OpenStreetMap Data Case Study

Map Area

Melbourne,Australia (9,990.5 km2)

- https://www.openstreetmap.org/node/21579127#map=11/-37.8139/144.9632
- https://mapzen.com/data/metro-extracts/metro/melbourne_australia/

Chose this city randomely having metro extract greater than 50 Mb

Problems Encountered in Map Area(sample.osm)

- Overabbreviated street names ("*Intrepid Av*")
- Inconsistency in the Postal Code ("*(3006;3130),3206Unset,38058*")
- "Incorrect" postal codes (Melbourne area zip codes all begin with "300" and postal codes must be 4 digit only however a large portion of all documented zip codes were outside this region.)
- Most of the cities are not part of Melbourne but present on the outskirt of Melbourne. ("*Caroline Springs is a suburb of Melbourne, Victoria, Australia, 25 km west of Melbourne's Central Business District.*")
- Typo Errors or City name Inconsistencies =>Misspelled city("*Moridalloc* => *Moriadlloc* ")

Overabbreviated Street Names

In [16]:

```
import xml.etree.cElementTree as ET
from collections import defaultdict
import re
import pprint

OSMFILE = "sample.osm"
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)


expected = ["St","St.","Rd.","Av","Gr",'Stg',"Rd"]

cities =[""]
```

```
# UPDATE THIS VARIABLE
mapping = { "St": "Street",
            "St.": "Street",
            "Rd.":"Road",
            "Av":"Avenue",
           "Gr":'Grove',
           'Stg':"Street"
            }


def audit_street_type(street_types, street_name):
    abbr=False
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type in expected:
            abbr=True
            street_types[street_type].add(street_name)
    return abbr


def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")

def audit(osmfile):
    osm_file = open(osmfile, "r")
    street_types = defaultdict(set)
    for event, elem in ET.iterparse(osm_file, events=("start",)):

        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_street_name(tag):

                    abbr=audit_street_type(street_types, tag.attrib['v'])
                    if(abbr):
                        tag.attrib['v']=update_name( tag.attrib['v'],
mapping)

    osm_file.close()
    return street_types

def audit_city(filename):
    osm_file = open(filename, "r")
    count=0
    cities=0
    city_list = set()
    for event, elem in ET.iterparse(osm_file, events=("start",)):
        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if tag.attrib['k'] == "addr:city":
                    cities=cities+1
                if tag.attrib['k'] == "addr:city" and tag.attrib['v'] != "Me
lbourne":
                    count=count+1
                    city_list.add(tag.attrib['v'])
    print "Total number of cities: "+ str(cities)
    print "Total number of suburbs or Neighbouring areas: "+str(count)
    return city_list

def update_name(name, mapping)
```

```
def update_name(name, mapping):
    m = street_type_re.search(name)
    for key,value in (mapping).iteritems():
        if key==m.group():
           name=name.replace(key,value)


    return name


def test():
    st_types = audit(OSMFILE)
    city_list = audit_city(OSMFILE)
    pprint.pprint(dict(st_types))
#     pprint.pprint(city_list)
    print "\n Few suburbs:"
    pprint.pprint([[x for x in city_list][j] for j in range(5)] )
    print "\n Misspelled City:"
    pprint.pprint([x for x in city_list if(x=="Moridalloc")] )
    print "\n After Auditing Street Names:"
    for st_type, ways in st_types.iteritems():
        for name in ways:
            better_name = update_name(name, mapping)
            print name, "=>", better_name


if __name__ == '__main__':
    test()
```

```
Total number of cities: 7220
Total number of suburbs or Neighbouring areas: 6762
{'Av': set(['Intrepid Av']),
 'Gr': set(['McCarthy Gr']),
 'St': set(['Queen St']),
 'Stg': set(['Leigh Stg'])}

 Few suburbs:
['Caroline Springs',
 'North Warrandyte',
 'Geelong',
 'reservoir',
 'Rosebud West']

 Misspelled City:
['Moridalloc']

 Auditing in Street Names:
Intrepid Av => Intrepid Avenue
Leigh Stg => Leigh Street
McCarthy Gr => McCarthy Grove
Queen St => Queen Street
```

Suburbs near Melbourne

Most of the cities are few Kms distant from main city of Melbourne.The 90% of datset comprises of suburbs of Melbourne i.e. is the a separate residential community within commuting distance of a city of Melbourne.

Inconsistency in Postal Code

When auditing the postal code removed the unwanted text and found the sets to postal codes then took the first postal code more relavant to describe the city.

In [18]:

```
import xml.etree.cElementTree as ET
from collections import defaultdict
import re
import pprint

OSMFILE = "sample.osm"
postal_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)
zip_code_re = re.compile(r'^\d{4}$')
fix_zipcode_state_short = re.compile(r'\d{4};\d{4}$')


def audit_postal_type(postal_types, postal_name):
    m = postal_type_re.search(postal_name)
    if m:
        postal_type = m.group()
        if len(postal_type)>4 or len(postal_type)<4:
            postal_types[postal_type].add(postal_name)

def audit_zip_code(zip_code):
    change=False
    new_code=zip_code
    zip_code = zip_code.strip()

    m = zip_code_re.search(zip_code)
    if zip_code[4:9] == 'Unset':
        change=True
        new_code = zip_code[0:4]
        return change,new_code,zip_code
    if fix_zipcode_state_short.search(zip_code):
        change=True
        new_code = zip_code[0:4]
        return change,new_code,zip_code
    if m:
        return change,new_code,zip_code
    if zip_code=='':
        return True,"none","none"
    else:
        return change,new_code,zip_code


def is_postal_name(elem):
    return (elem.attrib['k'] == "addr:postcode")


def audit(osmfile):
    osm_file = open(osmfile, "r")
    postal_types = defaultdict(set)
    print "After Auditing:"
    for event, elem in ET.iterparse(osm_file, events=("start",)):
```

```
        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_postal_name(tag):
                    audit_postal_type(postal_types, tag.attrib['v'])
                    c,n,code=audit_zip_code(tag.attrib['v'])
                    if c:

                        print str(tag.attrib['v']) +"=>"+str(n)
    osm_file.close()
    return postal_types



def test():
    postal_types = audit(OSMFILE)
    print "\n"
    print "Inconsistent Postal Codes:"
    pprint.pprint(dict(postal_types))

if __name__ == '__main__':
    test()
```

```
After Auditing:
3206Unset=>3206
3006;3130=>3006


List of Inconsistent Postal Codes:
{'3006;3130': set(['3006;3130']),
 '3206Unset': set(['3206Unset']),
 '38058': set(['38058'])}
```

Now applying sql queries on smaller database from *sample1.osm*

In [ ]:

```
SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags
      UNION ALL
      SELECT * FROM ways_tags) tags
WHERE tags.key='postcode'
GROUP BY tags.value
ORDER BY count DESC lIMIT 10;
```

Here are the top ten results, beginning with the highest count:

In [ ]:

```
3150,213
3195,105
3152,81
3805,73
3094,37
3149,27
3168,26
3196,23
3088,21
3170,20
```

considering these postal codes from sample1,I checked them on https://postcodes-australia.com/ and found that most of them or not even in melbourne. That struck me as surprisingly high to be a blatant error, and found that the number one postal code “3150” lie in Glen Waverley, SC. So, I performed another aggregation to verify a certain suspicion...

Sort cities by count, descending

In [ ]:

```
sqlite> 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
LMIT 10;
```

And, the top 10 results were:

In [ ]:

```
Glen Waverley|129
Wheelers Hill|66
Wantirna South|62
Mount Waverley|39
Melbourne|37
Montmorency|37
Mulgrave|20
Scoresby|16
Vermont South|16
Clayton|13
```

These results confirmed my suspicion that this metro extract would perhaps be more aptly named “Metrolina” or the “Melbourne Metropolitan Area” for its inclusion of surrounding cities in the sprawl.

Data Overview and Additional Ideas

This section contains basic statistics about the dataset from *sample1.osm*, the sql queries used to gather them, and some additional ideas about the data in context.

File sizes

In [5]:

```
from pprint import pprint
import os
from hurry.filesize import size
#put your own path of folder for which size of file has to be computed
dirpath = 'C:\Users\AKANKSHA\Desktop\udacity\openstreet
project\sqlite_windows'

files_list = []
```

```
for path, dirs, files in os.walk(dirpath):
    files_list.extend([(filename, size(os.path.getsize(os.path.join(path, f
ilename)))) for filename in files])

for filename, size in files_list:
    print '{:.<40s}: {:5s}'.format(filename,size)
```

```
.DS_Store.............................: 6K
melbourne_australia.osm...............: 818M
nodes.csv.............................: 3M
nodes.db..............................: 2M
nodes_tags.csv........................: 287K
node_tags.db..........................: 305K
sample.osm............................: 82M
sample1.osm...........................: 8M
sqlite3.exe...........................: 655K
ways.csv..............................: 309K
ways.db...............................: 292K
ways.db,nodes.db......................: 0B
ways.db,nodes.db;.....................: 0B
ways_nodes.csv........................: 1M
ways_tags.csv.........................: 483K
ways_tags.db..........................: 508K
```

Number of nodes(nodes.db)

In [ ]:

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

38363

Number of ways(ways.db)

In [ ]:

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

5287

Number of unique users

Using Sql:

In [ ]:

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

769

Top 10 contributing users

In [ ]:

```
sqlite> SELECT e.user, COUNT(*) as num
FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
GROUP BY e.user
ORDER BY num DESC
LIMIT 10;
```

In [ ]:

```
CloCkWeRX|11871
Leon K|4539
melb_guy|2883
Glen|1844
AlexOnTheBus|1346
stevage|927
Canley|873
dssis1|781
Supt_of_Printing|766
nickbarker|744
```

Number of users appearing only once (having 1 post)

In [ ]:

```
sqlite> SELECT COUNT(*)
FROM
    (SELECT e.user, COUNT(*) as num
     FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
     GROUP BY e.user
     HAVING num=1)  u;
```

314

Additional Data Exploration

Top 10 appearing amenities

In [ ]:

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

In [ ]:

```
bench,20
cafe,14
parking,14
restaurant,14
toilets,12
waste_basket,11
bbq,9
```

```
bbq,9
pub,9
drinking_water,8
bicycle_parking,7
```

Biggest religion (no surprise here)

In [ ]:

```
sqlite> SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
    JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE
value='place_of_worship') i
    ON nodes_tags.id=i.id
WHERE nodes_tags.key='religion'
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 1;
```

In [ ]:

```
christian|2
scientologist|1
```

Most popular cuisines

In [ ]:

```
SELECT node_tags.value, COUNT(*) as num
FROM node_tags
    JOIN (SELECT DISTINCT(id) FROM node_tags WHERE value='restaurant') i
    ON node_tags.id=i.id
WHERE node_tags.key='cuisine'
GROUP BY node_tags.value
ORDER BY num DESC;

Sushi_Bar|1
chinese;vietnamese|1
italian|1
japanese;chinese|1
```

Contributor statistics

The contributions of users seems incredibly skewed, possibly due to automated versus manual map editing (the word “bot” appears in some usernames). Here are some user percentage statistics:

- Top user contribution percentage (“CloCkWeRX”) 27.19%
- Combined top 2 users' contribution (“CloCkWeRX” and “Leon K”)37.5%
- Combined Top 10 users contribution 60.8%
- Combined number of users making up only 1% of posts(about 41% of all users)

Conclusion

The OpenStreetMap data of Melbourne is of fairly reasonable quality but the typo errors caused by the human inputs are significant as it can be seen in the Postal codes (*3206Unset*) and city names

the human inputs are significant as it can be seen in the Postal codes (3206.0, float) and city names (*Moridalloc*). We have cleaned a significant amount of the data which is required for this project. But, there are lots of improvement needed in the dataset.This metro extract would perhaps be more aptly named “Metrolina” or the “Melbourne Metropolitan Area” for its inclusion of surrounding cities in the sprawl.This is proved by the cities and the postal codes of Neighbouring regions.The dataset contains very less amount of additional information such as amenities, tourist attractions, popular places and other useful interest. So, I think there are several opportunities for cleaning and validation of the data in the future.

After this review of the data it’s obvious that the melbourne area is incomplete,though I believe it has been well cleaned for the purposes of this exercise. It interests me to notice a fair amount of GPS data makes it into OpenStreetMap.org on account of users’ efforts.

Additional Ideas

Gamification: encouraging user participation through incentives

- Thinking about these user percentages, I’m reminded of “gamification” as a motivating force for contribution. In the context of the OpenStreetMap, if user data were more prominently displayed, perhaps others would take an initiative in submitting more edits to the map. And, if everyone sees that only a handful of power users are creating more than 90% a of given map, that might spur the creation of more efficient bots, especially if certain gamification elements were present, such as rewards, badges, or a leaderboard.
- They can keep competition where the person who found the problems and auditted the data appropriately will be provided prizez.(For Example: Kaggle where there are competitions for finding best results)

Clean Typo Errors

- We can build parser which parse every word input by the users.
- We can make some rules or patterns to input data which users follow everytime to input their data. This will also restrict users input in their native language.
- We can develope script or bot to clean the data regularly or certain period.

More Information from users

- The tourists or even the city people search map to see the basic amenities provided in the city or what are the popular places and attractions in the city or near outside the city. So, the users must be motivated to also provide these informations in the map.Motivation can be incentives.
- If we can provide these informations then there are more chances to increase views on the map because many people directly enter the famous name on the map.