OpenStreetMap Data Case Study: Pittsburgh, PA, USA

By Michaela Wonacott

Map Area

Pittsburgh , PA, United States

- https://www.openstreetmap.org/relation/188553 (https://www.openstreetmap.org/relation/188553)

The map area I chose the city Pittsburgh in Pennsylvania, USA. I am intersted in this area since I have always wanted to visit the Phipps Conservatory and Botanical Gardens

Problems Encountered in the Map

Once I created a sample size of the Pittsburgh area, the problems I encountered are:

- Overabbreviated street names
  - Examples: "Centre Ave", "Lincoln Avenue"
- Inconsistent formatting for postcodes
  - Examples: "152036122", "15232-2106", "15017"

Overabbreviated Street Names

To fix the overabbreviated street names, I decided to create a function that would replace the abbreviated street names with the full spelling of the steet name.

In [ ]:
```
# Clean street names
def update_street_name(name):
    m = street_type_re.search(name)

    if m:
        street_type = m.group()
        if street_type not in expected_street:
            if street_type in mapping:
                name = name.replace(street_type, mapping[street_type])

    return name
```

This updated all the street name strings that had overabbreviated street names.

- Example: "Centre Ave" becomes "Centre Avenue"

Inconsistent Formatting for Postcodes

To fix the inconsisten formating for postcodes, I decided to splice the postcode string so only the first 5 elments would be kept. This would remove the extra numbers or dashes not needed for the postcodes.

In [ ]:
```
# Clean postcodes
def update_postcodes(postcode):
    m = postcode_type_re.search(postcode)

    if m:
        postcode = m.group()
        if postcode not in expected_postcode:
            if len(postcode) > 5:
                postcode = postcode[0:5]
            else:
                postcode = postcode
    return postcode
```

This updated all the postcodes strings that have a length greater than 5 to be sliced.

- Example:'15232-2106' becomes '15232'

Overview of the Data

This section contains basic statistics about the dataset, the SQLlite queries used to gather them, and additional ideas about the data in context. I used the pittsburg_sample.osm to generate my CSVs.

File Sizes

- pittsburgh.osm: 472 MB
- pittsburgh_sample.osm: 6.09 MB
- pittsburgh.db: 2.75 MB
- nodes.csv: 2.29 MB
- nodes._tags.csv: 45.1 KB
- ways.csv: 284 KB
- ways_tags.csv: 337 KB
- ways_nodes.csv: 796 KB

Number of Nodes

In [ ]:
```
sqlite> SELECT COUNT(*) FROM nodes;
```

54967

Number of Ways

In [ ]:
```
sqlite> SELECT COUNT(*) FROM ways;
```

9133

Number of Unique Users

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

649

Number of Churches

In [ ]:
```
sqlite> SELECT value, count(*) as num
FROM ways_tags
WHERE value = 'church'
ORDER BY num DESC;
```

1

Additional Statistics

Top 5 Popular Cusines

In [ ]:
```
sqlite> SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
    JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='restaurant') i
    ON nodes_tags.id=i.id
WHERE nodes_tags.key='cuisine'
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 5;
```

- greek 1
- american 1

Most Popular Religion

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 5;
```

christian 7

Ideas about the Dataset

Additional Improvements

While verifying postcodes for Pittsburg, I noticed that some of the postcodes were not for Pittsburg. They were for neighboring areas. This includes, but not limited to, postcodes: 15017, 15031, 15034, 15108, 15112, 15136, 15139, 15143, 15145, 15146, 15321.

To resolve this issue, I would create a list of verified postcodes for Pittsburg that can be selected from a drop-down list instead of manual entry. This would prevent incorrect postcodes for an area to be added. Removing existing incorrect postcodes would mean comparing existing postcodes with the verified postcodes list and updating them.

Anticipated Problems

The biggest problem with this solution is that it required a large change in how postcode data is entered. Currently, adding or updating a postcode is manual. You type out the postcode you want to enter with limited restiction on what can be entered. Instead of this, it would require chaning the input to accept only from a list of options or limit the input to 5 integers with the option of extensions in a seperate input box.

For the first option to change, there is an additional issue. Verifying which postcode are for a specific area would require a resouce that has accurate, conintuely up-to-date listings of verified postcodes. Finding such a resource and using it to create drop-down options is tedious and may require a level of coding greater than what is capable of from users.

This only covers future additions to OpenStreetMap input. Exisitng postcode data would need an overhaul that could be done with the creation of an additional program that updates each postcode syntax. This would requied even more resources to create that could impact the entire input system if not done properly. Manual change of each postcode to the new format would also produce a tedious and long process due to how much data there is OpenStreetMap.

Conclusion

Even with my audit and cleaning, the data for the Pittsburgh area is still not cleaned completely. This project accomplished a cleaning of only two problem areas. There are possibly many more errors that require a meticulous evaluation of every tag in OpenStreetMap. Human input is the biggest culprit to data errors. A misplead street name or incorrect postcode can happen now and then for a user. Multiplying that by the number of users who use OpenStreetMap makes those tiny errors way more impactful to OpenStreetMap as a whole. Having a program that could run over the entire of OpenStreetMap data to correct these common errors would eliminate quite a few of these issues. Introducing more limitations to inputs for each element or tag could also help decrease the number of input errors.