Wrangling OpenStreetMap Data

July 27, 2019

1 Open Street Map Data Wrangling

1.1 Selected Map Area

Columbus, Ohio

- Relation Link: https://www.openstreetmap.org/relation/182706
- Data Export Link: https://www.openstreetmap.org/export#map=13/39.9596/-83.0105

I have lived in the Columbus area for the past few years, and enjoy the diversity of the city. I wanted to see if there's any locations that I may have missed in my travels throughout the city. The data export link provided above returns a 102mb OSM file. A sample of the data looked at every 10th element, so reduce processing times while auditing, and cleaning the dataset.

1.2 Problems Encountered in the Map Data

1.2.1 Street Name Audit

Challenges I encountered while auditing the "Street Name" element include inconsistent usage of Drive, Road, Street and Avenue. I also wanted to make sure all street names were appropriately formatted like book titles. The script used to audit street addresses is titled "audit_street_types.py": - Avenue - Boulevard - Court - Drive - Lane - Place - Road - Street

After running the initial audit, I found that: - Drive had 2 abbreviated occurrences, "Dr" and "Dr." - Road had 1 abbreviated occurrence, "Rd". - Street had 1 abbreviated occurence, "St". - Avenue had 1 abbreviated occurence, "Ave".

The function used to clean the above issues is detailed below:

[1]:
```
def clean_street_name_values(name, mappings):
    """ clean_street_name_value corrects the name supplied with the correct␣
 ↪name from mapping

    :param name: a street name that will be cleaned
    :param mappings: a mapping of bad values to good values
    :return: corrected street name
    """
    new_street_name = name.title()
    for street_name in mappings:
        if street_name in new_street_name:
            # cleans the name of the element with the predefined mapping pairs
            new_street_name = re.sub(
                r"\b" + re.escape(street_name) + r"$", mappings[street_name],␣
↪new_street_name
            )
    return new_street_nam
```

This function updated any street name such as "N Cassady Ave" to "N Cassady Avenue".

1.2.2 Amenities Audit

I encountered similar challenges while auditing the amenities. After researching https://wiki.openstreetmap.org/wiki/Map_Features to get a better understanding of some of the classifications, I was able to determine an appropriate route to clean the information.
The scriped used to audit amenities is titled "audit_amenities.py". There were several amenities I already expected such as restaurants, schools, and places of worship for example. After running my initial audit I found that:

- I wanted to consolidate "University", "College", and "kindergarten" to "school", since they all relate to educating residents and their families.
- There were multiple "centres" that could be consolidated in the dataset such as exhibition_centre and conference_centre. They will changed to community centers
- I also covered "post_office" to "post_box" for the purposes of indicating where mail could be dropped off.

Function used to clean the unexpected values.

[2]:
```
def clean_amenities_name_values(name, mapping):
    """ clean_amenities_name_value corrects the name supplied with the correct␣
↪name from mapping

    :param name: a amenities name that will be cleaned
    :param mappings: a mapping of bad values to good values
    :return: corrected amenities name
    """
    for amenity in mapping:
        if amenity in name:
            # cleans the name of the element with the predefined mapping pairs
            name = re.sub(r"\b" + re.escape(amenity), mapping[amenity], name)
    return name
```

1.2.3 Postal Codes Audit

Finally, I audited Postal Codes. I've always found it interesting how cities are broken up by a 5 digit number and wanted to get a better understanding of my current city. The postal codes provided an interesting challenge, I had to reference http://www.mapszipcode.com/ohio/columbus to validate some of my findings during the audit, which used the script "audit_postal_codes.py". The reference site I used provided a expected values list during the audit. After completing the audit there were unexpected values that needed to be cleaned.

- There were 3 difference Postal Codes that included a 4 digit that I wanted to remove for consistency
- There was a 4 digit only zipcode, which was most likely incorrectly data entered
- There was a full address as well which I wanted to remove from the dataset
- There was also 2 zip codes that were incorrect for the Columbus area
  - 43328 does not appear to be a valid zip code for the United States
  - 43029 is located over 45 miles to the west of the map area I selected

Function used to clean the unexpected values.

[3]:
```
def clean_postal_name_values(name, mappings):
    """ clean_postal_name_value corrects the name supplied with the correct␣
↪name from mapping

    :param name: a postal name that will be cleaned
    :param mappings: a mapping of bad values to good values
    :return: corrected postal name
    """
    for postal_code in mappings:
        if postal_code in name:
            # check if length is greater than 5, to determine how to clean␣
↪value
            if len(name) >= 6:
                name = re.split("[:;-]", name)[0]
            name = re.sub(r"\b" + re.escape(postal_code),␣
↪mappings[postal_code], name)
    return name
```

1.3 Overview of the Data

[4]:
```
# imports needed to execute sql queries
import os
from pathlib import Path
import sqlite3
from pprint import pprint as pp
```

[5]:
```
# Create a connection to the sqlite3 database file
engine = sqlite3.connect('openstreetmap_xml.db')
```

[6]:
```
# Assign cursor object to conn variable
conn = engine.cursor()
```

1.4 Data Statistics

Name and File sizes of the osm, database, and csv files.

[7]:
```
current_dir = Path(".")
for path in current_dir.iterdir():
    if ".csv" in path.name or ".osm" in path.name or ".db" in path.name:
        print(f"{path}\t {round(path.stat().st_size/1024**2, 2)}mb")
```

```
columbus-10th.osm        10.42mb
columbus.osm     102.46mb
nodes.csv        37.06mb
nodes_tags.csv   1.97mb
openstreetmap_xml.db     54.94mb
osml_xml.db      54.94mb
osm_xml.db       54.94mb
ways.csv         3.7mb
ways_nodes.csv   13.28mb
ways_tags.csv    7.06mb
```

Database Table Summary

[8]:
```
# Generate a summary of the tables in the database
table_list = ['ways', 'nodes', 'ways_tags', 'nodes_tags', 'ways_nodes']
for table in table_list:
    query = (f"SELECT count(*) FROM {table}")
    conn.execute(query)
    print(f"{table.capitalize()} Table:\t{conn.fetchall()[0][0]} items")
```

```
Ways Table:     62001 items
Nodes Table:    447148 items
Ways_tags Table:        211094 items
Nodes_tags Table:       55393 items
Ways_nodes Table:       579426 items
```

Number of Nodes

[9]:
```
count_of_nodes = """SELECT COUNT(*) FROM nodes;"""
```

[10]:
```
conn.execute(count_of_nodes)
print(f"Number of Nodes: {conn.fetchone()[0]}")
```

```
Number of Nodes: 447148
```

Number of Ways

[11]:
```
count_of_ways = """SELECT COUNT(*) FROM ways;"""
```

[12]:
```
conn.execute(count_of_ways)
print(f"Number of Ways: {conn.fetchone()[0]}")
```

```
Number of Ways: 62001
```

Number of cities by count, descending by count

[13]:
```
inspecting_cities_sql = """
SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags UNION ALL
    SELECT * FROM ways_tags) tags
WHERE tags.key = 'city'
GROUP BY tags.value
ORDER BY count DESC;"""
```

[14]:
```
conn.execute(inspecting_cities_sql)
pp(conn.fetchall())
```

```
[('Columbus', 2193),
 ('Upper Arlington', 479),
 ('Bexley', 32),
 ('Grove City', 21),
 ('Grandview Heights', 10),
 ('Grandview', 9),
 ('columbus', 4),
 ('Columbus, OH', 2),
 ('German Village', 2),
 ('Hilliard', 2),
 ('Obetz', 2),
 ('57', 1),
 ('Delaware', 1),
 ('Gahanna', 1),
 ('Marble Cliff', 1),
 ('Whitehall', 1)]
```

We can see here that the tag element 'city' could use some cleaning. There's 3 separate versions of Columbus in the list, and strangely there's a number value (57) present.

Top 20 Streets by count, descending by count

[15]:
```
inspecting_street_types = """
SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags UNION ALL
    SELECT * FROM ways_tags) tags
WHERE tags.key = 'street'
GROUP BY tags.value
ORDER BY count DESC
LIMIT 20;"""
```

[16]:
```
conn.execute(inspecting_street_types)
pp(conn.fetchall())
```

```
[('North High Street', 257),
 ('South High Street', 243),
 ('East Whittier Street', 122),
 ('South Third Street', 84),
 ('South Front Street', 80),
 ('Parsons Avenue', 71),
 ('East Main Street', 59),
 ('Neil Avenue', 57),
 ('East Livingston Avenue', 56),
 ('East Gay Street', 41),
 ('Olentangy River Road', 39),
 ('Wellesley Drive', 39),
 ('East Mound Street', 38),
 ('West Lane Avenue', 36),
 ('East Russell Street', 35),
 ('Vassar Place', 34),
 ('College Hill Drive', 31),
 ('Mount Holyoke Road', 31),
 ('Park Street', 29),
 ('West Broad Street', 29)]
```

It looks like the cleaning function updated our values correctly.

Number of Postal Codes by count, descending by count with "Invalid" value not included

[17]:
```
inspecting_postal_codes = """
SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags UNION ALL
    SELECT * FROM ways_tags) tags
WHERE tags.key = 'postcode'
AND tags.value != 'Invalid'
GROUP BY tags.value
ORDER BY count DESC;"""
```

[18]:
```
conn.execute(inspecting_postal_codes)
conn.fetchall()
```

[18]:
```
[('43215', 1090),
 ('43221', 493),
 ('43206', 326),
 ('43210', 182),
 ('43201', 133),
 ('43202', 116),
 ('43212', 105),
 ('43207', 87),
 ('43209', 49),
 ('43205', 39),
 ('43214', 29),
 ('43219', 19),
 ('43123', 17),
 ('43204', 17),
 ('43211', 16),
 ('43228', 12),
 ('43224', 9),
 ('43227', 8),
 ('43222', 7),
 ('43203', 6),
 ('43213', 6),
 ('43220', 5),
 ('43223', 4),
 ('43026', 2),
 ('43230', 2),
 ('43232', 2),
 ('43216', 1)]
```

In the SQL query, I used "AND tags.value != 'Invalid'" to prevent incorrect values from appearing on the list which were labeled invalid by the cleaning function.

Number of Unique Users contributing to this area.

[19]:
```
unique_users = """
SELECT COUNT(DISTINCT(e.uid))
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e;"""
```

[20]:
```
conn.execute(unique_users)
print(f"Number of Unique Users: {conn.fetchone()[0]}")
```

```
Number of Unique Users: 672
```

Top 20 Contributors in the Area, descending

[21]:
```
top_20_contributors = """
SELECT e.user, COUNT(*) as num
FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
GROUP BY e.user
ORDER BY num DESC
LIMIT 20;"""
```

[22]:
```
conn.execute(top_20_contributors)
for x,y in conn.fetchall():
    print(f"{x, y}")
```

```
('doktorpixel14', 121483)
('MerlinPendragon', 102857)
('woodpeck_fixbot', 47033)
('kbzimmer', 42919)
('Anononononon', 17242)
('S_H', 16794)
('Vid the Kid', 16357)
('TimC', 15469)
('duck57', 10113)
('TimberPete', 8657)
```

```
('alarkin328', 7081)
('craig118', 7028)
('Johnny Mapperseed', 6659)
('morgankevinj', 6327)
('DangerRave', 5540)
('bot-mode', 2922)
('Minh Nguyen', 2896)
('ncmh91', 2783)
('Nate Wessel', 2341)
('_ChrisR', 2192)
```

Number of Contributors that only posted once

[23]:
```
one_time_user = """
SELECT COUNT(*)
FROM (SELECT e.user, COUNT(*) as num
FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
GROUP BY e.user
HAVING num=1) u;"""
```

[24]:
```
conn.execute(one_time_user)
print(f"One Time Contributors: {conn.fetchone()[0]}")
```

```
One Time Contributors: 118
```

Top 10 appearing amenities by count, descending

[25]:
```
top_10_amenity_type = """
SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key='amenity'
GROUP BY value
ORDER by num DESC
limit 10;"""
```

[26]:
```
conn.execute(top_10_amenity_type)
for x,y in conn.fetchall():
    print(f"{x, y}")
```

```
('place_of_worship', 347)
('bench', 240)
('waste_basket', 230)
('restaurant', 185)
('bicycle_parking', 144)
('fast_food', 95)
('school', 87)
('bicycle_rental', 72)
('cafe', 62)
('parking_entrance', 57)
```

Top 3 Places of Worship by count, descending

[27]:
```
place_of_worship_counts = """
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 3;"""
```

[28]:
```
conn.execute(place_of_worship_counts)
for x, y in conn.fetchall():
    print(f"{x, y}")
```

```
('christian', 324)
('muslim', 2)
('jewish', 1)
```

Top 10 Most Popular Cuisine Type

[29]:
```
popular_food_types = """
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 10;"""
```

[30]:
```
conn.execute(popular_food_types)
for x, y in conn.fetchall():
    print(f"{x, y}")
```

```
('pizza', 17)
('chinese', 9)
('american', 8)
('italian', 7)
('asian', 4)
('indian', 3)
('burger', 2)
('ice_cream', 2)
('middle_eastern', 2)
('peruvian', 2)
```

[31]:
```
# conn.close()
```

1.5 Other Ideas about the Dataset

One of the areas I think could use improvement with the dataset is the way amenities are classified. There are currently 244 documented values according to https://wiki.openstreetmap.org/wiki/Key:amenity . There's several values that I believe overlap, should still be amenities, or don't belong. For example, "nursing_home" and "social_facility" have almost the exact same description per the wiki. Gym's are no longer considered an amenity, but instead classified as leisure:fitness_centre. There's private toilet as well as toilet which don't have a clear description on why they're separate entities.
As we can see in the below query, there's separation between parking_entrace and parking which can cause data entry issues especially for new contributors. Wedding_chapel also appears in my dataset and I cannot find a reference to it on the wiki page previously listed. Upon further inspection, the amenity key is allowed to have user define values which can cause additional issues such as various spelling of the same item, or extremely limited usages of the value. For example, with Wedding_chapel it is used a grand total of 12 times globally with amenity as the key.

[32]:
```
amenity_types = """
SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key='amenity'
GROUP BY value
ORDER by num DESC;"""
```

[33]:
```
conn.execute(amenity_types)
pp(conn.fetchall(), compact=True)
```

```
[('place_of_worship', 347), ('bench', 240), ('waste_basket', 230),
 ('restaurant', 185), ('bicycle_parking', 144), ('fast_food', 95),
 ('school', 87), ('bicycle_rental', 72), ('cafe', 62), ('parking_entrance', 57),
 ('post_box', 56), ('fountain', 47), ('bar', 40), ('atm', 25),
 ('bicycle_repair_station', 22), ('bank', 19), ('fuel', 16),
 ('vending_machine', 16), ('pub', 15), ('parking', 13), ('toilets', 12),
 ('pharmacy', 11), ('ice_cream', 10), ('library', 10), ('theatre', 9),
 ('doctors', 7), ('loading_dock', 7), ('drinking_water', 6), ('recycling', 6),
 ('clinic', 4), ('dentist', 4), ('motorcycle_parking', 4), ('nightclub', 4),
 ('police', 4), ('veterinary', 4), ('arts_centre', 3), ('cinema', 3),
 ('community_center', 3), ('courthouse', 3), ('grave_yard', 3),
 ('events_venue', 2), ('marketplace', 2), ('shelter', 2), ('telephone', 2),
 ('animal_shelter', 1), ('biergarten', 1), ('car_wash', 1),
 ('charging_station', 1), ('compressed_air', 1), ('grit_bin', 1),
 ('hospital', 1), ('planetarium', 1), ('prison', 1), ('ranger_station', 1),
 ('social_facility', 1), ('studio', 1), ('swimming_pool', 1), ('townhall', 1),
 ('waste_disposal', 1), ('wedding_chapel', 1)]
```

To improve upon this specific entity I believe stricter guidelines should be implemented; along with additional regular expression checking to verify it meets the new guidelines. The ability to add user defined values should also be removed, or a process put in place to approve of new tag values. The benefits of implementing the sticter guidelines will allow for more consistent values for the amenities field. Referencing the following page https://taginfo.openstreetmap.org/keys/amenity#values indicates that they are over 9200 different values currently in use. Implimenting the suggestion changes would slowly bring that number down to a more manageable number.

There's a few fairly substantial drawbacks to implementing the stricter guidelines. First, the implementation of the regular expression checks could impact the website and servers the OSM data resides on, because it can be processing intensive which can impact customer service. Veteran users who previously had the ability to define their own values may abandon their use and updating of the map data reducing the overall accuracy of the data overtime. Finally, perhaps the biggest challenge in implementing this change, is updating all current values to meet the new guidelines, and pass all the regular expression checks. There would have to be a consensus on which values to remove and which to keep on a global stage which would complicate matters.