Project 3: Wrangling OpenStreetMap Data using Python and SQL

Deepu Dilip

This work performs a data wrangling task on an openstreetmap (osm) file wherein some major errors present in the data are audited and cleaned programmatically using python. Following this, the data from the file is written into a .csv file, which is eventually loaded into an SQL database. Further exploration of the data is performed using SQL.

The map chosen for this project is that of Whitefield, which is located within the city of Bangalore in India. The map is of relatively small size (~ 80 MB), yet some major data cleaning tasks have been performed on the same. Given below is a list of major problems encountered and description of the solutions that has been applied to correct them.

LIST OF PROBLEMS IDENTIFIED AND CORRECTIONS MADE:

I. Since the entire auditing tasks done in this project were performed along the guidelines given through the case study provided by Udacity, the first aspect that was checked for was the street addresses listed in the map. And this revealed a major problem in the data file, which is perhaps unique to addresses in India.

In India, the addresses do not have a standardized format as is the case in the many of the western nations. Although we can often observe a certain order of precedence while writing the address components, (such as house name/no, followed by name of locality, followed by name of city, state, pincode), it is not often followed strictly, and not all address components mentioned above will be present in an address. This should be more clear from some examples of addresses found in the dataset.

Listed below are examples of some addresses, which are listed as values for the secondary tag “Addr:Street” in the osm file

1) "Outer Ring Road”
2) "Brookfield Road, Whitefield”
3) "1st Main Road, EPIP Area”
4) "Hoodi Main Road (Near Hoodi Circle)"
5) "B Block, AECS Layout"
6) "Outer Ring Road (Near Shiva Ganga Layout), Mahadevapura"
7) "Kaadubeesanahalli Rd, Devarabisanahalli, Bellandur,"
8) "Block 9A & 9B, Pritech Park, Special Economic Zone, Survey No 51 - 64/4, Sarjapur Outer Ring Road, Bellandur Village"
9) "7th Floor, Tower 2, 2B, Embassy Tech Village, Near New Horizon College, Devarabisanahalli"
10) "B Narayanapura Village, Whitefield, Bangalore East Taluk

11) “Kasvanahalli Road, Behind State Bank of Mysore, Sarjapur Road, Kaikondrahalli"

In the above list, addresses 1 & 3 can be taken as a proper street addresses, since it states “Outer Ring Road” in case 1 or “1st Main Road, EPIP Area” in the case of 3. Address 2 consists of not only the road name, but it also the name of the place which is “Whitefield”. Address 4 has additional text stating “(Near Hoodi Circle)”, which a fairly common practice while stating addresses in India. Address 6 is similar to value 4, with the addition of a place name as well. However, addresses 7-11 contain not just street names, but more likely includes the entire address of the associated node.

This is the result of having no standard format for reporting addresses in India. People put down their addresses in several such forms, resulting in mostly unstructured address information in a lot of places in the osm file. However, the open street map follows a structured format for storing their address information, providing several tags such as “addr:housenumber”, “addr:housename”, “addr:flats”, “addr:place”, “addr:postcode”, “addr:city”, etc., for breaking down an address into individual components, with “addr:street” being the one to list the street name. However, openstreet map also suggests the use of a separate tag called “addr:full”, for cases where address format follows no standard structure. Most of the examples shown above should ideally be entered under this tag, rather than the “addr:Street” tag.

Hence, the first “cleaning” task that was undertaken during this project was to correct the “addr:street” field, to conform to the standards of openstreetmap. Although this entailed some level of arbitrariness in deciding what constitutes proper street names, the approach taken can be argued to be logical for the most part. Following points list the action taken in this regard.

1) Identify a list of keywords that can be taken as “proper” end words for street addresses. This is where the arbitrariness comes to picture, since it is debatable whether a selected keyword fits the bill or not. Currently, the following keywords have been chosen:

   [“Avenue”, “Circle”, “Colony”, “Cross”, “Lane”, “Main, “Layout”, “Mall”, “Zone”, “EPIP Area”, “Road”, “Road x”, “Road no. x ”], (where x can be a number)

   Of these words, the words “Mall” & “EPIP Area”, although strictly cannot be considered as street names, were chosen due to their frequent occurrence in “addr:street”

2) Search for the above set of words (henceforth called end-words) in the “addr:street” tag for all nodes and ways. If the address value ends in any of these end-words, continue to the next node/way. However, if any of these end-words appear in the middle of the address string, split the string at the end-word, and store the part of the string till the end-word in the “addr:street” tag. However, the rest of the original string is not completely discarded. Instead, a new tag will be created by the name “addr:full”, which will contain the full address for the node/way. This includes all fields, including, those for house-name, house no., street, city, postcode and state. This newly created tag will not be inserted in the osm file, however, it will be written down in the CSV file, which will eventually be uploaded to the database.

II. Correction to phone numbers

In case of phone numbers, we find a similar problem as with the case of street addresses, i.e, a non-adherence to a standard format of reporting phone numbers. Note that the phone numbers have been entered as strings in the osm file. The script ‘check_phone numbers’, (provided in the code files) when run on the file “white_field.osm” collects all unique phone numbers in the file. Looking into

the phone number strings collected by the script, we can get an idea of the different formats in which phone numbers have been entered in the file. For instance, list below shows several of the phone number strings found in the file.

1) “(91)-22-40176400”
2) “+91-80-65838277”
3) “(91)-80-25401309”
4) “+91 78460 94310”
5) “+91 8 971254477”
6) “'+91 80 25618799;+91 80 25618789;+91 80 25618789”
7) “+91 80 41487799;+91 80 41725222”
8) “+91 80 65970953”
9) “+91-080-28454691 /4693 / 4464”
10) “+91-7676751499”

Before explaining the different formats present in the file, it would be better to clarify how the format of phone numbers should be in India. A typical landline phone number should be written as +91-xx-yyyyyyyy. Here, xx stands for area code, and yy... is the actual number. Now it should be noted that the length of area code xx, and even number yyyy can be variable depending on the locality within India. In case of Bangalore, to which Whitefield area belongs to, the area code is 80, and the length of the landline number (excluding country and area code) is typically 8 digits. In case of cellphone numbers, the usual format is + 91-xxxxxxxxxx, where the cell phone number is of 10 digits in length.

Although these are the standard formats of phone numbers, there are a number of other formats in which phone numbers are often expressed in India (and can be used while making calls as well) An obvious case is that +91 is not affixed when making a call within India, more so in case of landline numbers. If one has to use an area code without affixing the country code +91, then (s)he has to attach a 0 at the beginning of the number. For instance, a number +91-80-65838277 would become 080-65838277. While dialling a landline number belonging to the same area as the caller, then it is not required to attach the area code either, and hence this number would just be 65838277. In case of cell phones, it is sometimes common to attach a 0 at the beginning of the number if country code is not included, although this is not mandatory. Hence, 7676751499 and 07676751499 are both valid numbers. Sometimes, the area and/or country codes are enclosed in parenthesis while writing.

In case of the numbers found in our osm file, we see that there are a number of additional formatting differences. For instance, sometimes a space is used to separate country code, area code and the number, while other times a hyphen is used. Moreover, sometimes multiple numbers are listed, separated by a ‘;’. In this project, the script ‘check_phone_numbers.py’ was used to find the list of unique phone numbers found in the entire file (both sample as well as actual). Looking at this set of numbers, we get a brief idea of the different formats in which the numbers have been listed. The observations above were derived using this script. Additionally, we also find a number of errors in the phone numbers listed in the osm file. For instance, some numbers do not contain the required number of digits, indicating a possibly erroneous number. Another error that was found that could confuse someone not familiar with the system was the system of writing numbers such as +91-80-6583827 as +91-080-6583827. This is in fact wrong, and will throw an error if dialled.

Correction to Phone numbers.

It was deciding to ‘clean’ the different phone numbers found in this file, meaning to convert all the different formats of numbers listed in the file to a single standard format, for the sake of clarity and

avoid confusion to anyone who is not familiar with the Indian way of writing phone numbers. The standard format is country code – area code – number such as +91806583827 in case of landline numbers and +91xxxxxxxxxx in case of cell phone numbers. Hence, all phone numbers found in the file have been re-written in this format programmatically. The only numbers that were exempted were the hotline numbers that begin as '1800-xxxx..' The details of how the correction of phone numbers were done is explained in detail in the code documentation.

III. Correction to Pincodes:

The last item to be checked and corrected were the pincodes. Now, correcting the pincode is a difficult task, primarily because it is not known which pincodes are assigned for the different sub-localities present in the map. Also, there is no specific pattern of assigning the pincodes according to the area, so it is not possible to guess them either. Hence, only a very superficial correction could be made, meaning only the very obvious errors were caught. The obvious error in this case would a difference in the length of the pincode, which for the whitefield area should be 6 digits.

The postcodes for which the number of digits were less than or greater than 6 were scanned for in the entire file and only 7 instances were found. In 4 of these 7 cases, the error seemed most certainly due to an inclusion of an extra zero somewhere in the middle, probably a typo that happened while entering the value. The remaining three cases, the errors were missing digits. To correct them, the address of the element(node/way) was checked to identify the locality, and the pincodes were manually found out. These newly found pincodes were stored in a dictionary in the main code, and while entering the map data into the csv files, the pincodes for these problematic elements are mapped from this dictionary.

Please note that this "cleaning" of pincodes is not exhaustive, in that it only identifies those errors in pincodes associated with incorrect number of digits. Also, this method of manually identifying and correcting the erroneous pincodes works only when their number is small. A programmatic way of correcting instead of using a manually made dictionary (of element & their pincode) would be lot more complicated, and hence has not been explored in this work.

These were the corrections made as part of the data-cleaning process. Note that these corrections are not made to the osm file itself, rather, they are made while writing this data into the csv file. The entire code for correcting the data and writing them into csv file is given in the file 'main.py'. (description separate). The code for creating the database schema, as well as inserting the data in csv file into the database is given in the file "create_database.py". (descrption separate). Once after inserting the data, some analysis of the same was done using SQL.

DATA EXPLORATION USING SQL:

- The number of nodes in the dataset

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

```
Output:346583
```

- No. of unique contributors (users) to the map

```
SELECT COUNT(*) FROM
(SELECT DISTINCT(user) FROM nodes);
```

```
Output:453
```

- Get a list of the top contributors and a count of their contributions

```
SELECT user, count(*) AS number FROM nodes GROUP BY user ORDER BY
number DESC LIMIT 10;
```

```
Output:
anushapyata|26497
praveeng|24684
Ashok09|23826
vamshikrishna|20636
maheshrkm|19042
akhilsai|18311
anthony1|16332
udaykiran01|14985
samuelmj|14406
docaneesh|13666
```

- Number of nodes/ways with address tags

```
SELECT COUNT(*) FROM (SELECT DISTINCT(id) FROM nodes_tags WHERE
type = 'addr');
```

```
Output:
1061
```

```
SELECT COUNT(*) FROM (SELECT DISTINCT(id) FROM ways_tags WHERE
type = 'addr');
```

```
Output:
384
```

- Number of nodes/ways with tags for full address, i.e., number of nodes/ways with improper values for “addr:street” tag.

```
SELECT COUNT(*) FROM (SELECT id FROM nodes_tags WHERE key =
'full');
```

```
Output:
547
```

```
SELECT COUNT(*) FROM (SELECT id FROM ways_tags WHERE key = 'full');
```

```
Output: 100
```

It is seen that in case of more than half of the nodes and over a quarter of the ways, the addresses specified by users were not conforming to the standards of openstreetmap.

- Top Amenities in the area

```
SELECT value,count(value) AS num FROM nodes_tags WHERE
key='amenity' GROUP BY value ORDER BY num DESC LIMIT 10;
```

```
Output:

restaurant|236
atm|107
place_of_worship|81
bank|80
pharmacy|78
fast_food|70
hospital|57
cafe|49
school|48
clinic|42
```

The above query lists the top amenities in the area. The most frequent amenity is the restaurant, for which we find 236 of them which is suspiciously high a value for the Whitefield area. So is is the case of the other amenities, all of which are showing a high count for such a small area. One reason for this large number of restaurants has been identified and will be mentioned in a later section of this document.

- Top fast-food chains in the area

```
SELECT value, COUNT(*) FROM nodes_tags WHERE key="name" AND id IN
(SELECT id FROM nodes_tags WHERE value="fast_food") GROUP BY value
ORDER BY count(*) DESC LIMIT10;
```

```
Output:
KFC|6
Dominos Pizza|4
McDonald's|4
Pizza Hut|4
Domino's Pizza|3
Five Star Chicken|2
Papa John's|2
Subway|2
99 varieties of dosa|1
Abhiruchi|1
```

- Banks with the most branches in the area

```
SELECT value, COUNT(*) FROM nodes_tags where (key="name" or
key="operator")AND id IN (SELECT id FROM nodes_tags WHERE
value="bank") GROUP BY value ORDER BY COUNT(*) DESC LIMIT 10;
```

```
Output:
```

```
State Bank of India|24
ICICI Bank|6
Canara Bank|5
HDFC Bank|5
Axis Bank|4
Vijaya Bank|4
Karnataka Bank|3
Punjab National Bank|3
Andhra Bank|2
SBI|2
```

- Users who have the most contributions of addresses in the map

```
SELECT user FROM nodes JOIN nodes_tags ON nodes.id=nodes_tags.id
WHERE nodes_tags.type="addr" GROUP BY user ORDER BY count(*)
DESC LIMIT 15;
```

```
Output:
srideviM
indigomc
Avinashk
docaneesh *
Ravindra Singh S
Nand Kishore Gupta *
Prasad Zende
Bidya Bharati
abhik uniyal *
shree007 *
ASHISH R SINGH
pujarinee
chaitanyam
drishya
moghal shahataj *
```

- Users who have most number of address entries in incorrect format

```
SELECT user FROM nodes JOIN nodes_tags ON nodes.id = nodes_tags.id
AND  nodes_tags.key="full" GROUP BY  user ORDER BY count(*) DESC
LIMIT 15;
```

```
Output:

Avinashk
srideviM
indigomc
Ravindra Singh S
Prasad Zende
Bidya Bharati
ASHISH R SINGH
pujarinee
abhik uniyal
chaitanyam
```

```
drishya
Abhinav Chandrakar
sushma v
hemanthkumargoli
Rajan choudhary
```

The previous two queries respectively returns the top 15 uses with the most number of contributions in addresses and the top 15 users who have contributed addresses in the wrong format (which is found by the existence of “addr:full” tag. This tag was added only for cases with wrong addresses). Interestingly, the top 3 contributors of addresses are the top 3 contributors of wrong addresses as well. In fact, there are only 5 users from the top contributors list, whose name do not appear in the second list of top contributors of wrong addresses (docaneesh, Nand Kishore Gupta, Abhik Uniyal, shree007 & moghal sahataj).

Some thoughts on pincodes:

In the data auditing phase, a quick check for incorrect pincodes were done by looking at the number of digits in each pincode in the osm file. A pincode with more or less than 6 digits were classified as wrong pincodes, and corrections were applied to them manually. However, this method does not capture any other mistakes in the pincodes. Using SQL, we can broaden our search for wrong pincodes further. Bascially, the idea is to find those pincodes who appear the least number of times in the osm file. If a typo occured while entering the pincode, then the particualr typo is most likely to have occured only once, and hence would appear only once in the osm file. Using the following query, we obtain 20 pincodes, which appear the least number of times in the osm file, along with the number of times they were found.

```
SELECT value, COUNT(value) as num from nodes_tags WHERE nodes_tags.key='postcode' group by value ORDER BY num ASC limit 20;
```

Output:

```
500036|1
530103|1
560 036|1
560 037|1
560001|1
560024|1
560025|1
560063|1
560067,|1
560092|1
560097|1
560109|1
600042|1
560038|2
560068|3
560102|3
560049|4
560087|4
560017|5
560075|9
```

The above output shows that are only 14 pincodes which appear just once in the entire file. amongst Amongst these 14 pincodes, we can further identify and correct some other mistakes. For instance, it is most likely the case that the first two pincodes contain a typo, i.e., 500036 is probably 560036 and 530103 is most likely 560103. (Since Whitefield pincodes start with 560). The pincode 600042 is clearly wrong since pincode 600042 belongs to Chennai, in the state of Tamil Nadu in India. In this case, the correct pincode is most likely 560042. The fourth and fifth pincodes are not wrong, however, their format is different (3 digits followed by space followed by next 3 digits), and hence appears only once in the file. So is the case with the 10^{th} pincode, which has a trailing comma attached to it.
In case of the rest of the pincodes, we will need to manually check their addresses (using the node_id) and find out the pincodes. This is currently outside the scope of this work, however, it is something that can still be incorporated.

CONCLUSION

To summarize, the openstreetmap file for whitefield area in Bangalore, India was audited and cleaned using python. The major cleaning tasks performed were modifying the street addresses in the file to conform to the standards of openstreetmap, addition of a new addr:full tag for those cases where street addresses contained more than just street names, standardization of the phone numbers present in the file to an acceptable and clear format, and identification and correction of certain erroneous pincodes. The corrections were made not in the osm file, but in the csv file and hence the sql database into which the data from the file were inserted to.

Some Suggestions for additional improvments in the map:

1) Native language: Some of the entries in the map are made in the local language (Kannada), which can be seen in the nodes_tags table in the database. This has not been corrected in this present work, although it is something that can be done in the future. Since the number of such entries are limited, it is possible to identify and correct them manually, as was done in the case of the pincodes. A more robust way would be automatic translation in a programmatic way, which would however, would be much more complicated to task to to

2)Pincode: As mentioned earlier, there are certain pincodes which are suspiciously less frequent in the map. These pincodes need to be verified if they actually belong to the address along which they are mentinoed. However, this is also a task that needs can only be done manually.

3)Large Number of Restaurants (and other amenities): An earlier query revealed a suspiciously large number of restaurants in the map (database). Now, a query to list the name of all those restaurants (output not shown since there are 235 names) revealed that many amenities are not strictly restaurants, but are sometimes ice-cream parlours, juice stalls etc. This perhaps explains the large number for restaurants in the map. The classification of the amenities appropriately would constitute a complex task, which can be undertaken in the future.

Query to list names of restaurants:

```
SELECT n.value FROM nodes_tags n JOIN nodes_tags m WHERE n.id = m.id AND
n.key="name" AND m.value="restaurant";
```

4) More work on address tags: In this work, the street addresses were checked for and corrected so that the addr:tag contained only the street address, and if the value of addr:street tag in the osm file contained more than just the street name, they were extracted and put together (along with other information such as house no, house name etc.) into a new tag called addr:full. Ideally, what should

be done would be to analyze the entire address string found in the "addr:street" tag and classify the individual components in the string (if it contains more than just street names) into the appropriate tags, such as addr:place, addr:city, addr:state etc, wherever possible and applicable, rather than putting them all together into an "addr:full" tag. This can be incorporated in the future.