OpenStreeMap data wrangling using Python and SQL

Selected area to study

Milladgeville GA, United States

https://www.openstreetmap.org/search?query=milledgeville%20GA#map=12/33.0800/-83.2321 (https://www.openstreetmap.org/search?query=milledgeville%20GA#map=12/33.0800/-83.2321)

I have selected the area of Milledgeville, GA to use in this study. This is the area I recently moved in. From this project I hope to understand the area better, so, it will help to ease my day to day life in the area while improving my programming skills.

Using mapzen.com (http://mapzen.com), I have downloaded the data of Milledgeville and surrounding area. The uncompressed size of the dataset is 294 mb.

Exploring the dataset and, looking for errors and problems in the data

To get a peek of data a small part of data set was examined (size 25 mb). In the sample dataset, following main and sub nodes were observed: bounds, node, tag, way, nd, relation, osm and member.

When examining the sample dataset, it is observed that most of the data was entered in proper way. In other words, there are less errors in the data. However, there are a few discrepancies in the street type data as in the following example.

- Morris Stevens Rd
- Montpelier Ave
- Turtle Shoals Rd NE
- US 41 (GA)

As above, in some street names, the street type is shortened. These type non standard entries need to be cleaned to get unified street types throughout the dataset.I leave out the last one (US 41 (GA)) as it is, because, it is probably the best way to define that type of streets.

To convert the different street types to a standard way, the following dictionary was generated.

In [ ]:
```
mapping = { "St": "Street",
                        "St." : "Street",
                        "Ave" : "Avenue",
                        "Rd." : "Road",
                        "Rd" : "Road",
                        "NE" : "North East",
                        "SE" : "South East",
                        "NW" : "North West",
                        "SW" : "South West",
                        "E" : "East",
                        "S" : "South",
                        "W" : "West",
                        "N" : "North"
                        }
```

The above dictionary was created based on the sample dataset, however, it should be updated for complete dataset.

To get discrepancies for the whole dataset if any, the function "audit_street_type2" was modified as follows. In this function, if the street types are different than the "expected" street type list (expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "Lane", "Road", "Trail", "Parkway", "Commons"]), the discrepant types are collected into the dictionary street_types2. So, the collected discrepancy list can be used to correct the errors later on.

In [ ]:
```
def audit_street_type2(street_types2, street_name):
        ## Split the street names
        street_splts = street_name.split()

        ## Get the last name of the streets. Generally it is the street type
        str_type = street_splts[len(street_splts) -1]
        if str_type not in expected and str_type not in mapping.keys():
                ## Get unique data only
                street_types2[str_type] = street_name
```

Cleaning and updating the dataset

Once updated the "mapping" dictionary, the standardizing of street types can be done as in following function.

In [ ]:
```
def update_street_name(name, mapping):
        nme = str()
        splitted_name = name.split()
        #print(splitted_name)
        for i, nam in enumerate(splitted_name):
                if nam in mapping.keys():
                        splitted_name[i] = mapping[nam]
        for nm in splitted_name:
                nme += nm+' '

        name = nme.rstrip()
        return name
```

By implementing the Python program with above developed methodologies, all the encountered errors for the sample data set were addressed and exported the cleaned data into .csv format. The developed program was used to run the entire dataset.

For the whole dataset, following additional discrepancies were observed that could not find in the sample dataset.

For the road types,

- Holiday Cir
- Earnest Biles Dr
- Tommy Stalnaker Dr
- I-475 Industrial Blvd
- Watson Blvd
- Industrial Highway
- Pearl Stephens Way

and for the zip codes.

- Macon GA 31217

After fixing these discrepancies, following updated list and dictionary were obtained for the street types. With the corrected data, the .csv files were generated for entire dataset. The generated .csv files are nodes.csv, node_tags.csv, ways.csv, way_tags.csv and way_nodes.csv.

In [ ]:
```
expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Squar
e", "Lane", "Road",
                          "Trail", "Parkway", "Commons", "Highway", "Circle", "W
ay"]

mapping = { "St": "Street",
                          "St." : "Street",
                          "Ave" : "Avenue",
                          "Rd." : "Road",
                          "Rd" : "Road",
                          "NE" : "North East",
                          "SE" : "South East",
                          "NW" : "North West",
                          "SW" : "South West",
                          "E" : "East",
                          "S" : "South",
                          "W" : "West",
                          "N" : "North",
                          "Cir" : "Circle",
                          "Dr" : "Drive",
                          "Blvd": "Boulevard"
                          }
```

Analysing data with SQL codes

Since, the data is in .csv formats now, it is possible to import .csv file to database. Therefore, the data analyzing part carried out using SQL codes, hereafter. I used MySQL Workbench to run the required quarries.

Data in .csv files can be important to the database using following example SQL codes.

Create a fresh table vith empty column

In [ ]:
```
CREATE TABLE nodes (
        changeset INT NOT NULL,
        nod_id DOUBLE NOT NULL,
    user_nm VARCHAR(50) NOT NULL,
    uid INT NOT NULL,
    lon DECIMAL(30, 26) NOT NULL,
    lat DECIMAL(30, 26) NOT NULL,
    PRIMARY KEY (nod_id)
    );
```

Load the data to the table from a .csv file

In [ ]:
```
LOAD DATA LOCAL INFILE 'F:/Final_project/nodes.csv'
INTO TABLE nodes
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

Number of nodes and way nodes

In [ ]:
```
## Number of nodes
SELECT COUNT(user_nm) AS way_count
FROM nodes;

## Number of way nodes
SELECT COUNT(user_nm) AS way_count
FROM ways;
```

There are 1538489 nodes and 83571 way nodes.

Available elementary schools

I am interested in the names of elementary schools in the area. therefore, the following quarry extracts the names of elementary schools in the area.

In [ ]:
```
## Select elementary schools in the area
SELECT v AS Elemtry_School
FROM node_tags
WHERE k = 'name' AND v LIKE '%Elementary School%'
ORDER BY Elemtry_School;
```

Elementary Schools
Banks-Stephens Elementary School
Butler-Baker Elementary School
Byron Elementary School
Centerville Elementary School
Davis Elementary School (Historical)
Eagle Springs Elementary School
Gray Elementary School
Jeffersonville Elementary School
Jessie Rice Elementary School
Midway Elementary School
Miller Elementary School
Northside Elementary School
Old Gray Elementary School
Pearl Stephens Elementary School
Porter Elementary School
Springdale Elementary School
W T Morgan Elementary School
Walter P Jones Elementary School
Wilkinson County Elementary School

First 20 contributors who has put effort in this dataset

In [ ]:
```
CREATE TABLE user_summary AS
SELECT user_nm, SUM(usr_cunt) AS user_count
FROM    (SELECT user_nm,COUNT(user_nm) AS usr_cunt
                FROM ways
        GROUP BY user_nm
        UNION ALL
                SELECT user_nm, COUNT(user_nm) AS usr_cunt
        FROM nodes
        GROUP BY user_nm)  ways_nodes
GROUP BY user_nm
ORDER BY user_count DESC;

SELECT user_nm, user_count, (user_count/tot)*100 AS percentage
FROM user_summary
CROSS JOIN (SELECT SUM(user_count) AS tot
                        FROM user_summary) total
LIMIT 20;
```

User name	User count	Percentage
Liber	986938	60.8447
macon_cfa	259995	16.0287
woodpeck_fixbot	132242	8.1527
ROJordan	38790	2.3914
"Chris Lawrence"	24939	1.5375
maxerickson	22088	1.3617
mapmeld	19209	1.1842
mwebb97	17569	1.0831
maven149	11431	0.7047
ediyes	9813	0.6050
"Jack the Ripper"	7009	0.4321
DKNOTT	6586	0.4060
EdLoach	6169	0.3803
bot-mode	5694	0.3510
ELadner	5650	0.3483
jacobbraeutigam	4943	0.3047
DaveHansenTiger	3977	0.2452
samely	3247	0.2002
Kirbert	2987	0.1841
Kla_Ger	2916	0.1798

- Based on above data the user 'Liber' has the main contributor with about 61 % of data entry. Followed by 'macon_cfa' with 16 % contribution.

Number of unique users

In [ ]:
```
SELECT COUNT(username) AS usercount
FROM (SELECT DISTINCT(user_nm) AS username
        FROM user_summary) usecont;
```

Number of unique users are 331.

Top 10 zip codes

The dataset contains the information about following main zipcodes.

In [ ]:
```
SELECT v, COUNT(v) AS zipcodes
FROM (SELECT v
        FROM node_tags
        WHERE k LIKE '%post%'
        UNION ALL
        SELECT v
        FROM way_tags
        WHERE k LIKE '%post%') zipcds
GROUP BY v
ORDER BY zipcodes DESC
LIMIT 10;
```

Zip code	Count
31210	17
31201	13
31216	12
31061	12
31206	10
31093	10
31029	7
31032	7
31088	5
31217	5

Size of the data files

File name	Size	Type
Milledgeville_GA.osm	294 mb	xml
nodes.csv	83.5 mb	csv
node_tags.csv	1.3 mb	csv
ways.csv	2.9 mb	csv
way_tags.csv	12.6 mb	csv
way_nodes.csv	30.2 mb	csv
milledgeville_dbs	230 mb	MySQL database files folder

Additional improvements

The selected Milledgeville area should have the zip codes starting 31***. However, an inappropriate zip code was observed. The SQL quarry that extract the un matching zipcodes is given below.

In [ ]:
```
## Inappropriate zip codess
SELECT (v) AS Zipcode, COUNT(v) AS counts
FROM (SELECT v
        FROM node_tags
        WHERE k LIKE '%post%' AND v NOT like '%3%'
        UNION ALL
        SELECT v
        FROM way_tags
        WHERE k LIKE '%post%'  AND v NOT like '%3%') zipcds
GROUP BY Zipcode;
```

Zipcode	counts
51201	1

So, the questioning zip code belongs to some are in Iowa state, but the area of interest is in Georgia state. So, this might be a data entry mistake. Therefore, by writing quarries like above to identify errors like this, the data set can be improved.

Conclusion

Python codes were written to wrangling the data from OpenStreetMap. The cleaned data exported to .csv format and the data in .csv formats later exported to a MySQL database. While, the overall data quality is good, some small discrepancies observed. In this project, I specially paid attention on cleaning and standardizing street types and zip codes using techniques in Python program. In addition, I have written some SQL quarries to extract, summarize, and get out some statistics in the dataset using techniques in SQL language.

In [ ]: