WRANGLE OPENSTREETMAP DATA

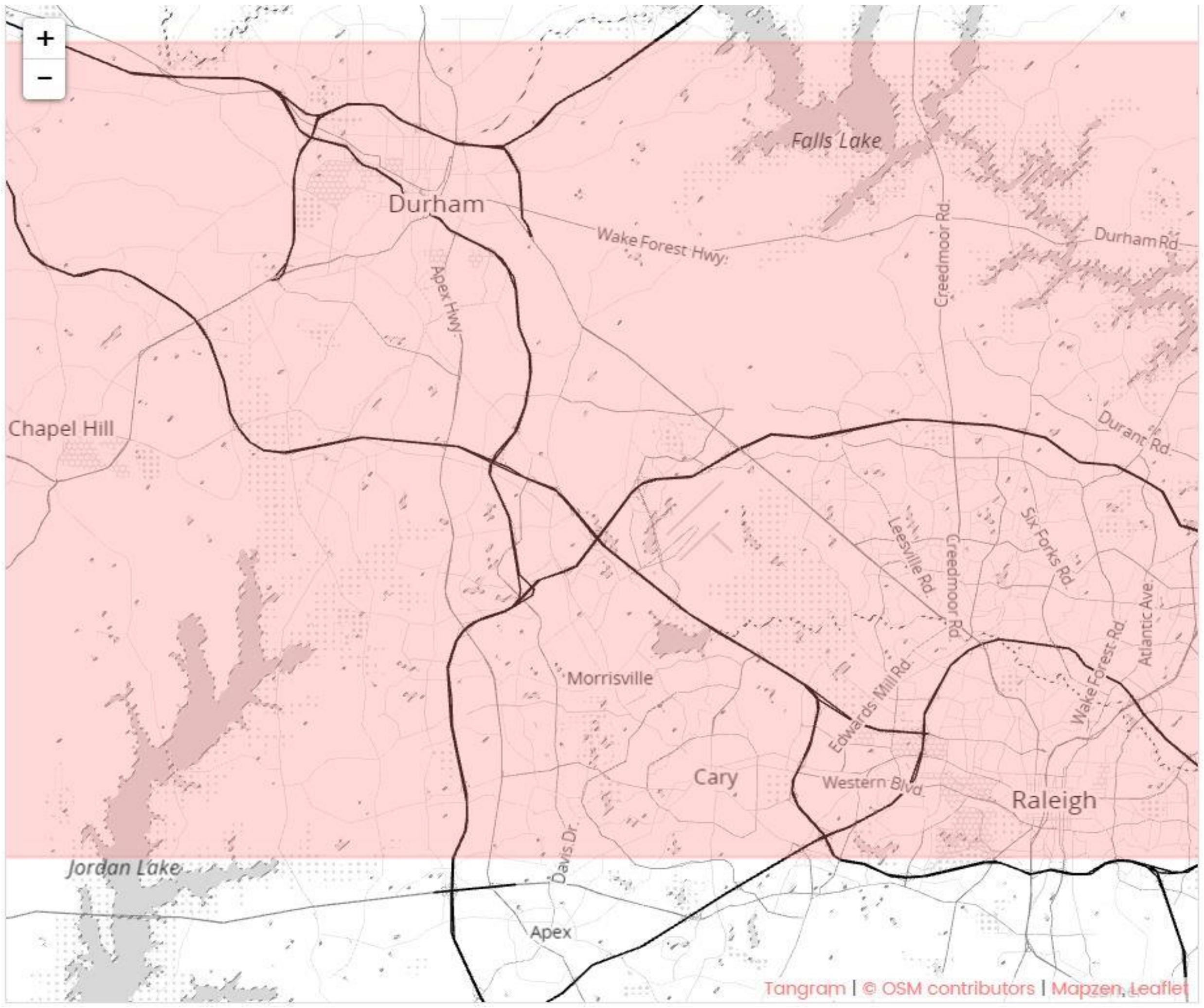
Introduction

The goal of this project was to use data munging techniques, such as assessing the quality of the data for validity, accuracy, completeness, consistency and uniformity, to clean the OpenStreetMap data for Raleigh, North Carolina. Raleigh was chosen as the location for this project since it was the first city I had visited for my first ever holidays in the USA.

Objectives

- Assess the quality of the data for validity, accuracy, completeness, consistency and uniformity.
- Parsing and gathering data from popular file formats such as. json, .xml, .csv, .html.
- Process data from multiple files or very large files that can be cleaned programmatically.
- Learn how to store, query and aggregate data using SQL.

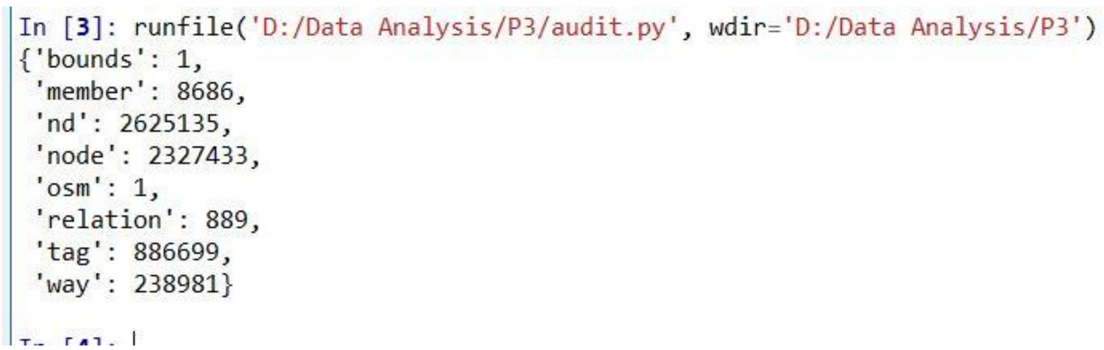
The following area, highlighted in red was chosen from the OpenStreetMap website to work with



The full dataset for the project can be found here.

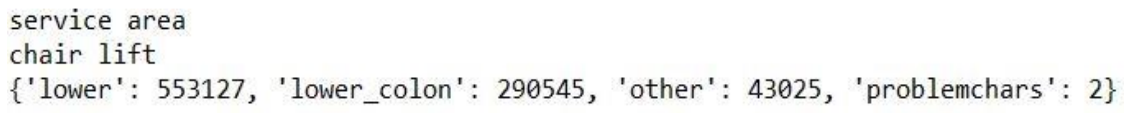
Auditing the Tag Types

To inspect the data, I had to find out what type of tags existed in the data and the *audit.py* code was used to iteratively parse through the file and return the tag type along with its count.



```
In [3]: runfile('D:/Data Analysis/P3/audit.py', wdir='D:/Data Analysis/P3')
{'bounds': 1,
 'member': 8686,
 'nd': 2625135,
 'node': 2327433,
 'osm': 1,
 'relation': 889,
 'tag': 886699,
 'way': 238981}
```

Also, three regular expressions were used to look at the different 'k' tags present in the dataset before importing the data into the dataset. The first expression looks for tags with only lowercase letters, the second one for tags with lowercase letters separated by a colon and the third one flags for any unwanted characters. The iterparse method of ElementTree was then used to compile a list of tags that fell into one of the three regular expression matches. The code for this can be found in the *tagTypes.py* file.

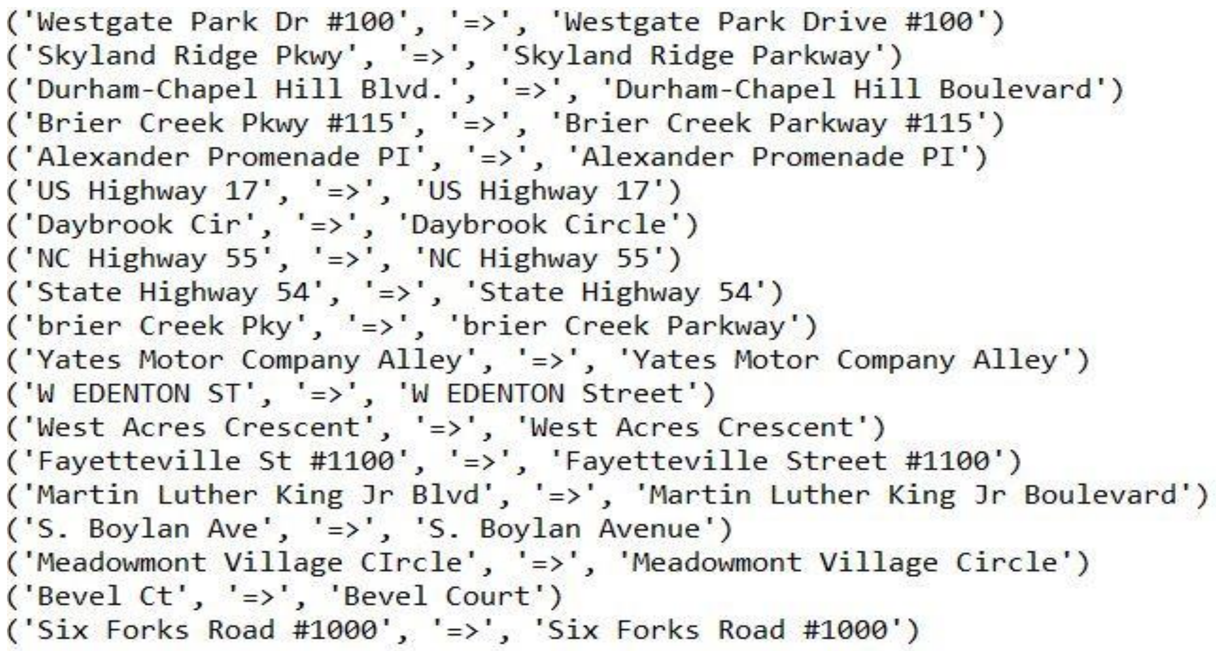


```
service area
chair lift
{'lower': 553127, 'lower_colon': 290545, 'other': 43025, 'problemchars': 2}
```

The *uniqueUsers.py* code was used to find out the different number of contributors to this area. It returns a result of 963 which is a high number and shows that the area chosen was significantly populated.

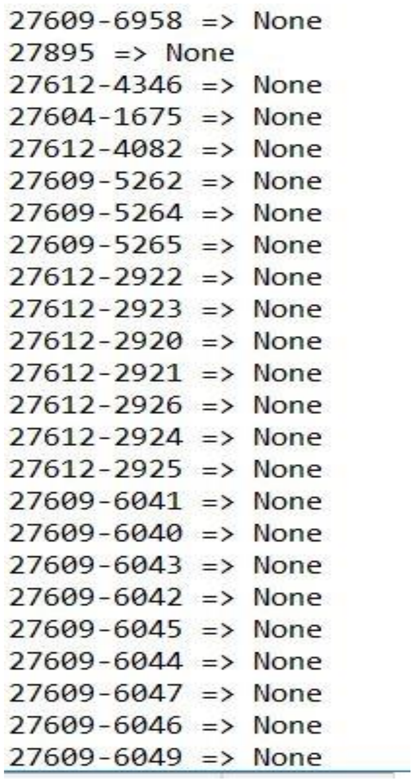
Problems Encountered

The main problem encountered was the street name abbreviations used throughout the dataset. To clean this type of data, a list of properly formatted words called 'expected' is created. Whenever the address tag words are not in this format, they will be substituted by words defined in 'mapping' list accordingly. (Only the last words of the tags are replaced) Functions for these operations are defined in the *improvingStreetNames.py*



```
('Westgate Park Dr #100', '=>', 'Westgate Park Drive #100')
('Skyland Ridge Pkwy', '=>', 'Skyland Ridge Parkway')
('Durham-Chapel Hill Blvd.', '=>', 'Durham-Chapel Hill Boulevard')
('Brier Creek Pkwy #115', '=>', 'Brier Creek Parkway #115')
('Alexander Promenade PI', '=>', 'Alexander Promenade PI')
('US Highway 17', '=>', 'US Highway 17')
('Daybrook Cir', '=>', 'Daybrook Circle')
('NC Highway 55', '=>', 'NC Highway 55')
('State Highway 54', '=>', 'State Highway 54')
('brier Creek Pky', '=>', 'brier Creek Parkway')
('Yates Motor Company Alley', '=>', 'Yates Motor Company Alley')
('W EDENTON ST', '=>', 'W EDENTON Street')
('West Acres Crescent', '=>', 'West Acres Crescent')
('Fayetteville St #1100', '=>', 'Fayetteville Street #1100')
('Martin Luther King Jr Blvd', '=>', 'Martin Luther King Jr Boulevard')
('S. Boylan Ave', '=>', 'S. Boylan Avenue')
('Meadowmont Village CIrcle', '=>', 'Meadowmont Village Circle')
('Bevel Ct', '=>', 'Bevel Court')
('Six Forks Road #1000', '=>', 'Six Forks Road #1000')
```

The other expected problem while auditing the dataset was the inconsistency among the format for the zip codes. Although there is some inconsistency in the formats for the zip codes, most of the zip codes are well formatted. They were found in the standard 5 digit and 9 digit formats. A postal code of the format NC-XXXXX was expected but not found in the dataset. While cleaning the data though, all the postal codes were kept in the standard 5-digit format. The code for auditing the postal codes can be found in the *improvpostalcodev2.py.*

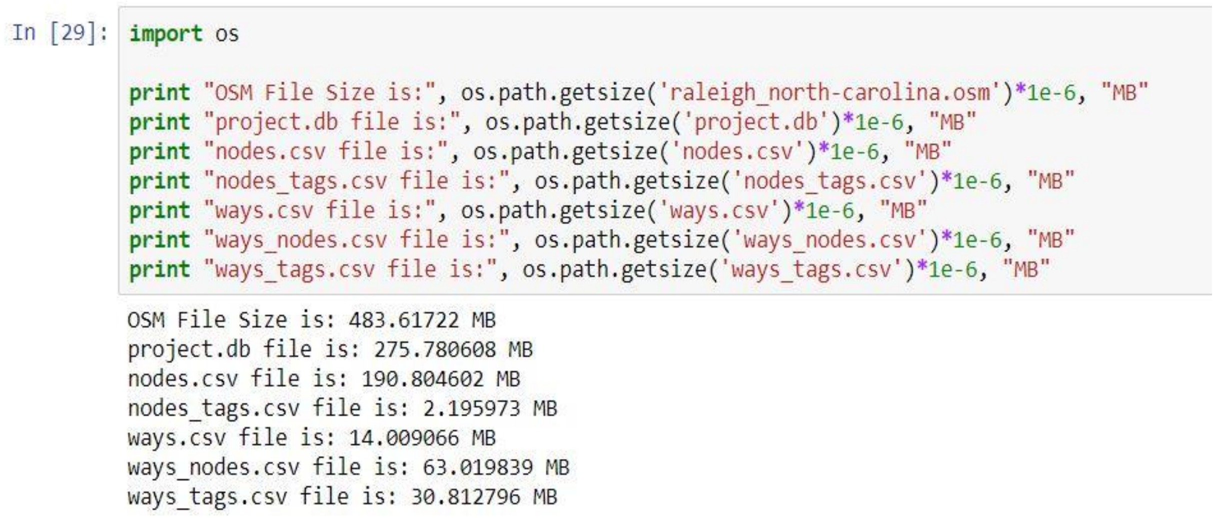


```
27609-6958 => None
27895 => None
27612-4346 => None
27604-1675 => None
27612-4082 => None
27609-5262 => None
27609-5264 => None
27609-5265 => None
27612-2922 => None
27612-2923 => None
27612-2920 => None
27612-2921 => None
27612-2926 => None
27612-2924 => None
27612-2925 => None
27609-6041 => None
27609-6040 => None
27609-6043 => None
27609-6042 => None
27609-6045 => None
27609-6044 => None
27609-6047 => None
27609-6046 => None
27609-6049 => None
```

The *data.py* file deals with the above-mentioned formatting problems and helps us create the necessary .csv files for analysis. All the data auditing functions are included and the names of the .csv files and their fields are defined. The 'shape_element' function helps us in parsing the data is also defined. The code check whether the tag is a 'node' or a 'way' and depending on the format of the tag, 'problemchar' or 'lower_colon' saves them separately. Finally, these saved tags are written in the earlier defined .csv files. The information needed for the .csv files to be created can be found in the file *schema.py*.

Overview of The Data

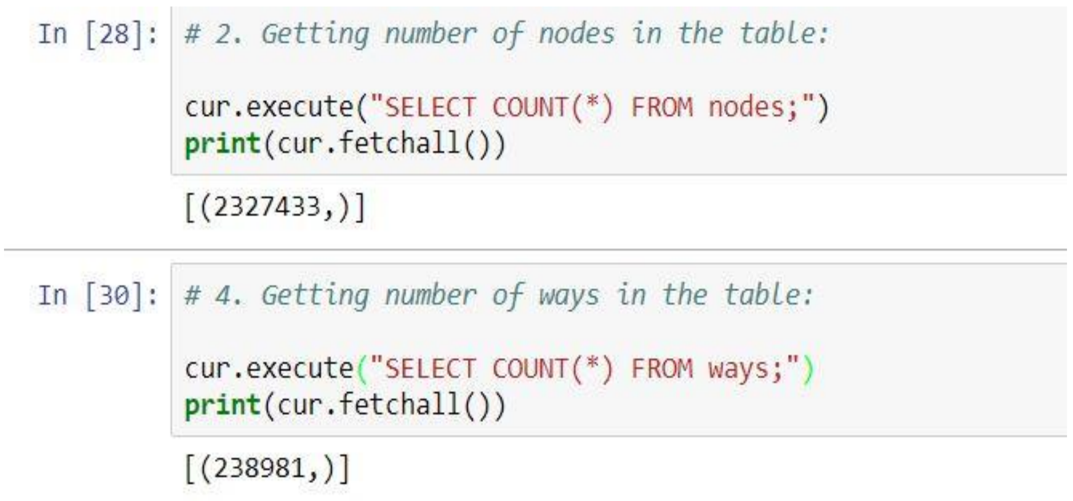
SQL queries were then used to get the statistical information out of the data. We start by looking at size of the files created.



```
In [29]: import os

         print "OSM File Size is:", os.path.getsize('raleigh_north-carolina.osm')*1e-6, "MB"
         print "project.db file is:", os.path.getsize('project.db')*1e-6, "MB"
         print "nodes.csv file is:", os.path.getsize('nodes.csv')*1e-6, "MB"
         print "nodes_tags.csv file is:", os.path.getsize('nodes_tags.csv')*1e-6, "MB"
         print "ways.csv file is:", os.path.getsize('ways.csv')*1e-6, "MB"
         print "ways_nodes.csv file is:", os.path.getsize('ways_nodes.csv')*1e-6, "MB"
         print "ways_tags.csv file is:", os.path.getsize('ways_tags.csv')*1e-6, "MB"

         OSM File Size is: 483.61722 MB
         project.db file is: 275.780608 MB
         nodes.csv file is: 190.804602 MB
         nodes_tags.csv file is: 2.195973 MB
         ways.csv file is: 14.009066 MB
         ways_nodes.csv file is: 63.019839 MB
         ways_tags.csv file is: 30.812796 MB
```

The. osm file size is well above the required 50 MB while the other files are comparatively smaller in size.

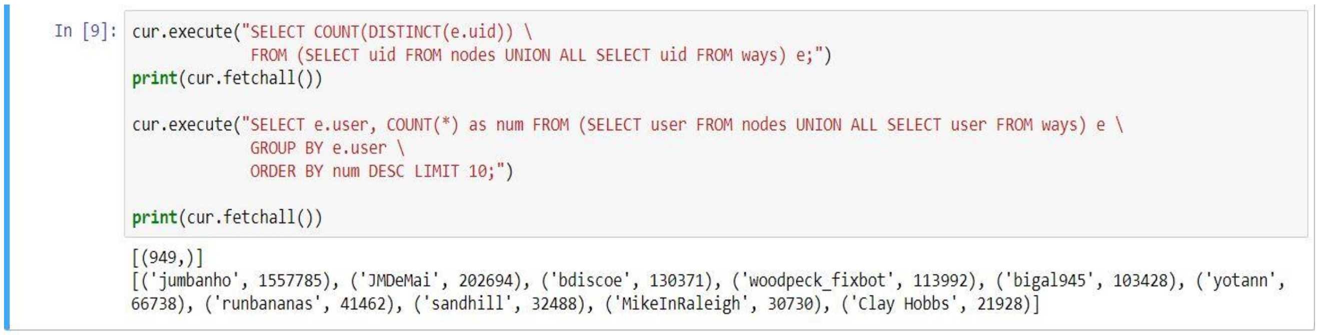
Let us start our querying by looking at the number of nodes and ways present in the dataset. All the tables required for this query are created in the 'project.db' file. Here are the results:



```
In [28]: # 2. Getting number of nodes in the table:

         cur.execute("SELECT COUNT(*) FROM nodes;")
         print(cur.fetchall())

         [(2327433,)]

In [30]: # 4. Getting number of ways in the table:

         cur.execute("SELECT COUNT(*) FROM ways;")
         print(cur.fetchall())

         [(238981,)]
```

We can see that the number of nodes is significantly higher than the number of ways.

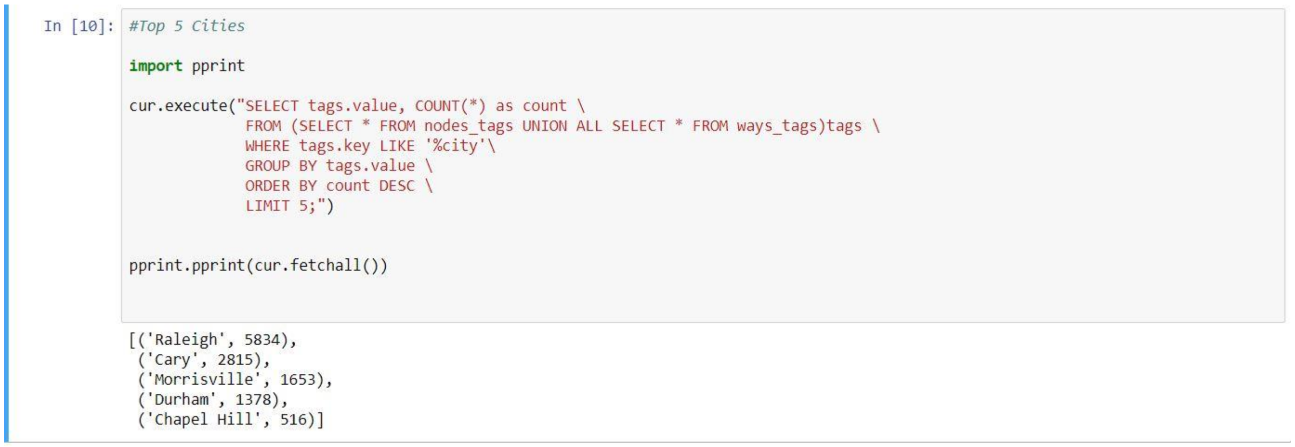
Looking at the number of unique users, I thought it would be interesting to find out how many users contributed to this dataset and who is the most common contributor of them all.



```
In [9]: cur.execute("SELECT COUNT(DISTINCT(e.uid)) \
                     FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e;")
        print(cur.fetchall())

        cur.execute("SELECT e.user, COUNT(*) as num FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e \
                     GROUP BY e.user \
                     ORDER BY num DESC LIMIT 10;")

        print(cur.fetchall())

        [(949,)]
        [('jumbanho', 1557785), ('JMDeMai', 202694), ('bdiscoe', 130371), ('woodpeck_fixbot', 113992), ('bigal945', 103428), ('yotann',
        66738), ('runbananas', 41462), ('sandhill', 32488), ('MikeInRaleigh', 30730), ('Clay Hobbs', 21928)]
```

There are 949 users who contributed to this dataset with ‘jumbanho’ being the most common contributor by a huge margin.

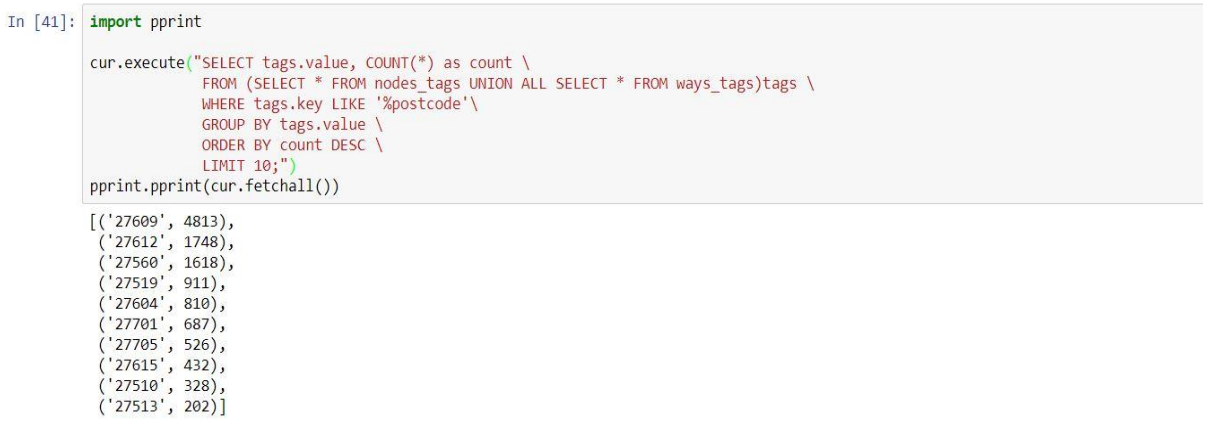
Since the map also included the surrounding cities near Raleigh, I thought it would be interesting to know the top 5 cities present in our dataset.



```
In [10]: #Top 5 Cities

         import pprint

         cur.execute("SELECT tags.value, COUNT(*) as count \
                      FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags)tags \
                      WHERE tags.key LIKE '%city'\
                      GROUP BY tags.value \
                      ORDER BY count DESC \
                      LIMIT 5;")


         pprint.pprint(cur.fetchall())

         [('Raleigh', 5834),
          ('Cary', 2815),
          ('Morrisville', 1653),
          ('Durham', 1378),
          ('Chapel Hill', 516)]
```

The only thing I found surprising about this result was the fact that Cary had more count than both Durham and Chapel Hill combined. Both Durham and Chapel Hill are homes to large universities and I assumed the number of incoming students helped their count.

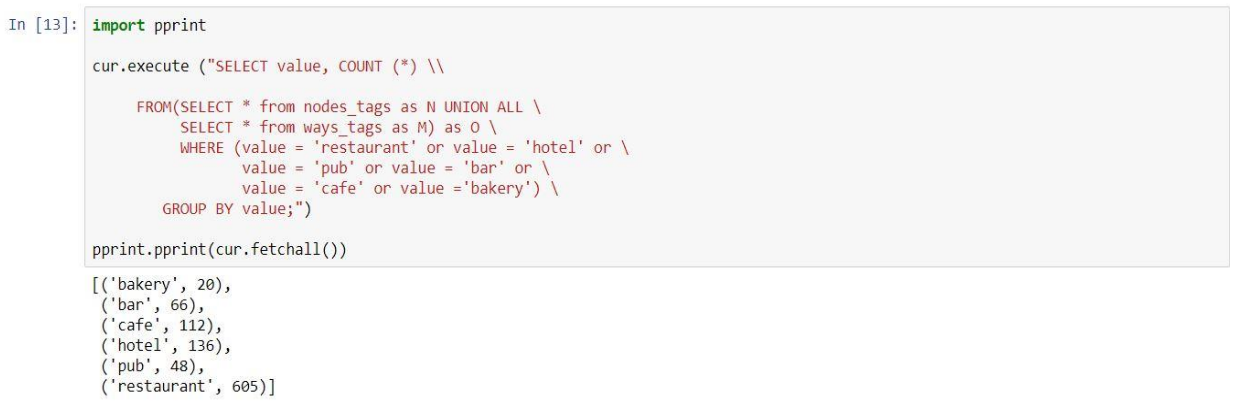
I also wanted to look at the most densely populated zip codes for the area chosen.



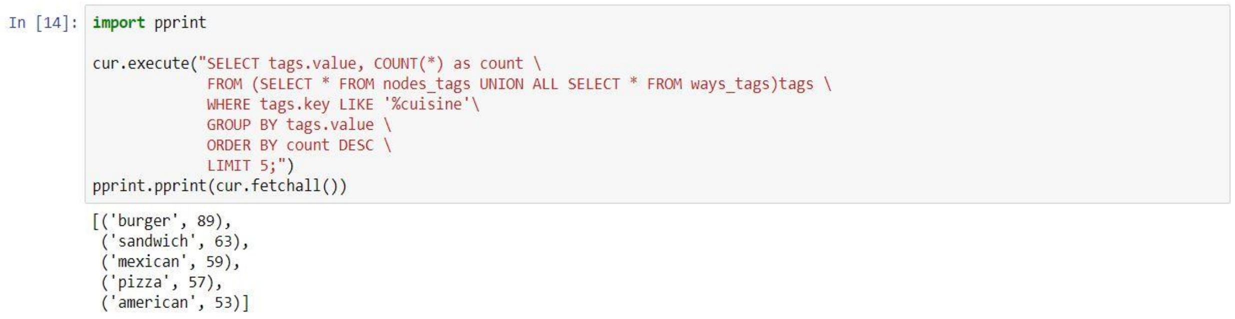
```
In [41]: import pprint

         cur.execute("SELECT tags.value, COUNT(*) as count \
                      FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags)tags \
                      WHERE tags.key LIKE '%postcode'\
                      GROUP BY tags.value \
                      ORDER BY count DESC \
                      LIMIT 10;")
         pprint.pprint(cur.fetchall())

         [('27609', 4813),
          ('27612', 1748),
          ('27560', 1618),
          ('27519', 911),
          ('27604', 810),
          ('27701', 687),
          ('27705', 526),
          ('27615', 432),
          ('27510', 328),
          ('27513', 202)]
```

It was surprising to see that ‘27609’ had almost twice the number of streets than the next postal code.

To further explore the map area, I decided to explore the various options one had the option to dine at. Firstly, I looked at the number of restaurants, hotels, pub, bar, café and bakeries available in the map area. For further analysis, I also looked at the preferred cuisine of the map area selected. The results of first query showed that the number of restaurants far outweighed the other places by a large margin. The result of the second query returned an interesting find in the sense that it considered ‘burgers’, ‘sandwich’ and ‘pizza’ as a separate cuisine. I didn’t know how to interpret this finding. However, per the result, the most preferred food of the region was burgers and sandwiches while the most preferred cuisine was Mexican.



```
In [13]: import pprint

         cur.execute ("SELECT value, COUNT (*) \\

              FROM(SELECT * from nodes_tags as N UNION ALL \
                   SELECT * from ways_tags as M) as O \
                   WHERE (value = 'restaurant' or value = 'hotel' or \
                          value = 'pub' or value = 'bar' or \
                          value = 'cafe' or value ='bakery') \
                 GROUP BY value;")

         pprint.pprint(cur.fetchall())

         [('bakery', 20),
          ('bar', 66),
          ('cafe', 112),
          ('hotel', 136),
          ('pub', 48),
          ('restaurant', 605)]
```



```
In [14]: import pprint

         cur.execute("SELECT tags.value, COUNT(*) as count \
                     FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags)tags \
                     WHERE tags.key LIKE '%cuisine'\
                     GROUP BY tags.value \
                     ORDER BY count DESC \
                     LIMIT 5;")
         pprint.pprint(cur.fetchall())

         [('burger', 89),
          ('sandwich', 63),
          ('mexican', 59),
          ('pizza', 57),
          ('american', 53)]
```

<u>Additional Ideas</u>

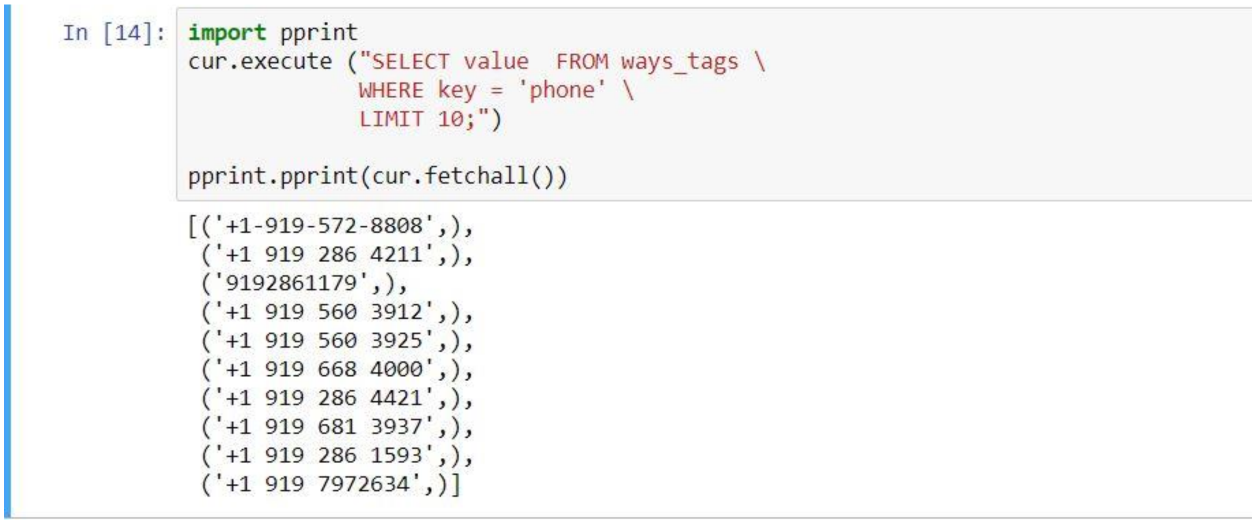
To further enhance the quality of the OpenStreetMap data, we could consider standardizing the information that is included with the node tags for places like restaurants, pubs, cafés and hotels. Information such as operational hours, cuisine, website and restaurant name could be included to further improve the consumer experience.

The dataset also contains the phone numbers used in the area. Upon querying the dataset for the different phone numbers available, I noticed that they were not in the same format. Some of the formats were:

1. +1-xxx-xxx-xxxx

2. +1 xxx xxx xxxx

3. xxxxxxxxxx



```
In [14]: import pprint
         cur.execute ("SELECT value  FROM ways_tags \
                      WHERE key = 'phone' \
                      LIMIT 10;")

         pprint.pprint(cur.fetchall())

         [('+1-919-572-8808',),
          ('+1 919 286 4211',),
          ('9192861179',),
          ('+1 919 560 3912',),
          ('+1 919 560 3925',),
          ('+1 919 668 4000',),
          ('+1 919 286 4421',),
          ('+1 919 681 3937',),
          ('+1 919 286 1593',),
          ('+1 919 7972634',)]
```

A similar approach to what was used for auditing and cleaning the postal codes could be used here. All the digits can be extracted, leaving additional characters like '- ','+1' and then replacing the original with the 10-digit phone number.

Conclusion

It is clear that this data is not 100 % clean although it is sufficiently cleaned for completing this project through SQL querying. Looking at the most common contributors, we see that the top 10 contributors almost make up 80 % of the contributions. To avoid bias and make more meaningful additions to the data, contributions should come from more number of users. To avoid the different phone number formats problem, the data could be taken in from apps like 'True Caller', so that the data is received in a standard format. While Raleigh is not famous as a tourist location, whatever data received from the arriving tourists can bring variety to the data already available. This can be achieved through utilizing the different hotel reservation and trip booking websites such as 'Expedia'.

More popular tools like Google Maps can also be used to keep the information updated. The main issue while dealing with using data from tools like Google Maps and other tourist websites will be parsing the data and converting it to a more standard format.

References

1.Udacity Data Wrangling Course (Case Study)

2. http://www.openstreetmap.org/#map=11/35.8401/-78.6449