                created[k] = v
            elif k == 'user':
```

```
                created[k] = v
            elif k == 'uid':
                created[k] = v
            elif k == 'lon':
                pos.append(float(v))
            elif k == 'lat':
                pos.append(float(v))

        #### child nodes
        for child in element:
            if child.tag == 'nd':
                node_refs.append(child.attrib['ref'])
            elif child.tag == 'tag':
                ktag = child.attrib['k']
                vtag = child.attrib['v']
                if not problemchars.match(vtag):
                    if ktag == "addr:city":
                        address['city'] = vtag
                    elif ktag == "addr:housenumber":
                        address['housenumber'] = vtag
                    elif ktag == "addr:postcode":
                        address['postcode'] = vtag
                    elif ktag == "addr:street":
                        address['street'] = vtag
                    elif ktag == 'amenity':
                        node[ktag] = vtag
                    elif ktag == 'cuisine':
                        node[ktag] = vtag
                    elif ktag == 'name':
                        node[ktag] = vtag
                    elif ktag == 'phone':
                        node[ktag] = vtag

##<tag k="addr:city" v="Chicago"/>
##<tag k="addr:housenumber" v="5157"/>
##<tag k="addr:postcode" v="60625"/>
##<tag k="addr:street" v="North Lincoln Ave"/>
##<tag k="amenity" v="restaurant"/>
##<tag k="cuisine" v="mexican"/>
##<tag k="name" v="La Cabana De Don Luis"/>
##<tag k="outdoor_seating" v="no"/>
##<tag k="phone" v="1 (773)-271-5176"/>
##<tag k="smoking" v="no"/>
##<tag k="takeaway" v="yes"/>
        if len(address) == 0:
            del node['address']
        if len(node_refs) == 0:
            del node['node_refs']
        if len(pos) == 0:
            del node['pos']
        #print "returning node: ", node
        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

def test():
    # NOTE: if you are running this code on your computer, with a larger dataset,
    # call the process_map procedure with pretty=False. The pretty=True option adds
    # additional spaces to the output, making it significantly larger.
    data = process_map('example.osm', True)
    #pprint.pprint(data)

    correct_first_elem = {
        "id": "261114295",
        "visible": "true",
        "type": "node",
        "pos": [-87.6866303, 41.9730791],
        "created": {
            "changeset": "11129782",
            "user": "bbmiller",
            "version": "7",
            "uid": "451048",
            "timestamp": "2012-03-28T18:31:23Z"
        }
    }
    #print "actual data[0]: ", data[0]
    #print "expected      : ", correct_first_elem
    assert data[0] == correct_first_elem
    assert data[-1]["address"] == {
                                    "street": "West Lexington St.",
                                    "housenumber": "1412"
                                      }
    assert data[-1]["node_refs"] == [ "2199822281", "2199822390",  "2199822392", "2199822369",
                                    "2199822370", "2199822284", "2199822281"]

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

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)
```

```
    <ipython-input-6-6f5a6c54c972> in <module>()
    254
    255 if __name__ == "__main__":
--> 256     test()


    <ipython-input-6-6f5a6c54c972> in test()
    246     #print "expected      : ", correct_first_elem
    247     assert data[0] == correct_first_elem
--> 248     assert data[-1]["address"] == {
    249                                       "street": "West Lexington St.",
    250                                       "housenumber": "1412"


    KeyError: 'address'
```

1.7 Dallas Area Data Exploration

In [ ]:
```
######################
######################
### over all comments ###
'''
file structure is divided into three categories
1) globals and utility functions like mapping, audit_street and shape_elements
these functions provide reference data and some utility based functionality
2) process map: this function iterates over xml file and creates appropriate
structures and returns them. idea is to iterate over this file once and gather
any / all information that we can in one shot. it is a HUGE data set.
3) insert_mongo: which creates data one by one into mongo db instance. this step
is bypassed for ease of testing and relplaced by
--------------------------------------------------------------------------------
mongoimport --db udacity --collection dallas --file dallas_texas.osm.json  --drop
--------------------------------------------------------------------------------
over all idea is to follow lesson 6 problem sets and twist and turn for this data set.

'''


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

'''
global regular expressions for determining
category of data
'''
lower = re.compile(r'^([a-z]|_)*$')
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
```

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

#### global street type categorizer
#### \bSt\. will pick up Street
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)

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

# mapping of incorrect street types to correct ones
# use this dict to replace incorrect street types
mapping = { "St": "Street",
            "St.": "Street",
            "Rd.": "Road",
            "Rd": "Road",
            "N.": "North",
            "Ave": "Avenue",
            "Tr" : "Trail",
            "Tr.": "Trail",
            "Trl": "Trail",
            "S." : "South",
            "W." : "West",
            "W" : "West",
            "S" : "South",
            "W" : "West",
            "Pkwy" : "Parkway",
            "Hwy" : "Highway",
            "E" : "East",
            "E." : "East",
            "Blvd" : "Boulevard",
            "Blvd." : "Boulevard"

            }

#### all the street types that are left out by the
### street_type_re expression are into this.
#### this needs a second look

incomplete_mapping = set()

'''
categorizes tags into lower, lower with a colon and problem chars
'''
def key_type(element, keys):
    if element.tag == "tag":
        kvalue = element.attrib['k']
        #print kvalue
        if lower.match(kvalue):
            #print '\tlower'
            if 'lower' in keys:
                keys['lower'] += 1
            else:
                keys['lower'] = 1
```

```
        elif lower_colon.match(kvalue):
            #print '\tlower_colon'
            if 'lower_colon' in keys:
                keys['lower_colon'] += 1
            else:
                keys['lower_colon'] = 1
        elif problemchars.match(kvalue):
            #print '\tproblemchars'
            if 'problemchars' in keys:
                keys['problemchars'] += 1
            else:
                keys['problemchars'] = 1
        else:
            #print '\tothers'
            if 'other' in keys:
                keys['other'] += 1
            else:
                keys['other'] = 1

    #print keys
    return keys

#### runs over al street types and assess them as expected or un expected
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:
            #print "adding street type to unexpected: ", street_type, street_name
            street_types[street_type].add(street_name)


#### if element attribute is a street addresss
def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")

#### update unexpected street names with provided correct street names
def update_name(name, mapping):
    m = street_type_re.search(name)
    if m:
        st = m.group()
        if st not in expected:
            #print "st not in expected", st
            if st in mapping:
                #print "st in mapping"
                name = re.sub(st, mapping[st], name)
            else:
                incomplete_mapping.add(st)
    return name


#### reshape the tags into an appropriate schema for mongo db repo
def shape_element(element):
    node = {}
```

```
if element.tag == "node" or element.tag == "way" :
    # YOUR CODE HERE
    if element.tag == "node":
        node['type'] = 'node'
    elif element.tag == "way":
        node['type'] = 'way'

    atts = element.attrib
    address = {}
    pos = []
    created = {}
    node_refs = []
    node['address'] = address
    node['pos'] = pos
    node['created'] = created
    node['node_refs'] = node_refs
    ### parent level attributes
    for k,v in atts.iteritems():
        if k == 'id':
            node[k] = v
        elif k == 'visible':
            node[k] = v
        elif k == 'amenity':
            node[k] = v
        elif k == 'name':
            node[k] = v
        elif k == 'phone':
            node[k] = v
        elif k == 'version':
            created[k] = v
        elif k == 'changeset':
            created[k] = v
        elif k == 'timestamp':
            created[k] = v
        elif k == 'user':
            created[k] = v
        elif k == 'uid':
            created[k] = v
        elif k == 'lon':
            pos.append(float(v))
        elif k == 'lat':
            pos.append(float(v))

    #### child nodes
    for child in element:
        if child.tag == 'nd':
            node_refs.append(child.attrib['ref'])
        elif child.tag == 'tag':
            ktag = child.attrib['k']
            vtag = child.attrib['v']
            if not problemchars.match(vtag):
                if ktag == "addr:city":
                    address['city'] = vtag
                elif ktag == "addr:housenumber":
```

```
                        address['housenumber'] = vtag
                    elif ktag == "addr:postcode":
                        address['postcode'] = vtag
                    elif ktag == "addr:street":
                        address['street'] = vtag
                    elif ktag == 'amenity':
                        node[ktag] = vtag
                    elif ktag == 'cuisine':
                        node[ktag] = vtag
                    elif ktag == 'name':
                        node[ktag] = vtag
                    elif ktag == 'phone':
                        node[ktag] = vtag

            if len(address) == 0:
                del node['address']
            if len(node_refs) == 0:
                del node['node_refs']
            if len(pos) == 0:
                del node['pos']
            #print "returning node: ", node
            return node
        else:
            return None

#### main operator function that iterates through the xml
### and gathers all info into multiple structures

def process_map(filename):
    keys = {"lower": 0, "lower_colon": 0, "problemchars": 0, "other": 0}
    users = set()
    tags = {}
    street_types = defaultdict(set)
    data = []
    context = ET.iterparse(filename, events=('start', 'end'))
    context = iter(context)
    file_out = "{0}.json".format(filename)
    fo = codecs.open(file_out, "w")
    for event, element in context:
        if event == 'start':
            if element.tag in tags:
                tags[element.tag] += 1
            else:
                tags[element.tag] = 1
            if element.tag == "node" or element.tag == "way":
                for tag in element.iter("tag"):
                    if is_street_name(tag):
                        audit_street_type(street_types, tag.attrib['v'])

        keys = key_type(element, keys)
        if 'user' in element.attrib:
            users.add(element.attrib['user'])
```

```
        el = shape_element(element)
        if el:
            data.append(el)
            fo.write(json.dumps(el, indent=2)+"\n")

    return tags, keys, users, street_types, data, file_out


#### insert into db one record at a time

def insert_mongo(data):
    client = MongoClient()
    dallasdb = client['udacity']
    dallasarea = dallasdb['dallas']
    for record in data:
        if 'address' in record:
            address = record['address']
            street = address['street']
            better_street = update_name(street, mapping)
            address['street'] = better_street
        nodeid = dallasarea.insert_one(record).inserted_id
        print "nodeid: ", nodeid, " inserted"
    total_records = dallasarea.find().count()
    print "total records: ", total_records


###### main run function that brings it al together
def run():
    filename = 'dallas_texas.osm'
    tags, keys, users, street_types, data, file_out = process_map(filename)
    #pprint.pprint(tags)
    #pprint.pprint(keys)
    print len(users)
    print len(street_types)
    #for st, ways in street_types.iteritems():
    #    for name in ways:
    #        better_name = update_name(name, mapping)
            #print "name: ", name, " in st: ", st, " corrected to: ", better_name
    print "incomplete mappings: ", len(incomplete_mapping)
    print len(data), " records in json"
    print "db json @ ", file_out
    ##############################################
    ##### THIS CAN TAKE VERY LONG ################
    ##### IT IS ADVISED TO USE mongoimport INSTEAD###
    # ------------------------------------------------------------------------------
    # mongoimport --db udacity --collection dallas --file dallas_texas.osm.json  --drop
    # ------------------------------------------------------------------------------
    ##############################################
    #insert_mongo(data)

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

1.7.1 Lets check some data and clean up some more

In [48]:
```
'''
now that data is into mongo db, it is time to
explore some of it and clean it further if necessary
start with some basic statistics and run some
aggregations. then try to clean up more street names
'''

import pymongo
from pymongo import MongoClient
import re

#### global with incorrect street types as regular expressions
### this is a little different than lesson 6 problem set

mapping = { "\bSt": "Street",
            "\bSt\.": "Street",
            "\bRd\.": "Road",
            "\bRd": "Road",
            "\bN\.": "North",
            "\bAve": "Avenue",
            "\bTr" : "Trail",
            "\bTr\.": "Trail",
            "\bTrl": "Trail",
            "\bS\." : "South",
            "\bW\." : "West",
            "\bW" : "West",
            "\bS" : "South",
            "\bW" : "West",
            "\bPkwy" : "Parkway",
            "\bpkwy" : "Parkway",
            "\bHwy" : "Highway",
            "\bE" : "East",
            "\bE\." : "East",
            "\bBlvd" : "Boulevard",
            "\bBlvd\." : "Boulevard"
            }

#### connect to local mongodb instance and
#### returns a db

def connect():
    client = MongoClient()
    return client['udacity']


#### do some simple stats and print out the results
def simple_counts(db):
    dallas = db['dallas']
    records = dallas.count()
    print "total records: ", records
    collstats = db.command("collstats", "dallas")
    print "data size: ", collstats['size']
```

```
    first_record = dallas.find()[0]
    print "first record: ", first_record
    users = dallas.aggregate([\
        {"$match": {"created.user": {"$exists": 1}}},\
        {"$group": {"_id": "$created.user", "count": {"$sum":1}}},\
        {"$sort": {"count": -1}}\
                            ])
    print "Top 5 Users"
    i = 0
    for u in users:
        if i >= 5:
            break
        print "User ", 5-i, u
        i += 1

    zipcodes = dallas.aggregate([\
        {"$match": {"address.postcode": {"$exists": 1}}},\
        {"$group": {"_id": "$address.postcode", "count": {"$sum": 1}}},\
        {"$sort": {"count": -1}}\
                                ])
    print "Top 5 Zip Codes"
    i = 0
    for z in zipcodes:
        if i >= 5:
            break
        print "Zip Code ", 5-i, z
        i += 1

### run some other finds and try to clean up more data

def clean_street_names(db):
    dallas = db['dallas']
    address_records = dallas.find({"address.street": {"$exists": 1}})
    print "address records: ", address_records.count()
    for i in range(0, 5):
        record = address_records[i]
        print "record : ", i+1, " : ", record
        address_street = record['address']['street']
        print "address.street: ", address_street
        for k, v in mapping.iteritems():
            updated_street_name = re.sub(k, mapping[k], address_street, re.IGNORECASE)
            #print "Old Street Name: ", address_street, "| New Street: ", updated_street_name
            if not address_street == updated_street_name:
                dallas.update({"address.street": address_street}, {"$set": {"address.street": u
                print "updated street", address_street, " with new name ", updated_street_name
        print " "


#### main controller that brings all together
def run():
    db = connect()
    simple_counts(db)
    clean_street_names(db)
```

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

```
total records:  5558972
data size:  1293651238.0
first record:  {u'id': u'26450261', u'_id': ObjectId('56f1d4af222cc489da99b3f0'), u'type': u'node', u'po
Top 5 Users
User  5 {u'count': 2254674, u'_id': u'woodpeck_fixbot'}
User  4 {u'count': 198416, u'_id': u'fmmute'}
User  3 {u'count': 176820, u'_id': u'TexasNHD'}
User  2 {u'count': 123490, u'_id': u'25or6to4'}
User  1 {u'count': 121506, u'_id': u'Chris Lawrence'}
Top 5 Zip Codes
Zip Code  5 {u'count': 1211, u'_id': u'75104'}
Zip Code  4 {u'count': 629, u'_id': u'75093'}
Zip Code  3 {u'count': 343, u'_id': u'75070'}
Zip Code  2 {u'count': 227, u'_id': u'75051'}
Zip Code  1 {u'count': 181, u'_id': u'75069'}
address records:  119075
record :  1  :  {u'created': {u'changeset': u'15830886', u'version': u'3', u'user': u'starnix', u'timest
address.street:  Blue Ribbon Road

record :  2  :  {u'created': {u'changeset': u'15830886', u'version': u'3', u'user': u'starnix', u'timest
address.street:  Blue Ribbon Road

record :  3  :  {u'amenity': u'place_of_worship', u'name': u'Ash Creek Baptist Church', u'created': {u'c
address.street:  South Stewart Street

record :  4  :  {u'amenity': u'place_of_worship', u'name': u'Ash Creek Baptist Church', u'created': {u'c
address.street:  South Stewart Street

record :  5  :  {u'amenity': u'school', u'name': u'Crockett Middle School', u'created': {u'changeset': u
address.street:  Hancock Street
```