AuditMapData

June 22, 2015

In [1]:
```
import xml.etree.cElementTree as ET
import pprint
from collections import defaultdict
import re

OSMFILE = "FernCreek-Highview_Kentucky.osm"  # The map data...
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)  # Regular Expression to pick off the
expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "Lane", "Road
            "Trail", "Parkway", "Commons", "Circle", "Way", "Trace"]  # Street types we expect


'''
Count all the tags found in the data.
Recieves the tags already found and the current element.  Inspect
the tag and add it to our counts.
'''
def count_tags(tags, elem):
    if tags.has_key(elem.tag):
        tags[elem.tag] += 1
    else:
        tags[elem.tag] = 1
    return tags

'''
Count attributes found in the data.
Recieves the tags already found and the current element.  Inspect
the tag and add it to our counts.
'''
def count_attribute(attribute, elem):
    if attribute.has_key(tag.attrib['v']):
        attribute[tag.attrib['v']] += 1
    else:
        attribute[tag.attrib['v']] = 1
    return attribute

'''
Count values found in the data.
Recieves the tags already found and the current element.  Inspect
the tag and add it to our counts.
'''
def count_value(attribute, value):
    if attribute.has_key(value):
        attribute[value] += 1
```

```
    else:
        attribute[value] = 1
    return attribute


'''
Pick off the street type and check it against a known list.
If we don't expect it, then record it along with the offending
street name for visual inspection.
'''
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)

'''
Main Code:
- open the map file
- initialize out counters and sets for storing what we find in the data
- process the data

The intial checks turned up very little wrong.  Only a couple of street types needed to be
corrected, so as I dug into the data, more and more components were checked.  All look
good.  So, then I started looking at combinations and discovered, in only a very few
instances, there was inconsistency in the amenity and building combinations.

- Once all the processing is compelte, print the results for review.  This will be used
to come up with a data cleaning plan.
'''
osm_file = open(OSMFILE, "r")
street_types = defaultdict(set)   # Initialize an empty variable to hold street types
tags = {}  # Initialize an empty variable for storing and counting unique tags
zipcodes = {}  # Initiailze an empty variable for zipcodes
states = {}  # Initiailze an empty variable for states
cities = {}  # Initiailze an empty variable for states
amenities = {}  # Initiailze an empty variable for amenities
worship_buildings = {}   # Initiailze an empty variable for worship buildings
fuel_buildings = {}   # Initiailze an empty svariableet for worship buildings
school_buildings = {}   # Initiailze an empty variable for school buildings


for event, elem in ET.iterparse(osm_file, events=("start",)):
    count_tags(tags, elem)  # Count all tags
    '''
    Since we are looking at combination of certain fields, we need variables to trigger
    when we find certain amenities we want to check.  In our case, we look for places_of_worshi
    fuel stations, and schools.
    '''
    worship = False
    fuel = False
    school = False
    building = None
```

```
    if elem.tag == "node" or elem.tag == "way":
        for tag in elem.iter("tag"):
            if tag.attrib.get('k') == "addr:street":
                audit_street_type(street_types, tag.attrib['v'])  # audit street types
            if tag.attrib.get('k') == "addr:postcode":
                count_attribute(zipcodes, elem)  # count unique zipcodes
            if tag.attrib.get('k') == "addr:state":
                count_attribute(states, elem) # count unique states
            if tag.attrib.get('k') == "addr:city":
                count_attribute(cities, elem)  # count unique cities
            if tag.attrib.get('k') == "amenity":
                count_attribute(amenities, elem) # count unique amenities

            if tag.attrib.get('k') == "amenity" and tag.attrib.get('v') == "place_of_worship":
                worship = True  # set if found a place_of_worship
            if tag.attrib.get('k') == "amenity" and tag.attrib.get('v') == "fuel":
                fuel = True  # set if found a fuel station
            if tag.attrib.get('k') == "amenity" and tag.attrib.get('v') == "school":
                school = True  # set if found a school
            if tag.attrib.get('k') == "building":
                building = tag.attrib.get('v')   # store the current building type
            if worship and building != None:
                count_value(worship_buildings, building) # count unique worship buildings
                worship = False  # Only count it once per node or way
            if fuel and building != None:
                count_value(fuel_buildings, building)  # count unique fuel buildings
                fuel = False  # Only count it once per node or way
            if school and building != None:
                count_value(school_buildings, building)  # count unique school buildings
                school = False   # Only count it once per node or way

# Output the audit results...

print "Tag Counts:"
pprint.pprint(tags)

print "\nStreet Types:"
pprint.pprint(dict(street_types))

print "\nZipcodes:"
pprint.pprint(zipcodes)

print "\nStates:"
pprint.pprint(states)

print "\nCities:"
pprint.pprint(cities)

print "\nAmenities:"
pprint.pprint(amenities)

print "\nWorship Buildings:"
pprint.pprint(worship_buildings)
```

```
        print "\nFuel Buildings:"
        pprint.pprint(fuel_buildings)

        print "\nSchool Buildings:"
        pprint.pprint(school_buildings)
```

```
Tag Counts:
{'bounds': 1,
 'member': 980,
 'meta': 1,
 'nd': 401916,
 'node': 370352,
 'note': 1,
 'osm': 1,
 'relation': 58,
 'tag': 262621,
 'way': 40541}

Street Types:
{'Chase': set(['Chloe Chase']),
 'Loop': set(['Outer Loop']),
 'North': set(['I 265 North']),
 'PT': set(['Pine PT']),
 'Pass': set(['Fern Valley Pass']),
 'Rd': set(['Beulah Church Rd']),
 'Run': set(['Hogans Run']),
 'South': set(['I 265 South']),
 'Walk': set(['Flirtation Walk'])}

Zipcodes:
{'40218': 820,
 '40219': 3027,
 '40225': 7,
 '40228': 7377,
 '40229': 1657,
 '40291': 16151,
 '40299': 3638}

States:
{'KY': 32612}

Cities:
{'Hollow Creek': 322,
 'Jeffersontown': 1356,
 'Louisville': 31046,
 'Spring Mill': 109}

Amenities:
{'fast_food': 11,
 'fire_station': 2,
 'fuel': 5,
 'grave_yard': 2,
 'parking': 26,
```

```
 'pharmacy': 4,
 'place_of_worship': 33,
 'police': 1,
 'post_box': 2,
 'post_office': 1,
 'restaurant': 14,
 'school': 15}

Worship Buildings:
{'church': 1, 'yes': 29}

Fuel Buildings:
{'roof': 1, 'yes': 3}

School Buildings:
{'yes': 4}
```

In [ ]: