OpenStreetMap Project

Map Area

Vancouver, BC, Canada

https://www.openstreetmap.org/relation/1852574

This map is one of my favorite places.

In [1]:

```
OSMFILE = 'vancouver.osm'
```

In [2]:

```
import mapparser
import xml.etree.cElementTree as ET
import update
import re
from collections import defaultdict
import schema
import csv
import codecs
import cerberus
import pandas as pd
import sqlalchemy

mapparser.count_tags(OSMFILE)
```

Out[2]:

```
defaultdict(int,
            {'member': 1355,
             'nd': 109258,
             'node': 87742,
             'osm': 1,
             'relation': 186,
             'tag': 32705,
             'way': 16904})
```

In [3]:

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


def key_type(element, keys):
    if element.tag == "tag":
        if lower.match(element.attrib['k']):
            keys['lower'] += 1
        elif lower_colon.match(element.attrib['k']):
            keys['lower_colon'] += 1
        elif problemchars.match(element.attrib['k']):
            keys['problemchars'] += 1
        else:
            keys['other'] += 1

    return keys



def process_map(filename):
    keys = {"lower": 0, "lower_colon": 0, "problemchars": 0, "other": 0}
    for _, element in ET.iterparse(filename):
        keys = key_type(element, keys)

    return keys
```

```
In [4]:
```

```
process_map(OSMFILE)
```

```
Out[4]:
```

```
{'lower': 28526, 'lower_colon': 3887, 'other': 292, 'problemchars': 0}
```

Similar to our case study, we can identify the structure of our vancouver.osm dataset.

'lower' : 28526 for valid tags with only lowercase letters. 'lower_colon' : 33527 for tags with a colon which are also valid. 'problemchars' : 0 for tags with special/problematic characters. 'other' : 4762 for other which are outside of the other groups.

Improving Street Names

With abbreviated street names, we could use a mapping dictionary to update with full names

```
In [5]:
```

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

expected = ["Court", "Place", "Square", "Lane","Trail", "Parkway", "Commons","Way",
            "Alley","Steeg","Avenue","Laan","Boulevard","Kringweg","Close","Crescent","Singel",
            "Drive","Rylaan","Place","Oord","Road","Weg","Street","Straat"]


mapping = { "St": "Street",
           "ST": "Street",
            "st": "Street",
           "st.": "Street",
           "st,": "Street",
            "street":"Street",
           'Sq.': 'Square',
            "Ave": "Avenue",
            "ave": "Avenue",
           'Ave.': 'Avenue',
            "Rd.": "Road",
           "Rd": "Road",
            "Cresent":"Crescent",
            "drive":"Drive",
           'HIghway': 'Highway',
           'Hwy': 'Highway',
            }

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)

# Create a dictionary for our postal codes
def audit_postal_code(postal_code_types, postal_code):
    ''' check if a given postal code is an expected type
    postal code is of valid format if it matches one of the following:
    X#X #X#, X#X-#X#, X#X#X#.

    ARGS:
    postal_types (Dict): Dictionary containing a set of invalid postal codes (gets updated in function)
.
        postal_code (String): Postal code.
    '''

    # Fix postal codes with extraneous letters
    postal_code = postal_code.strip('AB ,')
    postal_regex = re.compile(r'[T][0-9][A-Z][\-\s]*[0-9][A-Z][0-9]', re.IGNORECASE)
    match = postal_regex.search(postal_code)

    if not match:
        postal_types['invalid'].add(''.join(postal_code))


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

```
#tag checking
def is_postal_code(elem):
    return (elem.attrib['k'] == "addr:postcode")

def audit(osmfile):
    osm_file = open(osmfile, "r")
    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'])
                    tag.attrib['v']=update_name(tag.attrib['v'], mapping)
    return street_types

def update_name(name, mapping):
    for street_type in mapping:
        if street_type in name:
            name = re.sub(r'\b' + street_type+ r'\b\.?', mapping[street_type],name)
    return name

def update_postal_code(postal_code):
    '''Fixes postal code if in an improper format.

    Args:
        postal_code (String): Postal code.

    Returns:
        postal_code (String): Postal code in form X#X #X#.
    '''
    postal_code = postal_code.strip(',')

    # For the invalid key: we can't get the full code so we ignore
    if len(postal_code) > 7:
        return
    elif len(postal_code) == 6:
        postal_code = postal_code[:3] + ' ' + postal_code[3:]
    else:
        postal_code = postal_code[:3] + ' ' + postal_code[4:]

    return postal_code
```

In [6]:

```
audit(OSMFILE)
```

Out[6]:

```
defaultdict(set,
            {'108': {'8th Ave W #108'},
             'Broadway': {'East Broadway', 'West Broadway'},
             'Diversion': {'Victoria Diversion'},
             'E': {'37th Ave E'},
             'East': {'Grand Boulevard East'},
             'Esplanade': {'West Esplanade'},
             'Highway': {'Lougheed Highway'},
             'Jarvis': {'Jarvis'},
             'Kingsway': {'Kingsway'},
             'Mall': {'East Mall', 'Main Mall', 'Wesbrook Mall'},
             'Mews': {'Eldorado Mews', 'Menchions Mews'},
             'North': {'East Kent Avenue North'},
             'Rd.': {'Boundary Rd.'},
             'St': {'Robson St', 'Shaughnessy St', 'Whitchurch St'},
             'St.': {'Mainland St.'},
             'Streer': {'Water Streer'},
             'Terminal': {'Station Terminal'},
             'West': {'Grand Boulevard West'}})
```

Saving CSV Files

Preparing the data to be inserted into a SQL database. To do so we will parse the elements in the OSM XML file, transforming them from document format to tabular format, thus making it possible to write to .csv files. These csv files can then easily be imported to a SQL database as tables.

In [7]:

```
OSM_PATH = "vancouver.osm"

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

# Make sure the fields order in the csvs matches the column order in the sql table schema
NODE_FIELDS = ['id', 'lat', 'lon', 'user', 'uid', 'version', 'changeset', 'timestamp']
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_FIELDS,
                  problem_chars=PROBLEMCHARS, default_tag_type='regular'):
    """Clean and shape node or way XML element to Python dict"""

    node_attribs = {}
    way_attribs = {}
    way_nodes = []
    tags = []  # Handle secondary tags the same way for both node and way elements
    count=0

    if element.tag == 'node':
        # id = element.attrib['id']
        for item in node_attr_fields:
            node_attribs[item] = element.attrib[item]
        # code for 'node' element (the parent)

    if element.tag == 'way':
        for item in way_attr_fields:
            way_attribs[item] = element.attrib[item]
        # code for 'way' element (the parent)

    for child in element:
        id = element.attrib['id']
        # code for child elements

        if child.tag == 'tag':
            if problem_chars.match(child.attrib['k']):
                    continue
            else:
                fields={}
                fields['id'] =id
                fields['value'] = child.attrib['v']

                if child.attrib["k"] == 'addr:street':
                        # calling the update_name function
                        fields["value"] = update_name(child.attrib["v"], mapping)
                    # otherwise:
                else:
                        fields["value"] = child.attrib["v"]

                if child.attrib["k"] == 'addr:postcode':
                        # call the update_postal_code function
                        fields["value"] = update_postal_code(child.attrib['v'])
                else:
                        fields['value'] = child.attrib['v']

                if ':' in child.attrib['k']:
                    loc = child.attrib['k'].find(':')
                    key = child.attrib['k']
                    fields['type'] = key[:loc]
                    fields['key'] = key[loc+1:]
                else:
                    fields['key'] = child.attrib['k']
                    fields['type']= 'regular'
                tags.append(fields)
            # code for tag children

        if child.tag == 'nd':
            nds={}
            nds['id']=id
            nds['node_id']=child.attrib['ref']
            nds['position']=count
            count+=1
            way_nodes.append(nds)
            # code for 'nd' children

    if element.tag == 'node':
        return {'node':node_attribs, 'node_tags': tags}
    if element.tag == 'way':
        return {'way': way_attribs, 'way_nodes': way_nodes, 'way_tags': tags}
# ================================================== #
#               Helper Functions                     #
# ================================================== #
def get_element(osm_file, tags=('node', 'way', 'relation')):
    """Yield element if it is the right type of tag"""

    context = ET.iterparse(osm_file, events=('start', 'end'))
    _, root = next(context)
    for event, elem in context:
        if event == 'end' and elem.tag in tags:
            yield elem
            root.clear()


def validate_element(element, validator, schema=SCHEMA):
    """Raise ValidationError if element does not match schema"""
    if validator.validate(element, schema) is not True:
        field, errors = next(validator.errors.iteritems())
        message_string = "\nElement of type '{0}' has the following errors:\n{1}"
        error_string = pprint.pformat(errors)

        raise Exception(message_string.format(field, error_string))


class UnicodeDictWriter(csv.DictWriter, object):
    """Extend csv.DictWriter to handle Unicode input"""

    def writerow(self, row):
        super(UnicodeDictWriter, self).writerow({
            k: (v.encode('utf-8') if isinstance(v, unicode) else v) for k, v in row.iteritems()
        })

    def writerows(self, rows):
        for row in rows:
            self.writerow(row)


# ================================================== #
#               Main Function                        #
# ================================================== #
def process_map(file_in, validate):
    """Iteratively process each XML element and write to csv(s)"""

    with codecs.open(NODES_PATH, 'w') as nodes_file, \
         codecs.open(NODE_TAGS_PATH, 'w') as nodes_tags_file, \
         codecs.open(WAYS_PATH, 'w') as ways_file, \
         codecs.open(WAY_NODES_PATH, 'w') as way_nodes_file, \
         codecs.open(WAY_TAGS_PATH, 'w') as way_tags_file:

        nodes_writer = UnicodeDictWriter(nodes_file, NODE_FIELDS)
        node_tags_writer = UnicodeDictWriter(nodes_tags_file, NODE_TAGS_FIELDS)
        ways_writer = UnicodeDictWriter(ways_file, WAY_FIELDS)
        way_nodes_writer = UnicodeDictWriter(way_nodes_file, WAY_NODES_FIELDS)
        way_tags_writer = UnicodeDictWriter(way_tags_file, WAY_TAGS_FIELDS)

        nodes_writer.writeheader()
        node_tags_writer.writeheader()
        ways_writer.writeheader()
        way_nodes_writer.writeheader()
        way_tags_writer.writeheader()

        validator = cerberus.Validator()
        for element in get_element(file_in, tags=('node', 'way')):
            el = shape_element(element)
            if el:
                if validate is True:
                    validate_element(el, validator)

                if element.tag == 'node':
                    nodes_writer.writerow(el['node'])
                    node_tags_writer.writerows(el['node_tags'])
                elif element.tag == 'way':
                    ways_writer.writerow(el['way'])
                    way_nodes_writer.writerows(el['way_nodes'])
                    way_tags_writer.writerows(el['way_tags'])
```

In [8]:

```
data=process_map(OSMFILE, validate=False)
```

Taking a look at the CSV files

Now we can easily see the content of our csv files using Pandas.

In [9]:

```
csv_nodes = pd.read_csv("nodes.csv", encoding="utf-8")
csv_nodes.head()
```

Out[9]:

	id	lat	lon	user	uid	version	changeset	timestamp
0	25250662	49.197806	-123.102663	z-dude	135851	17	8895101	2011-08-01T20:35:13Z
1	25251476	49.204993	-123.140238	lokejul	2034065	7	30169891	2015-04-12T18:39:21Z
2	25251499	49.245394	-123.127659	lokejul	2034065	7	52386465	2017-09-26T14:38:16Z
3	25251514	49.258155	-123.048388	mattropolis	492807	7	18326670	2013-10-13T05:57:41Z
4	25477656	49.234331	-123.139615	pnorman	355617	83	20345972	2014-02-03T02:25:46Z

In [10]:

```
csv_nodes_tags = pd.read_csv("nodes_tags.csv", encoding="utf-8")
csv_nodes_tags.head()
```

Out[10]:

	id	key	value	type
0	25477656	source	Bing	regular
1	25477656	highway	traffic_signals	regular
2	26046289	barrier	bollard	regular
3	26270974	highway	traffic_signals	regular
4	26577982	highway	traffic_signals	regular

In [11]:

```
csv_ways = pd.read_csv("ways.csv", encoding="utf-8")
csv_ways.head()
```

Out[11]:

	id	user	uid	version	changeset	timestamp
0	4231652	keithonearth	154287	21	41078623	2016-07-28T06:44:38Z
1	4489462	fmarier	24555	43	51386611	2017-08-23T19:46:28Z
2	4520111	DustinDauncey	1355239	6	20095330	2014-01-19T23:21:46Z

	id	user	uid	version	changeset	timestamp
3	4645569	fmarier	24555	31	48548822	2017-05-10T03:04:28Z
4	4681261	keithonearth	154287	8	28734980	2015-02-09T19:08:09Z

In [12]:

```
csv_ways_nodes = pd.read_csv("ways_nodes.csv", encoding="utf-8")
csv_ways_nodes.head()
```

Out[12]:

	id	node_id	position
0	4231652	25251511	0
1	4231652	2884758539	1
2	4231652	251634126	2
3	4231652	2884758538	3
4	4231652	426297140	4

In [13]:

```
csv_ways_tags = pd.read_csv("ways_tags.csv", encoding="utf-8")
csv_ways_tags.head()
```

Out[13]:

	id	key	value	type
0	4231652	hgv	no	regular
1	4231652	name	South Grandview Highway	regular
2	4231652	is_in	Vancouver, BC	regular
3	4231652	oneway	no	regular
4	4231652	highway	secondary	regular

Insert Data

We can import the data saved in the csv files to the sqlite database using sqlalchemy package

In [14]:

```
disk_engine = sqlalchemy.create_engine('sqlite:///vancouver_db.db')
```

In [15]:

```
csv_nodes.to_sql('nodes', disk_engine, if_exists='replace', index=False)
```

In [16]:

```
csv_nodes_tags.to_sql('nodes_tags', disk_engine, if_exists='replace', index=False)
```

In [17]:

```
csv_ways.to_sql('ways', disk_engine, if_exists='replace', index=False)
```

In [18]:

```
csv_ways_nodes.to_sql('ways_nodes', disk_engine, if_exists='replace', index=False)
```

In [19]:

```
csv_ways_tags.to_sql('ways_tags', disk_engine, if_exists='replace', index=False)
```

Data Overview

File Size

File Size

In [20]:

```
import os
```

In [21]:

```
print "vancouver.osm: " + str(os.path.getsize(OSMFILE) / 1024 / 1024) + " MB"
print "nodes.csv: " + str(os.path.getsize("nodes.csv") / 1024 / 1024) + " MB"
print "nodes_tags.csv: " + str(os.path.getsize("nodes_tags.csv") / 1024 / 1024) + " MB"
print "ways.csv: " + str(os.path.getsize("ways.csv") / 1024 / 1024) + " MB"
print "ways_nodes.csv: " + str(os.path.getsize("ways_nodes.csv") / 1024 / 1024) + " MB"
print "ways_tags.csv: " + str(os.path.getsize("ways_tags.csv") / 1024 / 1024) + " MB"
print "vancouver_db.db: " + str(os.path.getsize("vancouver_db.db") / 1024 / 1024) + " MB"
```

```
vancouver.osm: 19 MB
nodes.csv: 7 MB
nodes_tags.csv: 0 MB
ways.csv: 1 MB
ways_nodes.csv: 2 MB
ways_tags.csv: 0 MB
vancouver_db.db: 10 MB
```

Number of unique users

In [22]:

```
result = pd.read_sql_query("""
SELECT COUNT(DISTINCT users.uid) AS num_of_unique_users
FROM (SELECT uid FROM Nodes UNION ALL SELECT uid FROM Ways) AS users;
""", disk_engine)
result
```

Out[22]:

	num_of_unique_users
0	472

Top 10 contributing users

In [23]:

```
result = pd.read_sql_query("""
SELECT users.user, COUNT(*) as num_of_contributions
FROM (SELECT user FROM Nodes UNION ALL SELECT user FROM Ways) users
GROUP BY users.user
ORDER BY num_of_contributions DESC
LIMIT 10;
""", disk_engine)
result
```

Out[23]:

	user	num_of_contributions
0	keithonearth	36574
1	michael_moovelmaps	11287
2	still-a-worm	9588
3	treeniti2	7480
4	keithonearth_imports	4438
5	pdunn	4143
6	muratc3	3697
7	WBSKI	3102
8	rbrtwhite	2606
9	Siegbaert	2135

In [24]:

```
result = pd.read_sql_query("""
SELECT COUNT(*) AS number_of_nodes FROM nodes;
""", disk_engine)
result
```

Out[24]:

	number_of_nodes
0	87742

Number of Ways

In [25]:

```
result = pd.read_sql_query("""
SELECT COUNT(*) AS number_of_ways FROM ways
""", disk_engine)
result
```

Out[25]:

	number_of_ways
0	16904

Most Popular Cuisine

In [26]:

```
result = pd.read_sql_query("""
SELECT nodes_tags.value AS cousine,
    COUNT(*) as num
FROM nodes_tags
    JOIN (SELECT DISTINCT id FROM nodes_tags WHERE value='restaurant') ids
    ON nodes_tags.id=ids.id
WHERE nodes_tags.key='cuisine'
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 10;
""", disk_engine)
result
```

Out[26]:

	cousine	num
0	japanese	6
1	chinese	5
2	vietnamese	5
3	italian	3
4	pizza	3
5	sushi	3
6	asian	2
7	Indian;vegetarian	1
8	Malaysian	1
9	fish	1

Most Popular Amenities

In [27]:

```
result = pd.read_sql_query("""
SELECT nodes_tags.value AS amenity,
    COUNT(*) as num
FROM nodes_tags
    JOIN (SELECT DISTINCT id FROM nodes_tags) ids
    ON nodes_tags.id=ids.id
WHERE nodes_tags.key='amenity'
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 10;
""", disk_engine)
result
```

Out[27]:

	amenity	num
0	bench	90
1	bicycle_parking	76
2	restaurant	76
3	cafe	47
4	fast_food	36
5	waste_basket	27
6	post_box	26
7	bicycle_rental	15
8	bank	13
9	toilets	11

Conclusion

With the overall dataset for Vancouver, BC, we did face an unusual issue where not all 'node' and 'way' elements have a 'user' and 'id' attribute. Also, the dataset did have some inconsistencies with the abbreviation for street names('Boundary Rd.' => Boundary Road}). However, we were able to update the dataset programmatically for the street names. Eventhough the dataset may have not been cleaned, we were amazed at how many people contributed to the project.

An interesting idea to improve the data analysis or dataset is to have a competition. For example, given a certain period of time the user that can produce the most accurate data be rewarded. With the right incentive, it can motivate users to contribute. A competition like this may require additional resources and may be a challenge to manage.

References

https://classroom.udacity.com/nanodegrees/nd002/parts/860b269a-d0b0-4f0c-8f3d-ab08865d43bf/modules/316820862075461/lessons/5436095827/concepts/54446302850923

http://wiki.openstreetmap.org/wiki/OSM_XML

https://gist.github.com/carlward/54ec1c91b62a5f911c42#file-sample_project-md

https://discussions.udacity.com/t/osm-data-project-getting-started-running-locally/232476

https://discussions.udacity.com/t/p3-project-combining-auditing-cleaning-and-csv-creation/231037

https://discussions.udacity.com/t/project-problem-cant-get-through-validate-element-el-validator/179544/28