OpenStreetMap - Data Wrangling with Python and MongoDB

By Sarah Hosking

Map area: Central Paris, France

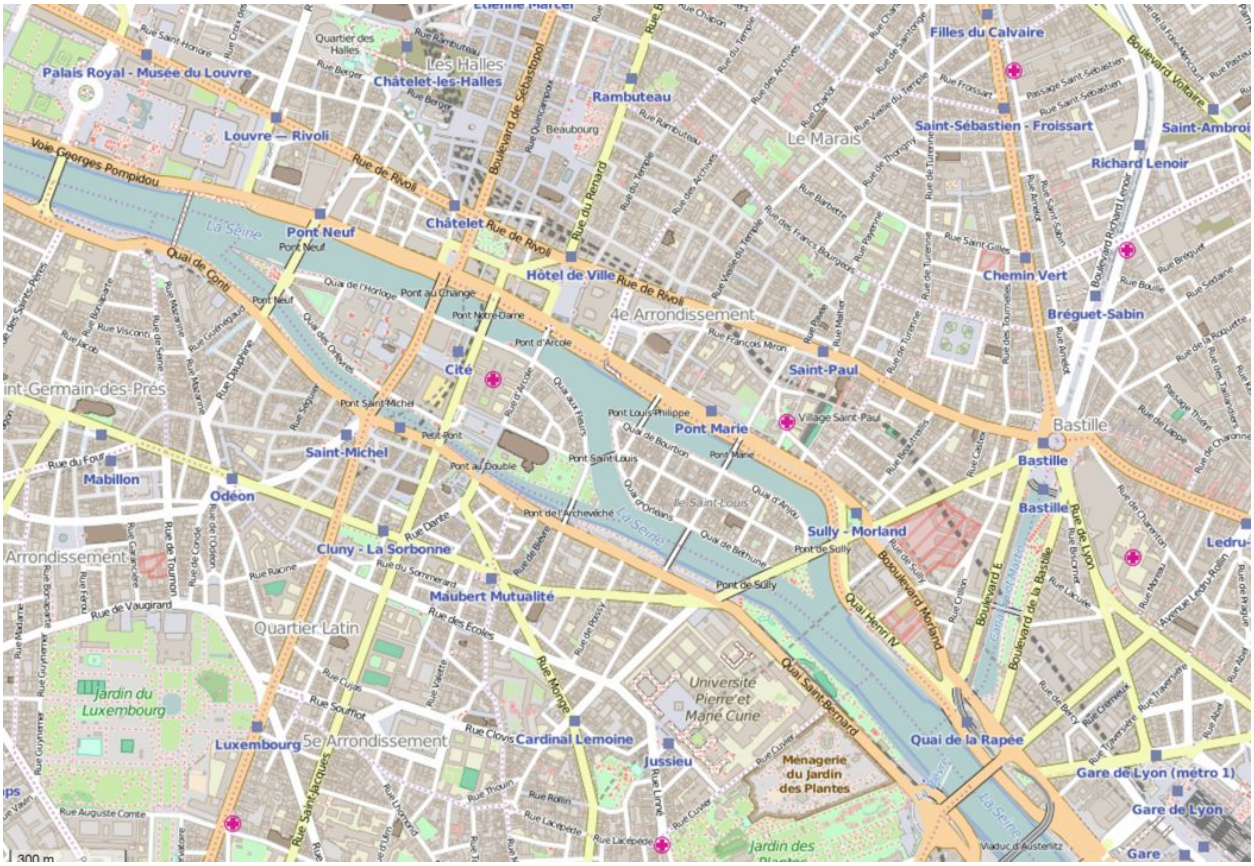

Table of Contents

1.0 Problems encountered in the map

The most obvious issues were:

- incomplete data
- inconsistent formats

1.1 Incomplete data

Many amenities are not included. For example, a quick visual inventory of the 200m along my block shows 3 restaurants, a mini-market, and a Vélib station. Of these, only 1 restaurant was listed in the map.

When querying the data, when I ran a count of amenities by type, there were 967 restaurants in my map sample. By contrast, TripAdvisor lists 13,955 restaurants in Paris. While my sample doesn't encompass the entire city of Paris, it does capture the centre, which has the densest population of restaurants.

An interesting tag attribute I noticed was "k"="wheelchair". This is extremely useful in a city that is horrible to navigate in a wheelchair: cobblestones, stairs, few elevators, tiny stores and restaurants.

However, a breakdown by postal code shows that of the 967 restaurants, 853 of them do not have postal code information. So it's little surprise that even fewer restaurants have information on wheelchair accessibility.

```
#query of restaurants, total & wheelchair-accessible, by arrondissement (postcode)
results = db.paris.aggregate([
                                {"$match" : {"amenity" : "restaurant"}},
                                {"$group" : {"_id" : "$address.postcode",
                                             "wh_count" : {"$sum": {"$cond":

                                                                    [{"$eq":["$wheelchair","yes"]},1,0}}
                                                                    ,
                                             "total_count" : {"$sum" : 1}}},

                                {"$project" : {"postcode" : "$postcode",
                                               "wheelchair_friendly" : "$wh_count",
                                               "total_restos" : "$total_count"}}])


#returned by query:
{u'_id': u'75003', u'total_restos': 1, u'wheelchair_friendly': 0}
{u'_id': u'75013', u'total_restos': 1, u'wheelchair_friendly': 0}
```

```
{u'_id': None, u'total_restos': 853, u'wheelchair_friendly': 9}
{u'_id': u'75011', u'total_restos': 7, u'wheelchair_friendly': 0}
{u'_id': u'75015', u'total_restos': 2, u'wheelchair_friendly': 0}
{u'_id': u'75004', u'total_restos': 32, u'wheelchair_friendly': 1}
{u'_id': u'75007', u'total_restos': 6, u'wheelchair_friendly': 0}
{u'_id': u'75001', u'total_restos': 1, u'wheelchair_friendly': 0}
{u'_id': u'75005', u'total_restos': 15, u'wheelchair_friendly': 1}
{u'_id': u'75006', u'total_restos': 22, u'wheelchair_friendly': 2}
{u'_id': u'75012', u'total_restos': 19, u'wheelchair_friendly': 0}
{u'_id': u'75014', u'total_restos': 7, u'wheelchair_friendly': 0}
```

1.2 Inconsistent formatting of values

Phone numbers had a lot of variation: no spaces, no country code, some with a zero before the city code, others without. If pulling this information into a mobile app, this would be annoying: a properly formatted phone number means you can click the number in the app and your phone will automatically call that number.

```
#sample returned by initial audit:
({u'11\xe8me Art': '+33143798169',
  u'A premi\xe8re vue': '+33143577466',
  'APEF': '0158514650',
  'ASVS Alarme Auto-surveillance': '0140010450',
  u'Acad\xe9mie Hotel': '+33145498000',
  'Allo Sushi': '01 44 61 93 73',
  'Amareto': '+33146280707',
  'Arnaout': '+33143431461',
  'Artmedium Cours de Dessin': '06 69 00 48 20',
  'Atelier 3': '+336 99 76 40 00',
  'Atelier de Paris': '01 44 73 83 50',
  'Atlassib': '01 44 74 64 01',
  'Au 105': '+33143453625',
```

To clean phone numbers, I needed to do the following before importing to MongoDB:

- verify the OSM standard in the documentation
- check for validity against a regex expression for the ITU-T E.123 pattern.
- for invalid formats, I removed the “+33” (if present)
- removed any spaces or periods
- used string slicing to arrange in valid format

```
#phone cleaning functions
import re
valid_phone = re.compile(r'^\+33\s[1-9]\s\d{2}\s\d{2}\s\d{2}\s\d{2}$')

def invalid_phone(n):
```

```
    p = valid_phone.search(n)
    if p:
        pass
    else:
        return n

def reformat(n):
    cc = "+33 "

    #remove spaces & periods
    n = n.replace(' ', '').replace('.', '')

    #if phone number is 9 char or longer, apply ITU-T E.123 standard
    if len(n) >= 9:
        n = cc + n[-9] + " " + n[-8:-6] + " " + n[-6:-4] + " " + n[-4:-2] + " " +
n[-2:]
    else:
        pass
    return n


def fix_phone(n):
    if n.startswith("+33"):
        n = n[3:]
        n = reformat(n)
    elif n.startswith("0"):
        n = reformat(n)
    else:
        pass
    return n

def clean_phone(n):
    if invalid_phone(n):
        clean = fix_phone(n)
        return clean
    else:
        pass
```

#sample after cleaning

```
{u'11\xe8me Art': '+33 1 43 79 81 69',
 u'A premi\xe8re vue': '+33 1 43 57 74 66',
 'APEF': '+33 1 58 51 46 50',
 'ASVS Alarme Auto-surveillance': '+33 1 40 01 04 50',
 u'Acad\xe9mie Hotel': '+33 1 45 49 80 00',
 'Allo Sushi': '+33 1 44 61 93 73',
 'Amareto': '+33 1 46 28 07 07',
 'Arnaout': '+33 1 43 43 14 61',
 'Artmedium Cours de Dessin': '+33 6 69 00 48 20',
```

```
 'Atelier 3': '+33 6 99 76 40 00',
 'Atelier de Paris': '+33 1 44 73 83 50',
 'Atlassib': '+33 1 44 74 64 01',
 'Au 105': '+33 1 43 45 36 25'
```

2.0 Data Overview

XML: 58.4 MB
JSON: 59.4 MB

#Count of documents

```
db.paris.find().count()
245530
```

#how many nodes?

```
219981
```

#how many ways?

```
25539
```

#Count of documents by arrondissement

```
results = db.paris.aggregate([ #{"$match" : {"type" : "node"}},

                                {"$group" : {"_id" : "$address.postcode",
                                             "count" : {"$sum" : 1}}},
                                {"$project" : {"_id" : 0,
                                               "postcode" : "$_id",
                                               "count" : "$count"}},
                                {"$sort" : {"count" : -1}},
                              ])
```

#returns

```
{u'count': 244786, u'postcode': None}
{u'count': 164, u'postcode': u'75004'}
{u'count': 150, u'postcode': u'75006'}
{u'count': 104, u'postcode': u'75007'}
{u'count': 101, u'postcode': u'75005'}
{u'count': 79, u'postcode': u'75012'}
{u'count': 50, u'postcode': u'75011'}
{u'count': 44, u'postcode': u'75015'}
{u'count': 20, u'postcode': u'75014'}
{u'count': 19, u'postcode': u'75001'}
{u'count': 8, u'postcode': u'75003'}
{u'count': 4, u'postcode': u'75013'}
{u'count': 1, u'postcode': u'75140'}
```

```
#count of unique users
db.paris.distinct("created.user").__len__()
702

#count of unique data sources
db.paris.distinct("source").__len__()
241

#count of distinct amenities
db.paris.distinct("amenity").__len__()
91
```

3.0 Digging a bit deeper

3.1 What are the most common amenities?

Restaurants are not a surprising 1st place, but I never would have expected “bench” and “vending_machine” to place 2nd and 3rd, respectively.

I find it telling of what most interests mappers: “bicycle_parking” makes the top 10 list, but car parking does not. While the Mayor of Paris would be thrilled if that really were the case, car parking (at least, on-street parking spaces) is still far, far more common than bike parking.

```
#Top 10 amenities
results = db.paris.aggregate([
                                {"$group" : {"_id" : "$amenity",
                                             "count" : {"$sum" : 1}}},
                                {"$project" : {"_id" : "$_id",
                                               "count" : "$count"}},
                                {"$sort" : {"count" : -1}},
                                {"$limit" : 10}
                              ])

#returns
{u'_id': None, u'count': 240404}
{u'_id': u'restaurant', u'count': 966}
{u'_id': u'bench', u'count': 412}
{u'_id': u'vending_machine', u'count': 301}
{u'_id': u'waste_basket', u'count': 300}
{u'_id': u'cafe', u'count': 295}
{u'_id': u'bicycle_parking', u'count': 241}
{u'_id': u'post_box', u'count': 197}
```

```
{u'_id': u'fast_food', u'count': 167}
{u'_id': u'bicycle_rental', u'count': 151}
```

3.2 Where does this info come from?

82% of documents do not have a source value.
14% of documents are based on data from the taxman, from *cadastre-dgi-fr source : Direction Générale des Impôts* (Director General of Income Tax). Apparently there is a project to automate the conversion of PDF cadastre documents into vector then osm data.

#Top 10 data sources

```
results = db.paris.aggregate([
                                {"$group" : {"_id" : "$source",
                                             "count" : {"$sum" : 1}}},
                                {"$project" : {"_id" : 0,
                                               "source" : "$_id",
                                              "count" : "$count"}},
                                {"$sort" : {"count" : -1}},
                                {"$limit" : 10}
                              ])
```

#returns [edited for readability]

```
{u'count': 200700, u'source': None}
{u'count': 23609,
 u'source': u'cadastre-dgi-fr source : Direction Générale des Impôts -
Cadastre. Mise à jour : 2010'}
{u'count': 5775,
 u'source': u'cadastre-dgi-fr source : Direction Générale des Impôts -
Cadastre. Mise à jour  : 2011'}
{u'count': 5215, u'source': u'opendata.paris.fr'}
{u'count': 4520,
 u'source': u'cadastre-dgi-fr source : Direction Générale des Impôts - Cadastre
; Mise à jour : 2010'}
{u'count': 912, u'source': u'survey'}
{u'count': 587,
 u'source': u'cadastre-dgi-fr source : Direction Générale des Impôts - Cadastre
; mise à jour : 2009'}
{u'count': 587,
 u'source': u'cadastre-dgi-fr source : Direction Générale des Impôts -
Cadastre. Mise à  jour : 2009'}
{u'count': 268, u'source': u'Mapillary 2015'}
{u'count': 266, u'source': u'Bing'}
```

What I find bizarre is how even data coming from government income tax references still doesn't include address information.

I filtered the data to only return documents from the top source, cadastre-dgi-fr source : Direction Générale des Impôts - Cadastre. Mise à jour : 2010 where the “address” field existed.

```
#If cadastre is source, is there address info?
results = db.paris.aggregate([  {"$match" : {"source" : "cadastre-dgi-fr source
: Direction Générale des Impôts - Cadastre. Mise à jour : 2010",
                                              "address" : {"$exists" : 1}}},
                                {"$group" : {"_id" : "$address.postcode",
                                             "count" : {"$sum" : 1}}},
                                {"$project" : {"_id" : 0,
                                               "postal code" : "$_id",
                                              "count" : "$count"}},
                                {"$sort" : {"count" : -1}},
                               ])
```

Out of 23609 documents tagged with this source, only 9028 (38%) have an address field. And of those 9028 documents with address info, only 63 (0.6%) include a postal code.

```
#returns
{u'count': 8965, u'postal code': None}
{u'count': 24, u'postal code': u'75006'}
{u'count': 22, u'postal code': u'75004'}
{u'count': 6, u'postal code': u'75001'}
{u'count': 4, u'postal code': u'75007'}
{u'count': 3, u'postal code': u'75005'}
{u'count': 1, u'postal code': u'75011'}
{u'count': 1, u'postal code': u'75015'}
{u'count': 1, u'postal code': u'75140'}
{u'count': 1, u'postal code': u'75012'}
```

Yet the data certainly exists. The government of France has put the cadastre online, and to query it you must specify a postal code and street address. So there seem to be some data limitations with the project to automatically import cadastre information into OSM.

https://www.cadastre.gouv.fr/scpc/afficherCarteFeuille.do?f=ZP105000AB01&dontSaveLastForward&keepVolatileSession=

3.3 Who is updating the map?

```
#Top 10 users
results = db.paris.aggregate([
                                    {"$group" : {"_id" : "$created.user",
                                                 "count" : {"$sum" :
1}}},
                                    {"$project" : {"_id" : 0,
                                                   "user" : "$_id",
                                                  "count" : "$count"}},
                                    {"$sort" : {"count" : -1}},
                                    {"$limit" : 10}
                                ])
#returns
{u'count': 80927, u'user': u'Pieren'}
{u'count': 28668, u'user': u'Mawie'}
{u'count': 25556, u'user': u'STA'}
{u'count': 19217, u'user': u'maouth-'}
{u'count': 10394, u'user': u'mat'}
{u'count': 8256, u'user': u'osmmaker'}
{u'count': 6884, u'user': u'cquest'}
{u'count': 6708, u'user': u'VinceBot'}
{u'count': 5741, u'user': u'cmif4'}
{u'count': 4144, u'user': u'dhuyp'}
```

3.4 What amenities are they adding?

For example, are the people who are adding bicycle_rentals (velibs) the same people who add the bicycle_parking, or car_rentals?

```
#top 5 users by amenity
results = db.paris.aggregate([
                                {"$match" : {
                                             "amenity" : "bicycle_rental"}},

                                {"$group" : {"_id" : "$created.user",
                                             "count" : {"$sum" : 1}}},
                                {"$project" : {"_id" : 0,
                                               "created_by" : "$_id",
                                               "count" : "$count"}},
                                {"$sort" : {"count" : -1}},
                                {"$limit" : 5}
```

```
                                ])
#returns
{u'count': 91, u'created_by': u'GeorgeKaplan'}
{u'count': 20, u'created_by': u'Truchin'}
{u'count': 11, u'created_by': u'wheelmap_visitor'}
{u'count': 5, u'created_by': u'cquest'}
{u'count': 5, u'created_by': u'Baptiste G'}
```

So does the top user (GeorgeKaplan) like to add other bike-related amenities?
In short, yes.

```
#Top 5 amenities added by user
results = db.paris.aggregate([
                                    {"$match" : {"created.user" : "GeorgeKaplan",
                                                 "amenity" : {"$exists" : 1}}},
                                    {"$group" : {"_id" : "$amenity",
                                                 "count" : {"$sum" : 1}}},
                                    {"$project" : {"_id" : 0,
                                                   "amenity" : "$_id",
                                                   "count" : "$count"}},
                                    {"$sort" : {"count" : -1}},
                                    {"$limit" : 5}
                                ])

#returns
{u'amenity': u'bicycle_parking', u'count': 183}
{u'amenity': u'bicycle_rental', u'count': 91}
{u'amenity': u'school', u'count': 35}
{u'amenity': u'restaurant', u'count': 18}
{u'amenity': u'cafe', u'count': 7}
}
```

However, the same did not apply when I ran the same queries for the top user (“zerpou”) for “car_rental” amenities - which furthers my hunch that map enthusiasts and bike enthusiasts often overlap.

```
#zerpou user’s (top car_rental) top 5 amenities
{u'amenity': u'pharmacy', u'count': 125}
{u'amenity': u'car_rental', u'count': 48}
{u'amenity': u'school', u'count': 34}
{u'amenity': u'post_box', u'count': 12}
{u'amenity': u'college', u'count': 7}
```

3.5 Where on earth are these public toilets?

To close on a light note…

When I ran the full query of distinct amenities, without a limit, I saw there are 91 public toilets in Paris. Where on earth are they???

```
#where ARE all these public toilets?!
results = db.paris.aggregate([ {"$match" : {"amenity" : "toilets"}},

                               {"$group" : {"_id" : "$pos",
                                            "count" : {"$sum" : 1}}},
                               {"$project" : {"_id" : 1,
                                              "name" : "$name",

                                              "count" : "$count"}},
                            ])
```

Unfortunately, the only location info for them is in the “pos” field.

```
#returns
{u'_id': None, u'count': 11}
{u'_id': [48.8575368, 2.3603524], u'count': 1}
{u'_id': [48.8445095, 2.3768246], u'count': 1}
...
```

4.0 Additional Ideas

Enhancing the Paris with more meta data combined with proximity search (NEAR:) could help the city promote environmental goals, improve accessibility, and facilitate commerce/tourism.

To encourage multi-modal transport: where is the closest Vélib (bike rental) stations around a given transit stop? This data already exists, it's simply a matter of combining them in one map.

Accessibility: I want to dine at a wheelchair-accessible restaurant/shop at a wheelchair-accessible store, is there a wheelchair-accessible transit stop nearby? While many transit stops do indicate if they're accessible, the same is not the case for shops and restaurants.

(A "nearest public washroom" would also be extremely welcome proximity search option as well!)

Choosing a restaurant: use permit data to add a "terrasse" (patio) tag so it can answer the all-important summertime question, which restaurants have a sidewalk terrasse, or better yet a hidden one in the courtyard? The data exists, but I am not sure how easily it can be accessed.

5.0 Conclusion

While it was interesting to dig into the data for central Paris, the data was so incomplete it was hard to take it seriously. The buildings and streets are there, but businesses, restaurants, address information….so much was missing.

On the other hand, when I initially looked at the data I found it quite clean. What addresses there were were entered consistently. I had to hunt a bit before I found a good candidate for clean-up in phone numbers.