P3 – Data Wrangling – OpenStreetMap Project

Map Area:

Chennai, Tamil Nadu, India

I have chosen the particular map data cause it's my home city and would really be excited to contribute my findings in OpenStreetMap.org, the map data has been obtained from the below mentioned websites.

- https://www.openstreetmap.org/relation/1766358
- https://mapzen.com/data/metro-extracts/metro/chennai_india/

The data sets have been downloaded as Raw OpenStreetMap datasets in OSM XML format and the datasets have been reduced to an appropriate systematic sample data by using the 'k' value as 7.

Steps Taken:

The following steps have been taken to clean the raw data and explore them;

1. *MapParser and Tags*
2. *Data Audit*
3. *Data Conversion and Overview*
4. *Data Exploration*
5. *Conclusion*

Step 1 – MapParser and Tags

In this step, I parse the OSM XML file to find out what tags are present and also count them. After parsing the OSM XML file, I get a dictionary with the tag name as the key and the number of times this tag can be encountered.

Output:

```
{'member': 512,
 'nd': 325567,
 'node': 261982,
 'osm': 1,
 'relation': 136,
 'tag': 66396,
 'way': 58731}
```

mapparser-step1.py is used to identify and count the tags.

Now, I analyze the dataset a bit more and hence, I check the 'k' value tag and find out if there are any potential problems in the dataset. We find out that there are certain patterns in the data set by creating four expressions such as lower, lower_colon, problemchars and others.

lower – for tags that contains lower case letter and are valid.
lower_colon - for otherwise valid tags with a colon in their names.
problemchars - for tags with problematic characters.
others - for other tags that do not fall into the other three categories.

tags-step2.py is used to check the data.

Output:

```
{'lower': 65442,
 'lower_colon': 868,
 'other': 85,
 'problemchars': 1}
```

Step 2 – Data Audit

In this step, I audit the data and find the various inconsistencies occurred in my data set, for example the street names have common errors associated with them.

Problems encountered in the DATASET:

A few problems faced in the dataset are mentioned below; The old names along with the corrected names are provided.

- *Spelling Errors*

  strret => Street

- *Regional Names*

  slai => Salai

  nagar => Nagar

- *Abbreviated Street Names*

  Rd => Road

  St => Street

Postal Codes:

Although a majority of the postal codes provided were of the actual inputs and also according to the normalized Indian standards, a few of these inputs had human errors associated with them. For example, 6000010 has 7 digits whereas the postal code should contain only 6 digits and hence these errors had to be dropped from the database. In the audit.py, I have these errors rectified.

SQL Code (Postal Codes):

```
sqlite> SELECT tags.value, COUNT(*) as count
   …> FROM (SELECT * FROM nodes_tags UNION ALL
   …> SELECT * FROM ways_tags) tags
   …> WHERE tags.key='postcode'
   …> GROUP BY tags.value
   …> ORDER BY count DESC
   …> LIMIT 5;
```

Output:

```
600015   11
600017   8
600042   7
600041   6
600032   5
```

Although a majority of the problems encountered in the datasets are only of the above mentioned four. Using ***audit.py*** I have rectified the errors and updated the names as given above.

Step 3 – Data Conversion and Overview

After auditing the data, I convert the OSM XML files into CSV files in order to insert it into a SQL Database.

The process for this transformation to take place is as follows:

- Use iterparse to iteratively step through each top level element in the XML.
- Shape each element into several data structures using a custom function.
- Utilize a schema and validation library to ensure the transformed data is in the correct format.
- Write each data structure to the appropriate .csv files.

The different files in which the data is split is mentioned as follows:

- node
- node_tags
- way
- way_nodes
- way_tags

Overview: (Data File Size)

- Chennai_india.osm = 56.4 MB
- chennai.db = 30.1 MB
- node.csv = 21.1 MB
- node_tags.csv = 141 KB
- way.csv = 3.46 MB
- way_nodes.csv = 7.75 MB
- way_tags.csv = 2.03 MB

The .csv files are now converted into a database with the appropriate tables and schema provided by Udacity.

Step 4 – Data Exploration

In this step I explore the chennai.db database using SQLITE3, it also contains various SQL queries I have used to obtain the output.

Number of Nodes:

```
sqlite> SELECT COUNT(*) FROM nodes;
```

Output: 261982

Number of Ways:

```
sqlite> SELECT COUNT(*) FROM ways;
```

Output: 58731

Number of Unique Users:

```
sqlite> SELECT COUNT(DISTINCT(e.uid))
   ...> FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e;
```

Output: 670

Top 15 Contributing Users:

```
sqlite> SELECT e.user, COUNT(*) as num
   ...> FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
```

```
...> GROUP BY e.user
...> ORDER BY num DESC
...> LIMIT 15;
```

Output:

maheshrkm	23724
PlaneMad	15107
venkatkotha	13399
shekarn	7881
praveeng	7873
sramesh	7656
anthony1	7584
Rahuldhanraj	7520
kushwanth	7468
venkatesh10	7098
saikumar	6754
rajureddyvudem	6715
ravikumar1	6548
harishvarma	6512
vamshiN	6309

Number of Users contributing only once:

```
sqlite> SELECT COUNT(*)
   ...> FROM
   ...> (SELECT e.user, COUNT(*) as num
   ...> FROM(SELECT user FROM nodes UNION ALL SELECT user FROM ways)e
   ...> GROUP BY e.user
   ...> HAVING num=1)u;
```

Output: 225

Additional Data Exploration

Top 10 Common Amenities:

```
sqlite> SELECT value, COUNT(*) as num
   ...> FROM nodes_tags
   ...> WHERE key='amenity'
   ...> GROUP BY value
   ...> ORDER BY num DESC
   ...> LIMIT 10;
```

Output:

restaurant	52
place_of_worship	45
atm	26
bank	24
pharmacy	20
school	17
fuel	16
bus_station	12
hospital	12
library	11

Top 2 Biggest Religion:

```
sqlite> SELECT nodes_tags.value, COUNT(*) as num
   ...> FROM nodes_tags
   ...> JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='place_of_worship')i
   ...> ON nodes_tags.id=i.id
   ...> WHERE nodes_tags.key='religion'
   ...> GROUP BY nodes_tags.value
   ...> ORDER BY num DESC
   ...> LIMIT 2;
```

Output:

Hindu	26
Christian	6

Top 5 Popular Cuisine:

```
sqlite> SELECT nodes_tags.value, COUNT(*) as num
   ...> FROM nodes_tags
   ...> JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='restaurant')i
   ...> ON nodes_tags.id=i.id
   ...> WHERE nodes_tags.key='cuisine'
   ...> GROUP BY nodes_tags.value
   ...> ORDER BY num DESC
   ...> LIMIT 5;
```

Output:

Regional	6
Indian	5
north_indian	2
Arabian	1
International	1

Conclusion

The OpenStreetMap data of Chennai, TN, India is of fair quality, this I can conclude due to the amount of unique contributors. Apart, from the typical grammatical errors present in the data set the information contributed is not very accurate, this may be due to information mismatch occurred when providing inputs. Moreover, there is very less information about the basic amenities provided in the data set, this may be due to the users providing very less information about the actual amenities such as the restaurants, tourist places and popular hangouts present in the city.

Suggestion's and Ideas:

- We need to control the grammatical errors by introducing some rules and guidelines through which a contributor can follow up his contributions and rectify his errors if need arises.
- Need to clean the data at regular intervals in order to make it more sufficient and dependable. This can be achieved by writing scripts which follow a SOP to clean the data at regular intervals.
- More information regarding basic amenities can be made available to the users by asking the users to contribute towards the data similar to what Google Maps does.

Limitation's:

There are a few limitations' in implementing my suggestions, for example, in my particular dataset we can notice that there 670 unique users out of which 225 have contributed only once, the users who have contributed only once will never bother to respond to the rules and guidelines, they may not bother to contribute more. Moreover, cleaning and maintaining the data at regular intervals might incur additional cost and may not be operationally feasible as each and every Country has its own unique city names, postal codes, phone number and addresses.

Attachments:

1. mapparser-step1.py – Parse Map
2. tags-step2.py – Find Tags and Counts
3. audit.py – Audit street names and postal code
4. data-step5.py – Code to build .csv files from OSM sample
5. databasecreation.py – Code to create the database
6. schema.py – Sample Schema
7. P3 OpenStreetMap.pdf

8. chennai_india.osm – Sample OSM file
9. chennai1.db

References:

1. https://www.openstreetmap.org/relation/1766358
2. https://mapzen.com/data/metro-extracts/metro/chennai_india/
3. https://gist.github.com/swwelch/f1144229848b407e0a5d13fcb7fbbd6f
4. https://gist.github.com/carlward/54ec1c91b62a5f911c42#file-sample_project-md
5. http://stackoverflow.com/questions/2887878/importing-a-csv-file-into-a-sqlite3-database-table-using-python
6. http://www.sqlitetutorial.net/sqlite-import-csv/
7. https://udacity.atlassian.net/wiki/display/BENDH/RDB+Lesson+4+Reference+Notes