Wrangle Vancouver OpenStreetMap Data

Justin Ibanez

Introduction

I started by running the XML file for Vancouver through the *data.py* code provided to convert it into CSVs, but an error occurred. After examination, it appeared the error was caused by data points that did not have a user or uid attribute. I edited the code to provide default values to such points. Below is the edit I made.

```
for field in NODE_FIELDS:
    # give default values to data points without user or uid
    if field not in element.attrib:
        if field == 'user':
            node_attribs['user'] = ''
        elif field == 'uid':
            node_attribs['uid'] = -1
        else:
            node_attribs[field] = element.attrib[field]
```

Auditing and Cleaning

I started by looking through the longitude and latitude of *nodes.csv* to check if they were within the boundaries as stated in OSM. The result was that all nodes were within the expected values. Afterwards, I looked at various keys and their values in *nodes_tags.csv* and *ways_tags.csv* using the audit_key function in *audit_csv.py* (with the exception of street names, which I wrote the audit_street_types function for) and found several cases of data that needed to be cleaned. The problems I encountered and how I approached them (including the functions written) are listed below.

is_in

The same information was represented in multiple ways for this key as seen below. I standardized this information by changing all values to not include the province name by using a dictionary. For example, 'Vancouver, British Columbia' was changed to 'Vancouver'.

```
is_in_dict = {
    'Vancouver, British Columbia': 'Vancouver',
    'Burnaby,British Columbia': 'Burnaby',
    'UBC,British Columbia': 'University of British Columbia',
    'Burnaby, BC': 'Burnaby',
    'Vancouver,British Columbia': 'Vancouver',
    'Richmond,British Columbia': 'Richmond',
    'North Vancouver,British Columbia': 'North Vancouver',
```

```
    'Vancouver,British Columbia,Canada': 'Vancouver',
    'Vancouver, BC': 'Vancouver'
}

def clean_is_in(is_in):
    return is_in_dict[is_in]
```

city

This values for this key were similar to the previous one in that the same information had multiple representations. The values were standardized to the two possible values, Vancouver and North Vancouver, using a dictionary.

```
city_dict = {
    'North Vancouver City': 'North Vancouver',
    'vancouver': 'Vancouver',
    'north vancouver': 'North Vancouver',
    'Vancouver, BC': 'Vancouver',
    'Vancovuer': 'Vancouver'
}

def clean_city(city):
    return city_dict[city]
```

height

I wanted the height to be in the form '# metres'. A regular expression was used to pull the floating number from the value to structure the data. One exception was a value in feet instead of metres. To confirm this was not a typo I pulled out the information for this value using the audit_values function in *audit_csv.py* and discovered it referred to a totem pole that was 50 feet tall, so I converted it into the proper height in metres.

```
def clean_height(height):
    if 'ft' in height:
        return '30.48 metres'
    height_number = re.findall(r"[-+]?\d*\.\d+|\d+", height)[0]
    return height_number + ' metres'
```

province

As expected, most of the values for this key referred to British Columbia in different ways ('BC', 'B.C.', etc.). However, there were two instances referring to Ontario instead. Again, the *audit_values* function was used to investigate further and I confirmed that both locations were indeed referring to places in British Columbia. This led me to believe this was simply human error. I standardized all values to 'British Columbia'.

```
def clean_province(province):
    return 'British Columbia'
```

postcode

I chose to structure the postal codes to the form 'A1B 2C3'. This could be accomplished for most values using string manipulation, but a dictionary needed to be used for some cases such as typos introducing a comma or an extra letter. Some values needed to be investigated further in order to find their proper postal code.

```
postcode_dict = {
    'V5P 3W7,': 'V5P 3W7',
    'V6': 'V6T 1W5',
    'V5Z': 'V5Z 0A5',
    '5368': 'V5Z 2Z8',
    'BC V6B 2E2': 'V6B 2E2',
    'V5T-4T1;V5T 4T1': 'V5T 4T1',
    'BC V6R 1P2': 'V6R 1P2',
    'V6C': 'V6C 3R9',
    'V6HH 2R5': 'V6H 2R5'
}

def clean_postcode(postcode):
    postcode = postcode.upper()
    if len(postcode) == 7:
        if ' '  in postcode:
            return postcode
        return postcode.replace('-', ' ')
    elif len(postcode) == 6:
        return postcode[0:3] + ' ' + postcode[3:]
    elif postcode in postcode_dict:
        return postcode_dict[postcode]
```

street

I wanted to use unabbreviated names (for example, 'Street' instead of 'St.'). This was done by looping through the words in the street name and replacing them if they were contained in a dictionary. A few special cases where street numbers appeared in the street name were cleaned using string manipulation before passing the value through the loop. The first word was then tested against another dictionary to clean abbreivations unique to the first word, but more importantly it was to account for abbreviations for 'Saint' being corrected to 'Street' during the first pass.

```
# see python file for dictionaries

def clean_street(street):
    # removes numbers
    if '#' in street:
        num_pos = street.find('#')
        street = street[:num_pos-1]
```

```
    # first pass corrects most errors
    for word in street.split():
        if word in street_dict1:
            street = street.replace(word, street_dict1[word])

    # second pass corrects street beginning
    # also compensates for shorthands for 'Saint' being changes to 'Street'
    if street.split()[0] in street_dict2:
        street = street.replace(street.split()[0], street_dict2[street.split()[0]],
1)
    return street
```

Once the data was cleaned, I imported it into an SQL database using *import_csv.py*.

Statistical Overview

File sizes

Filename	File size
vancouver.osm	173 MB
vancouver.db	87.3 MB
nodes.csv	67.3 MB
nodes_tags.csv	1.28 MB
ways.csv	9.41 MB
ways_tags.csv	7.44 MB
ways_nodes.csv	23.5 MB
cleaned_nodes_tags.csv	1.25 MB
cleaned_ways_tags.csv	7.21 MB

Interestingly, cleaning the data lowered the filesizes.

Number of unique users

```
SELECT count(uid)
FROM (SELECT uid FROM nodes UNION SELECT uid FROM ways) AS all_uid;
```

Result: 791

Number of nodes

```
SELECT count(*) FROM nodes;
```

Result: 800930

Number of ways

```
SELECT count(*) FROM ways;
```

Result: 154560

Number of shops

```
SELECT count(key) FROM nodes_tags WHERE key = "shop";
```

Result: 1459

Additional Examination

Checking the edits I made at the beginning

```
SELECT * FROM nodes WHERE user='' AND uid=-1;
```

Result:

```
30353228|49.2986292|-123.1211329||-1|1|96018|2007-06-14T16:54:18Z
30353231|49.2984913|-123.1201246||-1|1|96018|2007-06-14T16:54:20Z
30353302|49.306653|-123.1351983||-1|1|96018|2007-06-14T16:54:46Z
30353340|49.3085146|-123.1411967||-1|1|96018|2007-06-14T16:54:57Z
```

All of these contributions were made within a minute of each other and had the same changeset value. Therefore, I am confident they were made by the same person and my correction did not give credit for multiple contributions to one person.

Top 10 days for contributions

```
SELECT date(sq.timestamp), count(*)
FROM (SELECT timestamp FROM nodes UNION ALL SELECT timestamp FROM ways) AS sq
GROUP BY date(sq.timestamp)
ORDER BY count(*) DESC
LIMIT 10;
```

Result:

```
2015-07-01|92028
2015-07-02|79950
2015-06-30|68501
```

```
2015-06-26|59733
2015-06-29|44258
2015-07-03|31091
2016-01-18|6422
2015-01-01|5269
2014-12-31|5203
2016-01-17|4925
```

Users who contributed on the most popular day ordered by amount of contributions

```
SELECT sq.user, count(*)
FROM (SELECT user FROM nodes WHERE date(timestamp)="2015-07-01" UNION ALL SELECT
user FROM ways WHERE date(timestamp)="2015-07-01") AS sq
GROUP BY sq.user
ORDER BY count(*) DESC;
```

Result:

```
still-a-worm|35187
michael_moovelmaps|29444
treeniti2|16979
Siegbaert|5795
mame-stgt|3296
keithonearth|1321
muratc3|6
```

Top 5 common shop types

```
SELECT value, COUNT(*)
FROM nodes_tags
WHERE key="shop"
GROUP BY value
ORDER BY COUNT(*) DESC
LIMIT 5;
```

Result:

```
clothes|173
hairdresser|100
convenience|93
beauty|78
supermarket|77
```

I chose to explore the most popular shop type, clothes, more.

Top 5 common clothing shops

```
SELECT nodes_tags.value, COUNT(*)
FROM nodes_tags, (SELECT id, key, value FROM nodes_tags WHERE key="shop" AND
value="clothes") AS sq
WHERE nodes_tags.id=sq.id
AND nodes_tags.key="name"
GROUP BY nodes_tags.value
ORDER BY COUNT(*) DESC
LIMIT 5;
```

Result:

```
American Apparel|3
Exposure|3
Roots|3
Aritzia|2
Army and Navy|2
```

There are only three instances of the most common clothing store, so I decided to look at how many clothing stores appeared more than once.

Clothing shops that appear more than once

```
SELECT nodes_tags.value, COUNT(*)
FROM nodes_tags, (SELECT id, key, value FROM nodes_tags WHERE key="shop" AND
value="clothes") as sq
WHERE nodes_tags.id=sq.id
AND nodes_tags.key="name"
GROUP BY nodes_tags.value
HAVING COUNT(*) > 1
ORDER BY COUNT(*) DESC;
```

Result:

```
American Apparel|3
Exposure|3
Roots|3
Aritzia|2
Army and Navy|2
Banana Republic|2
Black & Lee|2
District Factory Outlet|2
Frank & Oak|2
GAP|2
Icebreaker|2
Motherland|2
Winners|2
```

This suggested the vast majority of clothing stores were unique. Although admittedly this data was not cleaned and thus there could be other reasons for this result.

While conducting this investigation I noticed one source of dirtiness in this data, which is elaborated on below.

How many times does the value 'yes' appear?

```
SELECT COUNT(*) FROM nodes_tags WHERE value="yes";
```

Result: 1317

Number of node tags

```
SELECT COUNT(*) FROM nodes_tags;
```

Result: 34507

This number is small relative to the number of node tags (a little less than 4%), but if these values are concentrated under a key then the statistics for that key could be misleading.

How many times does the value 'yes' appear under the key 'shop'?

```
SELECT COUNT(*) FROM nodes_tags WHERE key="shop" AND value="yes";
```

Result: 35

While this is a little less than 2% of the shops, if 'yes' were its own shop type then it would be the 11th most common type. In particular, since the number of each clothing store ranged from one to three, the statistics would be sensitive to small changes.

Suggestion for Improvement

For many keys, the corresponding value was simply 'yes' instead of something more descriptive. On the one hand this is better than nothing at all and in some cases this information may be all that is necessary. If one is simply interested in counting, for example, the number of shops then all that needs to be known is whether a shop exists rather than what type of shop exists. However, the detail lost could be detrimental. In another example, if one is conducting market research then they may wish to know the types of shops that exist in an area. The 'yes' values may cause misleading

statistics.

Therefore, my suggestion for an improvement is to reject any attempts to enter a value of 'yes'. The upside to this improvement is the increased level of detail. This will be useful to anyone conducting analysis of this area and will provide a clearer picture of the area for users. The possible downsides are the increased effort for data entry disuading some people and the data being more high-maintenance. For example, if a plaza is known for its high turnover rate of shops then it might just be beneficial to know that the area is used for shops rather than the specific shops that appear. Finally, a key might exist where a value of 'yes' is expected.

Conclusion

While there is still more that can be done to further clean this data, several of the major keys were cleaned and the result was a measureable decrease to the file sizes. After giving some overview statistics, the clothing stores in this area were investigated further and the result was that many clothing stores were unique. This could suggest a strong independent clothing manufacturer scene in Vancouver. However, it was also shown that generic values in the data could be throwing this statistic off; this would be in addition to other potential sources of dirtiness. The data would have to be further cleaned and investigated more rigorously to make any statistically significant conclusions.

I would like to thank Shanon Bradshaw for creating the lesson on data extraction and Carl for creating the course on SQL.