Project: Wrangle OpenStreetMap Data With SQL

Table of Contents

Introduction

First of all I downloaded small area of data around where I live from -

https://www.openstreetmap.org/export#map=16/53.2659/-2.8812 (https://www.openstreetmap.org/export#map=16/53.2659/-2.8812)

I used the websites export function and downloaded a small map file.

SmallMap.osm -> 28.6MB

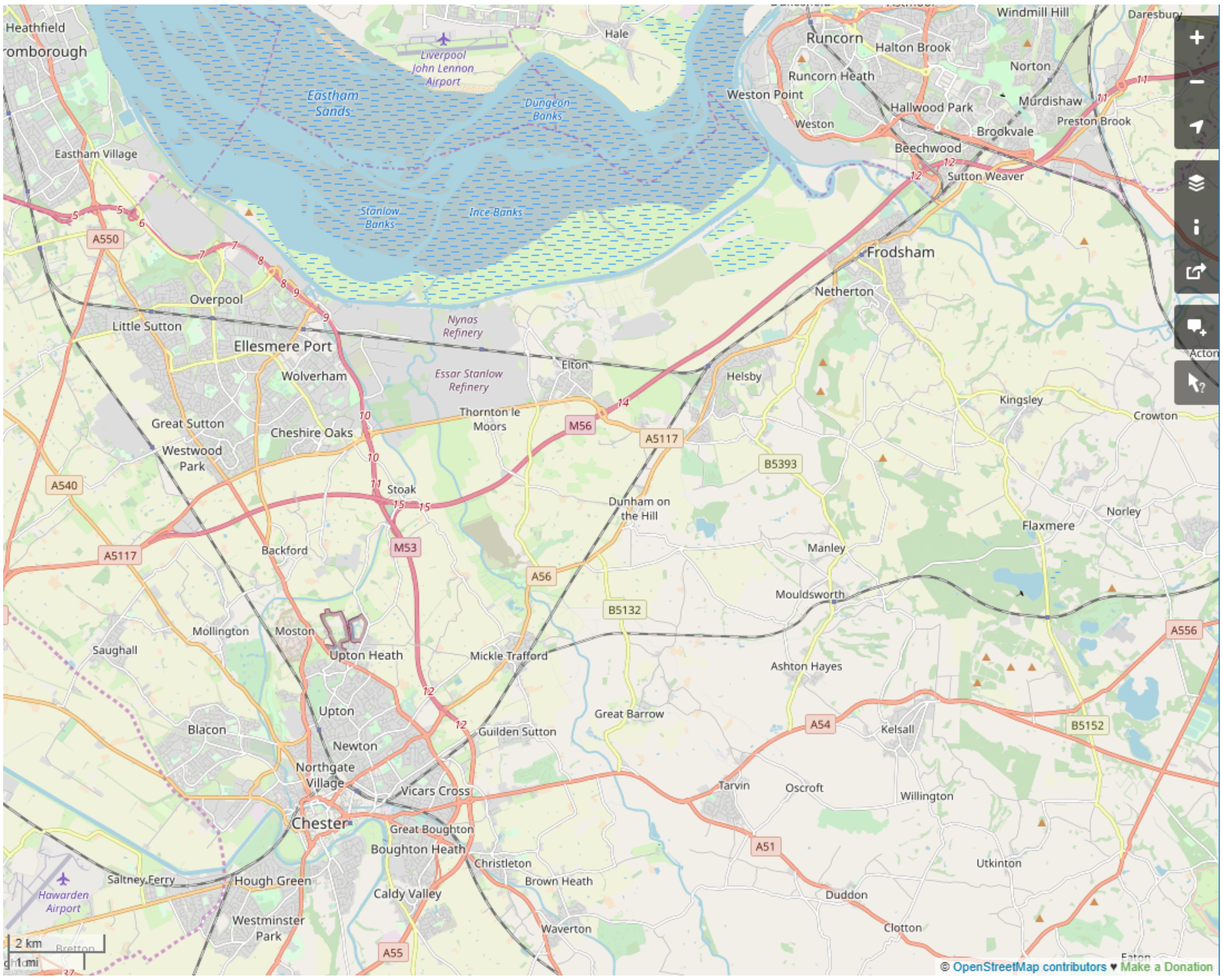
I opened the fie in Notepad++ (https://notepad-plus-plus.org/ (https://notepad-plus-plus.org/)) and searched for my postcode, which was not listed, but my street name was. Further searching in the data I found that the postcodes were only listed for business addresses and postboxes, and not for every street.

This is probably because the full UK postcode database is owned by Royal Mail, who charge for use of the full database, see -https://www.bbc.co.uk/news/business-26605375 (https://www.bbc.co.uk/news/business-26605375)

Map Area

I then expanded to a larger area, including Chester and Runcorn, and also Liverpool and Hawarden airports from -

[https://www.openstreetmap.org/export#map=12/53.2524/-2.8012 (https://www.openstreetmap.org/export#map=12/53.2524/-2.8012)](https://www.openstreetmap.org/export#map=12/53.2524/-2.8012)



Due to the large area, I had to download this using the "Overpass API Query Form" - [http://overpass-api.de/query_form.html (http://overpass-api.de/query_form.html)](http://overpass-api.de/query_form.html)

I used the following command - (node(**bottom,left,top,right**);<;);out meta;

With the map parameters from the open street map, which in my case was -

(node(53.1626,-2.9890,53.3419,-2.6134);<;);out meta;

This gave me a large map file.

LargeMap.osm -> 175MB

Cleaning The Data

I then set about cleaning the data in this Jupyter notebook, first loading the required libraries -

In [1]:

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

Then I parsed the file, counting all of te types of tags (first on the small map, before the large map) -

In [2]:

```
def count_tags(filename):
    counts = defaultdict(int)
    for event, node in ET.iterparse(filename):
        if event == 'end':
            counts[node.tag]+=1
        node.clear()
    return counts

tags = count_tags('LargeMap.osm')      #('SmallMap.osm')
pprint.pprint(tags)
```

```
defaultdict(<type 'int'>, {'node': 762937, 'member': 55116, 'nd': 1116253,
'tag': 367088, 'note': 1, 'meta': 1, 'relation': 2583, 'way': 146821, 'os
m': 1})
```

I then used regular expressions to check for patterns and possible character problems in the data (again first on the small map, before then the large map) -

In [3]:

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

def key_type(element, keys):
    if element.tag == "tag":
        if lower.search(element.attrib['k']):
            keys['lower'] = keys['lower'] + 1
        elif lower_colon.search(element.attrib['k']):
            keys['lower_colon'] = keys['lower_colon'] + 1
        elif problemchars.search(element.attrib['k']):
            keys['problemchars'] = keys['problemchars'] + 1
        else:
            keys['other'] = keys['other'] + 1
        pass
    return keys

def process_map(filename):
    keys = {"lower": 0, "lower_colon": 0, "problemchars": 0, "other": 0}
    for _, element in ET.iterparse(filename):
        keys = key_type(element, keys)
    return keys

keys = process_map('LargeMap.osm')      #('SmallMap.osm')
pprint.pprint(keys)
total = keys['lower'] + keys['lower_colon'] + keys['other']
print total
```

```
{'lower': 240880, 'lower_colon': 111160, 'other': 15048, 'problemchars':
0}
367088
```

I then parsed the file cheking for any errors in the street names (again first on the small map, before then the large map) -

The expected dictionary was created from all street types listed with more than street listed. I only found 5 errors, which I corrected with 4 entries in mapping. Only the 'St' set is shown for clarity.

In [4]:

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

expected = ["Arcades", "Avenue", "Bank", "Brow", "Close", "Cottages", "Court", "Crescen
t", "Croft", "Drive",
            "East", "End", "Farm", "Fold", "Gardens", "Grange", "Green", "Grove", "Heat
h", "Hey", "Heys", "Hill", "Hollow",
            "Meadows", "Mews", "North", "Lane", "Park", "Place", "Rise", "Road", "Row",
 "South", "Street", "Square",
            "Terrace", "View", "Village", "Walk", "Way", "West", "Wharf"]
# Street types with more than one listing added to this list after initially running th
is cell

mapping = { "Rd": "Road",
            "St": "Street",
            "green": "Green",
            "lane": "Lane" }
# Problems found - 'Liverpool Rd', 'Lower Bridge St', 'Phillip St', 'Mondrem green', 'I
rons lane'

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):
                    audit_street_type(street_types, tag.attrib['v'])
    osm_file.close()
    return street_types

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

def audit_street_type(street_types, street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type not in expected:
            street_types[street_type].add(update_name(street_name, mapping))

def update_name(name, mapping):
    m = street_type_re.search(name)
    if m:
        street_type = m.group()
        if street_type in mapping:
            name = re.sub(street_type_re, mapping[street_type], name)
    return name

st_types = audit(OSMFILE)
pprint.pprint(dict(st_types)['St'])
```

```
set(['Lower Bridge Street', 'Phillip Street'])
```

Create CSV Files

I then edited my earlier used **data.py** file in Spyder, to outut the map as csv files, converting using the **schema.py** file from the previous lesson.

Note that the CSV header lines are commented out, as when testing with the smaller area, I had import errors caused by the headers.

This output my 5 csv files -

nodes.csv -----------> 60.7MB

nodes_tags.csv ---> 1.52MB

ways.csv ------------> 8.40MB

ways_nodes.csv -> 26.5MB

ways_tags.csv ----> 10.5MB

Create The SQL Database

I then created a .bat (windows batch) file called **SQL.bat**, to create the database with sqlite3.

Bat file command -

sqlite3.exe EllesmerePort.db ".read CreateDB.sql"

This called up a modified version of the supplied schema - https://gist.github.com/swwelch/f1144229848b407e0a5d13fcb7fbbd6f (https://gist.github.com/swwelch/f1144229848b407e0a5d13fcb7fbbd6f) which I had renamed **CreateDB.sql**

I then ran the batch file, which output my database file -

EllesmerePort.db -> 96.4MB

SQL Queries

First I queried how many nodes and ways were present, which gave the same results as per the python 'count_tags' function earlier -

sqlite> SELECT COUNT (*) from nodes; 762937 sqlite> SELECT COUNT (*) from ways; 146821

I then counted how many unique users were present -

sqlite> SELECT COUNT (DISTINCT(users.uid)) FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) users; 595

And output the top 10 contributing users -

sqlite> SELECT users.user, COUNT (*) as num FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) users GROUP BY users.user ORDER BY num DESC LIMIT 10; daviesp12|479415

Nathan2006|86575 kjnpbr|85129 Dyserth|44091 smb1001|27092 Chris Morley|19145 mikh43|16637 Colin Smale|11479 pairmapper|10375 RobChafer|9701

Further SQL Queries

Then I counted which were the most popular amenities -

sqlite> SELECT value, COUNT (*) as num FROM nodes_tags WHERE key="amenity" GROUP BY value ORDER BY num DESC LIMIT 10; post_box|183 pub|116 bench|91 restaurant|72 telephone|71 fast_food|63 parking|55 cafe|47 post_office|31 fuel|30

As I live next to a large shopping outlet, I wanted to see which were the most common types of shops -

sqlite> SELECT value, COUNT (*) as num FROM nodes_tags WHERE key="shop" GROUP BY value ORDER BY num DESC LIMIT 10; clothes|92 convenience|30 yes|28 hairdresser|27 supermarket|14 car_repair|12 shoes|12 car|11 alcohol|7 doityourself|7

With the shopping outlet mainly being for designer clothes, this result was no suprise.

I then decided to look at which were the most popular food offerings, looking at restaurants, fast food and cafes together -

sqlite> SELECT nodes_tags.value, COUNT (*) as num FROM nodes_tags JOIN (SELECT DISTINCT(id) FROM nodes_tags ...> WHERE value="cafe" OR value="fast_food" OR value="restaurant") ...> food ON nodes_tags.id=food.id WHERE nodes_tags.key="cuisine" GROUP BY nodes_tags.value ORDER BY num DESC LIMIT 10; chinese|14 indian|14 coffee_shop|11 fish_and_chips|11 pizza|8 sandwich|7 italian|6 international|4 chicken|3 regional|3

Conclusions

I noticed that some areas of OpenStreetMap are out of date, such as the EPSV Sports Centre a the entrance to my housing estate, which is still shown as Cheshire Oaks High School. In order to update areas such as this, first someone needs to identify that the data is out of date, before someone can upload the new building data, before verifying that the information is correct.

Verifying OpenStreetMap data would be an ideal candidate for a more local based captcha system. Users could be shown only details from their current position, or from a wider area when then are based at home. Whilst having data checked more often would improve it's accuracy, users may not wish to see their location logged in such detail. Users would want to see the data managed by a trusted firm, with data use transparency being of the utmost importance.