Project3: OpenStreetMap Data Wrangling

Map Area / Data Overview

Denver Metro area, Colorado, United States, Earth. Sourced from Metro Extracts:
https://mapzen.com/data/metro-extracts/metro/denver-boulder_colorado/

One of my friends recently moved to this area, so I figured it could be fun to explore.

Summary Statistics

The .osm (XML) file covering this area is 876MB. Stripping the file down into csv based on the given schema and then populating a SQL database resulted in a 494MB .db file - still large, but a significant size reduction.

Within the database, there are:

```
SELECT COUNT(*) from nodes;
```

```
SELECT COUNT(*) from ways;
```

```
SELECT COUNT(DISTINCT(combo.uid))
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) combo;
```

4192064 nodes and **470334** ways, contributed by **2217** unique users.

Nodes make up ways

The ways are made up of between 1 and 1877 nodes, with an average of **10.28** nodes per way. Because of the additional position column in our way_nodes schema, the max value can be verified by simply querying for the max position without needing to first create an aggregate count of nodes by id.

```
SELECT AVG(nodes_per_way) from
(SELECT COUNT(*) as nodes_per_way
FROM way_nodes
GROUP BY id) /*way id*/
as avg;
```

```
SELECT MAX(position) from way_nodes;
```

What's on the map?

Supermarkets

Gotta have food. There are **305** nodes identified with k=shop and v=supermarket within the dataset.

```
SELECT COUNT(DISTINCT(alltags.id)) as foods from
(SELECT * FROM node_tags UNION ALL SELECT * FROM way_tags) alltags
WHERE key="shop" and value="supermarket";
```

Cycleways

I don't own a car and predominantly travel by bicycle. The dataset includes **4823** ways with k=highway and v=cycleway, the most accepted method of tagging a cycle path.

```
SELECT COUNT(DISTINCT(way_tags.id)) as cycleways
FROM way_tags
WHERE key="highway" and value="cycleway";
```

Who made the map?

The top 10 contributors are responsible for 66% of the ways and nodes in the dataset.

```
SELECT allitems.user, COUNT(*) as edits FROM
(SELECT user FROM nodes UNION ALL SELECT user FROM ways) allitems
GROUP BY allitems.user
ORDER BY edits DESC
LIMIT 10;
```

user	edits
chachafish	730400
Your Village Maps	704982
woodpeck_fixbot	343713
GPS_dr	316046
jjyach	282260
DavidJDBA	184898
Stevestr	170106
CornCO	139753

russdeffner	124392
Berjoh	84700

Problems in the Map

Before I get too far exploring, it's a good idea to check for any strange data that might be included in the set as downloaded. To do this, I'll target fields such as postal code that have well-defined authoritative values so that I can make reasonably objective assessments regarding the accuracy of the data.

Here's a python function I wrote to easily survey the values for a given tag/field:

```
def count_field(osmfile, field_test, skip_test=lambda x: False):
    '''
    return a Counter over the values of fields that match field_test
    field_test should take a tag element and return True if that element's
                "v" attribute should be counted
    skip_test should take a value and return True if the value should be
skipped
    '''
    osm_file = open(osmfile, 'r')
    count = Counter()
    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 field_test(tag):
                    v = tag.attrib['v']
                    if not skip_test(v):
                        count[v] += 1
    return count
```

Of course, doing this in python is *slow* when operating over the entire dataset. On my desktop it takes close to a full minute for this function to complete. By comparison, after loading the data into a SQL database, I can generate similar data almost instantaneously using a query like this one:

```
SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM node_tags UNION ALL SELECT * FROM way_tags) tags
WHERE tags.key='key_goes_here'
GROUP BY tags.value
ORDER BY count DESC;
```

Problem: Postal Codes

US Postal codes are five numbers, optionally followed by an additional four. So `80214` is the most common format, but `80214-1805` is also valid according to USPS, though the "zip+4" format is not widely used. The denver area has zipcodes between 80000 and 80700, and to be consistent, all the postcodes should just use the five digits and not zip+4. What data in the addr:postcode field doesn't fit this?

```
Counter({'80214-1803': 10, '80214-1801': 8, '80503-7570': 6, '80214-1833': 4,
'80210-2938': 3, 'Golden, CO 80401': 3, 'CO': 3, '80214-1805': 2, '801112': 2,
'801111': 2, '80214-1825': 2, '1800': 2, 'CO 80401': 1, '80214-1807': 1,
'80229-7923': 1, '80514-8502': 1, '80012-2543': 1, 'CO80219': 1, '80234-4154':
1, '80226-2975': 1, '80012-4014': 1, '802377': 1, '80501-6423': 1, '80444-
2000': 1, '80642-9615': 1, '80113-1523': 1, '80219-1535': 1, '80504-8601': 1,
'80026-2872': 1, 'CO 80223': 1, '80127-5008': 1, '80113-1525': 1, '80214-
1838': 1, '80303-1229': 1, 'CO 80439': 1, 'CO 80027': 1, 'CO 80305': 1,
'80504-6311': 1, '80214-1837': 1, '80011-3316': 1, '80504-7613': 1, '80305-
9998': 1, '80247-2121': 1, '80014-1319': 1, '80010-1425': 1, 'Highlands
Ranch,': 1, '8023': 1, '80012-2523': 1, '80504-5510': 1, '80124-5644': 1,
'90222': 1, '80002-4617': 1, 'Seventh Street': 1, '80504-3546': 1, '6210': 1})
```

Quite a few entries need some fixing. There are a couple things to do here. For any zip+4 values, I'll just remove the extra 4 digits so that the field is consistently 5 digits. Same thing for a few entries that have the state prefix "CO" ahead of a valid 5 digit value. After those are fixed via the code below, there are just a few issues that should be tackled on a case by case basis.

```
def is_denver_postcode(zip):
    zip = int(zip)
    if (zip > 80000) and (zip < 80700):
        return True
    return False

def fix_postcode(tag_val):
    ZIP = re.compile((r'^[0-9]{5}$'))
    ZIPPLUSFOUR = re.compile((r'^([0-9]{5})-[0-9]{4}$'))
    COZIP = re.compile((r'^CO\s*([0-9]{5})$'))
    if ZIP.search(tag_val):
        if is_denver_postcode(tag_val):
            return tag_val
        else:
            print "Unexpected ZIP:", tag_val
            return tag_val
    elif ZIPPLUSFOUR.search(tag_val):
        zip = ZIPPLUSFOUR.search(tag_val).group(1)
        if is_denver_postcode(zip):
            return zip
        else:
            print "Unexpected ZIP:", tag_val
            return zip
    elif COZIP.search(tag_val):
        zip = COZIP.search(tag_val).group(1)
        if is_denver_postcode(zip):
            return zip
        else:
            print "Unexpected ZIP:", tag_val
            return zip
    else:
        print "Invalid ZIP:", tag_val
        return tag_val
```

The remaining messed up zip codes are shown below. One of these, 90222, is a valid zipcode, but not for the area our dataset covers. The others appear to be typos or other human data entry error, and to handle this in the best way possible, it would be good to investigate each individual node or way.

```
Counter({'CO': 3, 'Golden, CO 80401': 3, '1800': 2, '801112': 2, '801111': 2,
'90222': 1, '802377': 1, 'Seventh Street': 1, '6210': 1, 'Highlands Ranch,':
1, '8023': 1})
```

Problem: State

The addr:state field is mostly the two-letter abbreviation "CO" but some "Colorado", and a few other typos:

```
Counter({'CO': 18006, 'Colorado': 624, 'co': 89, 'Co': 41, 'C': 1, 'cO': 1,
'80216': 1, 'Coloraod': 1})
```

I'm mostly just looking for consistency here and none of these look odd other than the zipcode in the wrong field, so I feel safe programmatically wiping and replacing values that are not CO with CO.

```
def fix_state(tag_val):
    if tag_val != 'CO':
        return 'CO'
    return tag_val
```

Additional Ideas

Open Street Map is a community edited map, and in order to be easily editable and adaptable, the tagging system is very open. This of course results in some very inconsistent data. A good example of this is the building field:

```
SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM node_tags UNION ALL SELECT * FROM way_tags) tags
WHERE tags.key='building'
GROUP BY tags.value
ORDER BY count DESC;
/* LIMIT 10; */
yes|45946
```

```
    house|41276
    apartments|13811
    garage|11266
    residential|3038
    retail|2688
    roof|2288
    terrace|1840
    industrial|1771
    shed|1727
```

In addition to the top 10 values (which already have a few kind of weird values like yes and roof), there is a long tail of very specific values (four "brewery", three "clubhouse", etc.) along with some that are likely mistakes or don't meet community established guidelines, such as "includes commercial use", "?", and "note 519591".

Given that OSM is fairly well populated at this point, at least in the US, I think there's a good argument to be made for starting to be a bit more strict with data input rather than allowing all values. One way to do this might be to query a surrounding area and display a warning or confirmation UI if there are no other tags in the area with the new value. This would still allow for growth of the map, but could help prevent bad data from entering the map. This isn't without cost - the query costs extra computation power for each update, and a lot of updates are programmatic batches, so an API reply warning might be just be ignored.