Case Study: Open Street Map Data of the West Seattle Area-WA State

Author: Huong (Ivy) Nguyen

Introduction:

The Open Street Map (OSM: openstreetmap.org) is a collaborative community-built and open-source project that helps creating free editable map of different locations around the world. The data collected from openstreetmap.org is mainly about roads, trails, cafe, railway stations, and much more with the emphasis on local knowledge. Moreover, the data can be shared under the xml format, which is designed to make it easy for humans to read, and can be parsed by using different xml-parsers such as those available in Python. However, the advantage of human readability also implies that xml files take a lot of time to parsee due to its size, which will make a big challenge for analyzing data obtained from OSM page.

For this project, I particularly chose the osm data file for the West Seattle area of Washington state. The analysis will include downloading the datafile, identiying problems, fixing the identified problems, processing the data using SQL database, and finally analyzing the obtained dataset. The following report will focus on these main topics:

1. Problems Encountered in The Map
2. Data Auditting
3. Data Overview
4. Additional Idea

Map Area Resources:

- https://www.openstreetmap.org/export#map=12/47.5711/-122.3868 (https://www.openstreetmap.org/export#map=12/47.5711/-122.3868)
- https://mapzen.com/data/metro-extracts/your-extracts/1669d3d1d300 (https://mapzen.com/data/metro-extracts/your-extracts/1669d3d1d300)

Why Seattle?

Have been living in Seattle for several years, I always appreciate the nature outside and the city lifestyle that Seattle offers. In this project, I chose the West Seattle area dataset as a starting point for my data wrangling learning process. I love West Seattle with the view of nature as well as many different type of ethnic foods, breweries, and activities that it offers.

1. Problems Encountered in the Map

This data analysis was started by using the search tool available on openstreetmap.org to locate the geographical boundaries for the West Seattle area of Washington State. The next step is to export and navigate to MapZen.com to download and extract the desired xml osm file of this area. The file is named 'west_seattle_wa.osm' with the uncompressed size of 168MB. The issues that were encountered include accuracy, completeness and consistency.

Here are a few main issues that were identified for fixing during the auditting process (part 2):

- Inconsistent street name format with the main issue in format abbreviation
  - For example: S Brandon St instead of South Brandon Street
- Inconsistent telephone number format
  - For example: the right-target phone format should be xxx-xxx-xxxx; however, some phone numbers were listed wit different formats such as in the cases of +1 206-257-1178 and (206) 624-2598.
  - In this study, all phone numbers that do not have 10 digits will be marked as NULL.
- Inconsistent format in postcode
  - For example: 98106-1499 instead of 98106.

2. Data Auditting

Auditting is the process that will help us identify inconsistency within the data. For this project, I only focused on auditting three of the following data type: streep type, telephone format, and postcode.

Auditting Street Types:

The audit step for streep types revealed that there are several inconsistency of how they are presented in the dataset. Therefore, it is necessary to update all streep types for data's uniformity. To start this part, I first parsed all streep type from name available from the osm file, and if the street type is not in the expected list of street name, it would be keept as key in a dictionary with the correponding name that it was originally found from. To find out where the name to be checked is located, I generated the is_street_name function, which would help me find where street type is located in which tag. The audtting process was then taken place through using the audit_street function. Finally, all the names were updated using the update_name and fixed_street_type functions. Here are some examples of the fixes that were made for streep types:

- 63rd Avenue Southwest => 63rd Avenue Southwest
- Renton Avenue S => Renton Avenue South

Auditting Telephone Format:

In the process of auditting telephone format, I first wrote the validNumber function to identify which phone number has the right format, which is xxx-xxx-xxxx. Then, I seperated the phone numbers into two list: the wrong_format list and the right_format list. Moreover, I located where the phone number value is located, I used the is_phone function. The audit_phone function would take place to seperate the phones into the two lists as mentioned. All wrong-format phone numbers would be updated to right format by using the update_phone_number function. Here are some examples of auditting wrong-format telephone to the right format:

- '+1 206 327-9932' => '206-327-9932'
- '2067741234' => '206-774-1234'

Auditting Postcode Format:

To audit the postcode format, I first checked where the postcode value is located through is_postcode function, then checked if that value has the right format using the audit_postcode_type and audit_postcode functions. The wrong-format postcodes would then be fixed using the fix_postcodes function. Here are some examples of fixing postcodes with wrong format to the right format:

- 98106-1499 => 98106
- 98109-5210 => 98109

3. Data Overview

Counting number of tags:

I parsed through the obtained file and counted the number of unique elements using the count_tags function. The result that I obtained is:

- way: 88553
- member: 11499
- osm: 1
- relation: 1140
- tag: 632956
- bounds: 1
- nd: 821032
- node: 746188

Counting number of 'k' value for different regex:

The next task is to check the 'k' value for each tag using the key_type and process_map functions. I conducted this checking process by comparing the 'k' value again three differetn regular expression. Specifically, the **'lower'** regular expression is for tags that contain only lowercase letters and are valid, the **'lower_colon'** is for other valid tags but with a colon in the 'k' value, and the **'problemchars'** is for tags that have problematic characters. The final result is obtained in a dictionary that has the regex as key and the count for each regex as value. Moreover, the dictionary will also have an additional key called **'other'** which include all other tags that did not satisfy any of the three regex.

The result that I obtained from running the above two functions is:

- lower: 285830
- lower_colon: 338470
- problemchars: 0
- others: 8665

Counting the number of contributors:

Next, I go on counting the number of contributors (represented as 'uid' value) using the count_uid function. The result that I obtained is 612 contributors.

Datasize:

The following csv files were generated using the data.py file. Then, the west_seattle.db database was generated by using the building_data.py file. The dataset was explored and investigated using SQLite.

- west_seattle_wa.osm:
- nodes.csv:61M
- nodes_tags.csv: 8.0M
- ways.csv: 5.1M
- ways_nodes.csv: 19M
- ways_tags.csv: 15M

Counting number of nodes using SQL database:

sqlite> SELECT COUNT(*) FROM nodes;

- Output: 746188

Counting number of ways using SQL database:

sqlite> SELECT COUNT(*) FROM ways;

- Output: 88553

Counting number of restaurants in the West Seattle area:

sqlite> SELECT COUNT(*) FROM nodes_tags where nodes_tags.value = 'restaurant';

- Output: 590

Counting number of schools in the West Seattle area:

sqlite> SELECT COUNT(*) as num_school FROM nodes_tags where nodes_tags.value = 'school';

- Output: 43

Counting the number of religion practice in the West Seattle area:

sqlite> SELECT nodes_tags.value, COUNT(*) AS num FROM nodes_tags INNER JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='place_of_worship') place ON nodes_tags.id=place.id WHERE nodes_tags.key='religion' GROUP BY nodes_tags.value ORDER BY num DESC;

- Output:

christian|84 buddhist|6 scientologist|1

4. Additional Idea

For this project, the data auditing and cleaning process was conducted before writing the data into csv files. In order to improve the quality of the data, I think there should be a specific set of rules that will stardardize how the data can be input if desired. For example, if the official right format for phone number is xxx-xxx-xxxx for the U.S., openstreetmap.org can make it impossible for contributors to enter phone number with a different format. This way, extracting and analyzing the data will be easier. And thus, the data can be used for other applications such as navigating where most of businesses are located based on zipcode. Let say: in particular zip code area, there is a lot of tech companies but not that many restaurants open, then business people can use the data from the osm webpage to locate that particular area to open their business and avoid competition.

Even though the given suggestion could easily be implemented, it could also potentially reduce the willingness to contribute to the database of OSM from many users. Moreover, some countries do not have a specific set of rules in terms of how postcode or telephone should be formatted. Therefore, it could make it difficult for OSM to set a specific set of formatting requirements for certain location.