osm_madrid

October 16, 2016

```
In [1]: #!/usr/bin/env python
        # -*- coding: utf-8 -*-
        %matplotlib inline
        import pprint
        import matplotlib.pyplot as plt
        from matplotlib.legend_handler import HandlerLine2D
        import seaborn as sns
```

1 Mongo DB data wrangling

In this document we are going to analyse the OSM dataset for Madrid , the dataset has been downloaded from :

https://mapzen.com/data/metro-extracts/metro/madrid_spain/101748283/Madrid/

Madrid is my home town so the spotting errors in the dataset should be easier. The dataset covers the whole province of Madrid but as it is defined as an square area in also includes some features of the neighbour provinces which are:

- Toledo
- Avila
- Segovia
- Guadalajara
- Cuenca

We need to take this into account when analysing the dataset.

For this analysis I have chosen MongoDB as the database.

1.1 Problems encountered in the map

After generating a reduced dataset with a 10th of the records.We proceed with the initial assessment of the data using a reduced dataset, the following fields have been analysed:

- street names
- leisures
- amenities
- postal codes

We haved generated a sample file with a 10th of the values for the data wrangling analysis usgin the create_sample_from_osm.py script. This script will also generate a sample with a 100th of the data more suitable for upload. The data from the sample file was then audited using the audit_sample_dataset.py script.

Based on this analysis some filtering or cleaning functions will be generated in order to clean the error encountered in the data.

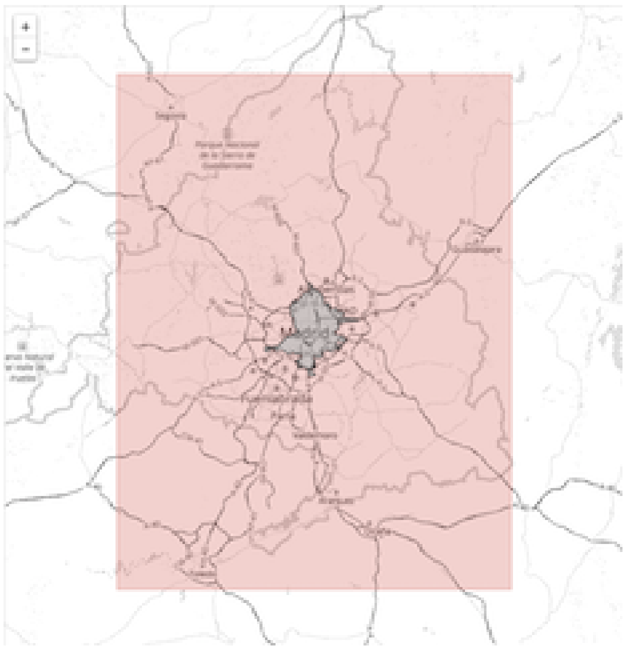


Figure 1: osm covered area

1.1.1 Street names

First we are going to clean the street names, so we will modify street abbreviations into the full name in order to have consistent street definitions across the dataset. We have to take into account that in spanish, contrary to english, the type of street is the first word. Finally we will use lower case in order to simplify the analysis.

We can find both abbreviations such as: - Av and Avd instead of Avenida (Avenue) - Pz instead of Plaza (Square)

and typos such as: - 'Call' instead of 'calle' (street) - 'carrera' instead of 'carretera' (Road), however "carrera" is a valid street type, this kind of errors can be easily missed, two of these cases have been found : 'CARRERA': set([u'CARRERA CARRETERA DE CRINON, S/N', 'CARRERA VALENCIA, KM 14.800'])})

Spanish roads use a common nomenclarture : a set of letters followed by a number e.g. A-6 M-604, we can match these entries using a regexp and add them to the type "carretera" (road). Assuming that we are using lowercase name we can use the folloing regexp

```
/[a-z]+\-[0-9]+/
```

Some fields are preceded by "CL" and then a valid address type, in these cases whe should remove the leading "CL". e.g. "CL AVENIDA" should be "avenida" (avenue).

The most common error is the use of streets names withouth the street identifier, so we will by default add street and the begining of these entries.

Next we found that C.N.-603-MARGEN should be "carretera n-603". Therefore we will change "C.N." into "carretera n-"

Another problem found was the presence of accented words, so in order to simplify the further analysis we will remove the accents of the words. e.g. we will use "travesia" instead of the gramatically correct "travesía", as this will simplify our database handling as well as filtering some errors in the inputs.

1.1.2 Leisures

We want also to check for kind of leisures are available and clean those fields. As we would like to make queries related with the number of green places in the city.

There are no big issues in this fields,the only fix we are going to apply is to equal “nature_reserve” to “natural_reserve” as they represent the same leisure. for our purpouse we could consider both “park” and “natural reserve”. We observed that most the entries for “leisures” are in english, but for the “pista de cemento” (cement pitch), fortunately this is not affecting our analysis so we are not going to modify these fields.

Nevertheless, the fields can be considered to be uniquely identified either in spanish or english, so we can keep them as they are as there is not any kind of leisure that is described in both languages at the same time.

1.1.3 Amenities

We want also to check what kind of amenities are available and clean those fields.

Unlike for leisures in amenities we can find that the same amenity is defined in some cases in both spanish and english, for example:

- “biblioteca” and “library”
- “espacio_trabajo” and “coworking_space”

It is convenient to use an unique language for the amenities definitions. Therefore, we will translate these entries into english.

1.1.4 Postal codes

The postal codes in spain consist of 5 numbers, of which the 2 first depend on the provinces, as we have 6 provinces in opur dataset, we can find 6 sets of postal codes:

- Madrid: 28XXX
- Toledo: 45XXX
- Avila: 05XXX
- Segovia: 40XXX
- Guadalajara: 19XXX
- Cuenca: 16XXX

From the sample dataset the following non conforming postcodes have been found:

- 21-23
- 2829
- 2839
- 288981
- 2945
- E28016

In some cases we can find that a postcode has a leading letter such as “E28016”, so we will remove this leading letters in order to have a consistent postcode dataset.

Therefore we can build a regexp to filter those postal codes that do not match one of the above. Using this approach we can see that there are postal codes that have not been properly set, for instance using 4 or 6 digits, in other cases there is a letter in the postal code or an invalid separator such as “-”. In order to have a clean dataset we will remove these entries.

Next, we are going to load our XML data into a more appropiate object representation that will be loaded into a JSON file in order to import it into the final mongoDB database. For this task we will use the python file create_json_from_osm.py , executing it from the command line.

1.1.5 Load into Mongo DB

We then import this dataset into mongodb using the mongoimport function.

```
mongoimport --db cities --collection madrid --file madrid_spain.osm.json
```

1.2 Data overview

This section contains basisc statistics about the dataset and the MongoDB database.
Size of the files:

- madrid_spain.osm : 858MB
- madrid_spain.osm.json : 962MB
- Number of documents in the database: 4,498,329

Data realted to the users:

- number of unique users: 2579

1.2.1 Unique users

We can see how many user have contributed to this dataset and which are the most actives ones:

```
In [2]: def get_db(db_name):
            from pymongo import MongoClient
            client = MongoClient('localhost:27017')
            db = client[db_name]
            return db
        db_madrid = get_db("cities").madrid
        pprint.pprint(len(db_madrid.distinct("created.uid")))
```

```
2579
```

1.2.2 Number of documents

```
In [3]: pprint.pprint(db_madrid.find().count())
```

```
4498329
```

1.2.3 Number of nodes

We can observe that this is the main kind of document accounting for 89% of the total documents in the database.

```
In [4]: pprint.pprint(db_madrid.find({"type":"node"}).count())
```

```
3993760
```

1.2.4 Number of ways

```
In [5]: pprint.pprint(db_madrid.find({"type":"way"}).count())
```

```
504110
```

1.2.5 Top postcodes

```
In [6]: result = [doc for doc in db_madrid.aggregate([{"$match":
                                                      {"address.postcode":{"$exists":1}}},
                                                  {"$group":{"_id":"$address.postcode",
                                                             "count":{"$sum":1}}},
                                                  {"$sort":{"count":-1}},{"$limit":5}])]
        pprint.pprint(result)
```

```
[{u'_id': [u'28330', u'28'], u'count': 5252},
 {u'_id': [u'28300', u'28'], u'count': 3968},
 {u'_id': [u'28017', u'28'], u'count': 3531},
 {u'_id': [u'28970', u'28'], u'count': 3135},
 {u'_id': [u'28830', u'28'], u'count': 2709}]
```

From the results above we can see that the postcode 28330 is the one with more nodes identified, it we take a look at the area in the map we can observe that this is a big but rather unpopulated area in the outskirts of Madrid.

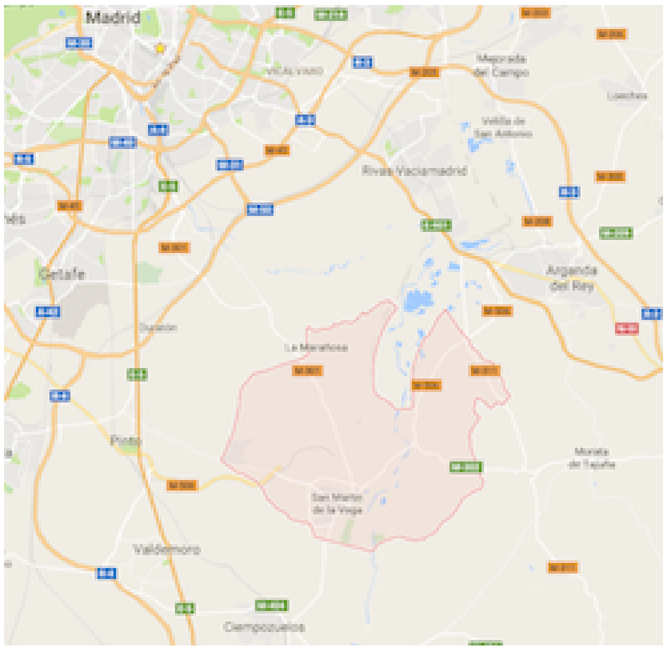


Figure 2: 28330 postcode area

So how is that this postcode is so popular in our database? maybe a very active user lives there. So we can list the users that populate nodes in that area. From this query we can see that most of the records in this area (95%) have been created by the user 3196761.

Thanks to that user this area is well populated.

```
In [7]: result = [doc for doc in db_madrid.aggregate([{"$match":{"address.postcode":'28330',
                                                                  "created.uid":{"$exists":1}}},
                                                      {"$group":{"_id":"$created.uid",
                                                                 "count":{"$sum":1}}},
                                                      {"$sort":{"count":-1}}])]
        pprint.pprint(result)
```

```
[{u'_id': u'3196761', u'count': 4992}, {u'_id': u'2240612', u'count': 260}]
```

1.2.6 Number of documents with an street field

```
In [8]: pprint.pprint(db_madrid.find({"address.street":{"$exists":1}}).count())
```

```
67757
```

1.2.7 Number of adresses with no postcode or invalid one

```
In [9]: result = [doc for doc in db_madrid.aggregate([{"$match":{"address.street":{"$exists":1}}},
                                                      {"$group":{"_id":"$address.postcode",
                                                      "count":{"$sum":1}}},
                                             {"$match":{"_id":None}}])]
        pprint.pprint(result[0]["count"])
```

```
15268
```

1.2.8 Most popular amenities

```
In [10]: result = [doc for doc in db_madrid.aggregate([{"$match":{"amenity":{"$exists":1}}},
                                              {"$group":{"_id":"$amenity",
                                                       "count":{"$sum":1}}},
                                              {"$sort":{"count":-1}},{"$limit":5}])]
         pprint.pprint(result)
```

```
[{u'_id': u'parking', u'count': 5513},
 {u'_id': u'restaurant', u'count': 3715},
 {u'_id': u'school', u'count': 3316},
 {u'_id': u'pharmacy', u'count': 2229},
 {u'_id': u'drinking_water', u'count': 2225}]
```

1.2.9 Most popular leisures

```
In [11]: result = [doc for doc in db_madrid.aggregate([{"$match":{"leisure":{"$exists":1}}},
                                              {"$group":{"_id":"$leisure",
                                                       "count":{"$sum":1}}},
                                              {"$sort":{"count":-1}},
                                              {"$limit":5}])]
         pprint.pprint(result)
```

```
[{u'_id': u'garden', u'count': 8180},
 {u'_id': u'pitch', u'count': 6302},
 {u'_id': u'swimming_pool', u'count': 5542},
 {u'_id': u'park', u'count': 3507},
 {u'_id': u'playground', u'count': 2029}]
```

1.2.10 Street of type "calle"

```
In [12]: result = [doc for doc in db_madrid.aggregate([
                                             {"$match":{"address.street":
                                                      {"$exists":1,
                                                       "$regex": '^Calle'}}},
                                             {"$group":{"_id":"Calle",
                                                      "count":{"$sum":1}}},
                                             {"$sort":{"count":1}}])]
         pprint.pprint(result[0]["count"])
```

```
57009
```

1.2.11 Most active users

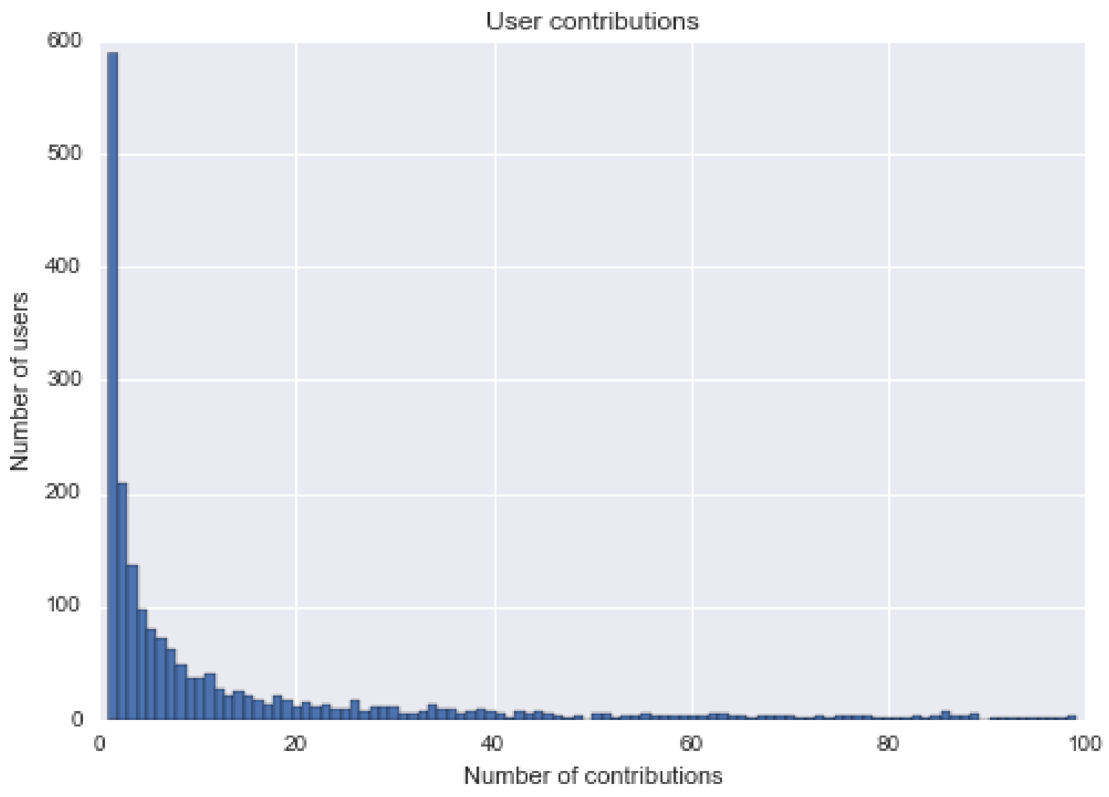
We can observer that the most active user accounts for the 10% of all the contributions.

```
In [13]: result = [doc for doc in db_madrid.aggregate([{"$match":{"created.uid":{"$exists":1}}},
                                             {"$group":{"_id":"$created.uid",
                                                      "name": {"$addToSet":"$created.user"},
                                                      "count":{"$sum":1}}},
                                             {"$sort":{"count":-1}},
                                             {"$limit":5}])]
      pprint.pprint(result)
```

```
[{u'_id': u'718437', u'count': 452285, u'name': [u'carlosz22']},
 {u'_id': u'468314', u'count': 350368, u'name': [u'rafaerti']},
 {u'_id': u'2245342', u'count': 260589, u'name': [u'cirdancarpintero']},
 {u'_id': u'777223', u'count': 194528, u'name': [u'Serfuen']},
 {u'_id': u'491515', u'count': 162791, u'name': [u'robertogeb']}]
```

We can see in the following histogram how users are distributed for those with less than 100 contributions.

```
In [14]: result = [doc for doc in db_madrid.aggregate([{"$match":{"created.uid":{"$exists":1}}},
                                               {"$group":{"_id":"$created.uid",
                                                          "name": {"$addToSet":"$created.user"},
                                                          "count":{"$sum":1}}},
                                               {"$match":{"count":{"$lt":100}}},
                                               {"$sort":{"count":-1}}])]
        y = [x["count"] for x in result]
        plt.hist( y,100)
        plt.title("User contributions")
        plt.xlabel('Number of contributions')
        plt.ylabel('Number of users')
        plt.show()
```



1.2.12 Number of green spaces

```
In [15]: result =  db_madrid.find({"leisure":
                                {"$in":["park","natural_reserve","garden"]}}).count()
         pprint.pprint(result)
```

```
11697
```

1.3 Additional ideas

In this dataset the address field uses the spanish labeling whereas most leisure data use english, it would be interesting to use a common language e.g. English. However, programmatic translations are not an easy task and can lead to other errors, especially when different languages are used in a same dataset where false friend may cause tranlation errors for example.

It would also be interesting to assess the closeness of some nodes to others for example, which bars are within 200m of a park? but this kind of queries are not stright forward to perform using MongoDB or an SQL database, maybe a graph database may help in this task.

Also wrong postcodes could be corrected using the address in order to obtain the postcode using a third party API that relates addresses with postcodes.

From the analysis of the contributions in the postcode 28330 area we can see how important are active users in order to have quality data in an area. One question that can arise from this observation is to quantify the quality of the data provided by the users. for e.g. we can ask, which users provide postcodes in their inputs? As in the query below, and we can observed that although the top contributor remains 'carlosz22' we can see that 'JavierSp' is now second, this is the user that contributed to the postcode 28330 area. a further analysis could be to see how the users contribute to different areas for example using the postcodes as references.

```
In [16]: result = [doc for doc in db_madrid.aggregate([{"$match":{"created.uid":
                                                                  {"$exists":1},
                                                                "address.postcode":
                                                                  {"$exists":1}}},
                                                      {"$group":{"_id":"$created.uid",
                                                                 "count":{"$sum":1}}},
                                                      {"$sort":{"count":-1}},
                                                      {"$limit":5}])]
         pprint.pprint(result)
```

```
[{u'_id': u'718437', u'count': 8454},
 {u'_id': u'3196761', u'count': 4995},
 {u'_id': u'1334676', u'count': 3785},
 {u'_id': u'2277975', u'count': 3655},
 {u'_id': u'777223', u'count': 3232}]
```

Another question we could make is to see who are the contributors that have set more number of different postcodes. i.e the contributors to more different areas in Madrid. We can see that the top user in this list has contributed to more than 100 areas, a great traveller.

```
In [17]: result = [doc for doc in db_madrid.aggregate([
                                                      {"$match":{"created.uid":{"$exists":1},
                                                                 "address.postcode":{"$exists":1}}},
                                                      {"$group":{"_id":"$created.uid",
                                                                 "postcodes":
                                                                    {"$addToSet":"$address.postcode"},
                                                                 "count":{"$sum":1}}},
                                                      {"$unwind":"$postcodes"},
                                                      {"$group":{"_id":"$_id","count":{"$sum":1}}},
                                                      {"$sort":{"count":-1}},
                                                      {"$limit":5}])]
                pprint.pprint(result)
```

```
[{u'_id': u'220932', u'count': 101},
 {u'_id': u'316694', u'count': 77},
 {u'_id': u'2153421', u'count': 70},
 {u'_id': u'26599', u'count': 64},
 {u'_id': u'42055', u'count': 57}]
```

It can be interest to see how different areas compare, for example we can which areas have more restaurants. However it should be taken into account thath not all areas are equal in size. and that some areas might not be well populated with data.

```
In [18]: result = [doc for doc in db_madrid.aggregate([{"$match":
                                                                {"amenity":"restaurant",
                                                                 "address.postcode":{"$exists":1}}},
                                                     {"$group":{"_id":"$address.postcode",
                                                                "count":{"$sum":1}}},
                                                     {"$sort":{"count":-1}},{"$limit":5}])]
         pprint.pprint(result)
```

```
[{u'_id': [u'28004', u'28'], u'count': 219},
 {u'_id': [u'28013', u'28'], u'count': 192},
 {u'_id': [u'28015', u'28'], u'count': 139},
 {u'_id': [u'28012', u'28'], u'count': 111},
 {u'_id': [u'28005', u'28'], u'count': 71}]
```

Another interesting analysis would be to compare the different top amenities in difference places, for example how the number of schools compare between different cities, etc.. This will imply a very large dataset and take into account other data in order to normalize the values e.g. schools per person.

1.4 Conclusion

After the review of the dataset it is clear that there are typos in the manual inputs even for postcodes that are easy to validate during the input phase, maybe it would be nice to have some schema and field validation when users incorporate data into the OSM dataset.

Furthermore, we can see that the language may impact the dataset as some fields tend to be filled in the local language whereas others such as the kind of leisure tend to be filled in english although it is not always the case and some field have mixed inputs which may difficult querying the database for users not familiar with the local language.

Finally, we can see that some areas are more complete and this seems to be highly related to the active users in that area, so it would be nice to have a way to incentivate or recompensate more active users, for example with badgets or lists of top users. This is important because the quality of the data depends directly on the users.