Project 3: OpenStreetMap Data Case Study

Map Area

Bengaluru, Karnataka, INDIA

https://mapzen.com/data/metro-extracts/metro/bengaluru_india/

Bengaluru is considered as the IT capital of India, The city saw tremendous growth and industrial revolution in the last decade. It also hosts many fortune 500 companies; this made me pick this city's map to do my case study.

Problems Encountered In The Map

I used the provided code, To make a OSM file for a smaller area (Using a larger value of k). In the initial inspection, I found the following problem with the map while auditing the data.

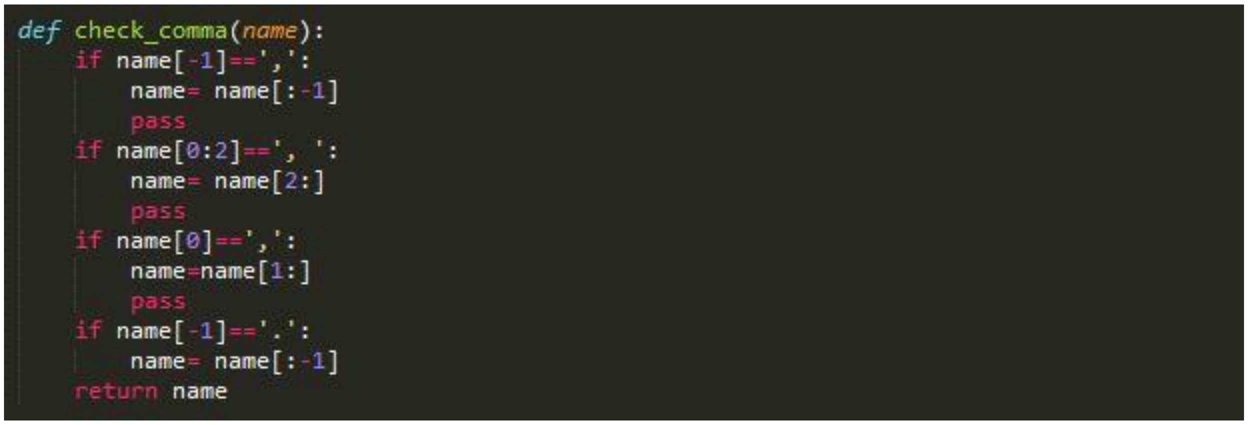
- Incorrect postal codes
- Abbreviated street names
- Errors in street tag
- Full address of the city in street tag
- Inconsistency in city tag

Errors in Street Tag:

I removed some errors while auditing the street tag. Here are some examples:

1. Comma in the end
2. Comma in the beginning
3. Full stop in the end

Solution:
We have used following function (check_comma) to remove un-necessary commas in the street tag. For example in our first 'if-statement' if we find comma in last position [-1], we update the name with all except last character [:-1].



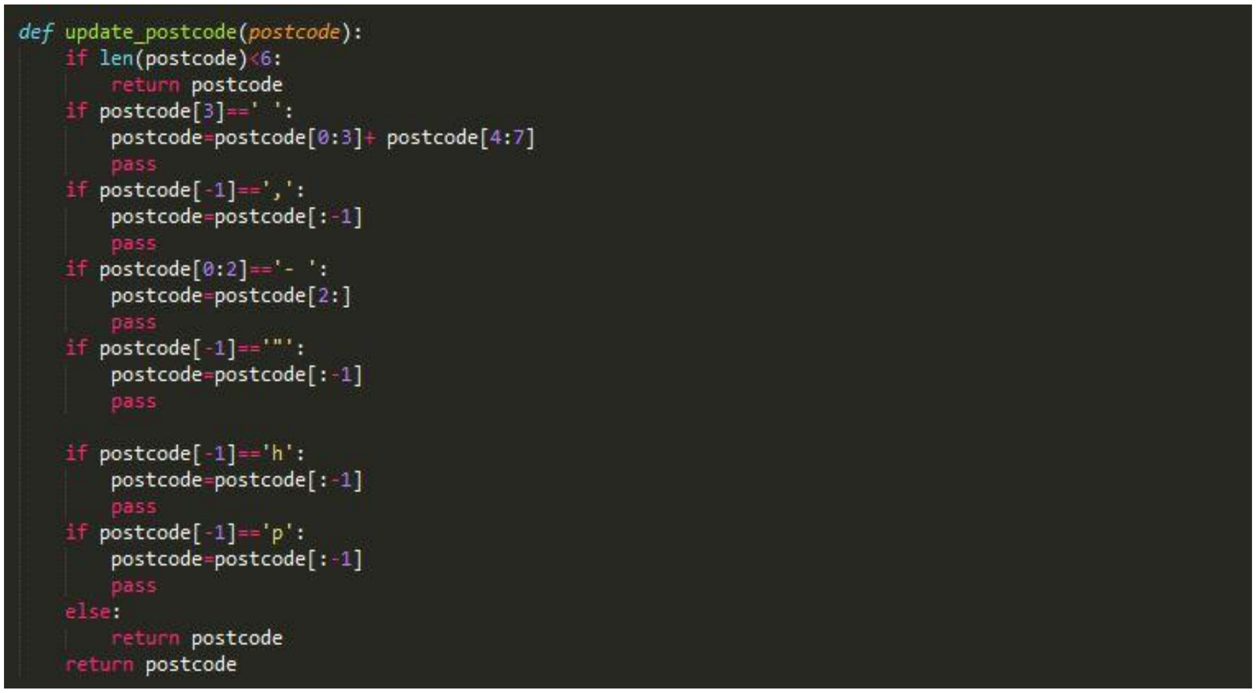
```
def check_comma(name):
    if name[-1]==',':
        name= name[:-1]
        pass
    if name[0:2]==', ':
        name= name[2:]
        pass
    if name[0]==',':
        name=name[1:]
        pass
    if name[-1]=='.':
        name= name[:-1]
    return name
```

Incorrect Postal Code:

I found much discrepancy while auditing the postal code. Some examples are mentioned below:

1. 560 001
2. – 560094
3. Bengaluru
4. 5600011
5. 560027"
6. 56006
7. 560001ph
8. iam in bang

Solution:
I corrected the postal code where it was possible, but where the postal code was incomplete or had a text in place, I removed it while updating the database. For example in our first 'if-statement' if postcode is less than 6 digit, we ignore it and return postcode as it is so that we can remove it later. We can also see how in 2nd if statement if there is a comma in last position [-1] we update the postcode with all except last position [-1], Similar if–statements are used for all our issues.



```
def update_postcode(postcode):
    if len(postcode)<6:
        return postcode
    if postcode[3]==' ':
        postcode=postcode[0:3]+ postcode[4:7]
        pass
    if postcode[-1]==',':
        postcode=postcode[:-1]
        pass
    if postcode[0:2]=='- ':
        postcode=postcode[2:]
        pass
    if postcode[-1]=='"':
        postcode=postcode[:-1]
        pass

    if postcode[-1]=='h':
        postcode=postcode[:-1]
        pass
    if postcode[-1]=='p':
        postcode=postcode[:-1]
        pass
    else:
        return postcode
    return postcode
```

Abbreviated Street Names:

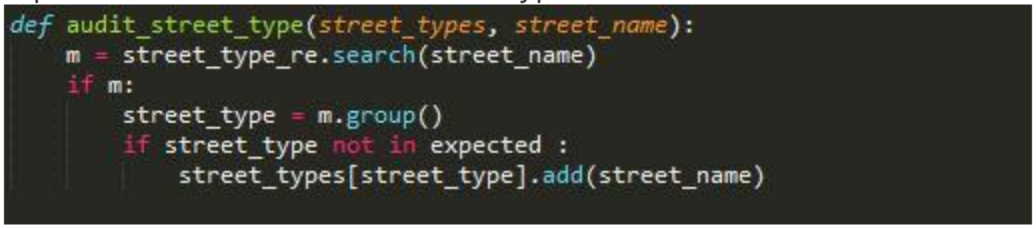
I found some street names and words to be abbreviated in street tag. Here are some of the examples:

1. "st": "Street"
2. "rd": "Road"
3. "blk": "Block"

I updated these abbreviated words respectively.

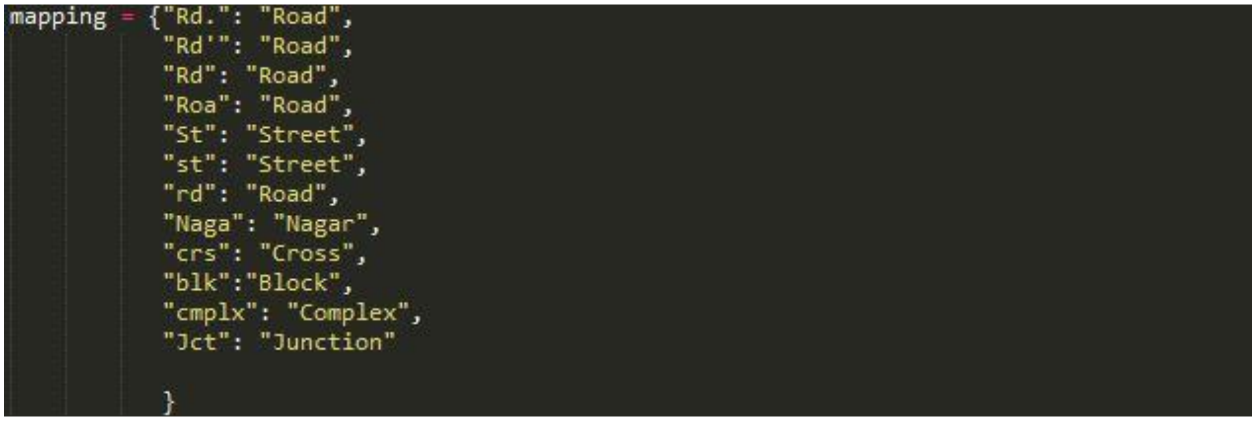
Solution:

I used the audit code that I prepared during udacity tutorial for updating abbreviated street names. We identified street types with below function if street name was not in expected list we add it to our street type list



```
def audit_street_type(street_types, street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type not in expected :
            street_types[street_type].add(street_name)
```

We can update the mapping list, It contains all the street names which are abbreviated and will be updated with update_name function.



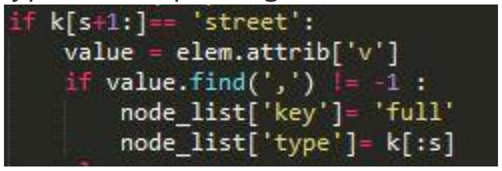
```
mapping = {"Rd.": "Road",
            "Rd'": "Road",
            "Rd": "Road",
            "Roa": "Road",
            "St": "Street",
            "st": "Street",
            "rd": "Road",
            "Naga": "Nagar",
            "crs": "Cross",
            "blk":"Block",
            "cmplx": "Complex",
            "Jct": "Junction"

            }
```

Full address instead of street name:

While auditing the street tag, I noticed that most values in the tag contain full address instead of just the street name. Here are some examples:

1. Main, I Phase, V Stage, BEML Layout, RajaRajeshwari Nagar
2. 1st Cross, 1st Main, 1st Phase, 5th Stage, BEML Layout
3. 13th cross, R.A. Road, Ejipura

Solution:
I found this problem in most of the tags to be corrected manually. To get the best of our map data, where ever I found comma in the street tag, I changed “street” to “full” type while updating the database. The below code can be found in data.py file.



```
if k[s+1:]== 'street':
    value = elem.attrib['v']
    if value.find(',') != -1 :
        node_list['key']= 'full'
        node_list['type']= k[:s]
```

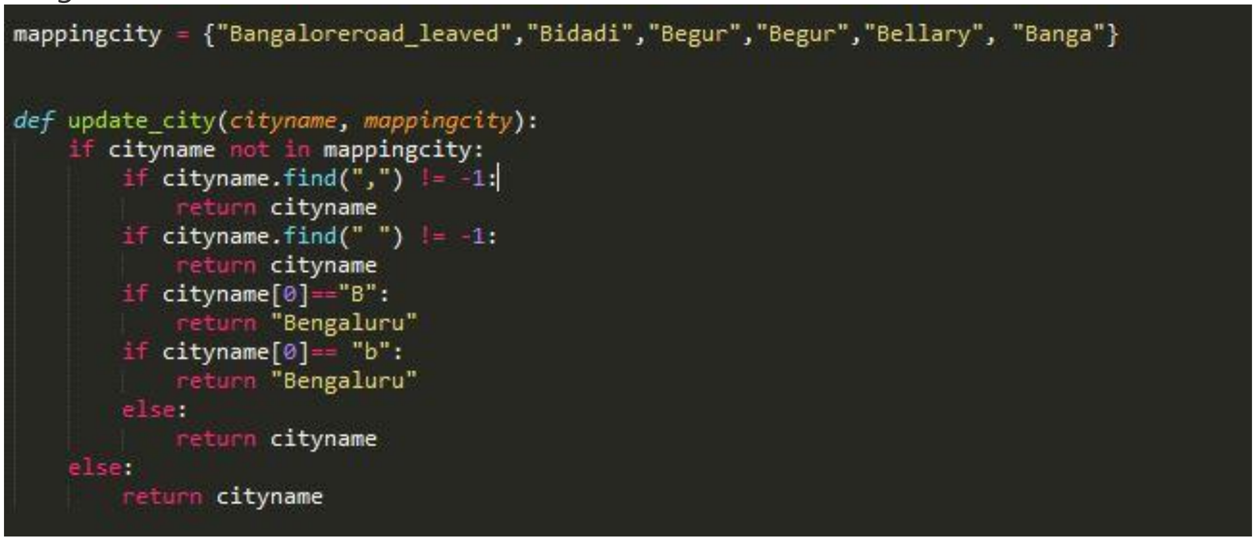
SPECIAL NOTE: The above change was done in recommendation from Udacity Forum Mentor, Here is the link of the details:
https://discussions.udacity.com/t/laptop-freezes-while-running-the-code/219216/5?u=deependranimiwal

Inconsistency in city tag

While auditing the city tag, I noticed some inconsistency:

1. Bangalore
2. bengaluru
3. Whitefield, Bangalore
4. Bellary
5. Whitefield

Solution:
Bangalore was the old city name which now has been named Bengaluru, Hence both of these are correct and used popularly. I also found some tags to have area information attached. I decided not remove this information as Bengaluru is a big metro city. For consistency purposes I updated city name to be “Bengaluru” in most of the cases and ignored others. We removed commas from front and end of the string. If cityname was not mapping city we updated all city name with first character ‘b’ or ‘B’ name to Bengaluru.



```
mappingcity = {"Bangaloreroad_leaved","Bidadi","Begur","Begur","Bellary", "Banga"}


def update_city(cityname, mappingcity):
    if cityname not in mappingcity:
        if cityname.find(",") != -1:
            return cityname
        if cityname.find(" ") != -1:
            return cityname
        if cityname[0]=="B":
            return "Bengaluru"
        if cityname[0]== "b":
            return "Bengaluru"
        else:
            return cityname
    else:
        return cityname
```

Data Overview And Additional Ideas

This section contains basic statistics about the dataset, the SQL queries used to gather them, and some additional ideas about the data in context.

FILE SIZES

bengaluru_india.osm:	612 MB
bengaluru_india.db:	350 MB
nodes.csv:	230 MB
nodes_tags.csv:	3.28 MB
ways.csv:	38.6 MB
ways_tags.csv:	23.3 MB
ways_nodes.cv:	84.4 MB

NUMBER OF NODES

```
sqlite> SELECT COUNT(*) FROM nodes;
```
2855613

NUMBER OF WAYS

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

654455

NUMBER OF UNIQUE USERS

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

1747

TOP 10 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;
```

jasvinderkaur,126140
akhilsai,119332
premkumar,116184
saikumar,115138
shekarn,100001
vamshikrishna,94386
PlaneMad,93977
himalay,88514
himabindhu,87410
sdivya,85202

NUMBER OF USERS APPEARING 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;
```

426

POPULAR AMMENITIE

```
sqlite> SELECT value, COUNT(*) as num
        FROM nodes_tags
        WHERE key="amenity"
        GROUP BY value
        ORDER BY num DESC
        LIMIT 1;
```

Restaurant: 1600

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;
```

Hindu: 416

CONCLUSION:-

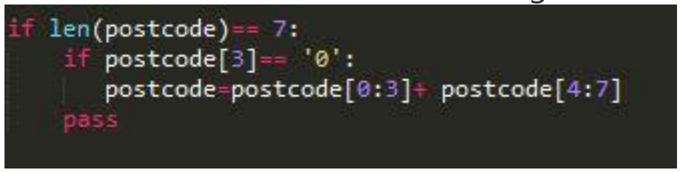
I encountered various issues while cleaning the dataset, I audited various tags and cleaned many of them, I believe there is still a lot that can be done to clean the data set. The Bengaluru city dataset that we used, was not a normal dataset and has various anamolies. We also saw many anomalies in postcode and street name, I removed many of them. I conclude that the dataset is now a lot more clean and consistent from before.

STRUGGLES:-

1. The important one was being ‘full address’ in the ‘street tag’. We changed the street tag to full tag while updating our database as recommended by the mentor.
2. While iterating all the tags from data, My laptop freezed and stop working. I took help from mentor and with some research I wrote new code with **root.clear() ,** This helped the iterparse function to clear RAM with each loop.

Additional Suggestions:

1. All the present full tags can be edited to make street tags, area tags and postcode or we can make.
   Idea: This can be done by dividing all street tags with comma and making a list. If 'street/road' word root is present then we enter it in street tag, if area word root is present then we add it to area tag, if 6 digits are present then we add it to postcode tag. Rest can be added to full address tag. This approach might generate anomalies and need more research.

2. In future a new approach can be thought to clean postcode like 5600011 which seems like a manual mistake of adding an extra 0 in the middle.
   

```
if len(postcode)== 7:
    if postcode[3]== '0':
        postcode=postcode[0:3]+ postcode[4:7]
    pass
```

   Idea: Like other updates in postcode, we can solve this issue with above mentioned code, But we have possibility of not getting a 100% accurate result because of other mistakes in postcode, For now, Not to add an inaccurate value to database seems is better.