

# Wrangle OpenStreetMap Data

*Orlando Castillo*

Map Area: Caracas, Venezuela

Source: [https://s3.amazonaws.com/metro-extracts.mapzen.com/caracas\\_venezuela.osm.bz2](https://s3.amazonaws.com/metro-extracts.mapzen.com/caracas_venezuela.osm.bz2)

## [1. Data conventions](#)

## [2. Problems Encountered in the Map](#)

[Unrecognized street types](#)

[Inconsistent tag keys for full addresses](#)

[Postal codes with invalid format](#)

[Invalid/inconsistent city values](#)

[Entries outside the city of Caracas](#)

## [3. Data Overview](#)

## [4. Additional Ideas](#)

[Stronger automated auditing](#)

[Additional data exploration using MongoDB queries](#)

## 1. Data conventions

To successfully audit, clean and analyze the data of the area of Caracas, Venezuela, it is important to understand the conventions followed to define a postal address. The [Universal Postal Union \(UPU\)](#) keeps track of the [postal addressing systems of member countries](#) and the conventions for Venezuela (as of December 30th of 2015) can be found [here](#). It is also important to keep in mind that the official language of Venezuela is Spanish and so is the case with most of the values in the map's data.

## 2. Problems Encountered in the Map

After downloading the data of the area of Caracas, Venezuela and running the function `osm_general_stats(osm_file)` in `src/utls.py` (the obtained stats can be found in `resources/original_osm_stats.txt`), I extracted the following subset of attributes and tag keys for closer inspection as I considered they are relevant enough to demand attention:

- **Attributes:** `timestamp`; `lat`; `lon`
- **Tag keys:** `addr:ful`; `addr:full`; `addr:full\x7f`; `addr:postcode`; `addr:street`; `\x7faddr:full`; `addr:city`; `addr:country`; `addr:state`

Using a combination of the `audit(osm_file)` function in `src/caracas_map_session.py` (results can be found in `resources/original_audit.txt`) and manual inspection of the data, I identified the following problems:

- Unrecognized street types.
- Inconsistent tag keys for full addresses.
- Postal codes with invalid format.
- Invalid/inconsistent city values.
- Entries outside the city of Caracas.

## Unrecognized street types

I extracted 113 unrecognized groups of street types from tags with the key `addr:street`. After manually inspecting this list, I was identified the following problems:

- **Abbreviated street types:** `'Av. Roosevelt'` instead of `'Avenida Roosevelt'`
- **Street types not capitalized:** `'carretera'` instead of `'Carretera'`
- **Street names without type:** `'Terepaima'` instead of `'Calle Terepaima'`
- **Street types with inconsistent formatting:** For example, there are cases of street types which use a colon to indicate a highway between two locations (`'Carretera: Petare - Santa Lucia'`) but most cases would avoid its use (`'Carretera Caracas - El Junquito'`)
- **Spelling mistakes:** `'Prolongacion'` instead of `'Prolongación'` (notice the diacritic mark)
- **Locations instead of streets:** `'Aeropuerto La Carlota'` is a reference to the [Generalissimo Francisco de Miranda Air Base](#), so in this case `addr:place` would probably be better to store this value.

Using the `clean_up_map(original_osm, clean_osm)` function in `src/caracas_map_session.py`, I managed to programatically clean 431 entries and reduce the amount of unrecognized groups to 91. To clean the rest of the values, it seems it would require a higher degree of manual work.

## Inconsistent tag keys for full addresses

From the values and the element structures, I found what I assume are several undesired variations of the `addr:full` key: `addr:ful`, `addr:full\x7f` and `\x7faddr:full`. I managed to programatically clean all 7 instances with the `clean_up_map(original_osm, clean_osm)` function in `src/caracas_map_session.py`.

## Postal codes with invalid format

I discovered 4 `addr:postcode` types that didn't follow the 4-digits convention: `'1010-A'`, `'10'`, `'321'`, `'Bloque 14 de UD-3'`. Below I explain in more detail the context and strategy for cleaning:

- **‘Bloque 14 de UD-3’**: One element had a child tag with this value and I found it was the same as the value for the key *addr:housename*, so it makes sense to assume it was just copied by mistake. Given that it was only one element, I manually searched for the proper postal code with the value of *addr:street* and updated the element.
- **‘1010-A’**: I found this value in the node representing the city of Caracas. Given that this node represents the city, I would not expect this field to exist as this element doesn’t represent an address. The element already has the field *openGeoDB:postal\_codes* with the desired range of values. With all the previous information in consideration, I decided to remove the *addr:postcode* tag from this element.
- **‘321’**: One ‘way’ element had a child tag with this postal code and it seems to reference a location outside of the city of Caracas. I didn’t update this element as I’m not sure if the proper cleaning should be to remove the element or to clean up the values of the child tags.
- **‘10’**: I found 18 elements with this postal code value, and from those I only cleaned the element with id ‘2420112600’ as it was the only one I could validate it was inside the city of Caracas. I manually explored the coordinates of some of the other elements and they are valid, but the values of the address seem to be invalid and at this point I stopped as I’m not sure what’s the best cleaning procedure without further context.

## Invalid/inconsistent city values

I extracted 60 unrecognized groups of city values. I programmatically fixed the capitalization of 4 of ‘caracas’ values, the rest seems it would require non-trivial work of mapping a node location to a particular city as there’s no guarantee the elements are inside the city Caracas.

## Entries outside the city of Caracas

As mentioned in some of the previous groups of problems, I observed elements outside the city of Caracas. I’ll summarize the different scenarios I observed:

- Post codes with values outside the [database](#) (link in Spanish) of valid values of Caracas.
- City values like ‘[Maracay](#)’, which is another city of Venezuela.
- State values like ‘[Vargas](#)’ instead of ‘[Distrito Capital](#)’
- Entries with latitude, longitude values outside the [boundaries](#) of the capital district.

I’m not sure what’s the best way to handle this values so I decided to leave them unchanged, but it seems to represent an area of improvement and relevant to keep in mind during the analysis of the data.

## 3. Data Overview

This section contains basic statistics about the dataset and the MongoDB queries used to gather them.

## File sizes

```
caracas_venezuela.osm ..... 51.8 MB
caracas_venezuela_clean.osm ..... 52.5 MB
caracas_venezuela_clean.osm.json .. 57.2 MB
```

### # Number of documents

```
> db.caracas.find().count()
246399
```

### # Number of nodes

```
> db.caracas.find({"type":"node"}).count()
225445
```

### # Number of ways

```
> db.caracas.find({"type":"way"}).count()
20950
```

### # Number of unique users

```
> db.caracas.distinct("created.user").length
326
```

### # Top 5 contributing users

```
> db.caracas.aggregate([{"$group":{"_id":"$created.user",
"count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":5}])
{ "_id" : "albertoq", "count" : 96163 }
{ "_id" : "Schandlers", "count" : 31112 }
{ "_id" : "algarinr", "count" : 14916 }
{ "_id" : "bolollo", "count" : 12516 }
{ "_id" : "kriscarle", "count" : 11682 }
```

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

```
> db.caracas.aggregate([{"$group":{"_id":"$created.user",  
  "count":{"$sum":1}}}, {"$group":{"_id":"$count",  
  "num_users":{"$sum":1}}}, {"$sort":{"_id":1}}, {"$limit":1}])  
[ {"_id":1,"num_users":58} ]  
# "_id" represents postcount
```

## 4. Additional Ideas

### Stronger automated auditing

Many of the issues I found in the dataset could be prevented with stronger automated auditing during the uploading process of the data. I can see there's already several [Quality Assurance](#) tools available and I believe it should be possible to bring some of the tools developed in this project for the community to use. In particular, the following are the particular areas which could benefit from the auditing in this project:

- **Format validation of street addresses:** With the current logic, it should be possible to identify problematic street addresses in any region of Venezuela, and with some tweaking generalize for more countries in South America. Currently the logic could find false negatives, but this is something that I'm sure could be mitigated.
- **Format validation of postal codes:** The postal code format in Venezuela is well defined, so the developed validation logic should be complete and useful already for any region of the country.
- **Validation of city, state and country values:** These are all well defined values and the current auditing logic should be complete for the region of Caracas, Venezuela. With some amount of tweaking, it could be generalized for any region of the country.

Some other areas of improvements which were not addressed in this project could be the following:

- **Cross validation of street addresses:** It should be possible to use another source of map data to validate a street address and potentially match it to the correct value. In some cases, I manually achieved this after using some of the other tag values of an element as input for Google Maps.
- **Area validation:** I'm not sure if it was intended, but many of the elements in the map extract are not inside the region of Caracas, Venezuela. Using what seems like a well defined [boundary](#) for the region, it should be possible to validate that latitude and longitude values are within this area.

## Additional data exploration using MongoDB queries

### # Top 5 appearing amenities

```
> db.caracas.aggregate([{"$match":{"amenity":{"$exists":1}},
{"$group":{"_id":"$amenity", "count":{"$sum":1}}},
{"$sort":{"count":-1}}, {"$limit":5}])
```

```
{ "_id" : "bank", "count" : 790 }
{ "_id" : "school", "count" : 410 }
{ "_id" : "parking", "count" : 391 }
{ "_id" : "restaurant", "count" : 367 }
{ "_id" : "pharmacy", "count" : 348 }
```

### # Top 5 cuisines

```
> db.caracas.aggregate([{"$match":{"amenity":{"$exists":1},
"amenity":"restaurant", "cuisine":{"$exists":1}},
{"$group":{"_id":"$cuisine", "count":{"$sum":1}}},
{"$sort":{"count":-1}}, {"$limit":5}])
```

```
{ "_id" : "chinese", "count" : 46 }
{ "_id" : "regional", "count" : 42 }
{ "_id" : "steak_house", "count" : 41 }
{ "_id" : "italian", "count" : 27 }
{ "_id" : "chicken", "count" : 23 }
```

### # Bank with highest amount of ATMs

```
> db.caracas.aggregate([{"$match":{"amenity":{"$exists":1},
"amenity": "bank", "atm":{"$exists":1}, "atm": "yes"}},
{"$group":{"_id": "$name", "count":{"$sum":1}}},
{"$sort":{"count":-1}}, {"$limit":1}])
```

```
{ "_id" : "Banesco", "count" : 98 }
```