

# OpenStreetMap Data Case Study

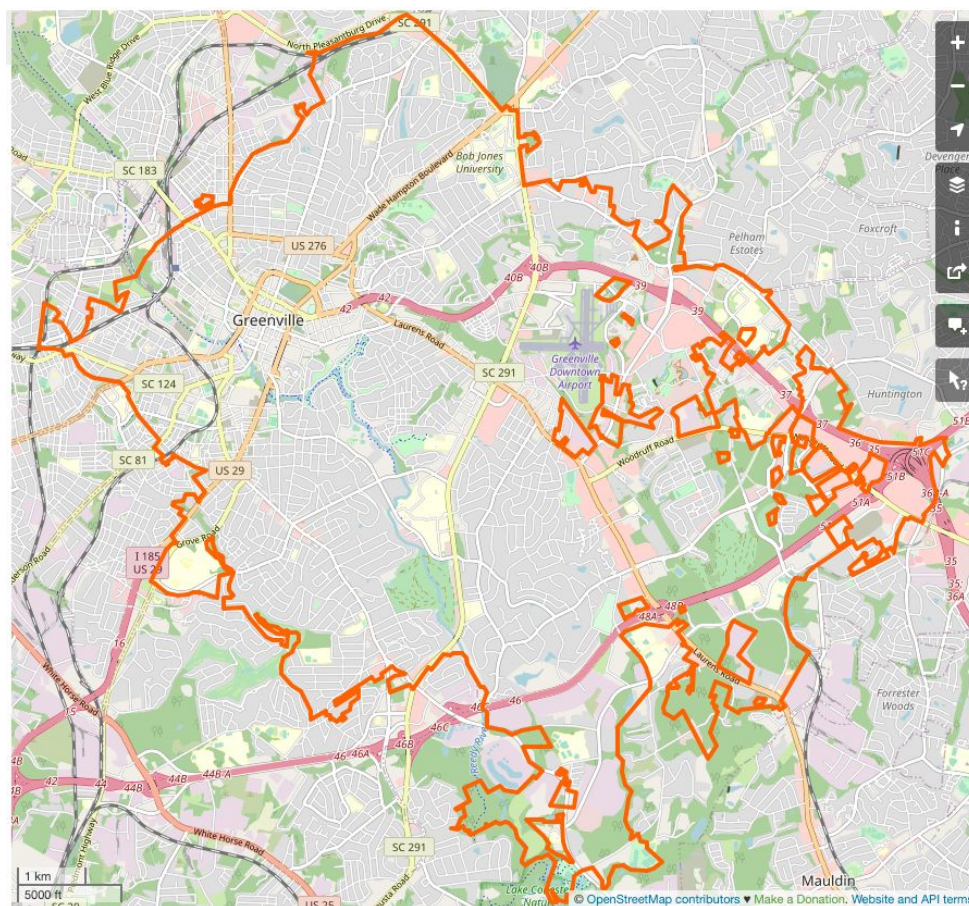
## Map Area

Greenville, SC

- <https://www.openstreetmap.org/relation/193989>

I selected Greenville, SC because we are searching for our forever city after my husband's military retirement. This may help me learn additional information about our potential hometown.

Source: OpenStreetMap.org.



# Problems Encountered in the Map

---

## Audit of Street Names

Output showed

```
defaultdict(<class 'set'>, {'Riverplace': {'Riverplace'}, 'Woodruff': {'Woodruff'}, 'Palza': {'West Lewis Palza'}, 'Crossing': {'Woods Crossing'}, 'Rd': {'Laurens Rd', 'Woods Crossing Rd', 'Woodruff Rd'}, '29611': {'17 P and N Drive, Greenville SC 29611'}, 'Point': {'Independence Point'}, 'A': {'Williams Street Suite A'}, 'Extension': {'Roper Mountain Road Extension', 'Hampton Avenue Extension'}, 'Row': {'Varsity Row', 'Faculty Row'}, 'E10': {'South Pleasantburg Drive Ste E10'}, 'Curve': {'Collegiate Curve'}, 'Ridge': {'University Ridge', 'Cross Ridge'}, 'Park': {'Falls Park'}, 'B': {'Shaw Street Unit B'}, 'Us': {'Haywood Rd @ Toys R Us'}, 'Terrace': {'Livingston Terrace', 'Braemar Terrace'}, 'Glen': {'Cross Glen'}, 'Pointe': {'Cross Pointe'}, 'Hollow': {'Cross Hollow'}, 'Knob': {'Mary Knob'}})
```

Out[12]:

```
defaultdict(set,
  {'Riverplace': {'Riverplace'},
   'Woodruff': {'Woodruff'},
   'Palza': {'West Lewis Palza'},
   'Crossing': {'Woods Crossing'},
   'Rd': {'Laurens Rd', 'Woodruff Rd', 'Woods Crossing Rd'},
   '29611': {'17 P and N Drive, Greenville SC 29611'},
   'Point': {'Independence Point'},
   'A': {'Williams Street Suite A'},
   'Extension': {'Hampton Avenue Extension',
                 'Roper Mountain Road Extension'},
   'Row': {'Faculty Row', 'Varsity Row'},
   'E10': {'South Pleasantburg Drive Ste E10'},
   'Curve': {'Collegiate Curve'},
   'Ridge': {'Cross Ridge', 'University Ridge'},
   'Park': {'Falls Park'},
   'B': {'Shaw Street Unit B'},
   'Us': {'Haywood Rd @ Toys R Us'},
   'Terrace': {'Braemar Terrace', 'Livingston Terrace'},
   'Glen': {'Cross Glen'},
   'Pointe': {'Cross Pointe'},
   'Hollow': {'Cross Hollow'},
   'Knob': {'Mary Knob'}})
```

Problems encountered with the map of the Greenville area street types are as follows:

- Returned Unit or Suite value instead of street type
- Abbreviation of Road
- Misspelling of Plaza

- One postal code was present in data not a street type

Count of different Street Types:

---

29611:	1
A:	1
Avenue:	234
B:	1
Boulevard:	154
Circle:	77
Court:	137
Crossing:	1
Curve:	3
Drive:	704
El0:	1
Extension:	18
Glen:	2
Highway:	41
Hollow:	3
Knob:	12
Lane:	138
Palza:	3
Park:	1
Parkway:	34
Place:	22
Plaza:	3
Point:	8
Pointe:	7
Rd:	3
Rd.:	1
Ridge:	10
Riverplace:	4
Road:	992
Row:	6
Street:	797
Terrace:	4
Trail:	8
Us:	1
Way:	64
Woodruff:	1

---

## Postal Codes

Post Code count came out as follows:

```
29601: 442
29602: 1
29605: 318
29607: 954
29607-3817: 1
29609: 466
29611: 139
29614: 101
29615: 313
29617: 15
29650: 1
29662: 109
29687: 3
```

I used the same code provided in the street type count and changed the attribute to postcode to get the count for each postal code.

There was one that has the additional 4 digits on the postal code, so that will need to be cleaned up.

The expected postal codes for Greenville, SC are as follows:

29601, 29605, 29607, 29609, 29611, 29614, 29615, 29662

There are several that were not expected and belong to neighboring cities/suburbs of Greenville.

29687- belongs to Taylors, SC, but is in Greenville County

If postal code is limited to the 5 digit code it will be better for grouping when doing further analysis. And those from Taylors should be excluded as well to limit the analysis to Greenville City.

## CONFIRMED Zip Code Cleanup in database:

```
sqlite> SELECT nodes_tags.value
...> FROM nodes_tags
...> JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE type='addr') i
...> ON nodes_tags.id=i.id
...> WHERE nodes_tags.key='postcode'
...> GROUP BY nodes_tags.value;
29601
29605
29607
29609
29611
29614
29615
29662
29687
```

## STATE AUDIT

Through the state audit I discovered that 18 times someone had used Carolina as the state. SC was the majority of state types.

---

Carolina: 18  
SC: 1647

## Confirmed Cleaning after database creation:

```
sqlite> SELECT nodes_tags.value
...> FROM nodes_tags
...> JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE type='addr') i
...> ON nodes_tags.id=i.id
...> WHERE nodes_tags.key='state'
...> GROUP BY nodes_tags.value;
SC
```

Since this returned only SC there appears to be no South Carolina or Carolina values for the state in the table.

# Sort cities by count, descending

---

```
sqlite> 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;
```

And, the results, edited for readability:

```
Greenville,2736
Mauldin,109
Taylors,2
"Greenville, SC",2
Greer,1
```

## Data Overview

---

### File sizes

#### Size of Data Set

```
Downloads — -zsh — 80x24
Last login: Wed Oct 28 06:34:42 on ttys001
(base) cynthiapottin@Cynthias-MacBook-Pro-2 ~ % cd Downloads
(base) cynthiapottin@Cynthias-MacBook-Pro-2 Downloads % ls -l Greenville
-rw-r--r--@ 1 cynthiapottin  staff  91927947 Oct 18 08:07 Greenville
(base) cynthiapottin@Cynthias-MacBook-Pro-2 Downloads %
```

Data set size: approximately 92 MB

## Types of Tags

```
In [1]: import xml.etree.cElementTree as ET
        from collections import defaultdict
        import re
        import pprint

        tree = ET.parse('Greenville')

        mylist = []
        for elem in tree.iter():
            mylist.append(elem.tag)

        mylist = list(set(mylist))

        print(mylist)

['node', 'member', 'nd', 'osm', 'note', 'relation', 'tag', 'meta', 'way', 'bounds']
```

Used an element tree and list to see the types of tags in the OSM xml file. It returned ['node', 'member', 'nd', 'osm', 'note', 'relation', 'tag', 'meta', 'way', 'bounds'].

## Tag Count

```
In [1]: import xml.etree.ElementTree as ET
from collections import defaultdict
import re
import pprint
import re
import xml.etree.cElementTree as ET
```

```
In [2]: ## code from udacity lessons

osmfile = 'Greenville'

def count_tags(filename):
    tags = {}
    for event, elem in ET.iterparse(filename, events=('start', )):
        if elem.tag not in tags:
            tags[elem.tag] = 1
        else:
            tags[elem.tag] += 1
    return tags

def test():
    tags = count_tags(osmfile)
    pprint.pprint(tags)

if __name__ == "__main__":
    test()

{'bounds': 1,
 'member': 14783,
 'meta': 1,
 'nd': 459960,
 'node': 377190,
 'note': 1,
 'osm': 1,
 'relation': 1472,
 'tag': 210744,
 'way': 60916}
```

```
In [ ]:
```

Bounds: 1  
Member: 14783  
Meta: 1  
Nd: 459960  
Node: 377190  
Note:1  
Osm:1  
Relation: 1472  
Tag: 210744  
Way: 60916

## User Analysis

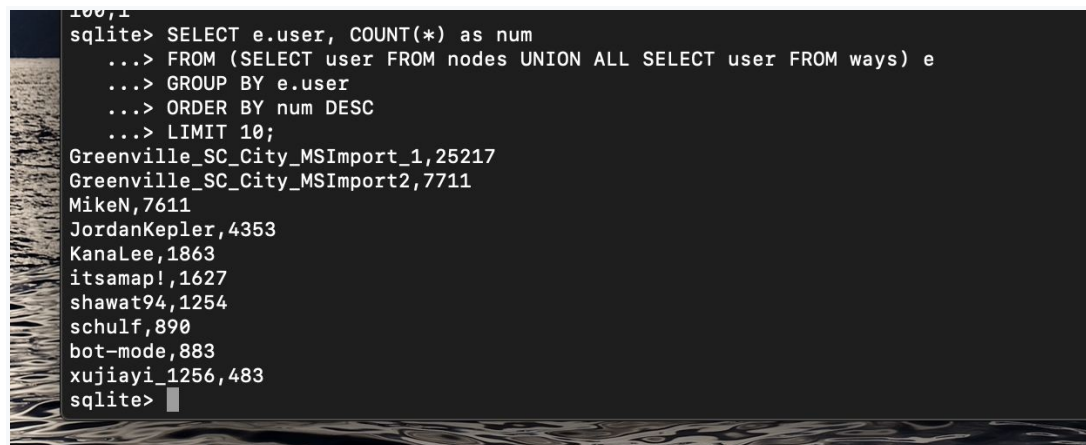


## Number of unique users

583

## Top 10 contributing users

```
sqlite> 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;
```



```
100,1
sqlite> 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;
Greenville_SC_City_MSImport_1,25217
Greenville_SC_City_MSImport2,7711
MikeN,7611
JordanKepler,4353
KanaLee,1863
itsamap!,1627
shawat94,1254
schulff,890
bot-mode,883
xujiayi_1256,483
sqlite>
```

```
Greenville_SC_City_MSImport_1,25217
Greenville_SC_City_MSImport2,7711
MikeN,7611
JordanKepler,4353
KanaLee,1863
itsamap!,1627
shawat94,1254
schulff,890
bot-mode,883
xujiayi_1256,483
```

## Number of users appearing only once (having 1 post)

```
sqlite> SELECT COUNT(*)
FROM
  (SELECT e.user, COUNT(*) as num
   FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
   GROUP BY e.user
   HAVING num=1) u;
```

72

```
sqlite> SELECT COUNT(*)
...> FROM
...>   (SELECT e.user, COUNT(*) as num
...>     FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
...>     GROUP BY e.user
...>     HAVING num=1) u;
72
sqlite> █
```

## Additional Data Exploration

---

### Top 10 appearing amenities

```
sqlite> SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key='amenity'
GROUP BY value
ORDER BY num DESC
LIMIT 10;
```

```
bench,267
restaurant,261
fast_food,84
place_of_worship,76
cafe,47
bicycle_parking,39
fuel,35
toilets,29
post_box,29
Parking_entrance,29
```

Top amenity is a bench, which I found surprising to find on the list of amenities. I would not have thought of a bench as an amenity. Restaurants came in at #2 with fast food and cafe also listed in the top 10. To me these are similar but we will take a look at the types of restaurants, fast food and cafes to see how they differ.

# Amenities Analysis

## House of Worship

```
sqlite> SELECT nodes_tags.value, COUNT(*) as num
...> FROM nodes_tags
...> JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='place_of_worship') i
...> ON nodes_tags.id=i.id
...> WHERE nodes_tags.key='religion'
...> GROUP BY nodes_tags.value
...> ORDER BY num DESC
...> LIMIT 1;
```

**Christian,76**

```
sqlite> SELECT nodes_tags.value, COUNT(*) as num
...> FROM nodes_tags
...> JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='place_of_worship') i
...> ON nodes_tags.id=i.id
...> WHERE nodes_tags.key='denomination'
...> GROUP BY nodes_tags.value
...> ORDER BY num DESC
...> LIMIT 10;
```

baptist,26

presbyterian,4

methodist,4

wesleyan,1

orthodox,1

evangelical,1

catholic,1

anglican,1

The biggest religion in Greenville was Christian and I searched for others but it only returned Christian, so I figured we could take a look at denomination as well. It was not much of a surprise to find that Baptist was the number 1 denomination in this southern city.

## Most popular cuisines

### Restaurants

```
sqlite> SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='restaurant') i
ON nodes_tags.id=i.id
WHERE nodes_tags.key='cuisine'
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 10;
```

regional,35

mexican,31

pizza,23

```
american,18
japanese,14
italian,14
sandwich,9
chinese,8
thai,6
Seafood,6
```

## Fast Food

```
SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
  JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='fast_food') i
  ON nodes_tags.id=i.id
WHERE nodes_tags.key='cuisine'
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 10;
```

```
sandwich,24
burger,17
pizza,9
chicken,6
mexican,3
hot_dog,3
mediterranean,2
japanese,2
donut,2
Tex-mex,1
```

## Cafes

```
SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
  JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='cafe') i
  ON nodes_tags.id=i.id
WHERE nodes_tags.key='cuisine'
GROUP BY nodes_tags.value
ORDER BY num DESC;
```

```
coffee_shop,8
juice,2
regional,1
dessert,1
coffee,1
```

## Suggestions for improving the data

Per Review submit suggestions to improve the data.

I think it would benefit many people to have a more detailed description of the food and amenities. “Regional” food or cafe is too generic of a description for those assessing if they would like to visit the restaurant.

Also, looking at the amenities the top listed amenity was a bench. I feel like there could be more information about the location of the bench. Is it in the park, on a greenway, or for a bus stop? This additional information would be helpful for users of Open Street Maps to figure out if they wanted to visit those benches.

It would be difficult to implement because of the extra details. It would create a lot of categories of restaurants and amenities but I think the more detail and extra description would be helpful for the users.

Another improvement would be to or its analysis. The suggestions are backed up by at least one investigative query.

Another way to improve would be to set character limits or reject values outside the expected range. Example if the OSM limited the creation of a zip code to a character length of 5 there would be less clean up of the data. If they limited the state to the two character abbreviation the data would be cleaner, as well.

## Conclusion

---

In conclusion, Greenville data was relatively clean. I believe this is due to the fact that the top contributors to the OSM was Greenville\_SC\_City\_MSImport\_1 and Greenville\_SC\_City\_MSImport2 not as much human error when the information is coming from an imported source.

References:

<https://docs.python.org/3.3/library/xml.etree.elementtree.html>

<http://docs.buildbot.net/0.9.4/developer/py3-compat.html>

[https://www.w3schools.com/python/ref\\_list\\_count.asp](https://www.w3schools.com/python/ref_list_count.asp)

<https://eli.thegreenplace.net/2012/03/15/processing-xml-in-python-with-elementtree>

<https://www.geeksforgeeks.org/defaultdict-in-python/>

<https://www.python.org/dev/peps/pep-0469/>

<https://www.geeksforgeeks.org/break-continue-and-pass-in-python/>

<https://www.guru99.com/python-regular-expressions-complete-tutorial.html>

Side Note:

This was a challenging project that was made more challenging with outdated material. I spent a good bit of my time finding ways to work around things. Broken links in some of the lessons and the simple roadblocks such as changes from what appears to be Python 2 to Python 3 syntax. (<https://www.python.org/dev/peps/pep-0469/>)

I have done my best to explain my code and my thought processes while working through this project.