Submission document : Wrangle OpenStreetMap Data

The Data

First step , given the experience we got with the case study, is to familiarize with the data.

Data , in this case, is in XML format. The riginal file downloaded from OSM metro extracts for the city of Buenos Aires, were I live, is around 400 MB in size so I reduced the file to a sample of 5 mb created using dd (count=20000) .

The Buenos Aires subway is the 1st subway system in Latinamerica and I used it a lot so my first guess was how the OSM project managed to describe subway stations and lines in general in maps.

My initial idea was to extract subway information from the provided OSM file , insert it in SQL and create some reports and statistics based on that. What I actually found was , as we saw in the course, that the data was not complete or with the expected details.

Subway information in the provided OSM extract is aggregated in Node elements. Here an example:

```
    <node id="256241854" lat="-34.639835" lon="-58.458014" version="13" timestamp="2017-05-03T14:31:10Z" changeset="48367783" uid="51045" user="Geogast">
        <tag k="name" v="Varela"/>
        <tag k="public_transport" v="stop_position"/>
        <tag k="railway" v="station"/>
        <tag k="start_date" v="1985-10-31"/>
        <tag k="station" v="subway"/>
        <tag k="subway" v="yes"/>
        <tag k="wikidata" v="Q3296869"/>
    </node>
```

So initially I found two things to fix :

a) The “name” of the stations was not normalized. The official reference of the underground system is http://www.buenosaires.gob.ar/subte/mapa-y-combinaciones but they only provide a JPG file for users to review (mapa_subte.jpg). The name of the stations on that official picture are different from the one I found in the OSM file (at least some of them).

b) No information relates the stations with their corresponding lines. There are 7 subway lines in Buenos Aires identified by the following letters (A,B,C,D,E,H and P).

The Data auditing process

For the data auditing process I will follow the flow of the code in the jupyter notebook osm_project_code_v2 referencing examples and comments with notes like “REF 1” that you can easy search in the notebook.

So given findings a and b above I basically had to go through the following actions :

1) Provide a programmable reference to correct data .

As I mentioned in point “a” the city of Buenos Aires only provide a JPG image of the whole map of lines and subway stations. Since some of the stations names are not the official ones in the OSM map I needed a normalized reference. So, basically I packaged a data structure of official station names and lines (based on JPG info) and a couple of methods to obtain the whole list of correct names or the list of stations by line

REF1 :

the normalized_stations.py module provides the following kind of structure :

"B" : ["Leandro N. Alem", "Florida", "Carlos Pellegrini", "Uruguay", "Callao", "Pasteur Amia", "Pueyrredon", "Carlos Gardel", "Medrano", "Angel Gallardo", "Malabia O. Pugliese", "Dorrego", "F. Lacroze", "Tronador Villa Ortuzar", "De los Incas Parque Chas", "Echeverria", "Juan Manuel de Rosas Villa Urquiza"] },

those are the “official” names of the stations from the goberment provided source : mapa_subte.jpg.

When we look at the OSM file we found names with these differences :

`Tronador - Villa Ortuzar`

`or`

`Los Incas - Parque Chas`

or

`Malabia - Osvaldo Pugliese`

2) Sample the OSM file and clean station names

As I already mentioned original extracted file from OSM project is about 400 Mb (buenos-aires_argentina.osm) . The total number of stations in the underground system based on the official information is 104 . I sampled the original file to create a reduced size file of about 5 Mb (bs_as_subway.osm). This reduced files has 63 stations. My proposed work around this 63 stations is :

2.a) Parse the reduced osm file and process each node for start and end events for node “tags”.

REF2 :

parse_and collect_nodes routine.

2.b) Call process_every_node routine to extract ‘k’/’v’ pairs for every node isolated by the first routine.

REF3 :

so the process_every_node receives a node from REF2 and pass k_tags and v_tags to extract_stations

REF4:

2.c) Call extract_stations to filter nodes where tags contain the “subway” key.

```
  <tag k="subway" v="yes"/>
```

Not all the Nodes in the OSM file mark subway stations. So, instead of moving forward processing all the nodes the idea here was, before extracting the names, to filter those Nodes without subway info. If the Node actually contains subway info, I append its “name” tag to the “actual_stations” list capturing the station names :

```
Primera Junta
Plaza de los Virreyes - Eva Perón
Varela
Medalla Milagrosa
Emilio Mitre
José María Moreno
Avenida La Plata
Boedo
Jujuy
Humberto I
Pichincha
```

(….) list continues

2.d) Finally , since I used ascii coding to create my structure of the official stations name, I use a routine to decode actual_stations from unicode to ascii on to_ascii_actual_stations so they can be comparable. They build up a dictionary with an index and the station name after the unidecode process.

REF5 :

so now we have a dictionary with the ascii equivalents :

```
0 Primera Junta
1 Plaza de los Virreyes - Eva Peron
2 Varela
3 Medalla Milagrosa
4 Emilio Mitre
5 Jose Maria Moreno
6 Avenida La Plata
7 Boedo
```

```
8 Jujuy
9 Humberto I
10 Pichincha
```

(….) list continues

3) First pass stations and the rest

REF6 :

A first pass of the actual_stations collected from OSM through a compare with official stations (split_match_nomatch_stations routine) provide a mixed results : some of the stations are exact match and some of them dont . The first group is registered in the match_stations dictionary. The second group , in the nomatch_stations dictionary.

4) Dealing with the "no match" stations names

To deal with the stations names that dont match official names I had to "tokenize" the OSM names and copare the tokens , any of them, to official names. The reasson is that I could not found other way to match -neither a way to do it with regular expressions- because the names is composed of more than one word and some of them could match and others dont. So, first of all I dont want insignificant tokens like "de", "los", "la" (which are spanish terms for "of", "the", etc) neither line letters (like "C" or "D") attached to the name so I use a split_nomatch outine to split the string with the name in tokens and ignore those tokens_to_ignore.

REF7 :

so I collected the insignificant tokens first

tokens_to_ignore = [" de ", " los ", "-", " La ", " Los ", "los", "\(E\)", "\(C\)", "\(D\)"]

and the split_nomatch routine split the stations names that the above routine could not match and produces the following structure

```
1 Plaza   Virreyes   Eva Peron
6 Avenida Plata
11 Entre Rios   Rodolfo Walsh
14 Independencia
15 Independencia
24 General Urquiza
58 Malabia   Osvaldo Pugliese
52 Pasteur   AMIA
46 Inclan Mezquita Al Ahmad
56 Medrano Almagro
26 Tribunales
27 Callao
60 Federico Lacroze
61 Tronador   Villa Ortuzar
62 Los Incas   Parque Chas
31 Pueyrredon
```

I kept the decoded stations dict keys for cross reference. As you can see each value is a splited string of the characters I am interested in.

To continue with this process the idea is to use the power of sqlite3.

REF8 :

So , I first load several dictionaries to move them later to csv and from there to sqlite3. With the information of the different group of stations I created the following dictionaries, csv files and sqlite tables :

Dictionary	Description	CSV file	Sqlite table
final_review_nomatch	Dictionary with a decoded ID , the station name that did not match, the probable match for tokens and a counter	nomatch_tokens.csv	nomatch_tokens
nomatch_stations_dict_list	Dictionary with stations that did not match	nomatch.csv	nomatch_stations
match_stations_dict_list	Dictionary with stations that did match	match.csv	match_stations
decoded_stations_dict_list	Dictionary with match and no match stations found in the sample file	decoded.csv	decoded_stations
all_stations_dict_list	Dictionary with the list of official stations and lines from mapa_subte.jpg	Official.csv	official_stations

So, the last part of the process, having built the tables in the subway.db is to use a couple of queries to look for the best official station candidate name given the tokenized version that cames from pyton. The result , the final part, is a list where the wrong name of stations that did not match has their corresponding match name found.

REF9 :

so the query process is basically to use the key I saved for the nomatch stations tokenenized names (REF7) and count which is the name from official stations names with the highest number of matches. The reasson is that “Plaza” for example , still match more than one name. But, if more than one token is found is more probable that the name is the correct name.

Lets see an example.

```
sqlite> select nomatch_tokens.nomatch_station as normalizar,
   ...> count(nomatch_tokens.norm_station) as match,
   ...> official_stations.station_name as normal
   ...> from nomatch_tokens, official_stations
   ...> where nomatch_tokens.norm_station = official_stations.station_name
   ...> and nomatch_tokens.nomatch_key = 1
   ...> group by nomatch_tokens.norm_station
   ...> order by match desc
   ...> ;
Plaza de los Virreyes - Eva Peron|4|Plaza de los Virreyes Eva Peron
Plaza de los Virreyes - Eva Peron|1|Plaza Italia
Plaza de los Virreyes - Eva Peron|1|Plaza Miserere
Plaza de los Virreyes - Eva Peron|1|Plaza de Mayo
```

so as you can see , the first match is the most clear candidate because 4 tokens has matches.

Whislist

I would like to include a list of things -the ones I remembered- that can be improved and the ones that would be nice to have:

Solution #1

I included the line letter for each station . It could be used to create a k/v pair with that info to submit it to OSM.

Benefits # 1

The idea is to use the information from normalized_stations ro include something like the following to subway nodes :

```
<tag k="line" v="B"/>
```

the benefit is to be able to look up stations by line or even searching for all the nodes with the same station have an idea of the whole line.

Solution #2

I could investigate a less verbose procedure to try to match non-matching stations using python regex for examples

Benefits # 1

Less verbose code is less code to maintain and more clear ideas to share

Anticipated issue # 1

Trying to match more than one word in regex could be very difficult since I should first recognize each token and create "on the fly" the regex match pattern . So instead of helping clean the code it could get worst

Solution # 3

I could find a way to deal with XML big files (after sample them with DD) without caching exceptions

Benefits # 1

My actual code has to catch and exception because the original big file has being cut with dd. Instead of thet I should explore coding the process of the big OSM file using "yield" . That way the solution could handle biggest files because it wont load in memory the whole file.

I dont see any issue away from undestanding the right way to use the iteration protocol

Solution # 4

Work the unicode versions

Benefits # 1

Without decoding to ascii the stations names I would be able to reduce code and reuse the code for any other XML wrangling project because it will be almost always in unicode. I just simplified the solution.

dont see any issue away from undestanding the right way to manage unicode code.

Solution # 5

Propose the Buenos Aires city government to have the data in a programmatic source (ej XML + web services) instead of just the jpg image.

Benefits # 1

Clearly if the Buenos Aires government could maintain a structure of the information that match stations and lines (line the structure used in normalized_stations which is JSON compatible) people could use it to build more rich information sources and to match subway information with train information , with sites of interest , etc.