jaken551 / wrangling_osm

Branch: master | wrangling_osm / project_v1.md | Find file | Copy path

jaken551 Rename Untitled (1).md to project_v1.md be93cdb a minute ago

1 contributor

281 lines (215 sloc) 8.93 KB

Wrangling OpenStreetMap Data

1. Map Area Used:

Aurora, IL

https://www.openstreetmap.org/relation/124817#map=11/41.7308/-88.2422

I chose data from Aurora, Illinois because it is my hometown. I love being able to see data points that I have seen in real life as it really brings a much more realistic, intuitive feel to my project

2. Problems Encountered

The first problem I encountered was to be expected. There were several street abbreviations that needed to be standardized.

3. Cleaning The Data

In order to clean the data, I wanted to correct the abbreviated street names in the same process of turning the XML elements into JSON in an effort to be more efficient. To do so, I used the code below:

```
import xml.etree.cElementTree as ET
import pprint
import re
import codecs
import json


#Regular Expression for street types
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)


#Already Existing Fields in OSM Data
CREATED = ["uid", "version", "changeset", "timestamp", "user"]

#Mapping Of Street Names
MAPPING = { "St": "Street",
            "St.": "Street",
            "Ave": "Avenue",
            "Rd" : "Road",
            "Rd." : "Road",
            "Ln" : "Lane",
            "Dr" : "Drive",
            "Ct" : "Court",
            "Cir" : "Circle",
            "Blvd" : "Boulevard",
            "Blvd." : "Boulevard"
```

```
            }

    def update_name(name, mapping):
        '''
        This function cleans the street abbreviations.
        :param name:
        :param mapping:
        :return updated name:
        '''
        street_type = name.rsplit(' ')[-1]
        m = street_type_re.search(name)
        street_name = name.rsplit(' ', 1)[0]
        if street_type in mapping:
            name = street_name + ' ' + mapping[street_type]
        return name


    def shape_element(element):
        '''
        This function takes OSM XML Data and turns it into JSON objects.
        :param element:
        :return node:
        '''
        node = {}

        #Just Looking for node and way tags (see https://wiki.openstreetmap.org/wiki/Elements) for questions
        if element.tag == "node" or element.tag == "way":
            node['type'] = element.tag
            node['id'] = element.attrib['id']
            if 'visible' in element.attrib:
                node['visible'] = element.attrib['visible']
            if 'lat' in element.attrib and 'lon' in element.attrib:
                node['pos'] = []
                node['pos'].append(float(element.attrib['lat']))
                node['pos'].append(float(element.attrib['lon']))
            node['created'] = {}
            for i in CREATED:
                node['created'][i] = element.attrib[i]

            #Taking Special Consideration for the address as they are tagged as
            address = {}
            for i in element.iter('tag'):
                if 'addr:' in i.attrib['k']:
                    if 'street:' in i.attrib['k'].split(':', 1)[1]:
                        pass
                    else:
                        #Cleaning Street Abreviations by calling update_name()
                        address[i.attrib['k'].split(':', 1)[1]] = update_name(i.attrib['v'], MAPPING)
                else:
                    node[i.attrib['k']] = i.attrib['v']

            if address:
                node['address'] = address


            return node
        else:
            return None


    def process_map(file_in, pretty=False):
        '''
        This function returns write data to a JSON file
        :param file_in:
        :param pretty:
        :return JSON Data:
        '''
        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


data = process_map('aurora_il.osm', True)
```

4. Overview of Data

Size of Files:

aurora_il.osm : 169.9 MB aurora_il.osm.json : 220.6 MB

Number of Unique Users:

> db.aurora_il.distinct('created.user').length

Result: 796

Number of Nodes:

> db.aurora_il.find({"type":"node"}).count()

Result: 801984

Number of Ways:

> db.aurora_il.find({"type":"way"}).count()

Result: 74256

Exploring More of the Data:

Number of Restaurants:

> db.aurora_il.find({"amenity": "restaurant"}).count()

Result: 332

Top Ten Cuisines:

> db.aurora_il.aggregate({"$match":{"cuisine":{"$exists":1}}},{"$group":{"_id":"$cuisine","count":{"$sum":1}}}, {"$sort":{"count": -1}}, {"$limit":10})

Result: { "_id" : "burger", "count" : 67 } { "_id" : "pizza", "count" : 36 } { "_id" : "mexican", "count" : 35 } { "_id" : "sandwich", "count" : 26 } { "_id" : "coffee_shop", "count" : 22 } { "_id" : "chinese", "count" : 19 } { "_id" : "american", "count" : 15 } { "_id" : "italian", "count" : 12 } { "_id" : "ice_cream", "count" : 11 } { "_id" : "chicken", "count" : 9 }

Different Types of Cuisine:

> db.aurora_il.distinct("cuisine").length

Result: 48

Data Statistics:

The top user 'Umbugbene' makes up 31.8% of all the entries in the data set. Here is the detailed breakdown of the users and their contributions:

```
import pprint
from pymongo import MongoClient

def get_db(db_name):
    '''
    Setting Up MongoDB Connections
    :param db_name:
    :return db:
    '''
    client = MongoClient('localhost:27017')
    db = client[db_name]
    return db

def make_pipeline():
    pipeline = [{"$match":{"created.user":{"$exists":1}}},
                {"$group":{"_id":"$created.user","count":{"$sum":1}}},
                {"$sort":{"count": -1}},
                {"$project": {"perc": { "$multiply": [ { "$divide": ["$count", 876295] }, 100 ] }}},
                {"$limit": 30}]
    return pipeline

def aggregate(db, pipeline):
    return [doc for doc in db.aurora_il.aggregate(pipeline)]


if __name__ == '__main__':
    db = get_db('opendata')
    pipeline = make_pipeline()
    result = aggregate(db, pipeline)
    pprint.pprint(result)
```

```
[{'_id': 'Umbugbene', 'perc': 31.836424948219495},
 {'_id': 'woodpeck_fixbot', 'perc': 7.323218779064128},
 {'_id': 'alexrudd (NHD)', 'perc': 5.651749696163963},
 {'_id': 'patester24', 'perc': 4.852019011862444},
 {'_id': 'cowdog', 'perc': 4.694651915165555},
 {'_id': 'Deo Favente', 'perc': 4.4626524172795685},
 {'_id': 'jimjoe45', 'perc': 3.400110693316748},
 {'_id': 'mpinnau', 'perc': 3.240461260192059},
 {'_id': 'Mundilfari', 'perc': 3.1287408920511925},
 {'_id': 'TacoBeans44', 'perc': 2.44267056185417},
 {'_id': 'Marga_Dela', 'perc': 2.2137522181457157},
 {'_id': 'mappy123', 'perc': 1.6171494759184977},
 {'_id': 'TIGERcnl', 'perc': 1.5822297285731404},
 {'_id': 'knottga3', 'perc': 1.2809613201033898},
 {'_id': 'g246020', 'perc': 1.1441352512567116},
 {'_id': 'Mulad', 'perc': 1.0236278878688114},
 {'_id': 'Tharsis', 'perc': 0.7605886145647299},
 {'_id': 'asdf1234', 'perc': 0.6718057275232656},
 {'_id': 'raykendo', 'perc': 0.6695233910954644},
 {'_id': 'Sundance', 'perc': 0.62068139154052},
 {'_id': 'Ru55Ht', 'perc': 0.528474999857354},
 {'_id': 'maxerickson', 'perc': 0.5232256260734114},
 {'_id': 'bot-mode', 'perc': 0.4930987852264363},
 {'_id': 'Radek_trz', 'perc': 0.4547555332393771},
 {'_id': '42429', 'perc': 0.42782396339132367},
 {'_id': 'N219JK', 'perc': 0.4140158280031268},
 {'_id': 'BeatlesZeppelinRush', 'perc': 0.35102334259581536},
 {'_id': 'dhansen22', 'perc': 0.34874100616801423},
 {'_id': 'MGH', 'perc': 0.3269446932825133},
 {'_id': 'Kev-H', 'perc': 0.31119657193068545}]
```

5. Other Ideas About the Data Set:

The data set could be improved by making bulk inputted data easier to filter out. Bots and automatic data dumps are very present in the data set as can be seen above. Elements from National Hydrography Dataset (NHD) were dumped in this case mainly by alexrudd(NHD) but only discovered after close investigation. Also, Topologically Integrated Geographic Encoding and Referencing data (tiger) data is spread throughout the data set. (See below for queries). While this data is important to some, it takes away the importance of user inputted data. I think focusing on user-inputted data, and highlighting its existence would help improve the data.

```
db.aurora_il.find({"NHD:way_id":{"$exists":1}}).count()
```

815

```
db.aurora_il.find({"tiger:reviewed":{"$exists":1}}).count()
```

14335

6. Conclusion:

After extensively looking through the OpenStreetMap data, I have fonud how much of it is inputted by automatic methods. I would like to see more light shed on the user-inputted part of the data set. Although this data set is extremely large, there are obviously still room for improvement when it comes to maintaing consistency of the data set.

Sources

Project Examples

https://gist.github.com/carlward/54ec1c91b62a5f911c42

https://docs.google.com/document/d/1F0Vs14oNEs2idFJR3C_OPxwS6L0HPliOii-QpbmrMo4/pub

Udacity OpenStreetMap Case Study

https://www.youtube.com/watch?v=JyX6j00Q2Cg&feature=youtu.be

MongoDB Documentation

https://docs.mongodb.com/