Wrangle OpenStreetMap data

---

1. Problems encountered in the map

- Street addresses were inconsistent. For example, *“3rd mn rd”* actually means *“3rd Main Road”* etc.
- Some content was in local language called Kannada which was unreadable and had a different encoding.
- I live in this city and a lot of street names that I know have been misspelled as it is user entered data. I could manually clean them for few places.

---

2 a. Overview of the data

- The actual metro extract data is 648.7 MB which can be downloaded from here : https://s3.amazonaws.com/metro-extracts.mapzen.com/bengaluru_india.osm.bz2
- It took lot of time to compute and generate data (waited for almost an hour) , therefore ran this on a sample of the actual metro extract.
- The intermediate files generated :
  - bi_nodes_tags.csv : 2.3 MB
  - bi_nodes.csv : 2 MB

```
SBs-MacBook-Pro:4_Wrangle_OpenStreetMap_data sb$ python e_query_data.py

The largest leisure club :-  ('sports_centre', 32)

Total number of nodes are :-  24934

Total number of ways are :- 157

Total number of unique users :-  353
```

```
The top most contributing users :-  [('PlaneMad', 3856), ('Divjo',
3623), ('Guillaume Audirac', 2206), ('Praveen', 1758), ('user_634020',
1373), ('docaneesh', 1281), ('lawgoff', 813), ('Gururaja Upadhyaya',
808), ('PR_Mapper', 738), ('indigomc', 667)]
```

---

2 b. Database Queries used

To calculate total number of unique users :

SELECT COUNT(DISTINCT(allusers.uid)) FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) allusers

To calculate total number of nodes :

SELECT COUNT(*) FROM nodes

To calculate total number of ways :

SELECT COUNT(*) FROM ways

To calculate biggest leisure club :

SELECT value, COUNT(*) as num FROM nodes_tags WHERE key="leisure" \

GROUP BY value ORDER BY num DESC LIMIT 10

To calculate most contributing users :

SELECT allusers.user, COUNT(*) as num FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) allusers GROUP BY allusers.user ORDER BY num DESC LIMIT 10

3. Other Ideas about the dataset

- Each area of locality can be considered as subset of data, which in turn requires careful data wrangling tests for the subset data. This becomes very big project to work on.

- To understand the transport scenario of Bengaluru we can investigate the number of highways in the city. The approach is to execute to the following steps :
  - Parse and record data to extract tags '*tag'* with '*highway'* attributes.
  - This in turn should be grouped by the unique '*highway'* values.

- The user entered data should be validated from henceforth because it turns out to be very complex to wrangle if the street names and street codes are written in its own ways. There is a need to standardise it. This can be achieved in following ways :
  - The UI or UX should be simple and clutter-free while collecting data. It should also provide auto-suggest features while the user is typing out details. This ensures better collection of data.
  - A small data wrangling script in the data collection application can make sure zip codes and other encoding type errors to stay away.

    - **Benefits of the suggestion** : Less burden on data scientists to spend too much time on the wrangling.
    - **Anticipated problems** : It might affect performance of data collection app. There might be cases when the user-entered data isn't in the auto-suggest. There is a need to verify the details of user entered data to make sure it is true and factual. Probably users can take pictures of the street address boards or address mentioned in those

boards, using ML techniques we can verify the data. Or just gather and generate data from pictures itself.

---

4. Audited info examples

```
tomap = {
        "Blr": "Bengaluru",
        "bangalore": "Bengaluru",
        "Bangalore": "Bengaluru",
        "nagar": "Nagar",
        "Ngr": "Nagar",
        "Rd.": "Road",
        "road": "Road",
        "Rd,": "Road,",
        "road,": "Road,",
        "road": "Road",
        "Rd": "Road",
        "layout,": "Layout,",
        "layout": "Layout",
        "main": "Main Road",
        "main rd.": "Main Road",
        "Main": "Main Road",
        "Main Road": "Main Road",
    }

exp = ["Bengaluru", "Colony", "Layout", "Road", "Nagar", "Main Road”]
```

```
def street_audit(street_type, street_name):
    match = reg.search(street_name)
    if match:
        street_t = match.group()
        if street_t not in exp:
            street_type[street_t].add(street_name)
```

The above function looks for similar street type words and replaces it based on the dict. For example, as shown in table below :

Original data	After auditing
Yelechenahalli mn rd.	Yelechenahalli Main Road
Anjaniputra Ngr	Anjaniputra Nagar
Dollors colony	Dollors Colony
Blr - 560099	Bengaluru - 560099