OpenStreetMap Data Wrangling Project

Bandung, Indonesia

The city that I choose to wrangle for this project is Bandung (https://en.wikipedia.org/wiki/Bandung) located in Indonesia!

https://mapzen.com/data/metro-extracts/metro/bandung_indonesia/ (https://mapzen.com/data/metro-extracts/metro/bandung_indonesia/)

I chose this city because I am from Indonesia and although my hometown is Jakarta and not Bandung, I often visit Bandung for the culinary and panoramic adventures!

Problems Encountered

There are several problems that I encounter when I wrangle the data set but I am going to list the top three problems that I experienced.

1. There are inconsistencies in the street naming, especially in the abbreviation and missing street type. (eg: Taman Sari instead of Jalan Taman Sari)
2. There are missing user and userID that unables to import the data into csv while adhering to the schema.
3. Key names are inconsistent to each other and there are unusual keys. (capacity vs capacity:persons, 'Waktu Belajar' which is Indonesian language for 'Time to Study', etc.)

But previous to running the audit codes, I made sure that there are no problematic tags by running tags.py. The results from tag.py show that all tags are in an acceptable format. (Note: code edited for better readability)

Inconsistent Street Naming

A little background, unlike the U.S, Indonesia does not have street types like Avenue, Road or Street. In Indonesia, all the roads or streets or avenues is called "Jalan" which basically means pathway" and encompasses all different types of street. After running the audit-streetname.py and checking for "Jalan", I discover that there are a lot of streets that is missing this description (such as as well as incosistent abbreviation use like "JL", "Jl.", "Jl", etc.

To update the street name, I used a similar code with the case study that utiilzes regular expressions in order to check the first word of the street name and map the correct format (eg: from 'Jl.' to 'Jalan') or add 'Jalan' to the street name if it is missing in the first place.

```
street_mapping = { "Jl.": "Jalan",
            "JL": "Jalan",
            "JL.":"Jalan",
            "Jl.Kartini":"Jalan Kartini",
            "Jl.soekarno":"Jalan Soekarno"
            }

expected_street_type=['Jalan']

def update_street_name(street_mapping):
    ...
    tree=ET.iterparse(osmfile, events=("start",'end'))
    for event, elem in tree:
        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if (tag.attrib['k'] == "addr:street"):
                    name=tag.attrib['v']
                    m=street_type_re.search(name)
                    if m:
                        street_type= m.group()
                        if street_type=="hjvhj":
                            tag.set('v','unknown')
                            print ('street: hjvhj -> unknown')
                        elif street_type not in expected_street_type and
street_type in street_mapping:
                            new_name = re.sub(street_type_re, street_mapp
ing[street_type], name)
                            tag.set('v',new_name)
                            print ('street: ' +name + ' -> ' + new_name)
                        elif street_type not in expected_street_type:
                            new_name='Jalan '+ name
                            tag.set('v',new_name)
    ...
```

Because there are some inconsistencies in the street name, I decided to check the postal code as well. Using audit-postcode.py, I have ensured that all entries for post code are in the correct format - starting with 4 and is 5 digits long.

Inconsistent keys

I was looking for inspiration for what other keys that I can play around with and I realize that there are inconsistent keys and unusual or "wrong" keys. I used keys.py to give me a list of all the existing and looked for keys that only appear less than 5 (to show that it is ununsual). After that, I manually choose which ones to change via "mapping", similar to the street type change in the case study.

```
key_mapping = { "Akreditasi": "accreditation",
            "food": "amenity:restaurant",
            "capacity:persons":"capacity:people",
            "capacity":"capacity:people", "Jml Rombel":"fixme",
            "lit":"fixme", "iata":"fixme","waktu belajar":"fixme",
            }

def update_keys(key_mapping):
    ...
    for event, elem in ET.iterparse(osm_file, events=("start",'end')):
        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if (tag.attrib['k'] in key_mapping):
                    print ('key: ' + tag.attrib['k'] + ' -> ' + key_mappi
ng[tag.attrib['k']])
                    tag.set('k',key_mapping[tag.attrib['k']])
                elif tag.attrib['k']==None:
                    tag.set('k','fixme')
                    print ('key: None -> fixme')
    ...
```

Missing User and User ID

When I first run the existing data.py from the Case Study, I got an error during validation because there are missing user and user id values. I have decided to replace the missing values with 'anon' as user and 000 as user id using this code.

```
def update_user():
    ...
    street_types = defaultdict(set)
    for event, elem in ET.iterparse(osm_file, events=("start",'end')):
        if elem.tag == "node" or elem.tag == "way":
            if 'user' not in elem.attrib and 'uid' not in elem.attrib:
                    elem.set('user','anon')
                    elem.set('uid','000')
                    print ('user: None -> anon\n' + 'uid: None -> 000')
    ...
```

After adding 'anon' and '000' to the data set, I am able to transform the data into csv and validate it to the available scheme (scheme.py) and perform queries to the data set via SQL!

All the codes for updating the file is located in the update.py

Overview of the Data with Queries

In this part of the report, I am going to show the overview of the data from the bandung_indonesia.osm file that I acquired from the link in the beginning of the report. I will also show some of the insights that I discover through performing SQL queries. (Note: output edited for better readability)

File sizes

```
bandung_indonesia.osm ......... 70.6MB
bandung.db .................... 38.8 MB
nodes.csv ..................... 26.8 MB
nodes_tags.csv ................ 0.60 MB
ways.csv ...................... 3.8 MB
ways_tags.csv ................. 3.0 MB
ways_nodes.cv ................. 9.8 MB
```

Number of nodes

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

320286

Number of ways

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

62818

Number of unique users

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

613

Additional Exploration

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

```
restaurant|426
place_of_worship|141
cafe|111
school|105
bank|90
atm|60
hospital|54
fast_food|43
fuel|42
university|31
```

Not surprised! The number of restaurants dominate the amenities in Bandung. As I have mentioned earlier, Bandung is one of the most popular destinations for food lovers (like me!) Now let's see what cuisines it offers.

```
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
```

```
regional|26
Indonesian_Restaurant|5
japanese|5
pizza|5
Chinese_Restaurant|3
```

Interesting... Most of the restaurants' cuisine in Bandugng is either regional or Indonesian but surprisingly Japanese and pizza comes in third. This definitely show the influence of different culture on the food scene in Bandung.

Next, I want to look at the tourism in Bandung.

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

```
hotel|205
viewpoint|18
guest_house|12
attraction|10
information|8
museum|8
motel|7
hostel|4
theme_park|4
artwork|3
```

That is a quite large number of hotels but I expected more viewpoint and attractions in Bandung as it is a popular tourist destination in Indonesia.

The last thing I want to look is at religion and places of worship in Bandung.

```
SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
    JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='place_of_worsh
ip') i
    ON nodes_tags.id=i.id
WHERE nodes_tags.key='religion'
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 3;
```

```
muslim|98
christian|18
buddhist|2
```

Also not surprising, Indonesia is an Islamic country with around 87% of Muslims. Although according to Wikipedia, Hinduism is the third religion in Indonesia but Buddhism seems to be in the third place in Bandung.

Other Ideas

Something that I have noticed is that there are a lot of variations of the same input. For example after querying for cities in Bandung, Indonesia and count them in descending order using this query:

```
SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags UNION ALL
      SELECT * FROM ways_tags) tags
WHERE tags.key LIKE '%city'
GROUP BY tags.value
ORDER BY count DESC
LIMIT 5;
```

I got this result:

```
Bandung|711
Bandung, Jawa Barat|189
Bandung Barat|130
bandung|16
Cimahi|9
```

The top 4 cities are Bandung, but inputted in different variations that makes the count wrong because all of the nodes are actually referring to the same thing.

Also, when I was running keys.py to list all the keys available, I realized that there are a lot of None values in the keys.

I think this problem can be removed by providing a guideline in the form of some kind of visualization of what keys and what values you can input. This way, when a user wants to update or input a new node, he or she can just look at this guideline and choose what key name to use and value to input along with the key name so that the new node/way is consistent with the rest.

I know that something similar like this exists already: https://taginfo.openstreetmap.org (https://taginfo.openstreetmap.org) but this is still too broad as it encompasses the whole OpenStreetMap. Maybe something that is more specific such as only limititing it to the cities in MapZen will be more practical and useful.

However, the challenge itself is to create such documentation when the whole thing right now is already messy and inconsistent. Before starting the documentation, someone has to tidy up the current existing data and then create a proper documentation.

Or another thing that can be done is to create an automated program that tries to match a new key or value inputted to an existing one. If no match is one, this program will then generate request and wait for admin approval before the new key or value is added. But the problem with this is, it will require someone or a group of people that act as admin and monitor every single request that does not match with an existing one which will be quite time consuming considering the large amount of user contribution to OSM Project every day.

Conclusion

From wrangling the Bandung OSM, I realize that there is a lot of variations in the data input and there still a lot of cleaning to be done, especially to ensure the validity and consistency of the data set. Also, I found out that there is some error that results in unexisting user and uid for some of the nodes and ways, which can be prevented if a rule that requires user and uid to create nodes/ways is enforced. This is definitely an interesting project that allows me to know more about one of my favorite cities in Indonesia!