OpenStreetMap Project Data Wrangling with SQL

1. Problems found in the map(Singapore):

1.1 To understand the osm_file:

The count of different types of the tags in the file:
{'node': 621126, 'nd': 763686, 'bounds': 1, 'member': 56949, 'tag': 299163, 'relation': 1004, 'way': 91522, 'osm': 1}

Note: parse_file.py

1.2 To understand types of the tag

In this study, the tags are grouped into 4 categories in the dictionary:

a. "lower", for tags where only lowercase letters are there and valid
b. "lower_colon", for valid tags with colon in the name
c. "problemchars", for tags with problematic characters
d. "other", for tags not belong to the above

Python regular expressions are used to define them:

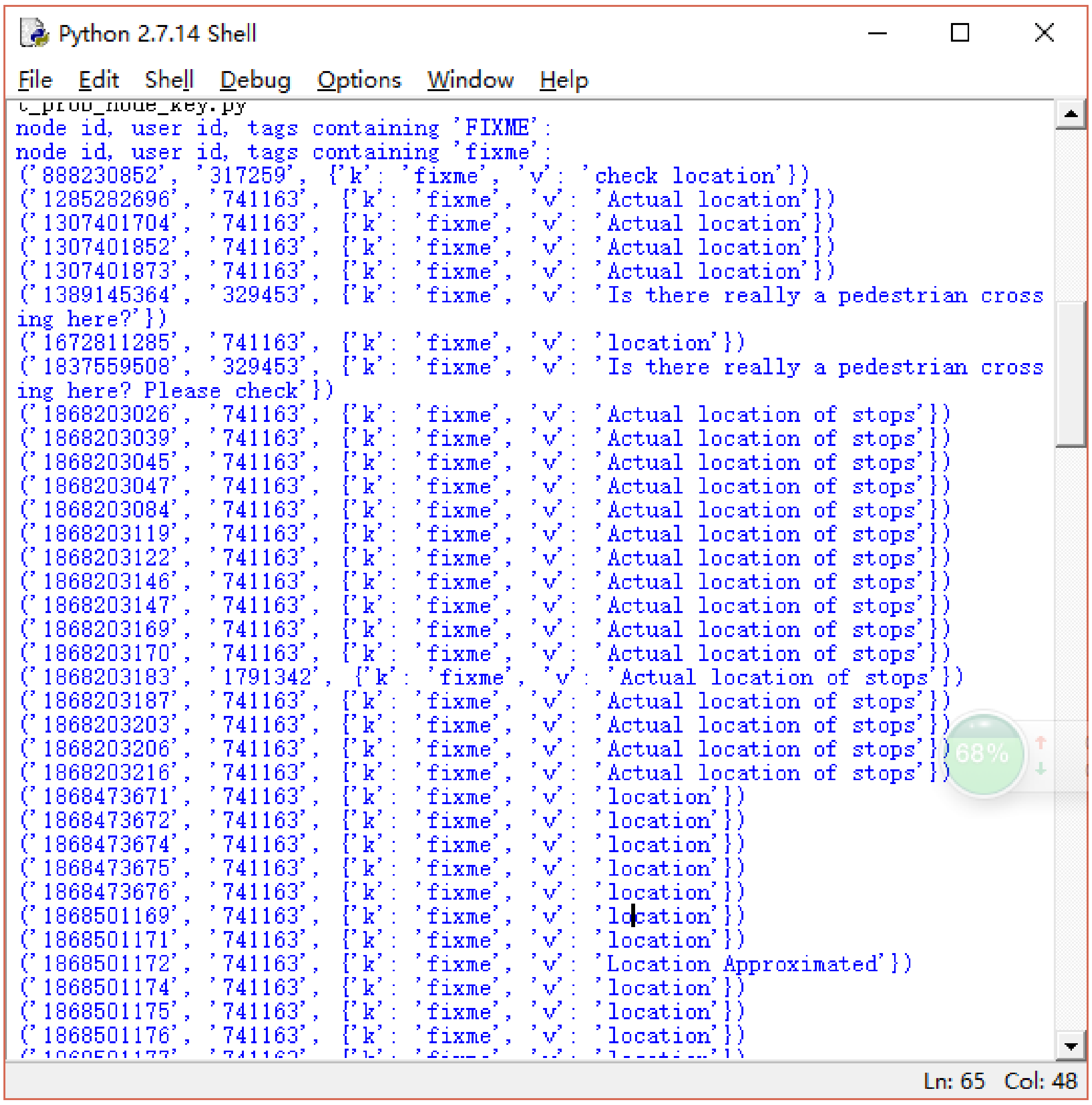
```
lower = re.compile(r'^([a-z]|_)*$', re.IGNORECASE)
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$', re.IGNORECASE)
problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]',
re.IGNORECASE)
```

The number of keys for each type is:
{'problemchars': 23, 'lower': 156, 'total': 782, 'lower colon': 272, 'others': 331}

Note: audit_node_tag.py

1.3 FIXME or fixme:

These tags represent missing or uncertain values.
A few examples are:



```
t_prob_node_key.py
node id, user id, tags containing 'FIXME':
node id, user id, tags containing 'fixme':
('888230852', '317259', {'k': 'fixme', 'v': 'check location'})
('1285282696', '741163', {'k': 'fixme', 'v': 'Actual location'})
('1307401704', '741163', {'k': 'fixme', 'v': 'Actual location'})
('1307401852', '741163', {'k': 'fixme', 'v': 'Actual location'})
('1307401873', '741163', {'k': 'fixme', 'v': 'Actual location'})
('1389145364', '329453', {'k': 'fixme', 'v': 'Is there really a pedestrian cross
ing here?'})
('1672811285', '741163', {'k': 'fixme', 'v': 'location'})
('1837559508', '329453', {'k': 'fixme', 'v': 'Is there really a pedestrian cross
ing here? Please check'})
('1868203026', '741163', {'k': 'fixme', 'v': 'Actual location of stops'})
('1868203039', '741163', {'k': 'fixme', 'v': 'Actual location of stops'})
('1868203045', '741163', {'k': 'fixme', 'v': 'Actual location of stops'})
('1868203047', '741163', {'k': 'fixme', 'v': 'Actual location of stops'})
('1868203084', '741163', {'k': 'fixme', 'v': 'Actual location of stops'})
('1868203119', '741163', {'k': 'fixme', 'v': 'Actual location of stops'})
('1868203122', '741163', {'k': 'fixme', 'v': 'Actual location of stops'})
('1868203146', '741163', {'k': 'fixme', 'v': 'Actual location of stops'})
('1868203147', '741163', {'k': 'fixme', 'v': 'Actual location of stops'})
('1868203169', '741163', {'k': 'fixme', 'v': 'Actual location of stops'})
('1868203170', '741163', {'k': 'fixme', 'v': 'Actual location of stops'})
('1868203183', '1791342', {'k': 'fixme', 'v': 'Actual location of stops'})
('1868203187', '741163', {'k': 'fixme', 'v': 'Actual location of stops'})
('1868203203', '741163', {'k': 'fixme', 'v': 'Actual location of stops'})
('1868203206', '741163', {'k': 'fixme', 'v': 'Actual location of stops'})
('1868203216', '741163', {'k': 'fixme', 'v': 'Actual location of stops'})
('1868473671', '741163', {'k': 'fixme', 'v': 'location'})
('1868473672', '741163', {'k': 'fixme', 'v': 'location'})
('1868473674', '741163', {'k': 'fixme', 'v': 'location'})
('1868473675', '741163', {'k': 'fixme', 'v': 'location'})
('1868473676', '741163', {'k': 'fixme', 'v': 'location'})
('1868501169', '741163', {'k': 'fixme', 'v': 'location'})
('1868501171', '741163', {'k': 'fixme', 'v': 'location'})
('1868501172', '741163', {'k': 'fixme', 'v': 'Location Approximated'})
('1868501174', '741163', {'k': 'fixme', 'v': 'location'})
('1868501175', '741163', {'k': 'fixme', 'v': 'location'})
('1868501176', '741163', {'k': 'fixme', 'v': 'location'})
```

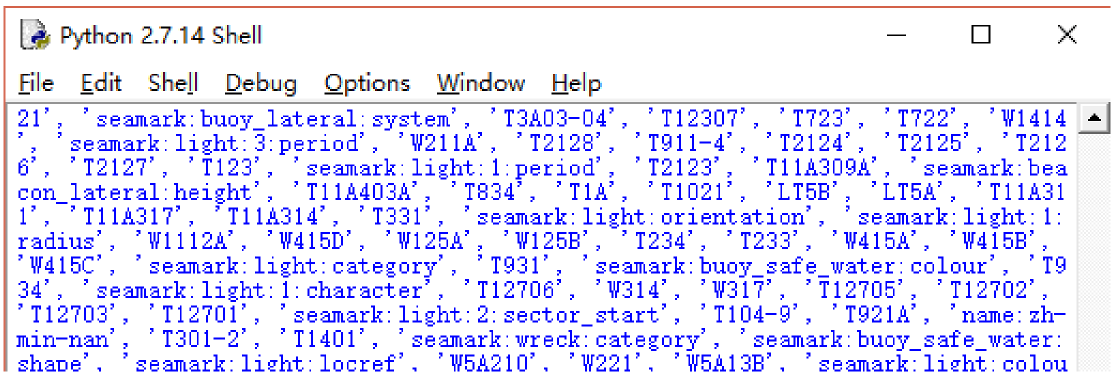
To further check these points, I found that the locations are accurate within 30m. For questions like “Is there really a pedestrian crossing here”, the answer is yes. For values like “continuation”, it may suggest that it is the end of the way. This will be fixed later.

1.4 node tag keys with problem chars:

set(['W3110/A', 'T12501/02', 'W3110A/B', 'T12503/04', 'T12309/10', 'T1451-2/4-6', 'T12401/02', 'W414A/B', 'Family Entertainment', 'T12303/04', 'Jalan Wong Ah Fook', 'T12505/06', 'T332/3', 'T1032-3/4', 'T12305/06', 'Singapore Poly', 'T12201/02', 'T12301/02', 'T12403/04', 'LT 7', 'T12405/06', 'T1472/4-5', 'W413A/B'])

These keys contain white space and ‘/’. To make data consistent, it’s better to change them into underscore (if therer is ‘.’, it could be changed into colon).

1.5 node tag keys with others:



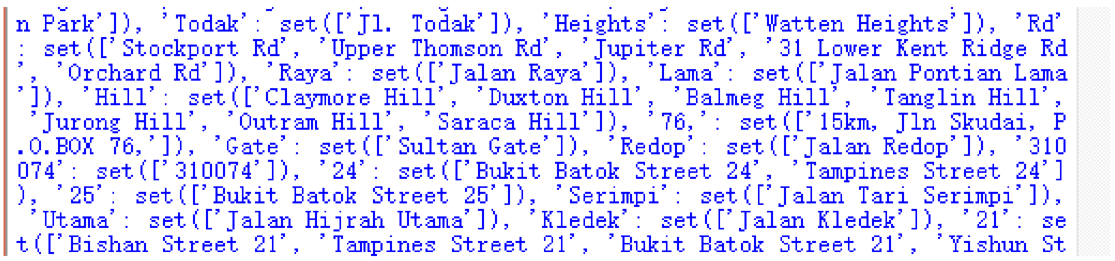
```
21', 'seamark:buoy_lateral:system', 'T3A03-04', 'T12307', 'T723', 'T722', 'W1414
', 'seamark:light:3:period', 'W211A', 'T2128', 'T911-4', 'T2124', 'T2125', 'T212
6', 'T2127', 'T123', 'seamark:light:1:period', 'T2123', 'T11A309A', 'seamark:bea
con_lateral:height', 'T11A403A', 'T834', 'T1A', 'T1021', 'LT5B', 'LT5A', 'T11A31
1', 'T11A317', 'T11A314', 'T331', 'seamark:light:orientation', 'seamark:light:1:
radius', 'W1112A', 'W415D', 'W125A', 'W125B', 'T234', 'T233', 'W415A', 'W415B',
'W415C', 'seamark:light:category', 'T931', 'seamark:buoy_safe_water:colour', 'T9
34', 'seamark:light:1:character', 'T12706', 'W314', 'W317', 'T12705', 'T12702',
'T12703', 'T12701', 'seamark:light:2:sector_start', 'T104-9', 'T921A', 'name:zh-
min-nan', 'T301-2', 'T1401', 'seamark:wreck:category', 'seamark:buoy_safe_water:
shape', 'seamark:light:locref', 'W5A210', 'W221', 'W5A13B', 'seamark:light:colou
```

It seems that there are multiple colons in some keys; they will be fixed later. In addition, there are numbers in some keys, they will be fixed, too (to be done).

1.6 Audit Street names of nodes and ways

Note: audit_street_names.py

```
expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place",
"Square", "Lane", "Road",
           "Trail", "Parkway",
"Alley","Plaza","Commons","Broadway","Expressway","Terrace","Center","C
ircle",
           "Crescent","Highway","Way"]
```



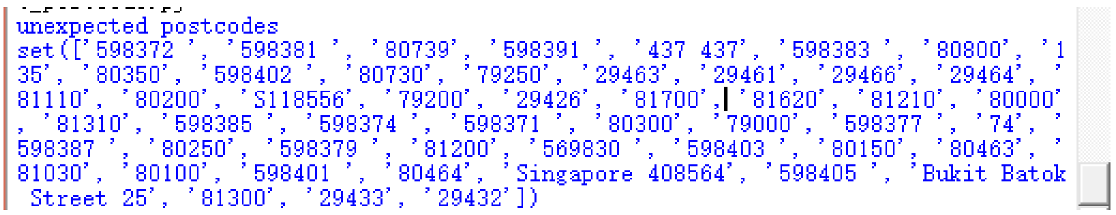
```
n Park']), 'Todak': set(['Jl. Todak']), 'Heights': set(['Watten Heights']), 'Rd'
: set(['Stockport Rd', 'Upper Thomson Rd', 'Jupiter Rd', '31 Lower Kent Ridge Rd
', 'Orchard Rd']), 'Raya': set(['Jalan Raya']), 'Lama': set(['Jalan Pontian Lama
']), 'Hill': set(['Claymore Hill', 'Duxton Hill', 'Balmeg Hill', 'Tanglin Hill',
 'Jurong Hill', 'Outram Hill', 'Saraca Hill']), '76,': set(['15km, Jln Skudai, P
.O.BOX 76,']), 'Gate': set(['Sultan Gate']), 'Redop': set(['Jalan Redop']), '310
074': set(['310074']), '24': set(['Bukit Batok Street 24', 'Tampines Street 24']
), '25': set(['Bukit Batok Street 25']), 'Serimpi': set(['Jalan Tari Serimpi']),
 'Utama': set(['Jalan Hijrah Utama']), 'Kledek': set(['Jalan Kledek']), '21': se
t(['Bishan Street 21', 'Tampines Street 21', 'Bukit Batok Street 21', 'Yishun St
```

There are different variations and typos of those expected street names e.g., Rd (short of Road); they will be fixed by using mapping = {

```
"Ave":"Avenue",
"Ave.":"Avenue",
"Avene":"Avenue",
"Aveneu":"Avenue",
"ave":"Avenue",
"avenue":"Avenue",
"Blv.":"Boulevard",
"Blvd":"Boulevard",
"blvd":"Boulevard",
"Broadway.":"Broadway",
"Ctr":"Center",
"Pkwy":"Parkway",
"Plz":"Plaza",
"Rd":"Road",
"Rd.":"Road",
"ST":"Street",
"St":"Street",
"St.":"Street",
"Steet":"Street",
"Streeet":"Street",
"st":"Street",
"street":"Street"
}
```

1.7 Audit postcode inconsistency

Note: audit_postcode.py



```
unexpected postcodes
set(['598372 ', '598381 ', '80739', '598391 ', '437 437', '598383 ', '80800', '1
35', '80350', '598402 ', '80730', '79250', '29463', '29461', '29466', '29464', '
81110', '80200', 'S118556', '79200', '29426', '81700', '81620', '81210', '80000'
, '81310', '598385 ', '598374 ', '598371 ', '80300', '79000', '598377 ', '74', '
598387 ', '80250', '598379 ', '81200', '569830 ', '598403 ', '80150', '80463', '
81030', '80100', '598401 ', '80464', 'Singapore 408564', '598405 ', 'Bukit Batok
 Street 25', '81300', '29433', '29432'])
```

It can be seen that the unexpected values mainly come from the white space, or only 5 digits, or Singapore or ‘S’ is included in the postcode. In addition, there is one value “Bukit Batok Street 25”, which should be the address. They will be fixed.

2. Overview of the data

2.1 Clean XML data

2.1.1 Clean node tag keys

Clean the data as discussed in 1.3, 1.4 and 1.5. A new file is outputted as Singapore_clean_node_tag_keys.osm.

Note: clean_node_tag_keys.py

Run audit_node_tag.py on cleaned file,
Result:The number of keys for each type is:
{'problemchars': 1, 'lower': 155, 'total': 749, 'lower colon': 306, 'others': 287}

It can be seen that both number of problemchars and number of others are reduced compared with the original file. This time the numeric is treated same as letter.
Result: The number of keys for each type is:
{'problemchars': 1, 'lower': 370, 'total': 749, 'lower colon': 342, 'others': 36}
There are still 36 ‘others’ to be cleaned.

The reason new files are created includes:

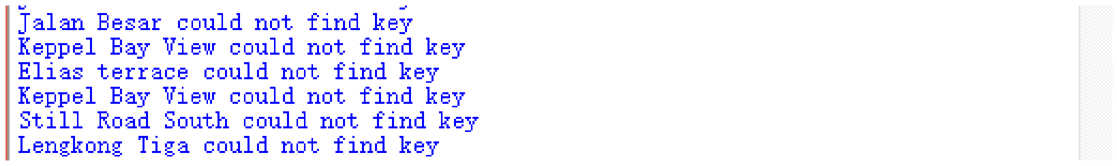
1. Other functions like audit_node_tag can be applied to the cleaned file to double check the clean process.
2. The cleaned file could be used by the community.

2.1.2 Clean Street Names and postcode

Clean data as discussed in 1.6 and 1.7.
Note: clean_street_names_postcode.py on singapore_clean_node_tag.osm and output to singapore_cleaned.osm.

A lot of street names could not match the expected and the mapping, hence we should update the mapping based on these street names.

Part of the result:



```
Jalan Besar could not find key
Keppel Bay View could not find key
Elias terrace could not find key
Keppel Bay View could not find key
Still Road South could not find key
Lengkong Tiga could not find key
```

Some updates:
Walk, View, Quay, Park, Hill, Link are valid full street types in Singapore.

In addition, there are some typos, e.g.: “Bayfront Avebue” should be “Bayfront Avenue”

Multiple cycles are running in order to clean the street names.

If audit_postcode.py is running on the cleaned file, no wrong postcode exists. So the files are left here at the current moment.

2.2 Export XML to CSV file

With defined schema in myschema.py and export_to_csv_schema.py, the XML files are exported into different CSV files.

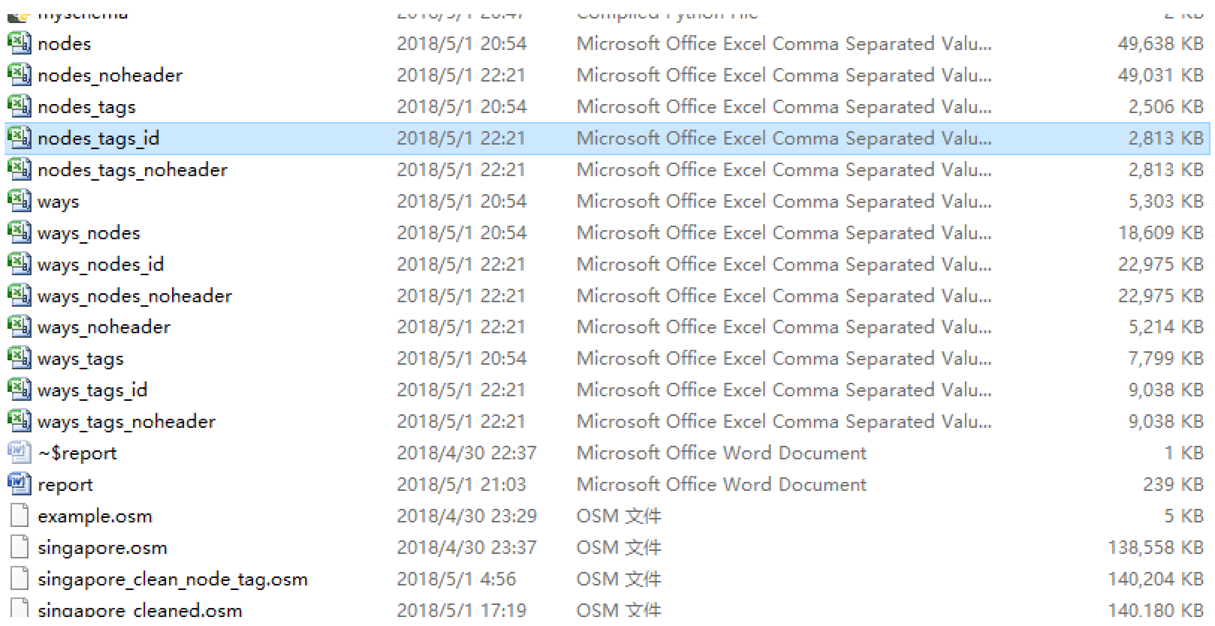
After the inspection on nodes_tag.csv, each node may have multiple tags and there is no unique key in the file. Therefore, I assign Integer column called id to the file. The same approach is used for ways_tags.csv and way_nodes.csv. The headers are also removed.

Note: skip_csv_header.py.

2.3 Import CSV into database

Using PostgreSQL as the software, with pdmin as the GUI, the csv could be imported into database with user-friendly operations.

2.4 General information of the files



myschema	2018/5/1 20:47	Compiled Python File	2 KB
nodes	2018/5/1 20:54	Microsoft Office Excel Comma Separated Valu...	49,638 KB
nodes_noheader	2018/5/1 22:21	Microsoft Office Excel Comma Separated Valu...	49,031 KB
nodes_tags	2018/5/1 20:54	Microsoft Office Excel Comma Separated Valu...	2,506 KB
nodes_tags_id	2018/5/1 22:21	Microsoft Office Excel Comma Separated Valu...	2,813 KB
nodes_tags_noheader	2018/5/1 22:21	Microsoft Office Excel Comma Separated Valu...	2,813 KB
ways	2018/5/1 20:54	Microsoft Office Excel Comma Separated Valu...	5,303 KB
ways_nodes	2018/5/1 20:54	Microsoft Office Excel Comma Separated Valu...	18,609 KB
ways_nodes_id	2018/5/1 22:21	Microsoft Office Excel Comma Separated Valu...	22,975 KB
ways_nodes_noheader	2018/5/1 22:21	Microsoft Office Excel Comma Separated Valu...	22,975 KB
ways_noheader	2018/5/1 22:21	Microsoft Office Excel Comma Separated Valu...	5,214 KB
ways_tags	2018/5/1 20:54	Microsoft Office Excel Comma Separated Valu...	7,799 KB
ways_tags_id	2018/5/1 22:21	Microsoft Office Excel Comma Separated Valu...	9,038 KB
ways_tags_noheader	2018/5/1 22:21	Microsoft Office Excel Comma Separated Valu...	9,038 KB
~$report	2018/4/30 22:37	Microsoft Office Word Document	1 KB
report	2018/5/1 21:03	Microsoft Office Word Document	239 KB
example.osm	2018/4/30 23:29	OSM 文件	5 KB
singapore.osm	2018/4/30 23:37	OSM 文件	138,558 KB
singapore_clean_node_tag.osm	2018/5/1 4:56	OSM 文件	140,204 KB
singapore_cleaned.osm	2018/5/1 17:19	OSM 文件	140,180 KB

2.5 Explore the database with SQL query

a. How many nodes and ways are there?

SELECT count(*) FROM “Node” (/”Way”)

There are 621126 nodes and 91522 ways.

b. How many nodes are highways? How many nodes are shops?

SELECT count(*)
FROM public."Node_tag"
WHERE "Key" = 'highway'

There are 6241 highways.

SELECT count(*)
FROM public."Node_tag"
WHERE "Key" = shop

There are 454 shops.

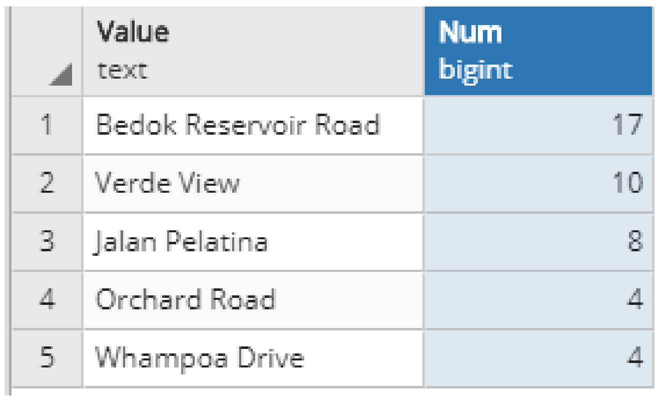
c. How many distinct street names? What are the top 5 names?

SELECT
count(distinct "Value")
FROM public."Node_tag"
WHERE "Key" = 'street'

There are 153 unique street names.

The top 5 popular street names are:

```
SELECT "Value",
count(*) as "Num"
FROM public."Node_tag"
WHERE "Key" = 'street'
GROUP By "Value"
ORDER By "Num" DESC
LIMIT 5
```



	Value text	Num bigint
1	Bedok Reservoir Road	17
2	Verde View	10
3	Jalan Pelatina	8
4	Orchard Road	4
5	Whampoa Drive	4

If we further check the full result, it seems that there are other popular street names, but the street number is included. Therefore, the count is not as large as others. This implies that further data cleaning is necessary to separate the street name and street number.

d. How many unique users?

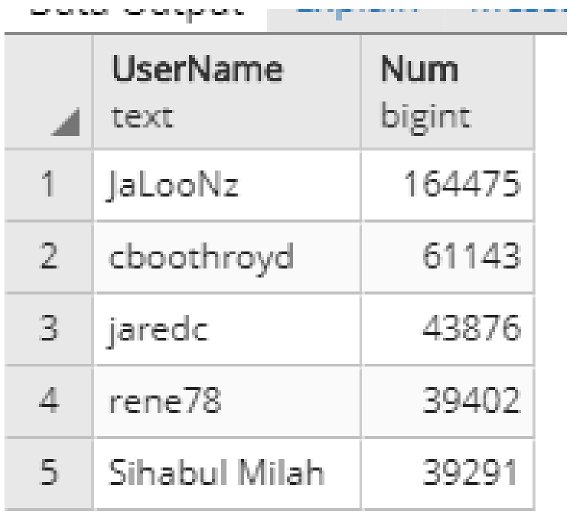
```
SELECT count(distinct (subq."UserId"))
FROM (SELECT "UserId" from public."Node" union all SELECT "UserId" from public."Way") as subq
```

There are 662 unique users.

e. Who are top 5 contributors?

```
SELECT subq."UserName",count(*) as "Num"
FROM (SELECT "UserName" from public."Node" union all SELECT "UserName" from public."Way") as subq
GROUP BY subq."UserName"
ORDER BY "Num" DESC
LIMIT 5
```

They are:



	UserName text	Num bigint
1	JaLooNz	164475
2	cboothroyd	61143
3	jaredc	43876
4	rene78	39402
5	Sihabul Milah	39291

3. Other ideas about the dataset

3.1 Automatic audit

The part 1 mostly discusses about the data quality. 1.3 (“Fixme”) is related to the data accuracy and completeness; while 1.4-1.7 are related to the data uniformity and format. Because the data could be contributed from different individual and in different language, the data could vary from each other. Implementation of the quality assurance approach would increase the data readability.

What has been done for the data:
a. Fix those “fixme” tag keys. By double checking the data, the data accuracy is increased.
b. Format character: problem chars has been converted to “_”.
c. Format street names: full names are used instead of abbreviation.
d. Format postcode: only 6 digits are allowed; in addition, “Singapore” or “S” has been removed in the postcode, leaving only the numeric values.

Areas not addressed:
1. Fix “Fixme” part is done manually. It would be great if automatic validation process could be done; a full list of such tags would also be useful for the validation process.
2. Format. In the input session, certain codes could be provided to clean the data before it is input. Therefore, all the data in the set would apply the same format.

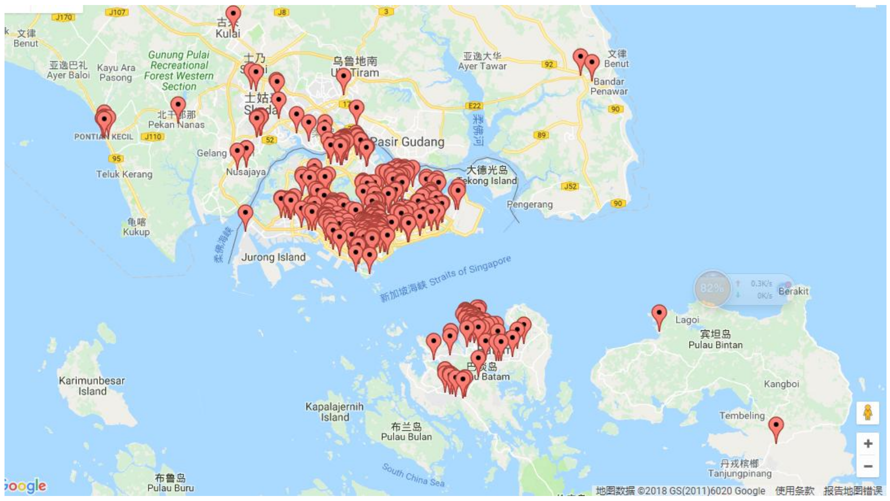
3.2 Explore shops

Shopping is very popular in Singapore. Therefore, it would be convenient that all the shops could be complied such that people can have a better shopping experience.

As in 2.5b, there are 454 shops in the map. All the latitude and longitude could be selected by:
SELECT public."Node"."Latitude",public."Node"."Longitude"
FROM public."Node", (SELECT DISTINCT("NodeId") from public."Node_tag" where "Key"

= 'shop') as subq
where public."Node"."NodeId"=subq."NodeId"

These nodes could be plotted on Google Map using www.darrinward.com/lat-long/



Most shops are located in central Singapore, and there are also a few choices which is close to Nanyang Technological University (where I stay for 9 years).

南洋理工大学
Nanyang Technological University
NTU Hall of Residence 1
NTU Hall of Residence 7
Chinese Heritage Centre
Yunnan Garden @ NTU
Nanyang Flyover
PIE
City Harvest Church
Pioneer Rd North
Jurong West Stadium
Yunnan Park
Pioneer
Gek Poh Ville CC
Boon Lay Secondary School
Frontier Primary School