Project 3: OpenStreetMap Data Wrangling with SQL

Name: Pratyush Kumar
Map Area: I have choosen Ahmedabad as it is near to my hometown.

- Location: Ahmedabad, India
- OpenStreetMap URL
- MapZen URL

1. Data Audit

Unique Tags

Looking at the XML file, I found that it uses different types of tags. So, I parse the Ahmedabad,India dataset using ElementTree and count number of the unique tags.
`mapparser.py` is used to count the numbers of unique tags.

- `'bounds': 1`
- `'member': 2298,`
- `'nd': 608543,`
- `'node': 525565,`
- `'osm': 1,`
- `'relation': 526,`
- `'tag': 91356,`
- `'way': 77866`

Patterns in the Tags

The `"k"` value of each tag contain different patterns. Using `tags.py`, I created 3 regular expressions to check for certain patterns in the tags.
I have counted each of four tag categories.

- `"lower" : 90228`, for tags that contain only lowercase letters and are valid,
- `"lower_colon" : 1104`, for otherwise valid tags with a colon in their names,
- `"problemchars" : 16`, for tags with problematic characters, and
- `"other" : 0`, for other tags that do not fall into the other three categories.

2. Problems Encountered in the Map

Street address inconsistencies

The main problem we encountered in the dataset is the street name inconsistencies. Below is the old name corrected with the better name. Using `audit.py`, we updated the names.

- **Abbreviations**
  - `Rd -> Road`
- **LowerCase**
  - `gandhi -> Gandhi`
- **Misspelling**
  - `socity -> Society`
- **Hindi names**
  - `rasta -> Road`
- **UpperCase Words**
  - `sbk -> SBK`

City name inconsistencies

Using `audit.py`, we update the names

- **LowerCase**
  - `ahmedabad -> Ahmedabad`
- **Misspelling**

  - Ahmadabad -> Ahmedabad

3. Data Overview

File sizes:

- ahmedabad_india.osm: 109.2 MB
- nodes_csv: 43.4 MB
- nodes_tags.csv: 163.4 KB
- ways_csv: 4.6 MB
- ways_nodes.csv: 14.7 MB
- ways_tags.csv: 2.8 MB
- ahmedabad.db: 78.5 MB

Number of nodes:

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

Output:

```
525565
```

Number of ways:

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

Output:

```
77866
```

Number of unique users:

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

Output:

```
226
```

Top 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 10;
```

Output:

```
uday01       177686
sramesh      136887
chaitanya110   123328
shashi2      49514
vkvora       22216
shravan91     21508
shiva05      19671
bhanu3       12645
Oberaffe     7042
PlaneMad      4969
```

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:

```
44
```

4. Additional Data Exploration

Common ammenities:

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

Output:

```
place_of_worship   47
restaurant        31
bank             21
school           18
fuel            14
library          14
hospital         13
cafe            12
fast_food        11
cinema           10
```

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 1;
```

Output:

```
Hindu : 31
```

Popular cuisines

```
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;
```

Output:

```
regional                   4
vegetarian                  3
pizza                     2
Punjabi,_SouthIndia,_Gujarati Thali    1
burger                     1
indian                     1
international                1
italian                    1
sandwich                    1
```

5. Conclusion

The OpenStreetMap data of Ahmedabad is of fairly reasonable quality but the typo errors caused by the human inputs are significant. We have cleaned a significant amount of the data which is required for this project. But, there are lots of improvement needed in the dataset. The dataset contains very less amount of additional information such as amenities, tourist attractions, popular places and other useful interest. The dataset contains very old information which is now incomparable to that of Google Maps or Bing Maps.

So, I think there are several opportunities for cleaning and validation of the data in the future.

Additional Suggestion and Ideas

Control typo errors

- We can build parser which parse every word input by the users.
- We can make some rules or patterns to input data which users follow everytime to input their data. This will also restrict users input in their native language.
- We can develope script or bot to clean the data regularly or certain period.

More information

- The tourists or even the city people search map to see the basic amenities provided in the city or what are the popular places and attractions in the city or near outside the city. So, the users must be motivated to also provide these informations in the map.
- If we can provide these informations then there are more chances to increase views on the map because many people directly enter the famous name on the map.

Files

- `Quiz/` : scripts completed in lesson Case Study OpenStreetMap
- `README.md` : this file
- `ahmedabad_sample.osm` : sample data of the OSM file
- `audit.py` : audit street, city and update their names
- `data.py` : build CSV files from OSM and also parse, clean and shape data
- `database.py` : create database of the CSV files
- `mapparser.py` : find unique tags in the data
- `query.py` : different queries about the database using SQL
- `report.pdf` : pdf of this document
- `sample.py` : extract sample data from the OSM file
- `tags.py` : count multiple patterns in the tags