Project: Wrangle OpenStreetMap Data - Pierce County WI

Heidi Raasch

Table of Contents

Introduction

For this project I have chosen to analyze the OpenStreetMap data for Pierce County Wisconsin. I chose this area as my Husband and I plan on moving there in the next 3 years. I will be using data mungling techniques to such as assessing the quality of the data for validity, accuracy, completeness, consistency and uniformity, to clean the OpenStreetMap data. Once the data has been cleand I will use SQL as the data schema for the remainder of the project.

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, .json, .xml, and .html
- Process data from multiple files or very large files that can be cleaned programmatically.
- Learn how to store, query, and aggregate data using SQL.

Auditing the Data

Identify Tag Types

In order to begin auditing the data I used identifyTags.py to identify the tags used in the datafile. The tags that I will be using for this project are node and way.

A node is a single point in space defined by its longitude, latitude, and node id. A way is an ordered list of nodes.

Auditing the "k" values

I will begin looking for any problems that may need attention before importing into a database.

First I will look for tags with only lowercase letters, then I will look for lowercase letters separated by a colon, lastly I will look for any problem characters.

In [ ]:

```
lower = re.compile(r'^([a-z]|_)*$')
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')
```

To do this I will use the interparse method of ElementTree to create a dictionary of tags that met those criteria. The full code can be found in auditingK.py.

In [ ]:

```
{'lower': 43556, 'lower_colon': 54836, 'other': 2509, 'problemchars': 0}
```

Auditing the Users

For fun I looked into how many users contributed to the map of Pierce County WI using uniqueUsers.py.

I found that 210 unique users made contributions.

Problems Encountered

- The format of the street names is not consistent. Some were abbreviated, some were in uppercase, and some used different abbreviations.
- Zip codes were in inconsistent formats.

Street Types

First issue identified when working with this dataset was the inconsistent use of abbreviations for street types in the street names. In the following code, I create a list of street types that I would expect. Then create a dictionary of the types that are not in my expected list. Using that list I map the abbreviations used in the dataset to the format I specify.

In [6]:

```
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)

expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square",
            "Trail", "Parkway", "Commons","East", "North", "West","South"]
```

I used the code in streetTypes.py to search through the datafile and compare the values to the expected list. I then print the list of values identified and print them out for review.

In [7]:

```
import xml.etree.cElementTree as ET
import pprint
import re
from collections import defaultdict

def audit_street_type(street_types, street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type not in expected:
            street_types[street_type].add(street_name)


def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")


def audit(osmfile):
    osm_file = open(osmfile, "r", errors = 'ignore')
    street_types = defaultdict(set)
    for event, elem in ET.iterparse(osm_file, events=("start",)):

        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_street_name(tag):
                    audit_street_type(street_types, tag.attrib['v'])
    osm_file.close()
    return street_types

pierce_street_types = audit(datafile)
pprint.pprint(dict(pierce_street_types))
```

```
{'47': {'47'},
 '5': {'East Main Street #5'},
 'Ave': {'W Race Ave'},
 'Circle': {'Bauer Circle',
            'Frederick Circle',
            'Hackberry Circle',
            'Melville Circle',
            'Sandpiper Circle'},
 'E': {'2nd St E'},
 'FLORA': {'CORNER OF WALNUT AND FLORA'},
 'Knoll': {'Highview Knoll'},
 'Ln': {'Learning Ln'},
 'N': {'LAKE ST N'},
 'Path': {'Neill Path'},
 'RD': {'ORRIN RD'},
 'Rd': {'Tyler Rd', 'Twin Bluff Rd'},
 'Rd.': {'Industrial Rd.'},
 'ST': {'BROAD ST'},
 'ST.': {'BROAD ST.', 'DEXTER ST.'},
 'STREET': {'2000 OLD WEST MAIN STREET'},
 'St': {'114th St'},
 'St.': {'N. Maple St.'},
 'WI-29': {'WI-29'},
 'Way': {'Village Way', 'Teal Way', 'Sherman Way', 'Glacier Way'},
 'Y': {'County Road Y'}}
```

There are quite a few that are abbreviated in the incorrect format. Now I will use the code in updateStreetTypes.py to update the street names with the name format I identified previously.

Zip Codes

Another problem with this dataset is the zip codes. Some zip codes were in different formats and some were missing completely. First I created a dictionary of the zipcodes that were in an invalid format.

The only issue with the zipcode appears that WI is included in some of them so I corrected those records.

The code for doing hte identification and cleaning of the postal codes can be found in zipCodes.py.

Prepare Data for SQL

Now that the data has been cleaned it's time to prepare the data for loading into SQL.

The XML data will be parsed through and converted into tabular format into CSV files. The CSV files can be imported into sqlite. The code used for this process can be found in sqlPrep.py.

In [12]:

```
NODES_PATH = "nodes.csv"
NODE_TAGS_PATH = "nodes_tags.csv"
WAYS_PATH = "ways.csv"
WAY_NODES_PATH = "ways_nodes.csv"
WAY_TAGS_PATH = "ways_tags.csv"

LOWER_COLON = re.compile(r'^([a-z]|_)+:([a-z]|_)+')
PROBLEMCHARS = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')

SCHEMA = schema.schema

NODE_FIELDS = ['id', 'lat', 'lon', 'user', 'uid', 'version', 'changeset', 'timest
NODE_TAGS_FIELDS = ['id', 'key', 'value', 'type']
WAY_FIELDS = ['id', 'user', 'uid', 'version', 'changeset', 'timestamp']
WAY_TAGS_FIELDS = ['id', 'key', 'value', 'type']
WAY_NODES_FIELDS = ['id', 'node_id', 'position']


def shape_element(element, node_attr_fields=NODE_FIELDS, way_attr_fields=WAY_FIEL
                  problem_chars=PROBLEMCHARS, default_tag_type='regular'):

    node_attribs = {}
    way_attribs = {}
    way_nodes = []
    tags = []  # Handle secondary tags the same way for both node and way element

    if element.tag == 'node':
        for name, value in element.attrib.items():
            if name in node_attr_fields:
                node_attribs[name] = value

        for secondary in element.iter():
            if secondary.tag == 'tag':
                if problem_chars.match(secondary.attrib['k']) is not None:
                    continue
                else:
                    new_dict = tag_dictionary(element, secondary, default_tag_typ
                    if new_dict is not None:
                        tags.append(new_dict)
        return {'node': node_attribs, 'node_tags': tags}

    elif element.tag == 'way':
        for name, value in element.attrib.items():
            if name in way_attr_fields:
                way_attribs[name] = value

        counter = 0
        for secondary in element.iter():
            if secondary.tag == 'tag':
                if problem_chars.match(secondary.attrib['k']) is not None:
                    continue
                else:
                    new_dict = tag_dictionary(element, secondary, default_tag_typ
                    if new_dict is not None:
                        tags.append(new_dict)
            elif secondary.tag == 'nd':
                newnd = {}
                newnd['id'] = element.attrib['id']
                newnd['node_id'] = secondary.attrib['ref']
                newnd['position'] = counter
                counter += 1
                way_nodes.append(newnd)
        return {'way': way_attribs, 'way_nodes': way_nodes, 'way_tags': tags}
```

Data Overview

Now that the data is available in SQL I will start exploring it using SQL queries. I will be using the ipython-sql module to connect to the database and run the queries.

In [1]:
```
%load_ext sql
%sql sqlite:///om_pierce_county.db
%config SqlMagic.feedback = False
```

Number of Nodes

In [4]:
```
%sql SELECT COUNT(*) AS No_Nodes FROM nodes
```

```
 * sqlite:///om_pierce_county.db
```

Out[4]:

No_Nodes
333470

Number of Ways

In [6]:
```
%sql SELECT COUNT(*) AS No_Ways FROM ways;
```

```
 * sqlite:///om_pierce_county.db
```

Out[6]:

No_Ways
22074

Number of Unique Users

In [7]:

```
%%sql
SELECT COUNT(DISTINCT(users.uid)) as No_Unique_Users
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) as users;
```

```
 * sqlite:///om_pierce_county.db
```

Out[7]:

No_Unique_Users
194

Top 5 Contributing Users

In [8]:

```
%%sql
SELECT users.user as 'User Name', COUNT(*) as Contributions
FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) as users
GROUP BY users.user
ORDER BY Contributions DESC
LIMIT 5;
```

```
 * sqlite:///om_pierce_county.db
```

Out[8]:

User Name	Contributions
jumbanho	117783
woodpeck_fixbot	46992
iandees	44623
Omnific	39211
PrometheusAvV	17081

Number of Entries

In [9]:

```
%%sql
SELECT SUM(no) as No_Entries
FROM
(SELECT COUNT(*) as no
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) as sub
GROUP BY sub.uid)
```

```
 * sqlite:///om_pierce_county.db
```

Out[9]:

No_Entries
355544

Number of Cemetaries

In [10]:

```
%%sql
SELECT count(*) as No_Cemeteries FROM nodes_tags
WHERE value like '%Cemetery%'
```

```
 * sqlite:///om_pierce_county.db
```

Out[10]:

No_Cemeteries
25

Cemetary Names

In [37]:
```
%%sql
SELECT nwt.value as Cemetery_Name
FROM nodes_tags as nwt
WHERE nwt.key='name' and nwt.value LIKE '%Cemetery%'
ORDER BY nwt.value;
```

```
 * sqlite:///om_pierce_county.db
```

Out[37]:

Cemetery_Name
Beldenville Cemetery
Big River Cemetery
Diamond Bluff Cemetery
Esdaile Cemetery
Farm Hill Catholic Cemetery
Free Home Cemetery
Gilman Lutheran Cemetery
Greenwood Valley Cemetery
Lost Creek Cemetery
Maiden Rock Cemetery
Maple Grove Cemetery
Mission Covenant Cemetery
Mount Olivet Cemetery
Oak Ridge Cemetery
Pine Glen Cemetery
Plum City Union Cemetery
Poplar Hill Cemetery
Porcupine Cemetery
Saint Josephs Catholic Cemetery
Saint Josephs Cemetery
Salem Lutheran Cemetery
Spring Lake Cemetery
Tabor Cemetery
Thurston Hill Cemetery
Zion Covenant Cemetery

Number of Post Offices

In [43]:

```
%%sql
SELECT count(distinct value) as No_Post_Office
FROM nodes_tags
WHERE value like '%Post Office'
```

```
 * sqlite:///om_pierce_county.db
```

Out[43]:

No_Post_Office
6

Post Office Names

In [40]:

```
%%sql
SELECT value as Post_Office
FROM nodes_tags
WHERE value like '%Post Office'
GROUP BY value
ORDER BY value
```

```
 * sqlite:///om_pierce_county.db
```

Out[40]:

Post_Office
East Ellsworth Post Office
Ellsworth Post Office
Hastings Post Office
Red Wing Post Office
River Falls Post Office
Welch Post Office

Number of Cuisines

In [19]:

```
%%sql
SELECT nwt.value as Cuisine, COUNT(*) as Number
FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) as nwt
      INNER JOIN
     (SELECT DISTINCT(id) FROM nodes_tags WHERE nodes_tags.value='restaurant'
      AND nodes_tags.id NOT IN
                       (SELECT ways_nodes.node_id FROM ways_nodes
                        JOIN
                       (SELECT DISTINCT(id) FROM ways_tags WHERE ways_tags.value=
                       ON ways_nodes.id = dnwt.id)
      UNION ALL
      SELECT DISTINCT(id) FROM ways_tags WHERE ways_tags.value='restaurant') as d
      ON nwt.id=dwt.id
WHERE nwt.key='cuisine'
GROUP BY nwt.value
ORDER BY Number DESC
LIMIT 10;
```

```
 * sqlite:///om_pierce_county.db
```

Out[19]:

Cuisine	Number
american	2
chinese	2
pizza	2
barbecue	1
chicken3	1
coffee_shop	1
ice_cream	1
italian;american;mexican	1
japanese	1

Fast Food Restaurants

In [25]:

```
%%sql
SELECT nwt.value as Restaurant, COUNT(*) as Number
FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) as nwt
      JOIN
     (SELECT DISTINCT(id) FROM nodes_tags WHERE nodes_tags.value='fast_food'
      AND nodes_tags.id NOT IN
                       (SELECT ways_nodes.node_id FROM ways_nodes
                        JOIN
                       (SELECT DISTINCT(id) FROM ways_tags WHERE ways_tags.value=
                       ON ways_nodes.id = dnt.id)
      UNION ALL
      SELECT DISTINCT(id) FROM ways_tags WHERE ways_tags.value='fast_food') as wt
      ON nwt.id=wt.id
WHERE nwt.key='name'
GROUP BY nwt.value
ORDER BY Number DESC
LIMIT 10;
```

```
 * sqlite:///om_pierce_county.db
```

Out[25]:

Restaurant	Number
Dairy Queen	3
McDonald's	2
Subway	2
Culver's	1
Jimmy John's	1
Randy's	1

Additional Ideas

Since the information is entered in by a variety of users and there is no standardization. To improve this a standard form for entry could be used. That would ensure that each of the amentities, such as restaurants, would have the same information available. This would result in more complete results and better statistics.

In [24]:

```
%%sql

SELECT nwt.id, COUNT(key) as Number
FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) as nwt
      INNER JOIN
     (SELECT DISTINCT(id) FROM nodes_tags WHERE nodes_tags.value='restaurant'
      AND nodes_tags.id NOT IN
                       (SELECT ways_nodes.node_id FROM ways_nodes
                        JOIN
                       (SELECT DISTINCT(id) FROM ways_tags WHERE ways_tags.value=
                       ON ways_nodes.id = dnwt.id)
      UNION ALL
      SELECT DISTINCT(id) FROM ways_tags WHERE ways_tags.value='restaurant') as d
      ON nwt.id=dwt.id

GROUP BY nwt.id
ORDER BY nwt.id DESC
```

```
 * sqlite:///om_pierce_county.db
```

Out[24]:

id	Number
802228084	2
802192124	2
5577469201	5
5291408314	4
481461235	3
458589351	5
4486234357	2
4486234327	3
3821674294	6
334633471	4
319416603	3
319416602	3
2912759369	7
2574354284	9
2491545055	3
2491545053	5
2491545051	2
2491543014	2
2491543006	2
2397909513	3
2397907496	3
2397905736	3

236315339	4
1833753344	3
125017937	4

References

OpenStreetMap Wiki: https://wiki.openstreetmap.org/wiki/Using_OpenStreetMap (https://wiki.openstreetmap.org/wiki/Using_OpenStreetMap)
Jupyter Magics with SQL: https://towardsdatascience.com/jupyter-magics-with-sql-921370099589 (https://towardsdatascience.com/jupyter-magics-with-sql-921370099589)
GitHub : https://github.com/tf-coreml/tf-coreml/issues/134 (https://github.com/tf-coreml/tf-coreml/issues/134)

```
In [ ]:
```