OpenStreetMap Project Final

September 8, 2017

0.1 Location –> Seattle

Though I live in Southern California, and have always lived here my entire life; I chose to look into Seattle because I had visited there once before years ago, and have always wanted to visit again to check out all the modern looking buildings and tech companies that reside there.
https://www.openstreetmap.org/relation/237385#map=11/47.6291/-122.2758
https://mapzen.com/data/metro-extracts/metro/seattle_washington/

```
In [1]: import sqlite3

        db = sqlite3.connect("osm.db")
        c = db.cursor()
```

0.2 Introduction

In the beginning I had explored the street types as a potential field for cleaning, but instead I had directed my attention to postal codes and cities because I had wanted to search results from these fields in my queries.

1 File Sizes

I copied these two function from a stackoverflow Q&A. https://stackoverflow.com/questions/2104080/how-to-check-file-size-in-python/19079887

```
In [2]: import os
        def convert_bytes(num):
            """
            this function will convert bytes to MB.... GB... etc
            """
            for x in ['bytes', 'KB', 'MB', 'GB', 'TB']:
                if num < 1024.0:
                    return "%3.1f %s" % (num, x)
                num /= 1024.0

        def file_size(file_path):
            """
            this function will return the file size
```

```
            """
            if os.path.isfile(file_path):
                file_info = os.stat(file_path)
                return convert_bytes(file_info.st_size)
```

```
In [3]: nodes_path = r"C:\Users\Chris\\nodes.csv"
        nodes_tags_path = r"C:\Users\Chris\\nodes_tags.csv"
        ways_path = r"C:\Users\Chris\\ways.csv"
        ways_tags_path = r"C:\Users\Chris\\ways_tags.csv"
        ways_nodes_path = r"C:\Users\Chris\\ways_nodes.csv"
        seattle_washington_osm_path = r"C:\Users\Chris\\seattle_washington.osm"
        seattle_washington_db_path = "C:\Users\Chris\\osm.db"
        print "nodes.csv ..........", file_size(nodes_path)
        print "nodes_tags.csv ..........", file_size(nodes_tags_path)
        print "ways.csv ..........", file_size(ways_path)
        print "ways_tags.csv ..........", file_size(ways_tags_path)
        print "ways_nodes.csv ..........", file_size(ways_nodes_path)
        print "seattle_washington.osm ..........", file_size(seattle_washington_osm
        print "osm.db ..........", file_size(seattle_washington_db_path)
```

```
nodes.csv ... 648.6 MB
nodes_tags.csv ... 41.1 MB
ways.csv ... 46.2 MB
ways_tags.csv ... 119.2 MB
ways_nodes.csv ... 207.8 MB
seattle_washington.osm ... 1.6 GB
osm.db ... 962.3 MB
```

2 Problems encountered

2.1 Problems with Postal Code data:

First I had audited postal codes and had found after listing the different kinds of postal codes that there were some postal codes that were undesirable. Listed below were the different kinds of postal codes that would come up in the data:

```
98528
Lacey, 98513
V9B2S3
186631629:18700775:186700777
Lacey, WA 98503
98004-4452
985370525
89370
Port Angeles
```

As you can see the postal codes ranged from a wrong postal code, to British Columbia postal code (V9…), to postal codes with cities attached with them, and others. I wanted a postal code

that had the form of '98...', and anything that wasn't in this form I would either have to rule out or trim it. To accomplish this I used a regular expression from the python standard library.

2.2 Problems with Cities/City names data:

Next came cleaning the city names. After listing out the cities that I would be dealing with, I had noticed that there was a good amount of city data that pointed to the British Columbia, rather than a city inside Washington state. I did not have a gold standard of accurate data listing all the Seattle cities and running this against the cities that I would encounter in the data, therefore I went for a more simpler approach. I had categorized cities as either 'matched' or 'non-matched'. 'Matched' meant that the city was associated with a postal code tag within the node/way element. 'Non-matched' meant that the city was not associated with a postal code tag within the node/way element. I chose this approach because it was easier to rule out a city if it was associated with a postal code, since I didn't have a gold standard to test whether a postal code belonged or did not. I want to note that every city in the 'matched' category that had a valid postal code associated with it was included into the cleaned csv files.

Although most cities in the non-matched category belonged to the British Columbia, there were some cities that belonged to Washington, such as the following below:

```
Silverdale
South Hill
Kenmore
Silvana
Oting
Allyn
Granite Fall
kirkland, wa
redmond
```

but were not included in the cleaned csv files. In actuality 'redmond' from the non-matched category is included in the 'matched' category but as 'Redmond' (with a capital 'R'). Specifically, 'redmond' had 1 instance in the dataset, and 'Redmond' had 623 instances in the data. Other cities in the non-matched category with another name of itself listed in the matched category also had similar proportions. Due to such insignificant proportions I had make the executive decision to not include these cities. Some cities in the non-matched category did not have a version of itself in the matched category, but had so few instances of that city showing up in the data that I thought it was insignificant for me to include them. The only huge loss was the throwing out of city 'South Hill', which had 127 occurances in the data, but I had decided to not include it with the matched category, because when I did queries to find the top 10 number of cities in Washington, I knew 'South Hill' wouldn't show up due to the large number of other city occurances. Therefore, every city listed in the non-matched category I did not include in my cleaning process.

Next came cleaning the cities within the matched category. After keeping every city that had a valid postal code, I then made sure cities that had duplicate names but written differently or attached with something, like 'Kirkland' and 'Kirkland, WA', would have one standard way of being written. To accomplish this I had split the city string whenever there was a comma and also capitalized the first letter of the city name (to account for cases like 'kirkland' and 'Kirkland'). Standardizing the city names in this way made the querying easier in counting cities.

2.3 Other Problems:

In addition to auditing and cleaning the postal codes and cities, I also audited nodes_id, ways_id, and ways_ref. I wanted to make sure that these id's and ref's was not null and had integer data. I also explored 'user' and 'uid' data. Some node/way elements either had missing user and uid data or had special characters in their user data. I kept the cases where the special characters were in the user data, but include 'n/a' in those with missing user and uid data. They seemed to come in pairs. The input of 'n/a' to missing data was a small fix so I didn't need to create a separate function for it, but added it to the main function.

3 Overview of the Data

These queries were copied from the SQL Sample Project that was given to us as an example for this project, because the queries that he carried out were the queries that I was interested in doing as well.

3.1 Number of Nodes:

```
In [4]: query = "SELECT COUNT(*) FROM nodes"
        c.execute(query)
        rows = c.fetchall()
        for row in rows:
            print row[0]

7889253
```

3.2 Number of Ways:

```
In [5]: query = 'SELECT COUNT(*) FROM ways;'
        c.execute(query)
        rows = c.fetchall()
        for row in rows:
            print row[0]

783487
```

3.3 Number of Unique Users:

```
In [6]: query = 'SELECT COUNT(DISTINCT(e.uid)) FROM (SELECT uid FROM nodes UNION AI
        c.execute(query)
        rows = c.fetchall()
        for row in rows:
            print row[0]

3528
```

3.4 Top 10 Postal Codes:

```
In [7]: query = "SELECT tags.value, COUNT(*) as count \
        FROM (SELECT * FROM nodes_tags \
                  UNION ALL \
              SELECT * FROM ways_tags) tags \
        WHERE tags.key='postcode' \
        GROUP BY tags.value \
        ORDER BY count DESC \
        LIMIT 10;"
        c.execute(query)
        rows = c.fetchall()

        for row in rows:
            print row[0], "|", row[1]
```

```
98034 | 22955
98033 | 19368
98115 | 18117
98103 | 16859
98118 | 14410
98117 | 13566
98125 | 12135
98105 | 9935
98144 | 9290
98108 | 9257
```

The first two postal codes belong to Kirkland, and the rest to Seattle. But shouldn't Seattle be number one in postal codes? Not really. The reasoning for this is because Kirkland has only three postal codes associated with it, and Seattle has upwards of 50 postal codes associated with it. Knowing this, we can understand that land area wise Kirkland will have bigger chunks than does Seattle. But as a whole, Seattle is bigger than Kirkland.

3.5 Top 10 Cities:

```
In [8]: query = "SELECT tags.value, COUNT(*) as count \
        FROM (SELECT * FROM nodes_tags UNION ALL \
              SELECT * FROM ways_tags) tags \
        WHERE tags.key='city' \
        GROUP BY tags.value \
        ORDER BY count DESC \
        LIMIT 10;"

        c.execute(query)
        rows = c.fetchall()
        for row in rows:
            print row[0], "--->", row[1]
```

```
Seattle ---> 203298
Kirkland ---> 42282
Mount vernon ---> 11744
Mill creek ---> 941
Kingston ---> 603
Tacoma ---> 559
Rochester ---> 526
Puyallup ---> 443
Gig harbor ---> 263
Woodinville ---> 258
```

As expected, Seattle has the most occurances in data about cities.

3.6 Top 10 Appearing Amenities:

```
In [9]: query = 'SELECT value, COUNT(*) as num \
        FROM nodes_tags \
        WHERE key="amenity" \
        GROUP BY value ORDER BY num DESC \
        LIMIT 10;'

        c.execute(query)
        rows = c.fetchall()
        for row in rows:
            print row[0], '--->', row[1]
```

```
bicycle_parking ---> 3375
bench ---> 3180
restaurant ---> 2844
waste_basket ---> 1454
cafe ---> 1395
fast_food ---> 1208
school ---> 856
parking ---> 803
toilets ---> 751
place_of_worship ---> 738
```

Seems like a lot of the seattleites like to ride their bicycles to work, maybe because of the really bad traffic. I also wouldn't expect nothing less that there's a lot of cafes, afterall Seattle is known for their coffee.

3.7 Most Prevalent Keys

```
In [10]: query = 'SELECT tags.key, COUNT(tags.key) as num \
                 FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags)
                 tags \
                 GROUP BY tags.key \
```

```
                ORDER BY num DESC \
                LIMIT 10;'
        c.execute(query)
        rows = c.fetchall()
        for row in rows:
            print row[0], '--->', row[1]
```

```
source ---> 503440
highway ---> 357078
building ---> 334589
housenumber ---> 312480
street ---> 312309
postcode ---> 268593
city ---> 265165
name ---> 226345
county ---> 136534
cfcc ---> 135955
```

Apart from source, we see that the number one most occuring key is 'highway'. Previously we saw that the number one appearing amenity was 'bicycle parking'. The large amount of bicycle parking affirms my common knowledge that Seattle has bad traffic. More advanced statistics would be useful in finding out how correlated bicycle parking and highways are.

There is an almost equal amount of building as there are 'highways'. I'd be interesting to find whether these buildings are dispersed around seattle and surrounding cities, concentrated in Seattle, or there's mostly just big buildings in Seattle and not many small buildings.

3.8 Most Prevalent Values

```
In [11]: query = 'SELECT tags.value, COUNT(tags.value) as num \
                FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags)
                tags \
                GROUP BY tags.value \
                ORDER BY num DESC \
                LIMIT 10;'
        c.execute(query)
        rows = c.fetchall()
        for row in rows:
            print row[0], '--->', row[1]
```

```
yes ---> 362936
Seattle ---> 203391
King County GIS;data.seattle.gov ---> 182912
no ---> 145049
residential ---> 144510
A41 ---> 126120
service ---> 85960
JOSM ---> 62443
data.seattle.gov ---> 56780
```

```
King, WA ---> 49213
```

4 Other Ideas about the Dataset

4.1 Which ways_id has the most node_id's in it?:

```
In [12]: query = 'SELECT id, COUNT(id) as num \
         FROM ways_nodes \
         GROUP BY id \
         ORDER BY num DESC \
         LIMIT 10;'

         c.execute(query)
         rows = c.fetchall()
         print('Ways_Id ---> # of nodes associated with this id:')
         for row in rows:
             print row[0], "--->", row[1]
```

```
Ways_Id ---> # of nodes associated with this id:
162337410 ---> 1731
507973144 ---> 1729
409051604 ---> 1676
37478893 ---> 1675
37468935 ---> 1667
508075899 ---> 1643
509640867 ---> 1604
408583046 ---> 1523
37298207 ---> 1507
409051603 ---> 1506
```

I checked ways_id '162337410' (the way_id with the greatest number of nodes) in ways_tags to see if there were any tags associated with it. It turns out there were none. So I couldn't really get any name as to what this way is describing. Further exploration will be needed as to why this way is highly populated with nodes. And to answer the question as to whether this is a popular spot for tourists or locals to visit?

5 How to Further Improve the Data

5.1 Additional Suggestions of Improvement:

As suggested by my first reviewer, I'd be interesting to explore how the popular game, Pokemon Go, can improve the dataset of Seattle, Washington. My first impression is that it offers improvements to building descriptions. The openstreetmap data categorizes buildings as keys and its values consist of types of buildings (apartments, hotels, house, etc.). Pokemon Go can provide names of popular landmarks such as Seattle's 'Space Needle', or its 'Pike Place Market'. I'd assume that these popular destinations/attractions that are selectively chosen by Pokemon Go, would also be

seen as such by the general consensus of non-Pokemon Go players. The drawback of including Pokemon Go data to supplement building descriptions, is that the process isn't comprehensive, meaning that not all buildings are seen as landmarks, so some buildings wouldn't have this additional description added. We would need another way to include supplemental data to these forgotten buildings.

Integrating data from Yelp to the Seattle, WA dataset is another useful improvement to the amenity:(restaurant/fast food) portion of the data. Integrating the data allows us to gain some more insights into the food culture of Seattle. It'd be interesting to see through the added restaurant descriptions whether Seattle, WA is partitioned into culture specific areas, as if the case for Los Angeles. One anticipated problem I see with integrating Yelp's data into Seattle, WA is how the process of integration. With the constant changes in ownership and shutting down of businesses, we would need to create a system that integrates the data in real time.

6 Conclusion

Just from the various queries I've performed we can get a general sense of the culture/landscape of Seattle and its surrounding cities. To paint a very brief picture, things that characterize Seattle and its surrounding cities are buildings, highways, bicycles, and coffee. Of course this doesn't do much justice to the city, because most major cities have most if not all of these. To draw more speculative/definitive insights I would have to search for specifics like what types of buildings are in Seattle, and which companies have the largest buildings in Seattle. As we know from job market data accessed through the internet, Seattle is known for its Tech Companies like Microsoft and Amazon.