# OpenStreetMap Analysis Report[¶](#OpenStreetMap-Analysis-Report)

Date:27/09/2017

**Author**: Mamadou Diallo

**Email:** mams.diallo@gmail.com

The project consists in the following files:

Source code for auditing and cleaning (separate files): '**wrangle.py**'

The .osm data set file '**Nanterre.osm**' is stored in compressed format here [Nanterre.osm.zip](https://github.com/mamsdiallo/DataWranglingOpenStreetMapData/blob/master/Nanterre.osm.zip)

The .osm Sample file which is part of the map region used: '**sampleNanterre.osm**'. It is stored in compressed format here [sampleNanterre.osm.zip](https://github.com/mamsdiallo/DataWranglingOpenStreetMapData/blob/master/sampleNanterre.osm.zip)

File containing the list of references (Web sites, books, forums, blog posts, github repositories): '**References.txt**' is stored here [References.txt](https://github.com/mamsdiallo/DataWranglingOpenStreetMapData/blob/master/References.txt)

## Data Wrangling Process[¶](#Data-Wrangling-Process)

The process iterates through the following activities in that order.

**1. GATHERING**:

I choose the dataset located in Nanterre because I lived there for years before I moved to my current place.

Nanterre is located 5 km from Paris.

It has been extracted from Map Zen.

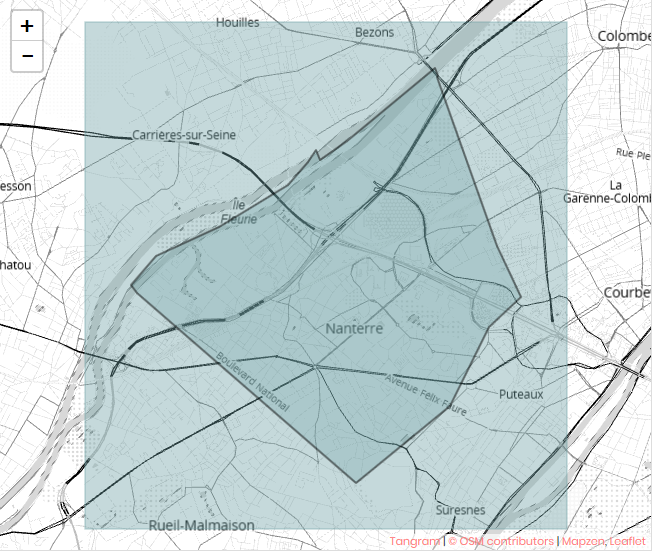
The bounding box is:

bounds {'minlat': '48.8688172',

'minlon': '2.1608126',

'maxlat': '48.9260304',

'maxlon': '2.2435479'}



**2. CLEANING**:

The following activities are conducted:

a- Audit the data in terms of Validity, Accuracy, Completeness, Consistency and Uniformity

Script to launch, by first activating the environment:

activate DAND

python wrangle.py

b- Create a data cleaning plan

- Identify Causes

- Define Operations

- Test

c- Execute the plan

d- Manual correct

- use of dedicated tool

- modification directly in the cvs files

**3. Storing**:

I choose SQL for completing the project.

From cleaned csv files, it is stored in SQL format stored in a SQLite3 Database.

**4. Analysis**:

This document '**Wrangle OpenStreetMap Data Notebook.ipynb**' provides the analysis

**Additional tooling**:

Manual correction is tedious and we have the opportunity to use powerfull tools like:

[OpenRefine](http://openrefine.org/)

## Process the Dataset[¶](#Process-the-Dataset)

### Issue: Header in cvs files[¶](#Issue:-Header-in-cvs-files)

Sqlite Commands to import and error messages:

sqlite> .import nodes.csv nodes

nodes.csv:1: INSERT failed: datatype mismatch

sqlite> .import ways.csv ways

ways.csv:1: INSERT failed: datatype mismatch

Extract of source file node.csv

id,lat,lon,user,uid,version,changeset,timestamp

644169,48.9049071,2.1959349,osmmaker,210173,3,26474158,2014-11-01T02:01:56Z

644179,48.913021,2.1659623,Marcussacapuces91,37548,2,432639,2008-12-16T23:03:54Z

645164,48.9071846,2.1923733,jmorais,5404,1,191925,2007-01-11T18:35:03Z

#### Solution:[¶](#Solution:)

Cutting all headers in csv files solved the problem. Since we have only 4 files, it was possible to do it manually.

### Issue: Encoding[¶](#Issue:-Encoding)

Problematic tags in the dataset Nanterre.osm:

<tag k="addr:street" v="Allée René Descartes"/>

<tag k="addr:street" v="Place de l&#39;Hôtel de Ville"/>

<tag k="addr:street" v="Avenue du Château de Malmaison"/>

SQL COMMANDS

SELECT \* FROM nodes\_tags

WHERE type="addr" AND key="street"

LIMIT 3;

Here are some problematic results:

id|key|value|type

150856317|street|AllΘe RenΘ Descartes|addr

180968674|street|Place de l'H⌠tel de Ville|addr

275556597|street|Avenue du ChΓteau de Malmaison|addr

#### Solution:[¶](#Solution:)

Change the code page to UTF-8 within powershell

Powershell command:

>chcp

Active code page: 850

>chcp 65001

Active code page: 65001

Change encoding to UTF-8 within sqlite with PRAGMA statement.

SQL command:

sqlite>PRAGMA encoding="UTF-8";

SELECT \* FROM nodes\_tags

WHERE type="addr" AND key="street"

LIMIT 3;

Here are the three results after changing parameters:

id|key|value|type

150856317|street|Allée René Descartes| addr

180968674|street|Place de l'Hôtel de Ville|addr

275556597|street|Avenue du Château de Malmaison|addr

### Issue: Variation in city names[¶](#Issue:-Variation-in-city-names)

SELECT key,value, COUNT(\*) AS Nb FROM ways\_tags

WHERE key='city'

GROUP BY value

ORDER BY value;

Result:

key|value|Nb

city|Bezons|7

city|Carrières-sur-Seine|4

city|Chatou|1

city|Colombes|2

city|Houilles|112

city|La Garenne-Colombes|3

city|Nanterre|1070

city|Puteaux|11

city|Rueil Malmaison|1

city|Rueil-Malmaison|16

city|Suresnes|2

Total Errors: 1 in ways\_tags Rueil Malmaison vs Rueil-Malmaison

SELECT key,value, COUNT(\*) AS Nb FROM nodes\_tags

WHERE key='city'

GROUP BY value

ORDER BY value;

Result:

key|value|Nb

city|Bezons|5

city|Carrières-sur-Seine|1

city|Chatou|4

city|Colombes|6

city|Courbevoie|16

city|Houilles|15

city|La Garenne Colombes|3

city|La Garenne-Colombes|4

city|NANTERRE|3

city|Nanterre|1266

city|Paris La Défense Cedex|1

city|Puteaux|98

city|RUEIL MALMAISON|2

city|Rueil-Malmaison|96

city|Suresnes|285

city|houilles|1

Total Errors: 4 in nodes\_tags

La Garenne Colombes vs La Garenne-Colombes

RUEIL MALMAISON vs Rueil-Malmaison

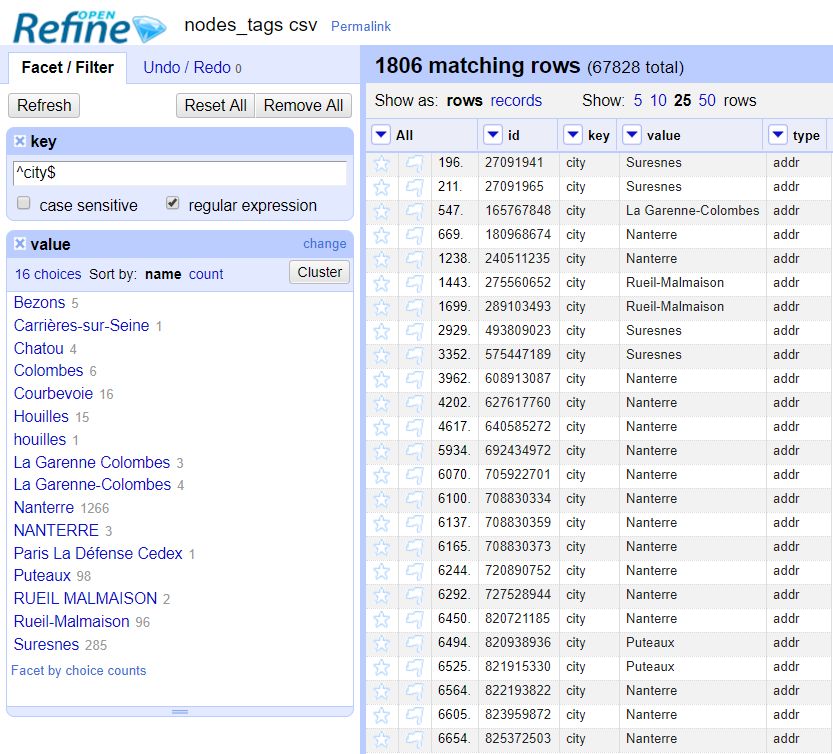
Houilles vs houilles

Nanterre vs NANTERRE

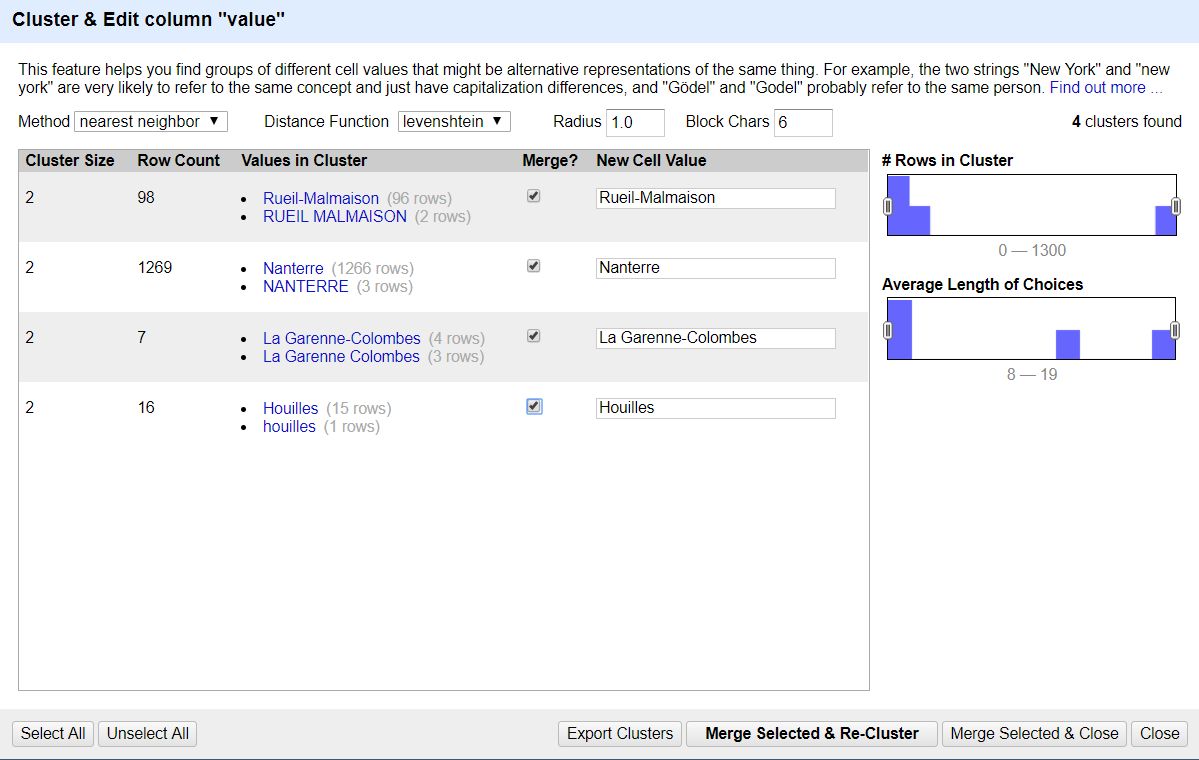
#### Solution:[¶](#Solution:)

The solution is based on post-processing operation from the two excel files (ways\_tags and nodes\_tags). Then, I apply mass correction - 1806 rows - with OpenRefine from the clustering feature.

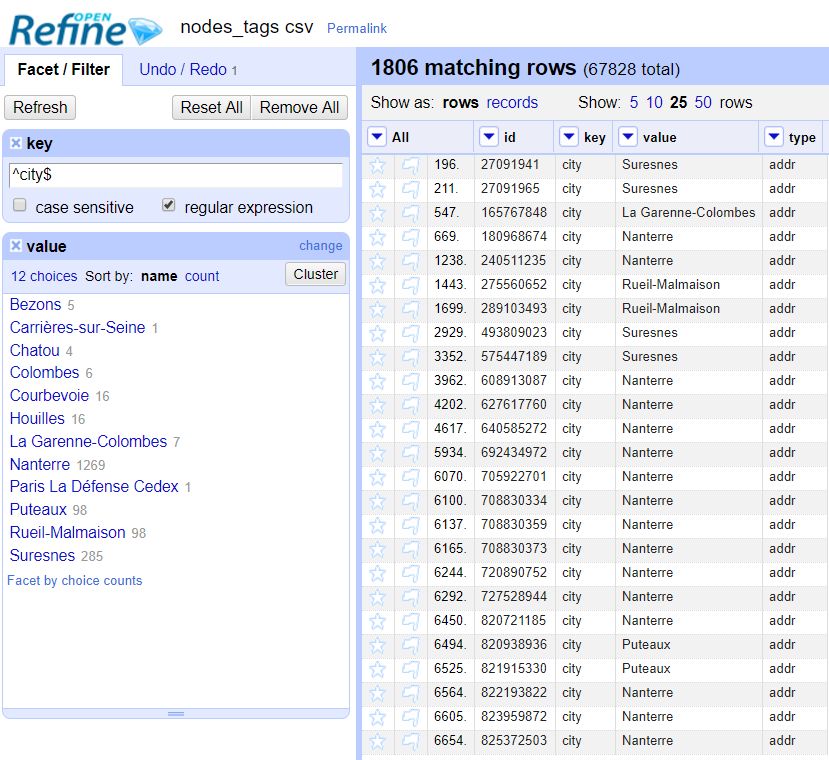
Here below, filtered elements for nodes\_tags.csv with key='city'. We have for sure the same results and numbers for values:



Use of the clustering feature (grouping and suggestion of merge done automatically) for mass correction:



Result of merging:



### Issue: variation with phone numbers[¶](#Issue:-variation-with-phone-numbers)

Phone numbers are not formated uniformly.

SELECT DISTINCT value FROM nodes\_tags WHERE key='phone';

Some remarkable outputs:

value

3631

39.92

01.49.61.06.34

+33 1 41193500

+33 1 47 78 95 99

08.92.68.69.25

0147493186

01.46.69..06.17

09 54 58 67 66

We have phone numbers:

* with country code and space separator (e.g. +33 1 47 78 95 99)
* with country code and no separator (e.g. +33 1 41193500)
* without country code and space separator (e.g. 09 54 58 67 66)
* without country code and dot separator (e.g. 01.49.61.06.34)
* without country code and no separator (e.g. 0147493186)
* typos (e.g. 01.46.69..06.17)
* special numbers (e.g. 3631 or 39.92)

#### Solution:[¶](#Solution:)

I chose to use country code with space as separator for readability: +33 N NN NN NN NN

Where +33 is the French country code. There should be 9 digits after the country code

Exception: special numbers. I chose to not use country code. 3631 => 3631 39.92 => 3992

The code is the following:

def update\_phone(phone\_num):

import string

whitelist = string.letters + string.digits + ' ' + '+'+';'

new\_s = ''

for char in phone\_num:

if char == ' ':

new\_s += ''

elif char in whitelist:

new\_s += char

else:

new\_s += ''

new\_s = new\_s.strip()

if new\_s[:4]=='0033':

new\_s = '+33'+new\_s[4:]

elif new\_s[:1]=='0':

new\_s = '+33'+new\_s[1:]

# change format for readability

if len(new\_s) == 4:

# special phone numbers

phone\_num\_parts = []

phone\_num\_parts.append(new\_s)

return ''.join(phone\_num\_parts)

else:

# regular phone numbers

phone\_num\_parts = []

phone\_num\_parts.append(new\_s[:3])

phone\_num\_parts.append(' ')

phone\_num\_parts.append(new\_s[3:4])

phone\_num\_parts.append(' ')

phone\_num\_parts.append(new\_s[4:6])

phone\_num\_parts.append(' ')

phone\_num\_parts.append(new\_s[6:8])

phone\_num\_parts.append(' ')

phone\_num\_parts.append(new\_s[8:10])

phone\_num\_parts.append(' ')

phone\_num\_parts.append(new\_s[10:12])

return ''.join(phone\_num\_parts)

The remarkable examples become:

value

3631

3992

+33 1 49 61 06 34

+33 1 41 19 35 00

+33 1 47 78 95 99

+33 8 92 68 69 25

+33 1 47 49 31 86

+33 1 46 69 06 17

+33 9 54 58 67 66

The following code used within 'shape\_element' function allows to process list of phone numbers separated by ';' delimiter:

phone = data["value"]

lst = phone.split(';')

s = []

for item in lst:

s.append(update\_phone(item))

ph = ';'.join(s)

data["value"] = ph

Here is the transformation:

'01.47.21.14.27; 01.47.24.08.68' => '+33 1 47 21 14 27;+33 1 47 24 08 68'

#### Way forward:[¶](#Way-forward:)

What is done for phone could be easily done for Fax.

Here is the request:

SELECT key,value FROM nodes\_tags

WHERE key='fax'

LIMIT 10;

Some results:

key|value

fax|01.47.21.81.21

fax|+33 1 41193515

fax|01.47.29.54.59

fax|01.41.37.52.53

fax|01.47.25.46.24

fax|+33 1 47 21 80 02

fax|01.47.08.24.47

fax|01.46.91.92.65

fax|01.47.21.29.33

fax|01.47.21.43.40

### Issue: Street types[¶](#Issue:-Street-types)

Do no confuse with next issue "Variation in street name"

In France, the street types are placed at the beginning.

Thus, we have to catch the first word.

The type of streets I met in Nanterre are the following:

expected = [u'Aire', u'Allée', u'Avenue',u'Boulevard', u'Chemin', u'Cours', \

u'Esplanade',u'Ile',u'Impasse',u'Jardins',u'Passage',u'Place',\

u'Quai',u'Résidence', u'Route', u'Rue',u'Square', u'Terrasse']

#### Solution:[¶](#Solution:)

Audit street type code here:

STREET\_TYPE2\_RE = re.compile(r'^\S+',re.IGNORECASE)

def audit\_street\_type(street\_types, street\_name):

m = STREET\_TYPE2\_RE.search(street\_name)

if m:

street\_type = m.group()

if street\_type not in expected:

street\_types[street\_type].add(street\_name)

The correction of streets is given by the following mapping:

mapping = { u'allée': u'Allée',

u'avenue': u'Avenue',

u'boulevard':u'Boulevard',

u'cours':u'Cours',

u'place':u'Place',

u'Pl':u'Place',

u'quai':u'Quai',

u'rue': u'Rue',

u'RUE': u'Rue',

u'Residence':u'Résidence',

u'terrasse':u'Terrasse'}

### Issue: Variation in street names[¶](#Issue:-Variation-in-street-names)

Do no confuse this issue with previous issue "street types"

Here are some street name request:

SELECT DISTINCT value FROM nodes\_tags,

(SELECT id AS idd FROM nodes\_tags WHERE key='street') as subquery

WHERE id = idd AND key='street'

ORDER BY value

LIMIT 20;

Here are some interesting issues:

"Avenue Georges Clemenceau" vs "Avenue Georges Clémenceau": accent

"Place de la Defense" vs "place de la Défense": accent and uppercase

"Place du 8 Mai 1945" vs "Place du 8 mai 1945": uppercase or lowercase

"Rue Gabriel Peri" vs "Rue Gabriel Péri":accent

"Rue Léon Maurice Nordmann" vs "Rue Léon-Maurice Nordmann": space instead of '-'

"Rue Marceau" vs "Rue Marceau,": typo

"Rue Noel Pons" vs "Rue Noël Pons": accent

"Rue du 11 Novembre" vs "Rue du 11 Novembre 1918":

#### Solution:[¶](#Solution:)

Post-processing operation from excel files.

Make all street adresses into uppercase avoiding issues with "rue" and "Rue".

We could get rid of accent (e.g. é, è, à) but it is debatable. I chose to keep accents.

Use of OpenRefine tool for clustering and merge the same way I did for issue related to Variation in city names.

### Issue: variation in letters within housenumber[¶](" \l "Issue:-variation-in-letters-within-hous)

In France, we could have house number followed by a complement (e.g. bis,ter,quater, etc...)

We have the following variations:

<tag k="addr:housenumber" v="9 bis"/>

<tag k="addr:housenumber" v="47 Bis"/>

<tag k="addr:housenumber" v="7 B"/>

<tag k="addr:housenumber" v="2 ter; 2 quater"/>

<tag k="addr:housenumber" v="1bis"/>

<tag k="addr:housenumber" v="118Ter"/>

<tag k="addr:housenumber" v="18 T"/>

<tag k="addr:housenumber" v="10B"/>

<tag k="addr:housenumber" v="10T"/>

<tag k="addr:housenumber" v="99 A"/>

<tag k="addr:housenumber" v="48Q"/>

<tag k="addr:housenumber" v="6, 6 bis, 6 ter"/>

SELECT key,value FROM nodes\_tags WHERE key='housenumber' AND value LIKE '%b%' LIMIT 50;

Here are some interesting outputs:

key|value

housenumber|9 bis

housenumber|47 Bis

housenumber|154 bis

housenumber|21 B

It is the same issue with ter, quart. And we have also list of housenumbers

key|value

housenumber|17 Ter

housenumber|2 ter; 2 quater

#### Solution:[¶](#Solution:)

We use code to audit and correct the issue.

We can create the following mapping based on the postal office recommendation:

NNN BIS, NNN Bis => NNN bis or NNN B (postal office recommendation). My decision: bis (lower case)

NNNBis,NNNbis => NNN bis or NNN B (postal office recommendation). My decision: NNN bis

TER, Ter => ter or T (postal office recommendation). My decision: ter

NNNTer, NNNter => NNN ter or NNN T (postal office recommendation). My decision: NNN ter

QUAT, Quat, Quater, QUATER => quater or q (postal office recommendation). My decision: quater

NNN Quater, NNN quater, NNN Quat => NNN quater or NNN Q (postal office reco). My decision: NNN quater

NNNQuater, NNNquater, NNNQuat => NNN quater or NNN Q (postal office reco). My decision: NNN quater

'Bâtiment B' should be kept as it is since 'Bâtiment' is not a number.

Here is the code:

def update\_housenb(nb, mapping):

m = housenumber\_re.search(nb)

if m:

complement = m.group(3)

if complement in mapping:

n = m.group(1)

#print n+' '+mapping[complement]

return n+' '+mapping[complement]

else:

return nb

else:

return nb

The following code used within 'shape\_element' function allows to process list of house numbers separated by various delimiters:

nb = data["value"]

lst = filter(None, re.split("[,;\-]+", nb))

s = []

for item in lst:

s.append(update\_housenb(item, bis\_ter\_quater))

better\_nb = ';'.join(s)

data["value"] = better\_nb

Here is the transformation:

'7-7 bis' => '7;7 bis'

### Issue: Empty values, leading and trailing space[¶](#Issue:-Empty-values,-leading-and-traili)

Query trailing space (no leading space found):

SELECT \* FROM nodes\_tags

WHERE value LIKE '% '

LIMIT 10;

Here is the unique result:

id|key|value|type

4423855431|designation|Epicerie fine |regular

Query Trailing space for ways\_tags table:

SELECT \* FROM ways\_tags

WHERE value LIKE '% '

LIMIT 10;

Unique result is:

id|key|value|type

295256565|description|Bibliothèque Universitaire Paris Nanterre Tél : 01 40 97 72 02 Maill : scd-bu@u-paris10.fr Bibliothèque de Documentation Internationale Contemporaine Tel : 01 40 97 79 00 |regular

Query empty values:

SELECT \* FROM nodes\_tags WHERE value = '';

Here is the unique result:

id|key|value|type

4855994331|operator||regular

#### Solution:[¶](#Solution:)

It is easy to audit and correct this type of issue directly in the code. Here is the code for auditing:

def empty\_value(element, keys):

# catch tags

if element.tag == "tag":

# get value

str = element.get('v')

if len(str.strip()) == 0:

keys['empty'] += 1

elif len(str.strip()) != len(str):

keys['leading\_trailing'] += 1

else:

keys['not\_empty'] += 1

return keys

## Explore the Database[¶](#Explore-the-Database)

### Preparation[¶](#Preparation)

* Create a database (OSM.db),
* Access the database,
* List all databases,
* Import SQL3 file into SQLite,
* Import CSV files (node.csv, nodes\_tags.csv, ways.csv,ways\_tags.csv,ways\_nodes.csv) into a table (node)

From Powershell:

>.\sqlite3.exe OSM.db

From SQLite:

.databases

.read db.sql

.mode csv

.import nodes.csv nodes

.import nodes\_tags.csv nodes

.import ways.csv nodes

.import ways\_tags.csv nodes

.import ways\_nodes.csv nodes

### Overview of the dataset[¶](#Overview-of-the-dataset)

Is the OSM XML large enough?

$ ls -lh \*Nanterre.osm\*

-rw-r--r-- 1 Diallo 197609 107M août 20 16:54 Nanterre.osm

-rw-r--r-- 1 Diallo 197609 11M sept. 23 08:42 Nanterre.osm.zip

-rw-r--r-- 1 Diallo 197609 11M août 20 21:51 sampleNanterre.osm

-rw-r--r-- 1 Diallo 197609 1,3M sept. 23 08:56 sampleNanterre.osm.zip

It is conform to the minimum requirement of 50 MB uncompressed for the original OSM file and conform to the maximum requirement of 10 MB for the sample file.

### Description of the database[¶](#Description-of-the-database)

SELECT tbl\_name, sql

FROM sqlite\_master

WHERE type='table';

Output is:

nodes|CREATE TABLE nodes (

id INTEGER PRIMARY KEY NOT NULL,

lat REAL,

lon REAL,

user TEXT,

uid INTEGER,

version INTEGER,

changeset INTEGER,

timestamp TEXT

)

nodes\_tags|CREATE TABLE nodes\_tags (

id INTEGER,

key TEXT,

value TEXT,

type TEXT,

FOREIGN KEY (id) REFERENCES nodes(id)

)

ways|CREATE TABLE ways (

id INTEGER PRIMARY KEY NOT NULL,

user TEXT,

uid INTEGER,

version TEXT,

changeset INTEGER,

timestamp TEXT

)

ways\_tags|CREATE TABLE ways\_tags (

id INTEGER NOT NULL,

key TEXT NOT NULL,

value TEXT NOT NULL,

type TEXT,

FOREIGN KEY (id) REFERENCES ways(id)

)

ways\_nodes|CREATE TABLE ways\_nodes (

id INTEGER NOT NULL,

node\_id INTEGER NOT NULL,

position INTEGER NOT NULL,

FOREIGN KEY (id) REFERENCES ways(id),

FOREIGN KEY (node\_id) REFERENCES nodes(id)

)

### Query on nodes\_tags table: What the top 20 keys used?[¶](#Query-on-nodes_tags-table:-What-the-top)

SELECT key,COUNT(\*) FROM nodes\_tags

GROUP BY key

ORDER BY COUNT(\*)

DESC

LIMIT 20;

Results:

key|COUNT(\*)

source|13374

housenumber|5055

highway|4925

amenity|3688

name|2987

natural|2744

street|2110

city|1806

country|1602

postcode|1433

wheelchair|1253

tactile\_paving|1097

barrier|951

type|934

operator|906

height|890

shop|846

bollard|779

entrance|756

public\_transport|660

### Query on nodes\_tags table: top 10 sources of information?[¶](#Query-on-nodes_tags-table:-What-are-the)

Query: It is given by key = source

SELECT key,value,COUNT(\*) AS Nb FROM nodes\_tags

WHERE key='source'

GROUP BY value

ORDER BY Nb

DESC

LIMIT 10;

Results:

key|value|Nb

source|survey|1455

source|cadastre-dgi-fr source : Direction Générale des Impôts - Cadastre. Mise à jour : 2014|1428

source|survey 2014|863

source|cadastre-dgi-fr source : Direction Générale des Impôts - Cadastre. Mise à jour : 2010|792

source|survey 2016|767

source|cadastre 2016 + survey|704

source|opendata.paris.fr|614

source|survey 2015|521

source|Microsoft Bing orbital imagery + survey|417

source|cadastre-dgi-fr source : Direction Générale des Impôts - Cadastre. Mise à jour : 2012|386

### Query on ways\_tags table: statistics of building heights[¶](#Query-on-ways_tags-table:-statistics-of)

I use cast function to avoid issues with values that are string. Issues: string ordering '90' > '110' and summary data like minimum and maximum.

The query with cast function:

SELECT DISTINCT id, key, CAST(value AS FLOAT) AS height FROM ways\_tags,

(SELECT id AS idd FROM ways\_tags WHERE key='building' ) AS sub

WHERE id=idd AND key='height'

ORDER BY height

Here are the 10 smallest heights:

id|key|height

47872151|height|1.0

71341167|height|1.0

71427150|height|1.0

71341236|height|1.5

71341248|height|1.5

71341262|height|1.5

71341312|height|1.5

109471844|height|2.0

109471847|height|2.0

71341265|height|3.0

Query to get the mean:

SELECT DISTINCT avg(CAST(value AS FLOAT)) AS mean\_height FROM ways\_tags, (SELECT id AS idd from ways\_tags WHERE key='building' ) AS sub WHERE id=idd AND key='height';

Here is the mean:

mean\_height

33.2682142857142

Query to get the minimum:

SELECT DISTINCT min(CAST(value AS FLOAT)) AS min\_height FROM ways\_tags, (SELECT id AS idd from ways\_tags WHERE key='building' ) AS sub WHERE id=idd AND key='height';

Here is the minimum:

min\_height

1.0

Query to get the maximum:

SELECT DISTINCT max(CAST(value AS FLOAT)) AS max\_height FROM ways\_tags, (SELECT id AS idd from ways\_tags WHERE key='building' ) AS sub WHERE id=idd AND key='height';

Here is the minimum:

max\_height

195.0

### Query on nodes table: histogram of activities[¶](#Query-on-nodes-table:-histogram-of-acti)

.head on

.mode csv

.output sumNodesDaily.csv

SELECT DATE(timestamp) AS date FROM nodes ORDER BY timestamp;

.exit

In [6]:

# Required Libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Get the Data and create the data frame

path = ''

filename = 'sumNodesDaily.csv'

nodes = pd.read\_csv(path+filename)

nodes['date'] = pd.to\_datetime(nodes['date'])

### Query on nodes table:[¶](#Query-on-nodes-table:)

In [7]:

import datetime as dt

import pandas as pd

nodes['Year']= nodes['date'].dt.year

In [8]:

nodes.head()

Out[8]:

|  | **date** | **Year** |
| --- | --- | --- |
| **0** | 2007-01-11 | 2007 |
| **1** | 2007-01-11 | 2007 |
| **2** | 2007-01-11 | 2007 |
| **3** | 2007-01-11 | 2007 |
| **4** | 2007-01-11 | 2007 |

In [9]:

by\_year=nodes.groupby("Year").size()

by\_year.head(n=12)

Out[9]:

Year

2007 172

2008 550

2009 1565

2010 251624

2011 96460

2012 24909

2013 20893

2014 26143

2015 14208

2016 25567

2017 5958

dtype: int64

In [10]:

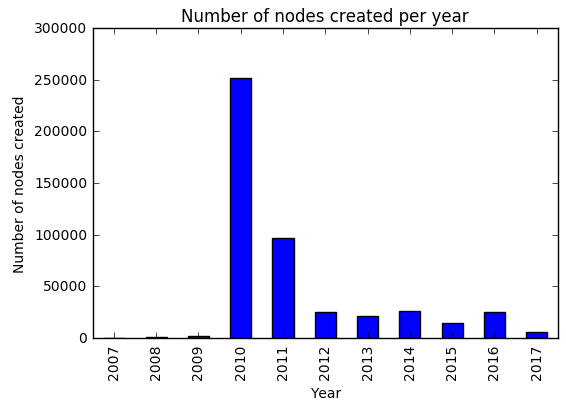
nodes.groupby("Year").size().plot(kind='bar')

plt.ylabel('Number of nodes created')

plt.xlabel('Year')

plt.title('Number of nodes created per year')

plt.show()



### Query on Dataset Statistics:[¶](#Query-on-Dataset-Statistics:)

It should be consistent with the Tags extracted from the code in python

{'bounds': 1,

'member': 12302,

'nd': 639866,

'node': 468049,

'osm': 1,

'relation': 938,

'tag': 279545,

'way': 75249}

Number of unique users

sqlite> SELECT COUNT(DISTINCT uid) AS nb\_distinct\_users FROM nodes;

Result:

nb\_distinct\_users

524

Number of nodes

SELECT COUNT(DISTINCT id) AS nb\_nodes FROM nodes;

Result confirming the tags counting:

nb\_nodes

468049

Number of ways:

SELECT COUNT(DISTINCT id) AS nb\_ways FROM ways;

Result confirming the tags counting:

nb\_ways

75249

Number of chosen type of nodes, like cafes, shops etc.

List of theatres:

SELECT key,value FROM nodes\_tags, (SELECT id AS idd FROM nodes\_tags where key='amenity' AND value='theatre') AS

subquery WHERE id=idd AND key='name';

Results:

key|value

name|Théâtre des Amandiers

name|Théatre par le bas

name|Théatre André Malraux

name|Bernard-Marie Koltès

List of motorways:

SELECT key,value FROM ways\_tags,

(SELECT id AS idd FROM ways\_tags WHERE key='highway' AND value ='motorway') AS sub

WHERE id=idd AND key='ref'

GROUP BY 2

ORDER BY 2;

Here are the results:

key|value

ref|A 14

ref|A 86

ref|N 1014

## Improvement[¶](#Improvement)

Using Mass correction with tools like OpenRefine is fast and efficient. it is part of the ideas presented here below. It brought me new ideas in data curation and data integration such as:

* Validate the data (e.g. street name checking)
* Data curation (as done in the delivered python code)
* Data exploration
* Statistical analysis of fields (good for detecting anomalies) or data profiling
* Transformation and add new sources of information thanks to ETL

We'll discuss here below about the pros and cons of using tools and associated processes (data governance, data quality assessment and data validation).

We can cite the remarkable tools such as: Trifacta, DataWrangler, OpenRefine, Paxata and Tamr.

If we look at history, ETL tools were used to build consolidated data warehouse.

It is the combination of 5 ideas/features - and not the selection of a single feature - and the proper processes that will bring the maximum impact to the improvement.

### Benefits[¶](#Benefits)

Using these tools might be a promise for better (data), cheaper and faster approach.

One main objective of these tools is cleaning in order to get better data. The data source is usually not created by others than the team applying data curation. Data are invariably dirty - humans are in the loop - and we need to clean or correct it. By applying data curation, we increase its quality.

Bad data quality has a cost. In this project, it is an opportunity to measure it.

How could we know that the data is correct (e.g. Street names)? By integrating more authoritative data source (e.g. public transport, opendata) for improve accuracy.

Here is an example. The valid street names in France could be found from here: [reference](https://www.data.gouv.fr/en/datasets/fichier-fantoir-des-voies-et-lieux-dits/)

It could be interesting to use the refence to check the validity of the street names. OSM France initiated an address centered project called "BANO" (Base d'Adresses Nationale Ouverte). This database is a composite from OSM address data, available opendata sets, cadastre data automatically extracted. The resulting composite dataset is available under ODbL license and can be used to improve OSM addresses and street names. For more detail see BANO project or <http://openstreetmap.fr/bano>.

If possible, by validating the data entered by the user before recording it, we make sure only correct data are in the database.

Many tools offer powerful visualisation and not only tablular views. Producing a tidy view of the data is an edge. Many tools are intuitive and easy to use and support a lot of format. All these features combined with automation improves speed.

The use of schema and workflow is a general practice (as in data integration) and make complex things easier to understand.

In conclusion, we can say that more data quality means better decision making based on accurate data and also higher trust in the data. This improvement if well done could bring higher value (e.g. fast and reliable decision making), lower cost (less rework) and better quality.

### Anticipated problems[¶](#Anticipated-problems)

Ingesting more and more data sources has a drawback. The situation become more complex and consequently the process could become too human intensive. Scaling to way more data sources is risky or unfeasible. Organisations don't scale easily because of increased complexity.

To validate some data, we might need experts.

The more we need people (experts included) or the bigger the organisation is, the more we could have silos.

The last pitfall identified is selecting a set of tools only usable by experts. It gives a bad learning curve and a low adoption.

Understanding the problems decribed here is a first step in alleviating the pains in the path of data wrangling. If done correctly, the benefits are greater than the cost of the improvement suggested.

## Conclusion[¶](#Conclusion)

Let's start with this citation:

A map is not the territory it represents, but, if correct, it has a similar structure to the territory, which accounts for its usefulness.

— Alfred Korzybski, Science and Sanity (1933, p. 58)

Another way to say it:

"All models are wrong but some are useful"

— George E. P. Box

Issues occurred with 2 tables out of 4.

| **table** | **remarks** |
| --- | --- |
| nodes | ok |
| nodes\_tags | issues |
| ways | ok |
| ways\_tags | issues |
| ways\_nodes | ok |

The processing and database construction depends on the country for streets names. The types of street in France are very numerous (about 330).

It requires way more time to complete a comprehensive cleaning process.