# C750

April 25, 2021

## 1 Auditing and Cleaning Open Street Map using Python and Data Wrangling Techniques

I chose the City of West Linn, Oregon for this project. West Linn is my hometown where I've recided for the first two decades of my life. As such, I am very familiar with its streets and layouts. I've always been a fan of the open source community so I figured this would be a perfect way to contribute back.

Let's get started.

The map in question can be found here: https://overpass-api.de/api/map?bbox=-122.6977,45.2379,-122.4220,45.4584

The map file was downloaded using the Overpass API. The full file is approximately 400 MB in size. The contents of the full file is what we will be working with here. In my GitHub is a sample file that you can run yourself.

The first thing I wanted to do was get a grasp of the file that we're working with. How many node tags? Ways tags? And so on.

[1]:
```
import xml.etree.cElementTree as ET

osmfile = "wlinn"

def tag_count(filename):

    """
    Counts how many tags are within the XML file and returns it as a dictionary
    """

    tag_dictionary = {}
    for event, elem in ET.iterparse(filename):
        if elem.tag not in tag_dictionary:
            tag_dictionary[elem.tag] = 1
        else:
            tag_dictionary[elem.tag] += 1

    return tag_dictionary
```

```
print (tag_count(osmfile))
```

```
{'note': 1, 'meta': 1, 'bounds': 1, 'node': 1643082, 'tag': 1180922, 'nd':
1898598, 'way': 223390, 'member': 50651, 'relation': 1998, 'osm': 1}
```

Here is a summary of what our function outputs:

- 134,382 nodes
- 223,390 ways
- 1,998 relations
- 50,651 members
- 1,898 nds

That's quite a lot to work with. But it's definitely not impossible. Let's discover which values in the tags will give us issues. Not all characters can be imported easily into a database and having consistency will enhance readability. It will be much easier later on to have consistency in our data. We are going to be using regular expresssions (REGEX) in Python. This is what we want to identify in our OSM File:

- Values with only lower case letters
- Values with only upper case letters
- Values that have characters that you wouldn't expect to be in a map

Let's handle this in the function below

[2]:
```
import re

lowercase = re.compile(r'^([a-z]|_)*$')
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
uppercase = re.compile(r'^([A-Z]|_)*$')
upper_colon = re.compile(r'^([A-Z]|_)*:([a-z]|_)*$')
problem = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')

"""
    This funtion will create a dictionary telling us how many entries in the
    dataset contain all lower case for "k=" values, all uppercase, if the "k="
    value contains any problem characters, values with either all lower or all
    upper with at least 1 colon, or if there are any characters not
    convered in the REGEX
"""

def k_type(element, key):

        if element.tag == 'tag': #find only elements named tag

            if lowercase.search(element.attrib['k']): #finds the "k" value in␣
 ↪the tag
                key['lowercase'] +=1
            elif lower_colon.search(element.attrib['k']):
                key['lower_colon'] +=1
```

```
            elif uppercase.search(element.attrib['k']):
                key['uppercase'] +=1
            elif upper_colon.search(element.attrib['k']):
                key['upper_colon'] +=1
            elif problem.search(element.attrib['k']):
                key['problem'] += 1
            else:
                key['other'] += 1
        return key


"""This fuction will parse through an XML file (the OSM file) and will
    execute the above function to count the different types of k values
    that we have.
"""

def process_tag(filename):

    # sets the key variable with 0 in all indexes
    key = {"lowercase": 0, "lower_colon": 0, "uppercase": 0, "upper_colon": 0,␣
 ↪"problem": 0, "other": 0}

    for _, element in ET.iterparse(filename):
        key = k_type(element, key)

    return key

tag_dictionary = process_tag(osmfile)
print (tag_dictionary)
```

```
{'lowercase': 617224, 'lower_colon': 557043, 'uppercase': 965, 'upper_colon':
2876, 'problem': 0, 'other': 2814}
```

Thankfully we haven't assesed anything that I would consider a problem in this dataset. However, we do have 2876 tags that do fit the "uppercase" description. Let's make a function that takes the dataset as an input and have it output the tags' k value associated with it. We will see what corresponds with the "uppercase" that our above function has identified.

[3]:
```
def get_key_with_issues(filename):


    #takes filename and returns a list of identified issues

    issue_list = []
    for _, element in ET.iterparse(filename):
        if element.tag == 'tag':
            if uppercase.search(element.attrib['k']):
                issue_list.append(element.attrib['k'])
```

```
    return issue_list

list = get_key_with_issues(osmfile)
list[:10]
```

[3]:
```
['NHS', 'NHS', 'NHS', 'NHS', 'NHS', 'NHS', 'NHS', 'NHS', 'NHS', 'NHS']
```

I truncated the list with list[:10], but there are a lot more results. This is a lot of NHS. Upon some further research, the "NHS" value was put in by one overzealous user named Peter Dobratz in 2016. NHS in this case stands for "National Highway System." This is acceptable to have and doesn't necessarily make sense to spell out the acronym in all 2876 cases. It seems our script has caught an erroneous error. I will leave this data untouched.

### 1.0.1 Cleaning and Auditing the Data

We will analyze the street types that are in this dataset and try to get an angle on how we want to find issues that arise.

[4]:
```
from audit import *

audit(osmfile)
```

```
defaultdict(<class 'int'>, {'Court': 13102, 'Road': 17588, 'Street': 26807,
'Drive': 19373, 'Rd': 7, 'Way': 4571, 'Boulevard': 2795, 'Lane': 6922, 'Avenue':
24919, 'East': 42, 'Circle': 2026, 'Highway': 504, 'Place': 3002, 'West': 72,
'Loop': 1431, 'Terrace': 1362, 'Alley': 2, '213': 107, 'Cervantes': 53,
'Summit': 25, 'Circus': 30, '212': 120, '224': 48, 'Parkway': 367, '97266': 1,
'Ave': 2, 'North': 44, 'Landing': 9, 'Botticelli': 7, 'Touchstone': 55, 'Point':
15, 'South': 47, '99E': 41, 'Vista': 4, 'Wheatland': 4, 'Run': 21, 'Crest': 42,
'Pointe': 2, 'Trail': 170, 'Grotto': 4, 'Downs': 29, 'Polonius': 5, 'Falstaff':
12, 'Pimlico': 4, 'Wheatherstone': 2, 'Woods': 15, 'Hotspur': 12, 'Greco': 1,
'Curve': 11, 'Path': 13, 'Miami': 17, 'Northbound': 1, 'Southbound': 1,
'Spinosa': 20, 'Pericles': 6, 'Commons': 37, 'View': 27, 'Fieldcrest': 46,
'TRL': 2}) %s: %d
```

[4]:
```
defaultdict(int,
            {'Court': 26204,
             'Road': 35176,
             'Street': 53614,
             'Drive': 38746,
             'Rd': 14,
             'Way': 9142,
             'Boulevard': 5590,
             'Lane': 13844,
             'Avenue': 49838,
             'East': 84,
             'Circle': 4052,
             'Highway': 1008,
             'Place': 6004,
```

```
'West': 144,
'Loop': 2862,
'Terrace': 2724,
'Alley': 4,
'213': 214,
'Cervantes': 106,
'Summit': 50,
'Circus': 60,
'212': 240,
'224': 96,
'Parkway': 734,
'97266': 2,
'Ave': 4,
'North': 88,
'Landing': 18,
'Botticelli': 14,
'Touchstone': 110,
'Point': 30,
'South': 94,
'99E': 82,
'Vista': 8,
'Wheatland': 8,
'Run': 42,
'Crest': 84,
'Pointe': 4,
'Trail': 340,
'Grotto': 8,
'Downs': 58,
'Polonius': 10,
'Falstaff': 24,
'Pimlico': 8,
'Wheatherstone': 4,
'Woods': 30,
'Hotspur': 24,
'Greco': 2,
'Curve': 22,
'Path': 26,
'Miami': 34,
'Northbound': 2,
'Southbound': 2,
'Spinosa': 40,
'Pericles': 12,
'Commons': 74,
'View': 54,
'Fieldcrest': 92,
'TRL': 4})
```

There are quite a few issues here. Using the above output, we have a lot of work to do.

- Roads are erroneously labeled as “Rd” and “Ave” should be spelt out.
- 97266 is an area/zip code and should not be in the street name field.
- Pimlico is the name of a Drive in my hometown and should be appended with “Drive” to aid in consistency.
- The value “TRL” should be spelt out fully as “Trail” to aid in consistency as well.
- Boticelli is a street.
- 97266 is a ZIP code and should not be in this field. The location refers to a delicious Mexican restaurant. The v field in the tag element is “8202 SE Flavel St, Portland, OR 97266”. It should just be “SE Flavel Street”.
- Wheatland is a road.
- Falstaff is a road.
- Pimlico is a Drive.
- Hotspur is a road.
- Southwest Miami is a street.
- Pericles is a loop.
- Polonius is a loop.
- El Greco is a street.
- Wheatherstone is a street
- View, Commons, Run, South, North, Circus, Summit, Downs, West, View, and East are all acceptable values. I will add them to the expected_values dictionary.
- Cervantes is a street.
- Touchstone is a road.
- Polonius is a street.
- Spinosa is a road.
- Southeast Fieldcrest is a road.

Based on the above findings, I’ve created a dictionary that will map an incorrect value to a correct value and a list of abbreviations with their corrected value.

[5]:
```
from map_cleaning import *

print("Expected Values:")
print(expected_values)
print()
print("Abbreivation Mappings:")
print(abbr_mapping)
```

```
Expected Values:
['Avenue', 'Alley', 'Road', 'Street', 'Trail', 'Landing', 'Pointe', 'Vista',
'Woods', 'Curve', 'Path', 'Freeway', 'Grotto', 'Court', 'Northbound',
'Southbound', 'Drive', 'Boulevard', 'Lane', 'Circle', 'Highway', 'Place',
'Loop', 'Terrace', 'Way', 'Crest', 'Parkway', 'Point', 'View', 'Commons', 'Run',
'South', 'North', 'East', 'Circus', 'Summit', 'West', '99E', '224', '213',
'View', '212', 'Downs']

Abbreivation Mappings:
{'Ave': 'Avenue', 'TRL': 'Trail', 'Hwy': 'Highway', 'Rd': 'Road', 'Ct': 'Court',
'Dr': 'Drive', 'Pl': 'Place', 'place': 'Place', 'Pkwy': 'Parkway', 'rd.':
'Road', 'Sq.': 'Square', 'St': 'Street', 'st': 'Street', 'ST': 'Street', 'St,':
```

```
'Street', 'St.': 'Street', 'street': 'Street', 'Street.': 'Street'}
```

Since we have expected values for streets and abbreviations taken care of, let's fix specific streets that won't be fixed by the above two dictionaries. Here is the spelling_fix dictionary used to fix specific issues:

[6]:
```
print(spelling_fix)
```

```
{'Falstaff': 'Falstaff Road', 'Pimlico': 'Pimlico Drive', 'Hotspur': 'Hotspur
Road', 'Pericles': 'Pericles Loop', 'El Greco': 'El Greco Street', '8202 SE
Flavel St, Portland, OR 97266': 'SE Flavel Street', 'Cervantes': 'Cervantes
Street', 'Touchstone': 'Touchstone Road', 'Polonius': 'Polonius Street',
'Spinosa': 'Spinosa Road', 'Boticelli': 'Boticelli Street', 'Southwest
Wheatland': 'Southwest Wheatland Road', 'Southwest Miami': 'Southwest Miami
Street', 'Wheatherstone': 'Wheatherstone Street', 'Southeast Fieldcrest':
'Southeast Fieldcrest Road'}
```

### 1.0.2 Now We Will Clean The Data!

**(This is the best part!)** With the above out of the way, let's clean the map so that we can put our above dictionaries to use.

[7]:
```
from map_cleaning import *

clean_map(osmfile)
```

```
4th Ave: 4th Avenue
7273 SE 92nd Ave: 7273 SE 92nd Avenue
Cervantes: Cervantes Street
Falstaff: Falstaff Road
Hotspur: Hostspur Road
Pericles: Pericles Loop
Pimlico: Pimlico Drive
Polonius: Polonius Loop
S Carus Rd: S Carus Road
S Penman Rd: S Penman Road
SE Stevens Rd: SE Stevens Road
SE Sunnyside Rd: SE Sunnyside Road
Southeast Hittay TRL: Southeast Hittay Trail
Southwest Borland Rd: Southwest Borland Road
Spinosa: Spinosa Road
SW Boones Ferry Rd: SW Boones Ferry Road
Touchstone: Touchstone Road
```

The data has now been cleaned. Let's create our CSV files, create our database with tables, and import our CSV files into our newly created database.

[8]:
```
from create_csvs import *
from createdb import *
```

```
process_map(osmfile, validate=False)
create_db()
```

### 1.0.3 Executing Queries in the Database

Here I will analyze the data to find the answers to a few questions I had.

**Who are the top contributing users? And how many contributions?**

[9]:
```
#Creates connection to the databse
conn = sqlite3.connect(sqlite_file)
c = conn.cursor()

QUERY = '''
SELECT DISTINCT user, COUNT(*)
FROM nodes
GROUP BY nodes.uid
ORDER BY COUNT(*) DESC
LIMIT 10;
'''

c.execute(QUERY)
all_rows = c.fetchall()
print(all_rows)
```

```
[('Peter Dobratz_pdxbuildings', 615824), ('lyzidiamond_imports', 218977),
('Peter Dobratz', 101784), ('justin_pdxbuildings', 95643), ('Mele Sax-Barnett',
87759), ('Darrell_pdxbuildings', 86402), ('Grant Humphries', 73924), ('baradam',
41090), ('cowdog', 25071), ('tguen', 23761)]
```

**What's the most common way tag?**

[10]:
```
#Creates connection to the databse
conn = sqlite3.connect(sqlite_file)
c = conn.cursor()

QUERY = '''
SELECT key,count(*)
FROM ways_tags
GROUP BY 1
ORDER BY count(*) DESC
LIMIT 10;
'''

c.execute(QUERY)
all_rows = c.fetchall()
print(all_rows)
```

```
[('building', 164482), ('ele', 141702), ('street', 120624), ('housenumber',
120525), ('city', 120378), ('postcode', 120275), ('height', 72492), ('highway',
```

```
50043), ('levels', 27997), ('name', 24671)]
```

**And the most common node tag?**

[11]:
```
#Creates connection to the databse
conn = sqlite3.connect(sqlite_file)
c = conn.cursor()

QUERY = '''
SELECT key,count(*)
FROM nodes_tags
GROUP BY 1
ORDER BY count(*) DESC
LIMIT 10;
'''
c.execute(QUERY)
all_rows = c.fetchall()
print(all_rows)
```

```
[('highway', 9152), ('street', 5321), ('housenumber', 5318), ('city', 5265),
('postcode', 5263), ('name', 3462), ('barrier', 2663), ('ref', 1790),
('public_transport', 1687), ('amenity', 1537)]
```

**What about the most common ammenities?**

[12]:
```
#Creates connection to the databse
conn = sqlite3.connect(sqlite_file)
c = conn.cursor()

QUERY = '''
SELECT value, COUNT(*) as Count
FROM nodes_tags
WHERE key='amenity'
GROUP BY value
ORDER BY Count DESC
LIMIT 25;
'''

c.execute(QUERY)
all_rows = c.fetchall()
print(all_rows)
```

```
[('bench', 176), ('bicycle_parking', 137), ('place_of_worship', 104),
('restaurant', 79), ('parking_space', 78), ('waste_basket', 67), ('post_box',
66), ('fast_food', 62), ('cafe', 56), ('parking', 55), ('doctors', 50),
('pharmacy', 43), ('toilets', 36), ('school', 35), ('dentist', 35),
('drinking_water', 34), ('letter_box', 32), ('fuel', 32), ('bank', 28),
('social_facility', 25), ('atm', 23), ('vending_machine', 22),
('public_bookcase', 19), ('pub', 18), ('parking_entrance', 18)]
```

**I see that we have 32 "fuel" locations. Let's look deeper. How many of them offer diesel fuel?**

[13]:
```
#Creates connection to the databse
conn = sqlite3.connect(sqlite_file)
c = conn.cursor()

QUERY = '''
SELECT nodes_tags.value, COUNT(*) as Count
FROM nodes_tags
JOIN
    (SELECT DISTINCT(id)
    FROM nodes_tags
    WHERE value='fuel') as f
ON nodes_tags.id=f.id
WHERE nodes_tags.key='diesel'
GROUP BY nodes_tags.value
ORDER BY Count DESC;
'''

c.execute(QUERY)
all_rows = c.fetchall()
print(all_rows)
```

```
[('yes', 8)]
```

**Let's investigate the most common place of worship.**

[14]:
```
#Creates connection to the databse
conn = sqlite3.connect(sqlite_file)
c = conn.cursor()

QUERY = '''
SELECT nodes_tags.value, COUNT(*) as Count
FROM nodes_tags
JOIN
    (SELECT DISTINCT(id)
    FROM nodes_tags
    WHERE value='place_of_worship') as Sub
ON nodes_tags.id=Sub.id
WHERE nodes_tags.key='religion'
GROUP BY nodes_tags.value
ORDER BY Count DESC;
'''

c.execute(QUERY)
all_rows = c.fetchall()
print(all_rows)
```

```
[('christian', 97), ('buddhist', 3), ('unitarian_universalist', 1), ('jewish',
```

```
1)]
```

The area apperas overwhelmingly Christian.

# 2 Conclusion

### 2.0.1 Closing Remarks

At first, I focused on cleaning up minor issues. I have lived in the West Linn area for two decades of my life so I am happy to be able to make changes to a database for an area I love.

Some common issues in the database is a lack of conformity in naming conventions and data entry. My guess is that OSM keeps the requirements of entry loose in order to decrease the barriers to entry. This allows OSM to obtain a much larger database. If OSM were to tighten the requirements for their database, I have a feeling that the amount of entries would be much less.

OSM is in a sweet spot because they are maintained by a supportive community. Other schools throughout the United States have programs wheeree students cleanup the database as well. It's a win-win situation; the database grows almost unhindered and it gets cleaned in the pursuit of knowledge.

### 2.0.2 Benefits

In the future, I would like to expand the maps size and possible cleanup a metropolitan area. To do so, however, I would need to refine my modifications to ensure accuracy. Additionally, I would like to clean up business and increase uniformity of businesses. For example, making sure that phone numbers are in a consistent format or ensuring proper capitalization of business names. Finally, benefits from the above suggestion would include contributing to an opensource community and

### 2.0.3 Anticipated Problems

Some issues with the above suggestions would include non-standard business names. For example, businesses that are intentionally named with all upper or lower case letters. Knowledge of the specific businesses would be key to making sure that incorrect modifications aren't made. Also, finding and correcting phone numbers would be a challenge. The first step would be to find businesses with incorrect phone numbers. The next step would be to query another database with known good numbers. And finally, the third step would be to make modifications. This could be time consuming and potentially bad for the business if an incorrect number is inserted so care and caution should be exercised before making large updates.

### 2.0.4 References

https://hadrien-lcrx.github.io/notebooks/Boston_Data_Wrangling.html
https://www.w3schools.com/sql/sql_syntax.asp https://wiki.python.org/moin/BeginnersGuide
https://github.com/ian-whitestone/data-wrangling-openstreetmap