OpenStreetMap - Data Wrangling

WGU | Data Wrangling
Udacity | Project: OpenStreetMap Data
Abbreviated Project Document

> The full project document with code blocks, output, and tables can be found here

Purpose

This project was created for Udacity's Data Analyst Nanodegree. An extract of xml data was downloaded for a selected city or region from OpenStreetMap (OSM). This document details the auditing, cleaning, transformation, and analysis performed on that raw dataset. After the raw data was cleaned and staged in a tabular format (csv), it was loaded into a database for additional analysis.

Selecting a Dataset

For this project I decided to work with data from Austin, TX. The selected map area is too large to export directly from OpenStreetMap,[1] but I found a suitable extract hosted by Interline.[2] This particular extract was a pbf file, so I had to convert it to osm file format before auditing the data. I used a command line tool called osmosis to make this conversion.[3]

Auditing

To begin the auditing process, I created three summaries; one each for elements, attributes, and keys. I created four functions to accomplish this: print_sorted_dict, count_elements, count_attributes, and count_keys.[4] Using these functions, I printed a few summaries.

The basic components of the OpenStreetMap data model are tags, and the most important to this project are:

- node - describes points in space
- way - describes area boundaries and features
- relation - describe how other elements work together
- tag - describes the element to which they are attached, and contains two attributes: key (k) and value (v)

Exploring Key Values

Next, I checked the top 10 keys - based on frequency of occurrence - to see where there are opportunities for data cleaning. I also checked a few others that look interesting, and I created a function to facilitate this portion of the audit, key_val_counter.[4] Because I was working in a python notebook, I couldn't just loop through the keys I was investigating. The printed data would get truncated well before all the keys' values were displayed. Instead, I decided to run each key in its own cell. Most of the keys' values for Austin, TX OpenStreetMap data were already very clean. I suspect there are other students and hobbyists who have completed similar projects.

Problem Tags

Although most of the top key-values are clean, there are a few with opportunities for cleaning or filtering. I'll outline how these tags were cleaned/filtered in the next section.

building

There are a few things that need to be cleaned-up in the values for the building key.

- There are spaces where there should be underscores. A simple string replace will correct those.
- A few other entries are incorrect or ambiguous; I'll correct those with a dictionary replace.

postcode

These data are mostly clean, but there are some post codes included that are not actually in Austin.[5] I'll filter those out while cleaning and staging the data.

surface

The values for the surface key need some cleaning. For some of them, I can figure out what the user intended - I can clean those with a dictionary. Some other values are less clear, and I'll remove those tags with a list.

city

Values for the addr:city tag are a little messy. To fix them, I'll capitalize just the first
letter of each word in each city name. A dictionary match should clean up the remainder.

state

There are a few non-Texas values in this key that need to be filtered out while cleaning and staging the data.

Other Considerations

I have a few additional cleaning steps to integrate into the data preparation function. There are also values for addr:postcode, addr:state, and surface that will be used to remove problematic elements. In addition to this, there are a set of characters that will cause problems when staging this data - any elements with these characters will be removed as well.

Cleaning & Transforming

To prepare the data for my database I need to clean and filter the raw OpenStreetMap data. Then, I need to transform the data from xml format to a tabular format (csv).

Cleaning
First, I wrote a function set for each of the problematic keys I outlined above.

building
For this key, I created a dictionary to correct a few bad values. The clean_building[6] function compares the input value to that dictionary; if the value is contained in the dictionary keys, it's replaced with the dictionary value. Next, the value is checked for spaces, any that are located are replaced with an underscore.

postcode
I created two functions for the addr:postcode key:

- The clean_postcode[7] function first takes the input value, splits on semicolon, and drops anything after the semicolon.
  - '12345; 98765' → '12345'
- Next, it drops the last four digits from any values that have the full 9 digit zip code
  - 12345-6789 → 12345
- Then, the filter_postcode[7] function checks a list of valid Austin, TX zip codes.[8] It returns *False* if that zip code is present on the list (meaning it should not be removed), and *True* if that zip code is not present on the list (meaning it should be removed).

surface
I created two functions for the surface key:

- For the clean_surface[9] function, I created a dictionary to correct a few bad values. Then, the input value is compared to that dictionary; if the value is contained in the dictionary keys, it is replaced with the dictionary value.
- The filter_surface[9] function checks a list of values to remove. It returns True if the input value is on that list (meaning it should be removed), and False if the value is not on that list (meaning it should not be removed).

city
The city key required the most cleaning among those I selected, and the function I created, clean_city,[10] has multiple steps:

1. First, the function splits any city names that have multiple words.
   - round rock → [round, rock]
2. Then, it capitalizes each of those words by looping through each item in the list created when splitting the value.
   - [round, rock] → [Round, Rock]
3. Next, it puts the city names back together, and retains a space between each word.
   - [Round, Rock] → Round Rock
4. Finally, the value is compared to a dictionary to clean up any lingering incorrect city names.

state
Creating a function just to filter for addr:state == TX would have been a textbook example of over-engineering a problem. Instead of creating a function, I just added that filter to the shape function outlined

below.

Transforming
After the cleaning and filtering functions were developed, I wrote a function, shape_element,[11] that shapes the node and way elements of the raw xml file, and returns them as a Python dictionary. Employing that function, the cleaning functions outlined above, and several helper functions provided by Udacity for this project;[11] I cleaned, filtered, transformed, and staged the data into csv format to prepare it to be loaded into a sqlite database.

Problems Encountered
I encountered several problems while working with the Austin OpenStreetMap data. Chief among them was file size. I did not anticipate how resource intensive working with a dataset of this size would be. If I were to do this project again I would select a smaller map to work with. Some of the files used are quite large.[12] To work through this problem, I created a sample file that only contains elements with ways tags for addr:state=TX. This step significantly sped up testing and development, reducing the working file size from 1.79 GB to 42 MB. I did this with the java-based osmosis tool used earlier in this document.[3] I also had a little trouble finding data to clean. Many of the keys' values were already very clean. I suspect that since Austin is a tech city other students and hobbyists like myself have done similar projects and cleaned the OpenStreetMap data for this metropolitan area.

Sqlite Database

After the data was cleaned, I created a sqlite database.[13] Then, I created a schema in that database to match the schema of my csv files.[14] After that, I loaded the data into their respective tables.

Overview of the Data

Sqlite Database
After the data was cleaned, I created a sqlite database.[13] Then, I created a schema in that database to match the schema of my csv files.[14] Next, I loaded the data into their respective tables.

SQL Stats
Before digging into the dataset, I took a look at some database stats to get an idea of how much data I'd be working with. To generate those stats, I used another command-line utility program called sqlite3_analyzer.[15] Then, I loaded the stats into a table in my database.[16] After the stats table was created, I wrote a simple query to check table sizes.[17]

Analysis
Now that the stats were collected I could start analyzing the clean data. As I was analyzing this dataset, I wrote several queries[18] to answer some investigative questions:

How many people have contributed to the Austin OSM dataset?

	contributors
1	2,973

Which years had the most contributions?

	year	contributions
1	2015	5,958,603
2	2021	754,611
3	2019	581,022

What are the average monthly and yearly contributions?

	avg_yearly	avg_monthly
1	586,036	53,929

How many total ways tags are in this dataset, and what are the most common tags?

	key	count_ways
-	Total	858,496
1	building	620,762
2	height	438,569
3	addr:street	261,635

How many total nodes tags are in this dataset, and what are the most common tags?

	key	count_nodes
-	Total	7,932,057
1	street	83,171
2	housenumber	83,135
3	postcode	62,267

What are the most common amenities in Austin, TX?

	value	count
1	restaurant	916

	value	count
2	bench	887
3	waste_basket	791

What kind of restaurants are popular in Austin?

	value	count_restaurants
1	sandwich	116
2	pizza	115
3	mexican	110

Other Ideas About the Dataset

I think one of the best things OpenStreetMap could do to improve their data would be to develop a robust, automated cleaning process. These scripts or bots could automatically make corrections of specific errors. It looks like there is a bot (woodpeck_fixbot) modifying elements in the Austin dataset, but the scope of that project must be relatively narrow because I still found simple corrections to make in the data.

The kind of bot I'm imagining is more extensive, one example would be Wall-E,[19] but it is currently only operational in Germany and Austria. Perhaps OSM is worried that an automated correction bot for a map as large as the United States could cause problems if it was making inaccurate 'corrections'. Those hurdles could be overcome through proper testing. Specifically, by testing features on a very small area, and slowly widening the bot's scope before unleashing it on the entire map.

Conclusion

These data went on a long journey before landing in a clean sqlite database.

- I converted pbf file to osm format using osmosis, a java-based command line tool.
- Then I investigated the raw data in a python notebook.
- Next, I cleaned, filtered, and transformed the data using a handful of python scripts and functions.
- Finally, I loaded the data into a sqlite database and analyzed it with SQL.

There is additional work that could be done on the OpenStreetMap data for Austin, TX. Also, data issues are likely to be a constant problem for OSM until someone implements a more widespread automated cleaning process.