DATA EXTRACTION AND WRANGLING

Wrangle OpenStreetMap Data

Project Report

Introduction

On the particular project, I am using data mungling techniques to assess the quality of OpenStreetMap's (OSM) data for the center of Singapore regarding their consistency and uniformity. The data wrangling takes place programmatically, using **Python** for the most of the process and **SQL** for items that need further attention while in the **PostgreSQL**.

The dataset describes the center of Singapore, covering an area from Clementi on the west, to Bedok on the east and from Serangoon on the north, to Sentosa Island on the south. The size of the dataset is 96 MB and can can be downloaded from here

Data Assessment

An initial exploration of the dataset revealed the following problems:

- Abbreviations of street types like 'Av' instead of 'Avenue' and 'Rd' instead of 'Road'.
- All lowercase letters like 'street' instead of 'Street'.
- Postcodes including the first letter (S xxxxxx) or the whole name (Singapore xxxxxx) of the country.
- Postcodes omitting the leading '0' (probably because of declared as integers at some point before their import to OpenStreetMap.)
- Multi-abbreviated amenity names.

The problems in the amenity names were to a small extent, and they were corrected directly in the database, the rest resolved programmatically using Python on the biggest part and a subtle portion of them needed further assessment, resolven in the database.

Auditing Street Types

To audit the Street Types I need to extract the Street Names from the XML.
The Street Names appear in two forms in the dataset:
In *Node* and *Way* elements, in the form of: "*< tag k="addr:street" v="**street_name**"/>*"

```
<node id="337171253" lat="1.3028023" lon="103.8599300" version="3"
timestamp="2015-08-01T01:38:25Z" changeset="33022579" uid="741163"
user="JaLooNz">
    <tag k="addr:city" v="Singapore"/>
    <tag k="addr:country" v="SG"/>
    <tag k="addr:housenumber" v="85"/>
    <tag k="addr:postcode" v="198501"/>
    <tag k="addr:street" v="Sultan Gate"/>
  </node>
```

In *Way* elements that have the "*< tag k="highway" ..../>*", and the '*v*' attribute is one of ['living_street', 'motorway', 'primary', 'residential', 'secondary', 'tertiary'], as"*< tag k="name" v="**street_name**"/>*".

```
<way id="4386520" version="23" timestamp="2016-11-07T12:03:39Z"
changeset="43462870" uid="2818856" user="CitymapperHQ">
    <nd ref="4486796339"/>
    <nd ref="1278204303"/>
    <nd ref="3689717007"/>
    <nd ref="246494174"/>
    <tag k="highway" v="primary"/>
    <tag k="name" v="Orchard Road"/>
    <tag k="oneway" v="yes"/>
  </way>
```

Although most of the Singaporean street names end with the street type (e.g., “Serangoon Road” or “Arab Street”) it is also common to end with a number instead (e.g. “Bedok North Avenue 1”). Thus, by using regular expressions, I am extracting from the streetnames, the last word that does not contain numbers.
It would be easy to populate the list of Common Street Types with some profound values like Street or Avenue, but guessing does not take into account any local peculiarity. Instead, I searched the dataset for all the different types and used the 12 with the most occurrences (From 13th position, abbreviations start to appear).

	Street Type	Occurrences		Street Type	Occurrences		Street Type	Occurrences
1	‘Road’	574	6	‘Geylang’	42	11	‘Link’	34
2	‘Avenue’	145	7	‘Crescent’	42	12	‘Terrace’	30
3	‘Street’	139	8	‘Walk’	40	13	‘Ave’	29
4	‘Drive’	87	9	‘Park’	39	14	‘Hill’	25
5	‘Lane’	80	10	‘Close’	37	15	‘Flyover’	23

To find the street names that need correction, I used the “get_close_matches()” function from the “difflib” module to find “close matches” of the 12 Common Street Types. This is what I found:

```
Road ['Road', 'road', 'Rd', 'Ria']
Avenue ['Avenue', 'Aenue', 'Avebue', 'Ave']
Street ['Street', 'street', 'See', 'Stangee']
Drive ['Drive', 'Grove', 'Grisek', 'Bridge']
Lane ['Lane', 'Lana', 'Lateh', 'Layang']
Geylang ['Geylang', 'Pelangi', 'Selangat', 'Selanting']
Crescent ['Crescent', 'Cresent', 'Cres', 'Green']
Walk ['Walk', 'walk', 'Wajek', 'Wakaff']
Park ['Park', 'park', 'Parkway', 'Paras']
Close ['Close', 'Cross', 'Circle', 'Flyover']
Link ['Link', 'link', 'Minyak', 'Bingka']
Terrace ['Terrace', 'Terrance', 'Ter', 'service']
```

The Python code was able to correct 998 problems both in addresses where the Street Type was at the end of the address
`Greenwood Ave ==> Greenwood Avenue (2 occurrences)`
and wher it was not:
`Eunos Ave 7A ==> Eunos Avenue 7A`

Auditing Postcodes

Postcodes in Singapore consist of 6 digits with the first two, denoting the Postal Sector and taking values between 01 and 80, excluding 74 (link).
I searched the dataset for this pattern, correcting whatever could be addressed automatically and added the rest to the “PROBLEMATICS” for further examination.
Postcodes were much more consistent than the street types with 3 problems fixed programmatically and 8 pending further inspection.

Assessment in the Database

After performing the most of the cleaning with Python, I stored the dataset in a database to explore it and examine further the PROBLEMATIC elements.
As a database I used PostgreSQL to present a generic solution although a lightweight database like SQLite might be a more appropriate choice for the size of the dataset.

Addresses

The small number of the elements that were requiring further attention (13) allowed me to examine them one by one. There were three categories of problems.
In the first category belong elements that some values have been placed to wrong attributes (e.g. housenumber in the place of postcode. These problems resolved just by checking the attributes and update the relevant tables with the righ keys/values relation.
Incomplete addresses with no self-explained errors belong to the second category. For these elements I defined a function that uses Google Maps API to resolve the full address from a partial address. This was helpful for the addresses with missing postcodes.
Finally, whatever could not be resolved with one of the above ways I used web search with any information available.
You may find the changes that took place during this phase in the following table.

Element id	Problematic Attribute	Original Value	Corrected Value
453243296	street	2	(street attribute removed)
453243296	housenumber	(missing)	2
453253763	street	2	(street attribute removed)
453253763	housenumber	(missing)	2
453227146	street	65	Alexandra Terrace
46649997	street	65	Alexandra Terrace
169844052	street	310074	Lor 4 Toa Payoh
169844052	postcode	74	310074
169844052	housenumber	(missing)	74
1318498347	postcode	135	(postcode attribute removed)
1318498347	housenumber	(missing)	135
3026819436	postcode	2424	238841
3756813987	postcode	05901	059011
4338649392	postcode	88752	088752
4338649392	housenumber	(missing)	279
4338649392	street	(missing)	New Bridge Road
4496749591	postcode	#B1-42	(element deleted)
23946435	postcode	437 437	437437
172769494	postcode	05901	059011

Amenities

In Singapore they refer to the banks with their abbreviations rather than their complete names. This fact along with their popularity of the ATMs on the street amenities list (will be presented later) makes them prone to mistakes. The assessment revealed only 5 issues:

- 'UOB' referred as 'Uob'
- 'POSB' referred as 'Posb'
- 'OCBC' referred as 'Overseas Chinese Banking Corporation'
- 2 completely irrelevant nodes marked as 'ATM'

Data Exploration

Dataset Specific

Size of the database

```
In [15]: size_in_bytes = %sql SELECT pg_database_size('Project_3');
         print "DB size: " + str(size_in_bytes[0][0]/1024**2) + ' MB'
```

```
DB size: 96 MB
```

Number of Unique Users

```
In [24]: %%sql
         SELECT count(DISTINCT(uid)) AS "Unique Users"
         FROM (SELECT uid FROM nodes
               UNION SELECT uid FROM ways) AS elements

Out[24]: [(847L,)]
```

Number of *Nodes* and *Ways*

```
In [18]: n_nodes = %sql SELECT COUNT(*) FROM nodes
         n_ways = %sql SELECT COUNT(*) FROM ways

         print "Number of 'nodes: " + str(n_nodes[0][0])
         print "Number of 'ways: " + str(n_ways[0][0])

Number of 'nodes: 409350
Number of 'ways: 66404
```

Area Specific

Most frequent amenities

No surprises here, Singaporeans love food! Restaurant are the first amenity with nearly 3 times more occurrences from the second amenity.

```
In [23]: %%sql
         SELECT value AS "Amenity", COUNT(value) AS "Occurrences"
         FROM (SELECT * FROM nodes_tags
               UNION ALL SELECT * FROM nodes_tags) AS tags
         WHERE key = 'amenity'
         GROUP BY value
         ORDER BY "Occurrences" DESC
         LIMIT 10

Out[23]:
          [(u'restaurant', 1562L),
          (u'parking', 654L),
          (u'taxi', 524L),
          (u'cafe', 396L),
          (u'fast_food', 252L),
          (u'atm', 190L),
          (u'toilets', 190L),
          (u'bar', 176L),
          (u'bank', 120L),
          (u'police', 120L)]
```

Most popular cuisine

```
In [8]: %%sql
        SELECT value AS "Cuisine", COUNT(*) AS "Restaurants"
        FROM (SELECT * FROM nodes_tags
              UNION ALL
              SELECT * FROM ways_tags) tags
        WHERE tags.key='cuisine'
        GROUP BY value
        ORDER BY "Restaurants"  DESC
        LIMIT 10

Out[8]: [(u'chinese', 99L),
         (u'japanese', 42L),
         (u'korean', 36L),
         (u'coffee_shop', 34L),
         (u'burger', 33L),
         (u'italian', 32L),
         (u'indian', 28L),
         (u'asian', 27L),
         (u'pizza', 17L),
         (u'french', 15L)]
```

Religion Singapore is well-known for its multicultural environment. People with different religious and ethnic heritages are forming the modern city-state. This is reflected in the variety of temples that can be found in the country.

```
In [25]: %%sql
         SELECT tags.value AS "Religion", COUNT(*) AS "Temples"
         FROM (SELECT * FROM nodes_tags
               UNION ALL
               SELECT * FROM ways_tags) tags
         WHERE tags.key='religion'
         GROUP BY tags.value
         ORDER BY "Temples" DESC;

Out[25]: [(u'christian', 73L),
          (u'muslim', 38L),
          (u'buddhist', 30L),
          (u'hindu', 9L),
          (u'taoist', 6L),
          (u'jewish', 1L),
          (u'sikh', 1L)]
```

Ideas for additional improvements.

There are two areas where the current project can be improved in the future. The first one is on the completeness of the data. All the above analysis is based on a dataset that reflects a big part of Singapore but not the whole country. The reason for this is the lack of a way to download a dataset big enough to include the entire Singapore without including parts of the neighboring countries.
As a future improvement, I would download the metro extract from MapZen and filter the non-Singaporean nodes and their references. The filtering would involve the creation of polygons from a suitable shapefile like this one with a GIS library like Fiona and the comparison of *Nodes*' coordinates with these polygons with a geometric library like Shapely.
The drawback of the above technique is that the comparison of each node against the polygons is a very time-consuming procedure with my initial tests taking 17-18 hours to produce a result.
The second area with room for future improvement is the exploratory analysis. Although the scope of the current project was the wrangling of the dataset, thus the exploration has been kept in a basic level, the dataset is full of information that can be extracted. Just to mention some:
- Distribution of commits per contributor. - Plotting of element creation date, per type, per day. - Popular franchises in the country (fast food, conventional stores, etc.)
And even more interesting such us:

- Distribution of distance between different types of amenities.
- Selection of a bank based on the average distance you have to walk for an ATM.
- Which area has the biggest parks and recreation spaces.

References

Udacity - https://www.udacity.com/
Wikipedia - https://www.wikipedia.org/
OpenStreetMap - https://www.openstreetmap.org
Overpass API - http://overpass-api.de/
Python Software Foundation - https://www.python.org/
Urban Redevelopment Authority of Singapore - https://www.ura.gov.sg
Catherine Devlin's Github repository - https://github.com/catherinedevlin/ipython-sql