Wrangling OSM data for Madrid

Jerome Vergueiro Vonk

December 25, 2017

1 Map Area

Madrid, Spain

- https://www.openstreetmap.org/relation/6426653
- Data obtained using the Overpass API with the following parameters: Latitude: 40.3757 N to 40.4958 N, Longitud: 3.8284 E to 3.5589 E

Madrid is a city I learned to love when I spent a couple summers for a temporary job. Perhaps, with this project, I can explain why I like this city so much with numbers! And, besides that, catch up with my Spanish skills :)

2 Problems faced

After running a few tests, I noticed a couple problems with the data, explained above:

- Missing street type in address (*“Fuencarral”* instead of *”Calle Fuencarral”*)
- Inconsistentent street types (*”Calle”* was written as *”CALLE”*, *”calle”*, *”CL”*, *”C/”*)

In Spain, the type of the street comes first, so the regular expression used to identify the street type is as follows:

```
street_type_re = re.compile(r'^[^\s]+', re.IGNORECASE)
```

Next step was to search in the internet for a list of valid street types in Spain. Then, I matched all the street types I found against the ”correct” list to find the differences.

The output of the script *4_improve_street_types.py* shows which address were corrected and which where not:

```
Via de las Dos Castillas changed to Vía de las Dos Castillas
Via De Los Poblados changed to Vía De Los Poblados
Not fixed: Ribera de Curtidores
Not fixed: Amor de Dios
Not fixed: Cava de San Miguel
```

As shown in the example, some typos were fixed programatically. Besides, consistency was applied by fixing all abbreviations for ”Calle”, ”Carretera”, etc.

Here is a code snippet of the script prepare_database.py, that actually makes some improvements to the addresses before adding them to the database:

```
mapping = { "CL": "Calle", "C/": "Calle", "calle": "Calle", "CALLE": "Calle",
"AUTOP.": "Autopista", "Avda.": "Avenida", "plaza": "Plaza",
"CR": "Carrera", "CTRA.": "Carretera", "Ctra": "Carretera", "Pasage":"Pasaje" }

expected = ["Calle", "Plaza", "Avenida", "Alameda", "Camino", "Pasaje", "Paseo", "Rambla",
"Carrera", "Carretera", "Ronda", "Cuesta", "Glorieta", "Costanilla"]
# If this is a street name, let's audit if the street type is valid
else:
        m = street_type_re.search(secondary.attrib['v'])
        if m:
                street_type = m.group()

                if street_type in expected:
                        # Street type is what we expect, all good here
                        secondary_dic['value'] = secondary.attrib['v']
                else:
                        # Try to improve the street type
                        try:
                                secondary_dic['value'] = secondary.attrib['v'].replace(street_type,
                                mapping[street_type])
                                improved_address += 1
                                #print("Corrected \'{}\' to \'{}\'".format(secondary.attrib['v'],
                                corrected) )

                        except:
                                # We could't fix this by automation
                                secondary_dic['value'] = secondary.attrib['v']
                                #print("Do not know how to fix '{}\'".format(secondary.attrib['v']))
```

The cases were the street type were missing couldn't be fixed programatically, unfortunately. After examining the output of the *4_improve_street_types.py* script, I picked some of the addresses that appeard the most, looked online for double-checking (I don't claim to know all the streets in Madrid yet, ;P) and fixed in the original data by 'Replace all' method in *Notepad++*.

Another improvement made was about the postcode. In Madrid, the postal always has 5 numbers and must be between 28000 and 29000. This code snippet shows how it was done:

```
def is_postal_code(elem):
        return (elem.attrib['k'] == "addr:postcode")

# If it is a postcode, let's check if the postcode looks legit
# (for Madrid, must have 5 numbers and start with 28)
if not is_postal_code(secondary):
        secondary_dic['value'] = secondary.attrib['v']
else:
        try:
                postcode = int(secondary.attrib['v'])
                if postcode >= 28000 and postcode <= 28999:
                        secondary_dic['value'] = secondary.attrib['v']
                elif postcode == 2839:
                        # This has been added after examining the output on the first run
                        #print("Correcting '2839' to '28039'")
                        secondary_dic['value'] = "28039"
                        improved_address += 1
                else:
                        #print("Postal code looks invalid: ", postcode)
                        return None
        except:
                if secondary.attrib['v'] == "E28016":
                        # This has been added after examining the output on the first run
                        #print("Correcting 'E28016' to '28016'")
                        secondary_dic['value'] = "28016"
                        improved_address += 1
                else:
                        #print("Postal is not a number: ", secondary.attrib['v'])
                return None
```

3 Overview of the data

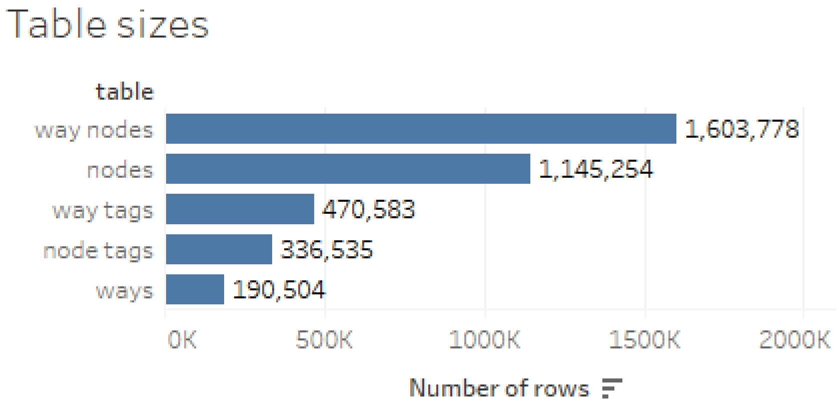
File sizes:

Madrid_custom_11122017.osm	280 MB
madrid.db	196 MB
nodes.csv	109 MB
nodes_tags.csv	15 MB
ways.csv	12 MB
ways_tags.csv	18 MB
ways_nodes.cv	37 MB

How many rows are there in each table?

Information	Query	Result
Number of nodes	SELECT COUNT(*) FROM nodes	1,145,254
Number of node tags	SELECT COUNT(*) FROM node_tags	336,535
Number of ways	SELECT COUNT(*) FROM ways	190,504
Number of way tags	SELECT COUNT(*) FROM way_tags	470,583
Number of way nodes	SELECT COUNT(*) FROM way_nodes	1,603,778

It's easier to visualize the size of the tables by looking at a chart:

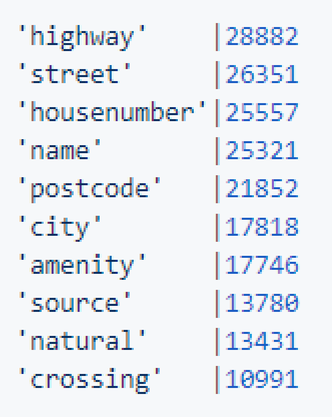


Most common keys in node tags

What's the most common key used in the node tags? We can list the top 10 with the following SQL query:

```
SELECT key, count(*) as count
FROM node_tags
GROUP BY key
ORDER BY count desc
LIMIT 10;
```

And the results are as follows:



```
'highway'    |28882
'street'     |26351
'housenumber'|25557
'name'       |25321
'postcode'   |21852
'city'       |17818
'amenity'    |17746
'source'     |13780
'natural'    |13431
'crossing'   |10991
```

Number of unique users that contributed

How many users contributed to this map? We can find out with the following SQL query:

```
SELECT COUNT( DISTINCT subquery.uid )
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways)
  as subquery;
```

The answer is 1728.

What's the maximum number of nodes that on way has?

We can unravel that with the following SQL query:

```
 select way_id, max(position) from way_nodes
```

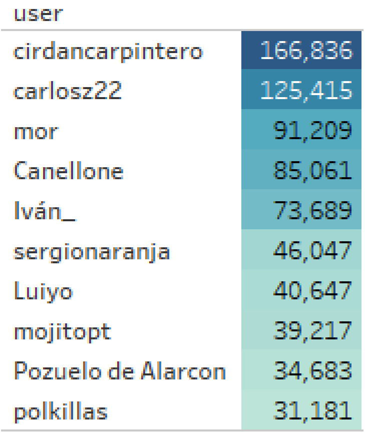
As a result, there is a way (32337047) that has 1023 nodes!

Which user has contributed the most?

We can list the top 10 contributors with the following SQL query:

```
SELECT user, count(*) as count
FROM nodes
GROUP BY user
ORDER BY count desc
LIMIT 10;
```

Way to go, *cidancarpintero*!? The results are:



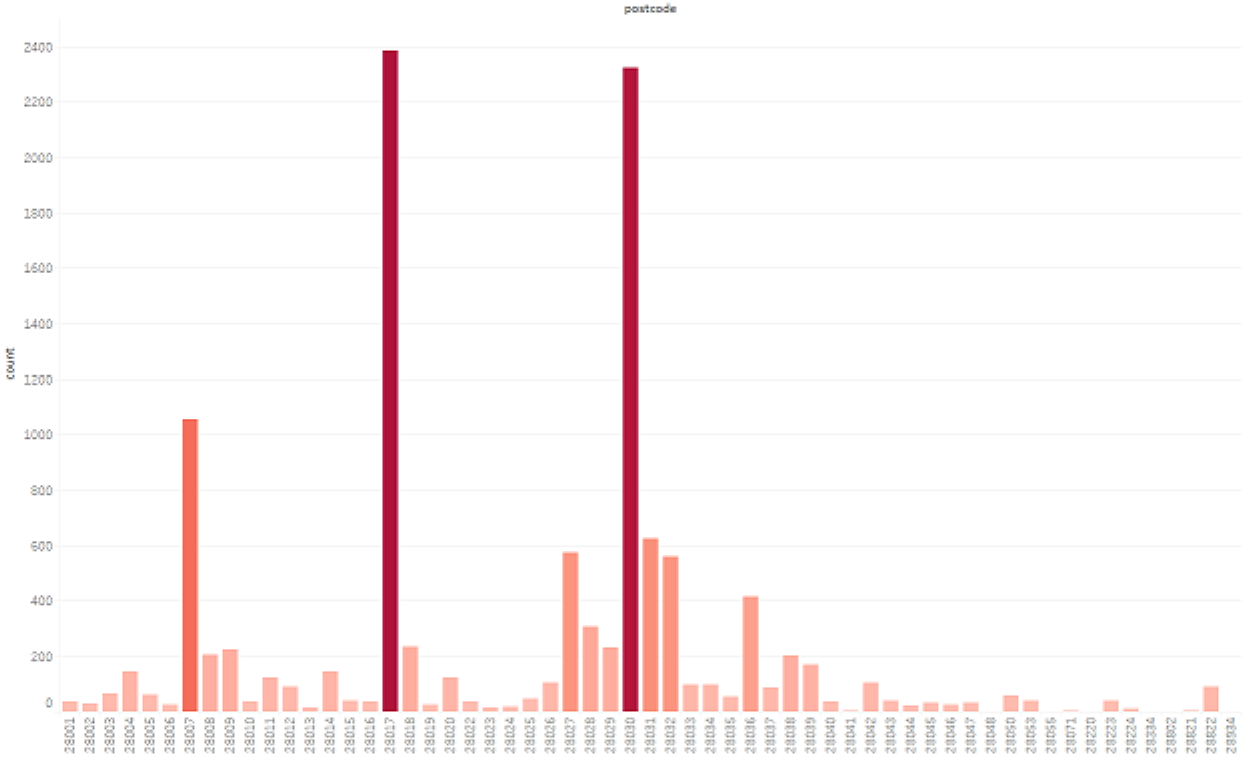
user	
cirdancarpintero	166,836
carlosz22	125,415
mor	91,209
Canellone	85,061
Iván_	73,689
sergionaranja	46,047
Luiyo	40,647
mojitopt	39,217
Pozuelo de Alarcon	34,683
polkillas	31,181

Which postcode appear more often?

We can list the top 10 with the following SQL query:

```
SELECT value, count(*) as count
FROM way_tags
WHERE key='postcode'
GROUP BY value
ORDER BY count desc
```

There are clearly two postcodes thar appear way more often. These postcodes are near the areas of Ventas (28017) and Moratalaz (28030). The complete results are:

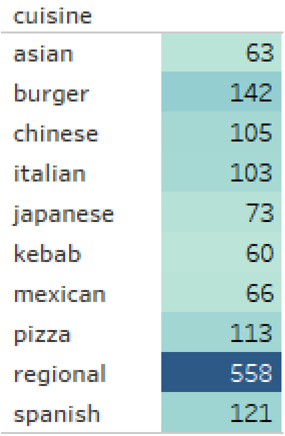


What type of cuisine is more frequently listed?

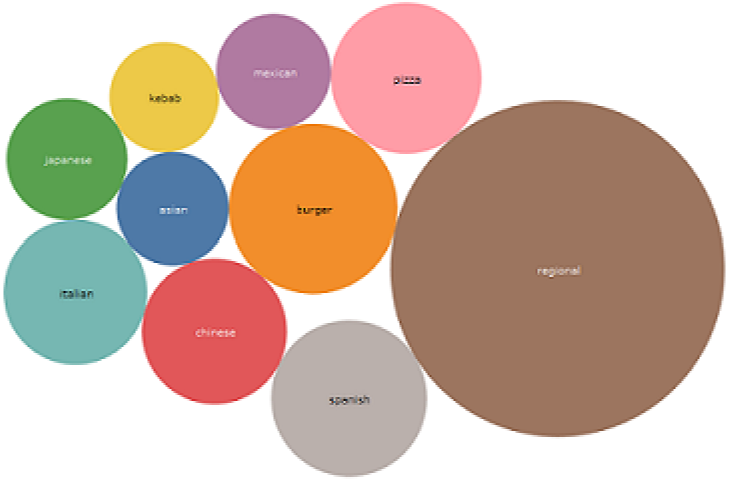
We can list the top 10 with the following SQL query:

```
SELECT value, count(*) as count
FROM node_tags
WHERE key='cuisine'
GROUP BY value
ORDER BY count desc
LIMIT 10;
```

No surprises here, right? The results are:



cuisine	
asian	63
burger	142
chinese	105
italian	103
japanese	73
kebab	60
mexican	66
pizza	113
regional	558
spanish	121



4 Additional ideas

Since the input data is taken from humans, we should come up with ways of forcing the user to enter the data correctly. For example, in this case study, I came accross lots of imcomplete address where the street type was missing. My suggestion would be to create a combo-box with all the available options of street types (might even be user generated, but from another form) and force the user to pick a value from the combo-box and then complete with the street name.

In this case, the options on the combo-box would be (*"Calle"*, *"Carrera"*, *"Carretera"*, *"Plaza"*, *"Ronda"*, etc).

Of course, that would need some testing before, because we don't want to make the process so daunting that the user will give up entering data.