OpenStreetMap Data Wrangling Project

In this notebook I summarize my steps and findings towards the Data Wrangling Project course, using OpenStreet Map as a Case Study. This is in the Scope of the Udacity Nanodegree, P3: Wrangle OpenStreetMap Data

Author : Marcelo Tyszler (mailto:tyszler.jobs@gmail.com)

Start Date : 04/Mar/2017

Today's Date : 28/Apr/2017

Choice of tools

For this project I opted for:

- Keep basic documentation and notes in the Jupyter Notebook
- Save files in Github
- Use PyCharm as Python environment
- Use DB Browser for SQLite as SQL environment (in combination with command line) for testing and exploration
- Having all SQL operations done via python using the sqlite API

Map Area

Nieuwegein, Netherlands:

- Mapzen extract: https://mapzen.com/data/metro-extracts/your-extracts/2ac3c83d0cf8 (https://mapzen.com/data/metro-extracts/your-extracts/2ac3c83d0cf8)
- OpenStreetMap link: https://www.openstreetmap.org/relation/419212#map=12/52.0300/5.0966 (https://www.openstreetmap.org/relation/419212#map=12/52.0300/5.0966)

I chose this area since this is where I currently live, and not a major city in the Netherlands. Therefore I believe I can collaborate better here due to my knowledge and have a marginal higher impact than to a major city like Amsterdam.

I downloaded the OSM XML option:
https://s3.amazonaws.com/mapzen.odes/ex_4nypWySr8p3tPS5D3gfvy2aPey7mV.osm.bz2 (https://s3.amazonaws.com/mapzen.odes/ex_4nypWySr8p3tPS5D3gfvy2aPey7mV.osm.bz2)

Initial exploration

After loading a small sample of the data (taking every 10th entry in the full data), I explored the database.

I noticed the (initial) issues:

- Many unique keys: for example, *postal_code* and *postcode* should be combined
- Post codes where not uniform (for example "3431 LN" and "3523ED")

Many unique keys

I ran the following query:

select key, count(*) as count from ways_tags group by key order by count

This returned 134 unique rows, which I found suspiciously high. I did not investigate all details, but I did compare the top returning with the lower returning keys. Here I noticed, for example, that *postal_code* appeared with count 2, as opposed to *postcode* with count 5614. Therefore I decided to merge those in the python import code:

correct for postal_code: if potential_key == "postal_code": potential_key == "postcode" potential_type == "addr"

Clean Postcodes

Browsing through the data revealed 2 problems with postcodes:

- Not uniform on the white space between 4 numbers and letters
- Incomplete postcodes having only 4 numbers

To solve this, I first removed any whites spaces by adding the codes into the python db_prep.py code:

In [6]:

```
def improve_postcode(postcode):
    return postcode.replace(" ", "")
```

if potential_key == "postcode": content.attrib['v'] = improve_postcode(content.attrib['v'])

To improve the postcodes with 4 digits only, I decided to use the information in the database to improve itself.

The approach was to find the closest node point with full postcode that started with the same 4 letters and use that full postcode as replacement.

This was done by combining the following subqueries:

Find the Matching postcodes select substr(value,1,4) as match_postcode, value as full_postcode, key, lat, lon from nodes_tags join nodes using(id) where key = "postcode" and length(value)>4)

And after matching on the (filtered) postcode, to compute squared distances and find the minimum distance

Putting it all together: select id, problem_postcode , full_postcode, min(distance) from (select id, problem_postcode , full_postcode, (problem.lat-solution.lat)*(problem.lat-solution.lat)+(problem.lon-solution.lon)*(problem.lon-solution.lon) as distance from (select value as problem_postcode, id, key, lat, lon from nodes_tags join nodes using(id) where key = "postcode" and length(value)<=4) as problem inner join (select substr(value,1,4) as match_postcode, value as full_postcode, key, lat, lon from nodes_tags join nodes using(id) where key = "postcode" and length(value)>4) as solution on problem_postcode = match_postcode) as lat_lon_distances group by problem_postcode

This was fed into the python create_db code

results = c.fetchall() #print results # results: id, problem_postcode, best_match_postcode for problem_postcode in results: sql = "UPDATE nodes_tags" + \ " SET value = '" + str(problem_postcode[2]) + "'" \

" WHERE id = " + str(problem_postcode[0]) + \ " AND key = 'postcode'" db.execute(sql) db.commit()

Further Exploration

After the initial exploration, further exploration was done with the full dataset

Verify Cities

I was curious to see which cities would be listed. I would expect Nieuwegein, but also the neighbouring areas of Utrecht, Vianen, Ijsselstein.

select value, count(*) from nodes_tags where key = 'city' group by value

City	Count
De Meern	44
Hagestein	12
Houten	24
IJsselstein	7673
Lopikerkapel	149
Nieuwegein	31427
Tull en 't Waal	12
Utrecht	14398
Vianen	3094

The above result is fine and shows other small cities on the neighbourhood

Other fields

Further inspection of the other fields did not reveal any (serious) problems. Street names were correct and did not suffer the abbreviation problems from typical US data.

Data Overview

File sizes

nieuwegein.osm: ..... 117 MB nieuwegein_sample.osm ..... 12 MB OpenStreeMap_Nieuwegein.sqlite... 74 MB nodes.csv ..... 37 MB nodes_tags.csv ..... 12 MB ways.csv ..... 5 MB ways_tags.csv ..... 13 MB ways_nodes.cv ..... 10 MB

Number of nodes

442787 rows returned in 339ms from: select * from nodes

Number of nodes_tags

369053 rows returned in 167ms from: select * from nodes_tags

Number of ways

73018 rows returned in 148ms from: select * from ways

Number of ways_tags

328890 rows returned in 160ms from: select * from ways_tags

Number of ways_nodes

566456 rows returned in 156ms from: select * from ways_nodes

Number of unique users

select user, count(*) as counter from (select user from nodes UNION ALL select user from ways) group by user order by counter DESC 336 rows returned in 2094ms

Top 10:

User	Contributions
Martin Borsje_BAG	193759
Zugführer_BAG	47604
3dShapes	45710
Hendrikklaas	41691
rivw_BAG	38711
Gertjan Idema_BAG	36189
ruudblank_BAG	19041
Christoph Lotz	13181
ligfietser	12619
cartinus	6849

Top 10 Amenities:

select value, count(*) as counter from nodes_tags where key = "amenity" group by value order by counter desc limit 10

Amenitiy	counter
waste_basket	501
bench	216
parking	138
recycling	64
post_box	63
bicycle_parking	48
fast_food	29
restaurant	25
school	24
waste_disposal	20

Other ideas:

- Further improvement can be done by cross check externally the postcodes
- Certain tags could be re-written to a more intuitively format. For example these tags refer to a cell tower:

```
<tag k="height" v="21" />
<tag k="man_made" v="tower" />
<tag k="technology" v="UMTS" />
<tag k="tower:type" v="communication" />
```

  It would be clearer to have all these 4 entries with a tag type of tower, instead of 'regular' and just one as 'tower', and the values as in there

In [ ]:

In [ ]: