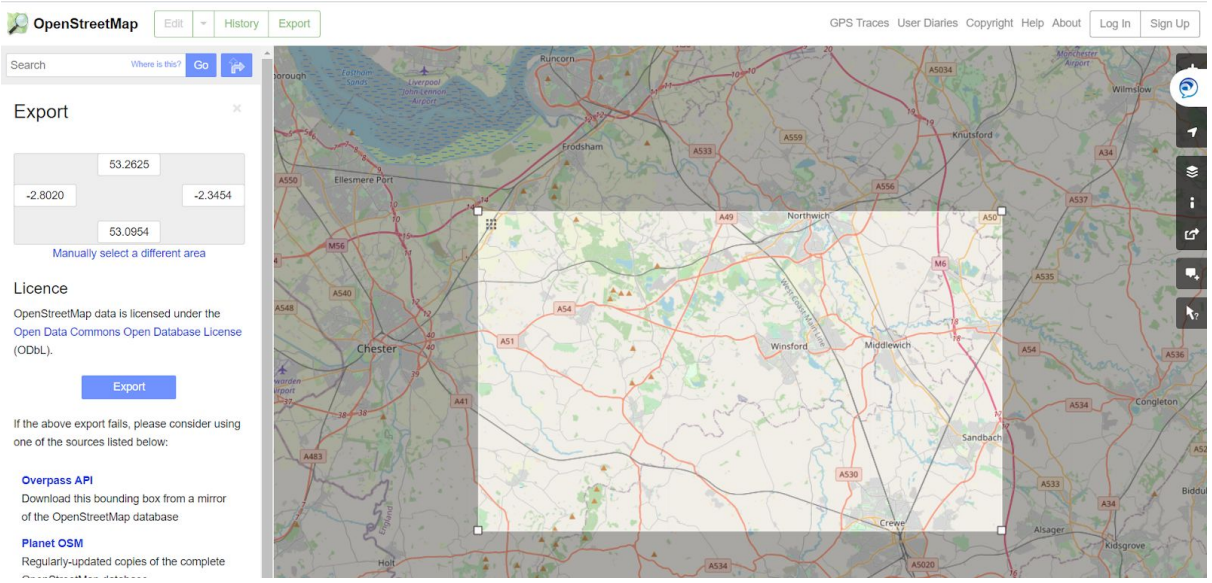
Data wrangling with SQL – Alex Challenor

Area downloaded from OSM:

I chose a large quite rural area which I was born and grew up in.



https://www.openstreetmap.org/export#map=11/53.1789/-2.5735

Data inside the XML file downloaded:

Using “tags hi level.ipynb”

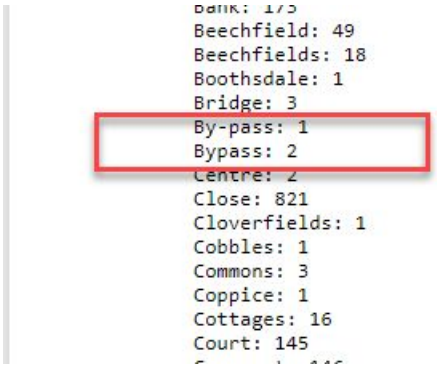
```
{'bounds': 1,
 'member': 43369,
 'meta': 1,
 'nd': 506000,
 'node': 403972,
 'note': 1,
 'osm': 1,
 'relation': 862,
 'tag': 197441,
 'way': 54669}
```

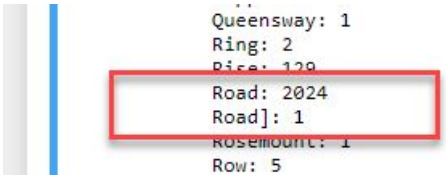
Problems with data:

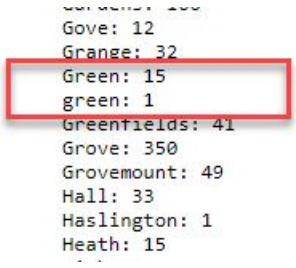
During the data wrangling phase python scripts where used to interrogate the XML data to find areas where the data might need to be “cleaned” going forward.

Below is three examples found when I ran “road names type.ipynb”. These errors are probably down to human errors whilst entering the data into OSM. Looking through the raw XML data you would probably not find this as there is 1000’s of lines of data to look though.

As I have stated human error is the most likely cause of these errors/differences but some of it will come down to the preference of the user that input the data. An example is “By-pass” or Bypass” both of these get the information to the user but because there is a difference it would not read the same to another script or program.







To correct the issues found whilst auditing the data i used a function to update the data. In this instance I corrected the By-pass/Bypass issue highlighted above. In the script that creates the .csv files a function corrrects any ‘By pass’ to ‘By-pass’.

This function will change all data to say ‘By-pass’ and will allow any analysis conducted on it to be consistent.

Update name function.

```
# Make sure the fields order in the csvs matches the column order in the sql table schema
NODE_FIELDS = ['id', 'lat', 'lon', 'user', 'uid', 'version', 'changeset', 'timestamp']
NODE_TAGS_FIELDS = ['id', 'key', 'value', 'type']
WAY_FIELDS = ['id', 'user', 'uid', 'version', 'changeset', 'timestamp']
WAY_TAGS_FIELDS = ['id', 'key', 'value', 'type']
WAY_NODES_FIELDS = ['id', 'node_id', 'position']

def update_name(name, mapping):
    for st_type, ways in st_types.items():
            for name in ways:
                better_name = update_name(name, mapping)
                print (name, "=>", better_name)
                if name == "Bypass":
                    assert better_name == "By-pass"


def shape_element(element, node_attr_fields=NODE_FIELDS, way_attr_fields=WAY_FIELDS,
                  problem_chars=PROBLEMCHARS, default_tag_type='regular'):
    """Clean and shape node or way XML element to Python dict"""

    node_attribs = {}
```

Files and file sizes:
OSM Chester and Tarporley.xml - 90.9mb
Nodes.csv – 42.9mb
Nodes_tags.csv – 2mb
Ways.csv – 4.2mb
Ways_nodes.csv – 15.2mb
Ways_tags.csv – 7.3mb
Chester.sqlite – 77.4mb

Number of Nodes and number of ways:

```
sqlite> select count(*) from ways;
count(*)
----------
54669
sqlite> select count(*) from nodes;
count(*)
----------
403972
sqlite>
```

Number of distinct users:

```
sqlite> SELECT COUNT(distinct("b'uid'")) FROM (SELECT "b'uid'" FROM nodes UNION ALL SELECT "b'uid'" FROM ways);
656
sqlite>
```

656 distinct users

Top user:

```
sqlite> SELECT "b'user'", ROUND ((100.0*cnt/(SELECT COUNT(*) AS total FROM(SELECT "b'user'" FROM nodes U
NION ALL SELECT "b'user'" FROM ways))),2) FROM(SELECT "b'user'", COUNT(*) AS cnt FROM(SELECT "b'user'" F
ROM nodes UNION ALL SELECT "b'user'" FROM ways) GROUP BY "b'user'" ORDER BY cnt DESC LIMIT 10);
b'blackadder'|22.63
b'daviesp12'|15.66
b'James Derrick'|7.85
b'JustStupid'|6.55
b'RichardB'|5.48
b'mikh43'|4.21
b'Colin Smale'|3.75
b'Mauls'|3.19
b'Kyral210'|2.53
b'smb1001'|2.41
```

Top user is ‘Blackadder with 22.63% of the input/changes.

Most frequent amenity in nodes_tags:

```
sqlite> SELECT "b'value'",COUNT(*) AS cnt FROM nodes_tags WHERE "b'key'"="b'amenity'" GROUP BY "b'value'
" ORDER BY cnt DESC LIMIT 10;
b'post_box'|168
b'bench'|93
b'parking'|80
b'pub'|75
b'telephone'|36
b'atm'|33
b'cafe'|28
b'place_of_worship'|28
b'post_office'|24
b'bicycle_parking'|19
```

The area I chose is quite rural and this could be why post box is more frequent than ATM’s.

Most frequent amenity in ways_tags:

```
sqlite> SELECT sql FROM sqlite_master WHERE type = 'table' AND tbl_name = 'ways_tags';
CREATE TABLE "ways_tags" (
"b'id'" TEXT,
  "b'key'" TEXT,
  "b'value'" TEXT,
  "b'type'" TEXT
)
sqlite> SELECT "b'value'",COUNT(*) AS cnt FROM ways_tags WHERE "b'key'"="b'amenity'" GROUP BY "b'value'"
 ORDER BY cnt DESC LIMIT 10;
b'parking'|762
b'school'|94
b'pub'|83
b'place_of_worship'|72
b'fast_food'|51
b'grave_yard'|36
b'cafe'|28
b'bar'|25
b'restaurant'|24
b'community_centre'|20
sqlite>
```

Most popular religion:

Having grown up in this area I don't remember any other religious building other than Christian. I will use the SQL query to see if there is any other religion in the section of the map.

```
sqlite> SELECT "b'value'",COUNT(*) AS cnt FROM ways_tags WHERE "b'key'"="b'religion'" GROUP BY "b'value'
" ORDER BY cnt DESC LIMIT 10;
b'christian'|83
b'buddhist'|1
b'muslim'|1
sqlite>
```

This query shows there is a Buddhist and Muslim place of worship in the selected area. This is information I was not aware of growing up in this area.

Ideas to improve data:

I have through that one way to improve the data being input into OSM would be to have a form of conditionally formatting when entering in the data. This would then alert the user that the data they have entered in not consistent with the current data standard. This would be quite complex as the formatting upon data entry would need to be specific to local areas. An example would be telephone numbers. Each country has a different format to telephone numbers (styles, number grouping etc) and this would need to be understood whilst making the formatting.

Conclusion:

OSM is a very powerful and useful tool but it is reliant upon user generated information. As such there are differences/errors in the data.

If possible I would get OSM to run similar scripts like the ones in this project all over the world and for each country come up with a standard naming (eg Bypass , By-pass) and have these included in the OSM documentation. Following these standards you could run large scale scripts to go and "tidy up" the user generated information. These scripts would be very very large and would need to be "batched up" some how.