Submission Document

February 5, 2018

1 Wrangling Data from OpenStreetMap for the Greater Salt Lake City Area

I have used Mapzen (https://mapzen.com/) to extract a dataset from OpenStreetMap. The area I have chosen is Greater Salt Lake City. The total uncompressed size of this file is 318 MB.

1.0.1 Cleaning the Data and Converting it to the CSV Format

In [6]:
```
import xml.etree.cElementTree as ET
from collections import defaultdict
import re
import pprint



street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE) # pulls out last word
                                                         # when tag attribute is "v"

expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place",
            "Square", "Lane", "Road", "Trail", "Parkway", "Commons", "Circle",
            "West", "East", "North", "South", "Temple", "Way", "Gateway", "Broadway"]

mapping = { "Rd.": "Road",
            "1300": "",
            "N": "North",
            "319": "",
            "A": ""
            }

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(SAMPLE_FILE):
        open(SAMPLE_FILE, "r")
        street_types = defaultdict(set)
        for event, elem in ET.iterparse(SAMPLE_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'])

        return street_types

    audit(SAMPLE_FILE)
```

```
Out[6]: defaultdict(set,
                    {'1300': {'1300'},
                     '319': {'S Sleepy Ridge Drive Suite 319'},
                     'A': {'S State St #A'},
                     'N': {'W 800 N'},
                     'Rd.': {'Portobello Rd.', 'West Portobello Rd.'}})
```

When I first ran this auditing script, I found a lot of addresses that ended with words that were not in the "expected" list but which are perfectly acceptable for Salt Lake City. These include addresses ending with "North,"South," "East," and "West." I also found some ending with "Circle" (and others), which are perfectly acceptable. Thus, I have added these words to the expected list above so that they will be ignored.

In my sample file, I am only left with one incorrect street name—one that ends in "Rd." instead of "Road."

```
In [7]: postal_type_re = re.compile('(?<!\d)\d{5}(?!\d)') # Looks for postal codes
                                                         # that are five digits.
        # Reference: https://stackoverflow.com/questions/3532947/python-regular-expression
        # -match-all-5-digit-numbers-but-none-larger

        def audit_postal_code(postal_types, postal_code):
            m = postal_type_re.search(postal_code) # Looks for postal codes that are #
                                                   # of digits and sets them equal to m.
            if m:                                  # If the postal code (m) is # digits,
                                                   # proceed.
                postal_type = m.group()            # Take the postal codes that are # digits,
                                                   # groups them, and sets them equal
                                                   # to postal_type.
                postal_types[postal_type].add(postal_code) # Take those postal codes that
                                                           # are # digits and adds them
                                                           # to the postal types dictionary.
```

```
        def audit1(SAMPLE_FILE):
            open(SAMPLE_FILE, "r")
            postal_types = defaultdict(set)
            for event, elem in ET.iterparse(SAMPLE_FILE, events=("start",)):
                if elem.tag == "node" or elem.tag == "way":
                    for tag in elem.iter("tag"):
                        if tag.attrib['k'] == 'addr:postcode':
                            audit_postal_code(postal_types, tag.attrib['v'])

            return postal_types

        audit1(SAMPLE_FILE)
```

```
Out[7]: defaultdict(set,
                    {'84005': {'84005'},
                     '84014': {'84014'},
                     '84020': {'84020'},
                     '84032': {'84032'},
                     '84043': {'84043'},
                     '84044': {'84044'},
                     '84047': {'84047'},
                     '84057': {'84057'},
                     '84058': {'84058'},
                     '84060': {'84060'},
                     '84061': {'84061'},
                     '84062': {'84062'},
                     '84081': {'84081'},
                     '84084': {'84084'},
                     '84087': {'84087'},
                     '84088': {'84088'},
                     '84095': {'84095'},
                     '84096': {'84096'},
                     '84097': {'84097'},
                     '84098': {'84098'},
                     '84101': {'84101'},
                     '84102': {'84102'},
                     '84103': {'84103'},
                     '84105': {'84105'},
                     '84106': {'84106'},
                     '84107': {'84107'},
                     '84108': {'84108'},
                     '84109': {'84109'},
                     '84113': {'84113'},
                     '84115': {'84115'},
                     '84116': {'84116'},
                     '84117': {'84117'},
                     '84118': {'84118'},
                     '84119': {'84119'},
```

```
                    '84120': {'84120'},
                    '84121': {'84121'},
                    '84124': {'84124'},
                    '84601': {'84601'},
                    '84604': {'84604'}})
```

Honestly, I played around with this code for hours to try and find an efficient way to determine if there were any zip codes in the data that were not five digits. While this code does this, it does not do it efficiently. One has to change the "5" in "postal_type_re = re.compile('(?<!)5(?!)')" to different numbers for this to work. I did this with digits 1-10 and found no addreses any number of digits other than 5 in the postcode field. As a result, no additional cleaning is necessary on thses fields.

1.1 2. Exploring the Database

```
In [21]: cur.execute('SELECT user, COUNT(*) as num FROM (SELECT user FROM nodes UNION ALL
                      SELECT user FROM ways) GROUP BY user ORDER BY num DESC LIMIT 20');
         all_rows = cur.fetchall()
         print('1):')
         pprint(all_rows)
```

```
1):
[(u'chadbunn', 9895),
 (u'osmjwh', 6360),
 (u'woodpeck_fixbot', 4288),
 (u'butlerm', 3495),
 (u'mash84121', 2866),
 (u'mvexel', 2747),
 (u'MelanieOriet', 2612),
 (u'Level', 2006),
 (u'carlotta4th', 1963),
 (u'OremSteve', 1735),
 (u'wrk3', 1614),
 (u'Ted Percival', 1387),
 (u'MasterOfKittens', 1305),
 (u'Val', 1259),
 (u'jackwiplock', 927),
 (u'TheDutchMan13', 903),
 (u'GaryOSM', 883),
 (u'nemmer', 799),
 (u'balcoath', 731),
 (u'bburgon42', 646)]
```

Here, I am counting the number of data entries each user contributed to the data in the OSM area I selected.

```
In [22]: cur.execute('SELECT value, COUNT(*) as num FROM nodes_tags WHERE key="amenity" GROUP
                      BY value ORDER BY num DESC LIMIT 10');
```

```
        all_rows = cur.fetchall()
        print('1):')
        pprint(all_rows)
```

```
1):
[(u'restaurant', 41),
 (u'place_of_worship', 32),
 (u'fast_food', 19),
 (u'fuel', 14),
 (u'parking', 14),
 (u'bank', 8),
 (u'bench', 8),
 (u'toilets', 8),
 (u'school', 6),
 (u'cafe', 5)]
```

Here, I am looking the number of each type of amenity found in the OSM area I selected.

```
In [23]: cur.execute('SELECT nodes_tags.value, COUNT(*) as num FROM nodes_tags WHERE
                    nodes_tags.key="religion" GROUP BY nodes_tags.value ORDER BY
                    num DESC LIMIT 5');
         all_rows = cur.fetchall()
         print('1):')
         pprint(all_rows)
```

```
1):
[(u'christian', 31)]
```

Here, I am looking at the number of places of worship by each religion type in the OSM area I selected.

```
In [24]: cur.execute('SELECT nodes_tags.value, COUNT(*) as num FROM nodes_tags WHERE
                    nodes_tags.key="denomination" GROUP BY nodes_tags.value ORDER BY
                    num DESC LIMIT 5');
         all_rows = cur.fetchall()
         print('1):')
         pprint(all_rows)
```

```
1):
[(u'mormon', 22),
 (u'latter_day_saints', 2),
 (u'catholic', 1),
 (u'jehovahs_witness', 1)]
```

Here, I am looking at the denominations represented in the places of worship in the OSM area I selected.

```
In [25]: cur.execute('SELECT nodes_tags.value, COUNT(*) as num FROM nodes_tags WHERE
                     nodes_tags.key="name" GROUP BY nodes_tags.value ORDER BY num
                     DESC LIMIT 5');
         all_rows = cur.fetchall()
         print('1):')
         pprint(all_rows)

1):
[(u'The Church of Jesus Christ of Latter-day Saints', 22),
 (u'7-11', 3),
 (u'7-Eleven', 3),
 (u"Arby's", 3),
 (u'Burger King', 2)]
```

Here, I was interested to see the names of the facilities of the places in my dataset.

```
In [26]: cur.execute('SELECT tags.value, COUNT(*) as xyz FROM (SELECT * FROM nodes_tags
                     UNION ALL SELECT * FROM ways_tags) tags WHERE tags.key="postcode"
                     GROUP BY tags.value ORDER BY xyz DESC LIMIT 10');
         all_rows = cur.fetchall()
         print('1):')
         pprint(all_rows)

1):
[(u'84105', 219),
 (u'84108', 127),
 (u'84106', 123),
 (u'84102', 56),
 (u'84116', 14),
 (u'84005', 11),
 (u'84096', 7),
 (u'84109', 7),
 (u'84084', 6),
 (u'84103', 5)]
```

And here, I wanted to see where most of the places in my OSM file were located. It looks like they are mainly concentrated around four zip codes.

1.2 3. Conclusions

I chose to look at every 25th data primitive in the OSM XML for Greater Salt Lake City because the full dataset crashed my computer when I tried to analyze it. This full file was .33 GB. The sample file was .013 GB.

Here are a few interesting things I found about the data:

- 794 unique users contributed data to the sample of the OSM XML I looked at
- A single user, "chadbunn" contributed 20% of entries of the top 20 contibutors (9895/48420).

- The most common amenity is a "restaurant"(41), which is followed by "places of worship" (32).
- The only religion found at these places of worship is Christianity.
- The denominations of these Christians are Mormon (24), Catholic (1), and Jehovas Witness (1). This seems reasonable considering Salt Lake City is the geographic center of the Mormon faith.
- In the area under investigation, most of the names of the places are "The Church of Jesus Christ of Latter-day Saints," and the rest are convienience stores and fast-food chains.
- There are no zip codes in the sample that seem to be erroneous; they all appear to be in areas within the OSM XML area that was extracted.

An area for further investigation would be to run an analysis on the entire data set and see how the religious makeup of the geographic area captured by the data in the OSM XLM aligns up with official statistics for religious affiliation for Salt Lake City and for Utah as a whole.

1.3 Possible Area of Improvement

In the nodes_tags SQL table, there are 3,708 data points, but there are only 33 named places out of all of them. I noticed that my company, Western Governors University, is not even named in OpenStreetMap. I think this is a deep flaw in OSM that hinders is usability. One way of filling in all of these gaps would be to create a very user-friendly OSM app with a clean, intuitive interface that people could download from their app store of choice. They could then use these apps and the GPS on their phones to add names for nodes, ways, and other data primitives in OSM. I beleive that their could be incentivized to do this by linking the app to social media where users would push the data about their update. Like Wikipedia, I think there is some intrinsic reason people would want to contribute to such a project in that it advances human knowledge. Also, their names could be immortalized in OSM if they are the first person to provide a name for a data point or are the first to update the data when it changes.

1.3.1 Benefits of Suggested Improvement

- The benefits of this suggested improvement would be that because updates could happen by more people, it would ensure the relability of the data. Users could review the updates of others and vote to accept or reject them based on their validity under a system similar to what Wikipedia does.
- Since more people would be providing data, updates to OSM would happen much faster, reflecting changes in the geopgraphy of human settlements as they happen.

1.3.2 Anticipated Problems about Suggested Improvement

- When we consider the number of amazingly fun apps that are on the market, one anticipated problem would be getting people to actually use the app. There may need to be an inventive system built into the system so that heavy users would get perks like being given free tickets to OSM conferences. It may also be even harder to get people to review other people;s contributions.
- The cost to develop an maintain the app may be prohibitive. Also, the funds to market the app may not be available.

```
In [ ]:
```