# **OpenStreetMap Data Case Study**

### **Map Area**

Rio de Janeiro, RJ, Brazil

* <https://www.openstreetmap.org/relation/2697338>
* [API Overpass](https://overpass-api.de/api/map?bbox=-43.7997,-23.1157,-43.0959,-22.7129)

I chose this area because this is the city I currently live in, and where my son was born. It would be nice to have the opportunity to contribute to its improvement.

## **Overview of the data**

* The file has **280 MB**.
* Number of unique users that have contributed to the map: **1,703**

code number\_users.py

* Number of nodes and ways:
  + **'node': 117,986**
  + **'way': 19,665**

code Iterparse.py

* Number of "k" values for each "<tag>":
  + **'lower': 522194,**
  + **'lower\_colon': 50664,**
  + **'other': 22223,**
  + **'problemchars': 2**

code tag\_types.py

## **Problems Encountered in the Map**

In order to find and fix problems more easily, I first subsetted the data (k = 1000), using [sampleCode.py](https://github.com/mablatnik/Wrangle-OpenStreetMap-Data/blob/master/sampleCode.py).

I noticed four main problems with the data, which I will discuss in the following order:

1. **Regarding street names (remember that names are in Portuguese):**

* Over­abbreviated names
  + "Av" and "Av." meaning "Avenida"
  + "Pça" and "Pça." meaning "Praça"
  + "R." meaning "Rua"
* Incorrect writing
  + "Ruo", "Rue" and "Ruas" meaning "Rua"
  + "Praca" meaning "Praça"
* Street Type missing (very common problem). See some examples below. All of them should be "Rua …." or "Avenida ….." .
  + {'15': {'15 de Novembro'},
  + '199': {'199'},
  + 'Afredo': {'Afredo Ceschiatti'},
  + 'Aires': {'Aires Itabaiana'},
  + 'Alfredo': {'Alfredo Ceschiatti'},
  + 'Apurinãs': {'Apurinãs'},
  + 'Arquias': {'Arquias Cordeiro'},
  + 'Assis': {'Assis Bueno'},
  + 'Augusta': {'Augusta Candiani'},
* Inconsistent postal codes *(“NC28226”, “28226­0783”, “28226”)*
* “Incorrect” postal codes (Charlotte area zip codes all begin with “282” however a large portion of all documented zip codes were outside this region.)

**2. Regarding Tags names**

* 594 keys used only once (not exactly a problem for all cases);
* Similar key tags with different names. Some examples:
  + "CEP\_LD" vs "cep:par" vs "zip\_right"
  + "CEP\_LE" vs "cep:impar" vs "zip\_left"
  + "Contact\_phone" vs "phone"

code tag\_types.py

**3. Inconsistent postal codes**

* 22410-000 *vs* 22410000

**4. Inconsistent contact phones**

* +55-21-9999-9999 *vs* (21) 9999-9999 *vs* 9999-9999, etc.

### **Over­abbreviated Street Names**

Once the data was imported to SQL, some basic querying revealed street name abbreviations and postal code inconsistencies. To deal with correcting street names, I opted not use regular expressions, and instead iterated over each word in an address, correcting them to their respective mappings in audit.py using the following function:

def update(name, mapping):   
 words = name.split()  
 for w in range(len(words)):  
 if words[w] in mapping:  
 if words[w­1].lower() not in ['suite', 'ste.', 'ste']:   
 # For example, don't update 'Suite E' to 'Suite East'  
 words[w] = mapping[words[w]] name = " ".join(words)  
 return name

This updated all substrings in problematic address strings, such that: *“S Tryon St Ste 105”* becomes: *“South Tryon Street Suite 105”*

### **Postal Codes**

Postal code strings posed a different sort of problem, forcing a decision to strip all leading and trailing characters before and after the main 5­digit zip code. This effectively dropped all leading state characters (as in “NC28226”) and 4­digit zip code extensions following a hyphen (“28226­0783”). This 5­digit restriction allows for more consistent queries.

Regardless, after standardizing inconsistent postal codes, some altogether “incorrect” (or perhaps misplaced?) postal codes surfaced when grouped together with this aggregator:

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;

Here are the top ten results, beginning with the highest count:

value|count  
28205|900  
28208|388  
28206|268  
28202|204  
28204|196  
28216|174  
28211|148  
28203|120  
28209|104  
28207|86

These results were taken before accounting for Tiger GPS zip codes residing in second­ level “k” tags. Considering the relatively few documents that included postal codes, of those, it appears that out of the top ten, seven aren’t even in Charlotte, as marked by a “#”. That struck me as surprisingly high to be a blatant error, and found that the number one postal code and all others starting with“297”lie in Rock Hill, SC. So, I performed another aggregation to verify a certain suspicion...

# **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:

Rock Hill 111   
Pineville 27   
Charlotte 26   
York 24   
Matthews 10   
Concord 4   
3000 3   
10 2   
Lake Wylie 2   
1 1   
3 1   
43 1   
61 1   
Belmont, N 1   
Fort Mill, 1

These results confirmed my suspicion that this metro extract would perhaps be more aptly named “Metrolina” or the “Charlotte Metropolitan Area” for its inclusion of surrounding cities in the sprawl. More importantly, three documents need to have their trailing state abbreviations stripped. So, these postal codes aren’t “incorrect,” but simply unexpected. However, one final case proved otherwise. A single zip code stood out as clearly erroneous. Somehow, a “48009” got into the dataset. Let’s display part of its document for closer inspection (for our purposes, only the “address” and “pos” fields are relevant):

sqlite> SELECT \*  
FROM nodes   
WHERE id IN (SELECT DISTINCT(id) FROM nodes\_tags WHERE key='postcode' AND value='48009')

1234706337|35.2134608|-80.8270161|movercash|433196|1|7784874|2011-04-06T13:16:06Z

sqlite> SELECT \* FROM nodes\_tags WHERE id=1234706337 and type='addr';

1234706337|housenumber|280|addr  
1234706337|postcode|48009|addr  
1234706337|street|North Old Woodward Avenue|addr

It turns out, *“280 North Old Woodward Avenue, 48009”* is in Birmingham, Michigan. All data in this document, including those not shown here, are internally consistent and verifiable, except for the latitude and longitude. These coordinates are indeed in Charlotte, NC. I’m not sure about the source of the error, but we can guess it was most likely sitting in front of a computer before this data entered the map. The document can be removed from the database easily enough.

# **Data Overview and Additional Ideas**

This section contains basic statistics about the dataset, the MongoDB queries used to gather them, and some additional ideas about the data in context.

### **File sizes**

charlotte.osm ......... 294 MB  
charlotte.db .......... 129 MB  
nodes.csv ............. 144 MB  
nodes\_tags.csv ........ 0.64 MB  
ways.csv .............. 4.7 MB  
ways\_tags.csv ......... 20 MB  
ways\_nodes.cv ......... 35 MB

### **Number of nodes**

sqlite> SELECT COUNT(\*) FROM nodes;

1471350

### **Number of ways**

sqlite> SELECT COUNT(\*) FROM ways;

84502

### **Number of unique users**

sqlite> SELECT COUNT(DISTINCT(e.uid))   
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e;

337

### **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;

jumbanho 823324   
woodpeck\_f 481549   
TIGERcnl 44981   
bot-mode 32033   
rickmastfa 18875   
Lightning 16924   
grossing 15424   
gopanthers 14988   
KristenK 11023   
Lambertus 8066

### **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;

56

# **Additional Ideas**

## **Contributor statistics and gamification suggestion**

The contributions of users seems incredibly skewed, possibly due to automated versus manual map editing (the word “bot” appears in some usernames). Here are some user percentage statistics:

* Top user contribution percentage (“jumbanho”) 52.92%
* Combined top 2 users' contribution (“jumbanho” and “woodpeck\_fixbot”) 83.87%
* Combined Top 10 users contribution 94.3%
* Combined number of users making up only 1% of posts 287 (about 85% of all users)

Thinking about these user percentages, I’m reminded of “gamification” as a motivating force for contribution. In the context of the OpenStreetMap, if user data were more prominently displayed, perhaps others would take an initiative in submitting more edits to the map. And, if everyone sees that only a handful of power users are creating more than 90% a of given map, that might spur the creation of more efficient bots, especially if certain gamification elements were present, such as rewards, badges, or a leaderboard.

## **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;

place\_of\_worship 580   
school 402   
restaurant 80   
grave\_yard 75   
parking 63   
fast\_food 51   
fire\_station 48   
fuel 31   
bench 30   
library 28

### **Biggest religion (no surprise here)**

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 571

### **Most popular cuisines**

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;

american 9   
pizza 5   
steak\_hous 4   
chinese 3   
japanese 3   
mexican 3   
thai 3   
italian 2   
sandwich 2   
barbecue 1

# **Conclusion**

After this review of the data it’s obvious that the Charlotte area is incomplete, though I believe it has been well cleaned for the purposes of this exercise. It interests me to notice a fair amount of GPS data makes it into OpenStreetMap.org on account of users’ efforts, whether by scripting a map editing bot or otherwise. With a rough GPS data processor in place and working together with a more robust data processor similar to data.pyI think it would be possible to input a great amount of cleaned data to OpenStreetMap.org.