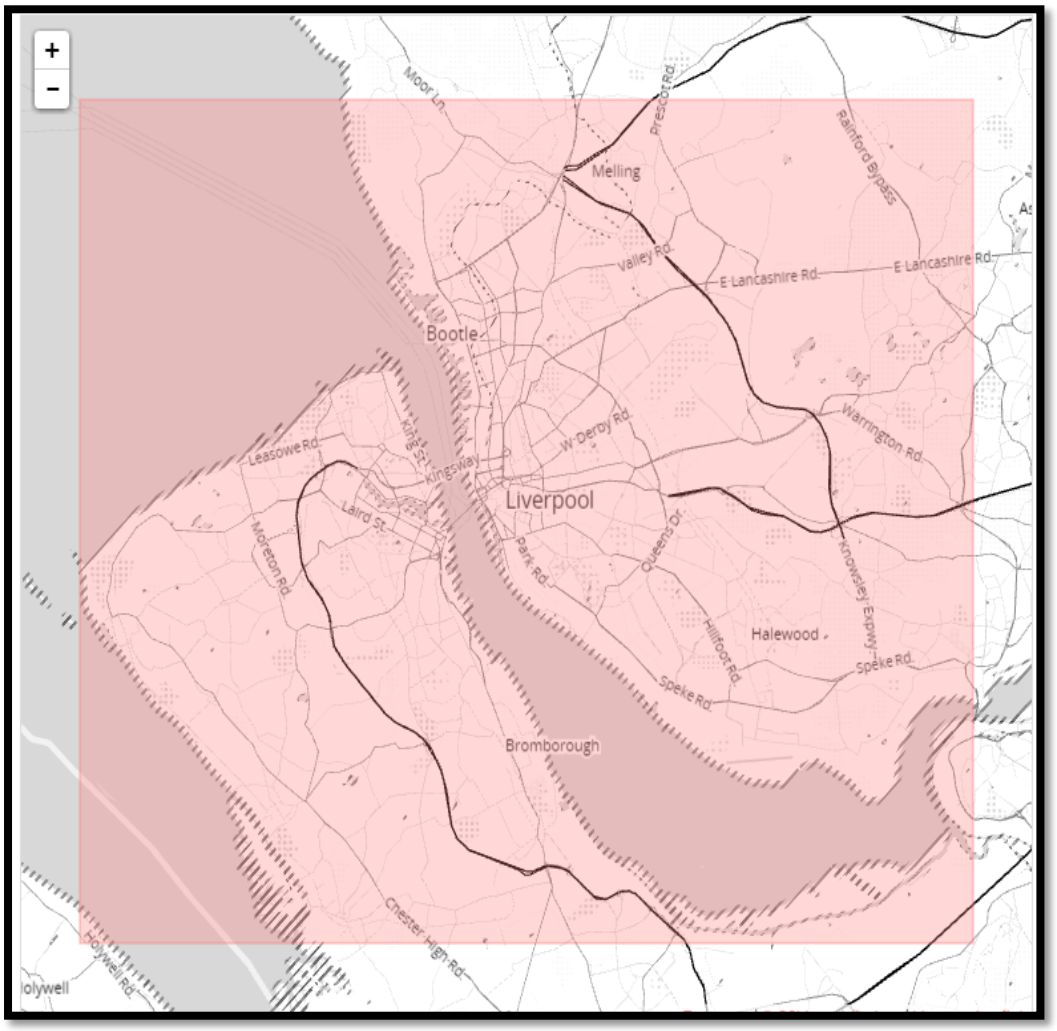
OpenStreetMap Data Wrangling with SQL

Husnuye Yasar

Map Area: Liverpool/England:
https://www.openstreetmap.org/relation/172987
https://mapzen.com/data/metro-extracts/metro/liverpool_england/



Project Overview

Using data munging techniquesto clean the OpenStreetMap data for a part of the world in**https://www.openstreetmap.org**.

Objectives:

- Assess the quality of the data for validity, accuracy, completeness, consistency and uniformity.
- Parse and gather data from popular file formats such as .csv,and .xml
- Process data from multiple files or very large files that can be cleaned programmatically.

- Learn how to store, query, and aggregate data using SQL.

Reason for selection:

- Liverpool / England: I want to see Liverpool and I wonder where and I chose it because it was ideal for file size, I think I will learn a lot of information about Liverpool

Auditing data

I audit street names, postal codes, and amenities as I suspected that these values could be problematic based on what I learned from the Udacity data wrangling module. I found errors in three of the four categories (street names, postal codes).

Problems:

Street types:

In **cleaning.py**, I print out uncleaned street names by using "pprint" module so that I can refer to the list to fix the mapping variables.E.g. mapping = {"Blvd": "Boulevard","St.": "Street", }.

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

After changing the variables, the update_name function correctsthe problematic street names to their respective mappings when it runs and the part of outcome is as follows.

```
street_mapping = {"151": "",
        "15th": "15th Street",
        "2": "",
        "Avenue,":"Avenue",
        "Avenue,Moreton":"Avenue",
        "203": "",
        "302": "",
        "3500": "",
        "3658": "",
        "4": "",
        "404": "",
        "502": "",
        "AVE": "Avenue",
        "Airport": "San Francisco International Airport",
        "Alameda": "Alameda Street",
        "Alto": "Alto Route",
        "Ave": "Avenue",
        "Ave. ": "Avenue",
        "Blvd": "Boulevard",
        "Blvd, ": "Boulevard",
        "Blvd.": "Boulevard",
```

"California": "California Street",
"Cres": "Crescent",
"Ctr": "Center",
"Dr": "Drive",
"Hwy": "Highway",
"Garden": "Gardens",
"Ln.": "Lane",
"North": "",
"Rd": "Road",
"road": "Road",
"road,": "Road",
"St": "Street",
"St.": "Street",
"broadway": "Broadway",
"square": "Square",
"st": "Street",
"street":"Street",
"Strand":"Street",
"VIllage":"Village",
"way":"Way",
}

Postal codes:

In **cleaning.py** , I use “regular expression”to get 5 valid numbers in each postal code to make them consistent.

```
def update_code(post_code):
    if(post_code[0] != '9' and post_code[0] != 'C'):
        post_code = 'Invalid Zip Code'
    elif(post_code == 'CA'):
        post_code = 'Invalid Zip Code'
    elif(len(post_code) > 5): #truncate zip codes that have secondary codes with a (-)
        if(post_code[:3] == 'CA ' or post_code[:3] == 'CA:'): # 'CA 94080' case
            post_code = post_code[3:]
        elif(post_code[:3] == 'CA9'):
            post_code = post_code[2:]
    return post_code
```

Cleaned Data Exploration (using SQL)

After cleaning the data and writing it into .csv files, I loaded the newly formed .csv files into a datebase named “husnuye.db”. I initially counted the number of nodes and ways, the basic units of this data set, to get a feel for how large the dataset was.

Data Analysis

Data Overview

Below is a basic statistical overview of the dataset:

liverpool.osm	**300 MB**
sample.osm	**3,02 MB**
husnuye.db	**1.68 MB**
ways_nodes.csv	**458 KB**
ways.csv	**159 KB**
nodes.csv	**1,07 KB**
ways_tags.csv	**190 KB**
node_tags.csv	**49,5 KB**

- Number of nodes query result
  - **SELECT COUNT(*) FROM nodes**
    - 13549
- Number of ways query result
  - **SELECT COUNT(*) FROM ways**
    - 2704
- Number of uniq users uid
  - **SELECT COUNT(DISTINCT(uid)) FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways)**
    - 296
- Top 10 contributing users
  - **SELECT user, COUNT(*) as num FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) GROUP BY user ORDER BY num DESC Limit 10**

    - [('daviesp12', 10546), ('khbritish', 920), ('jrdx', 510), ('UniEagle', 337), ('Dyserth', 279), ('duxxa', 204), ('F1rst_Timer', 197), ('xj25vm', 190), ('thewilk', 188), ('John Embrey', 183)
- Top 10 amenitites:
  - **SELECT value, COUNT(*) as num FROM nodes_tags WHERE key = "amenity" GROUP BY VALUE ORDER BY num DESC LIMIT 10**
    - [('post_box', 7),('restaurant', 6),('pub', 5),('fast_food', 3),('place_of_worship', 2), ('atm', 1), ('bank', 1), ('bench', 1), ('bicycle_parking', 1), ('cafe', 1)]
- Number of worship place
  - **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**
    - [('christian', 1)]

Conclusions

- **Final thought**:Even though many inconsistency and inaccuracy was seen in the dataset from the auditing process, there were not much problematic characters or language problems, which saved theresearcher a lot of work.
- **Limitations:** As long as OpenStreetMap allows volunteers to edit objects such as restaurants and schools, human input errors are inevitable.Furthermore, manual data cleaning process was harder for ambiguity.
- **Suggestions:** To bring more volunteers and encourage them to input clean data, there are several approaches I would recommend
    1. **Data auditing tools**:Those tools would be a great help to increase the level of data quality. They should carry out consistency, completeness, format, spelling and grammar check when users type in the map contents.
    2. Action: Validation check methods need following actions.
        - **Advisory action**: It should indicate participants that there is a validation issue with their input and show where the problemis detected.
        - **Verification action**: It parses every data andrestrictsin valid input such as foreign language or special characters. It can show recommended options to users when they are entering data e.g. frequently written word.
    3. **Gamification**: It is to make the website look more interesting and fun.Entertaining activities are involved.The detailed examples are ranking system, community context, evaluation and rewardsystems.Theycan motivate and challenge high performers to add more new data and correct the existing information.Visualizing their contribution is essential in this case.
    4. **Behavioral approaches**: OpenStreetMap should be well-organized to increase the chance to see the map for volunteers. There should be no barriers to their contents input.It can also send them notifications of where should be updated or if it has found a street signon the area they seem interested.
- **Benefits:**Suggestions mentioned above can lead to number of users increase and higher satisfaction among people who got rewarded. Furthermore, the higher data quality OpenStreetMap achieves, the more likely it will appeal to potential investors.
- **Anticipated issues:**First, it would cost much to implement gamification. Second is that keen competition may cause an accuracy problem because a few contributors might input fake data. In this case, cross referencing validation will be needed. Third, volunteer retention will be tough if validation check is too strict. Lastly, if the company focuses too much on gamification, the purpose of the project will become less important and faded.Volunteers will forget what they were willing to do.