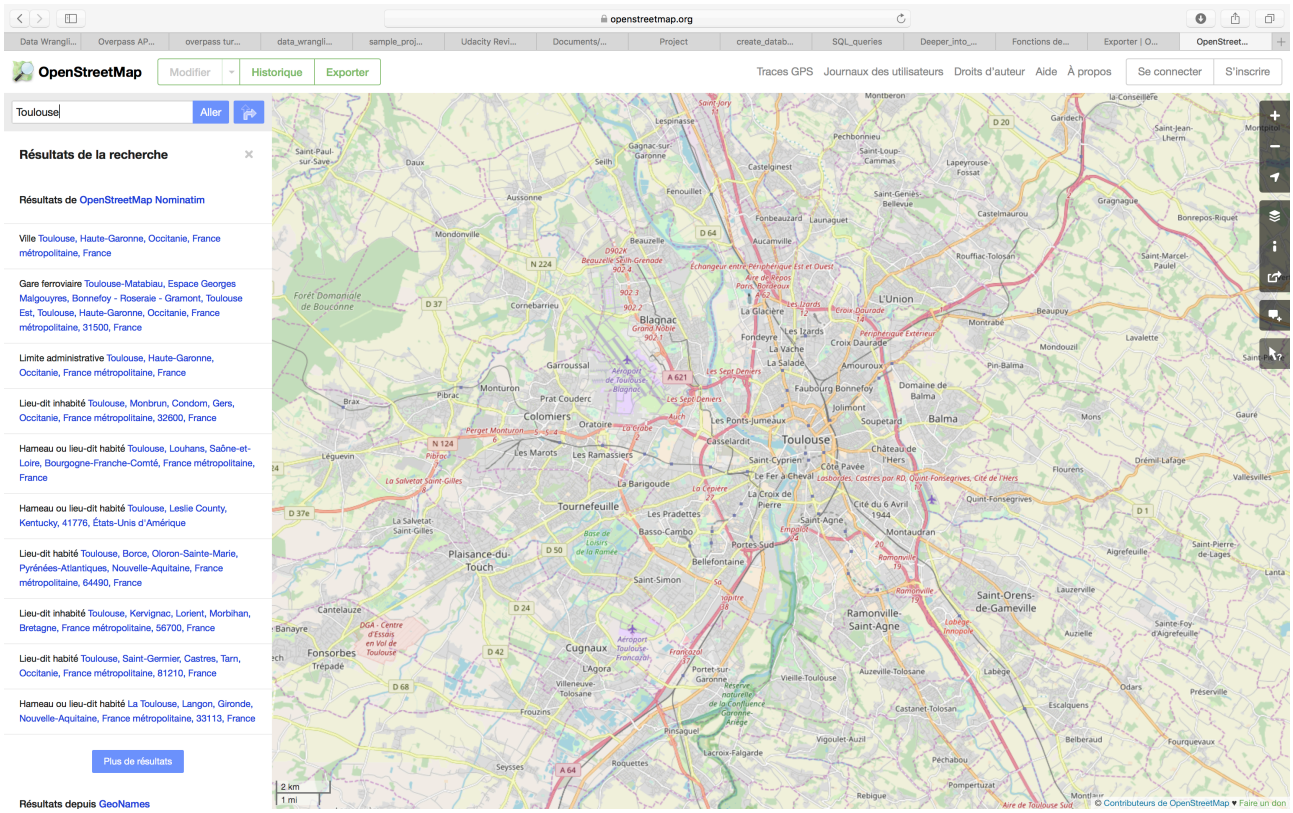
Introduction

For that project, I decided to use data from the town where I live: Toulouse.



No metro extract for this city exists.

I used the the overpass API (http://overpass-api.de/query_form.html) to download the data with following command:

Overpass API Query Form
(node(43.4846, 1.2047, 43.7168, 1.6613);<;);out meta;

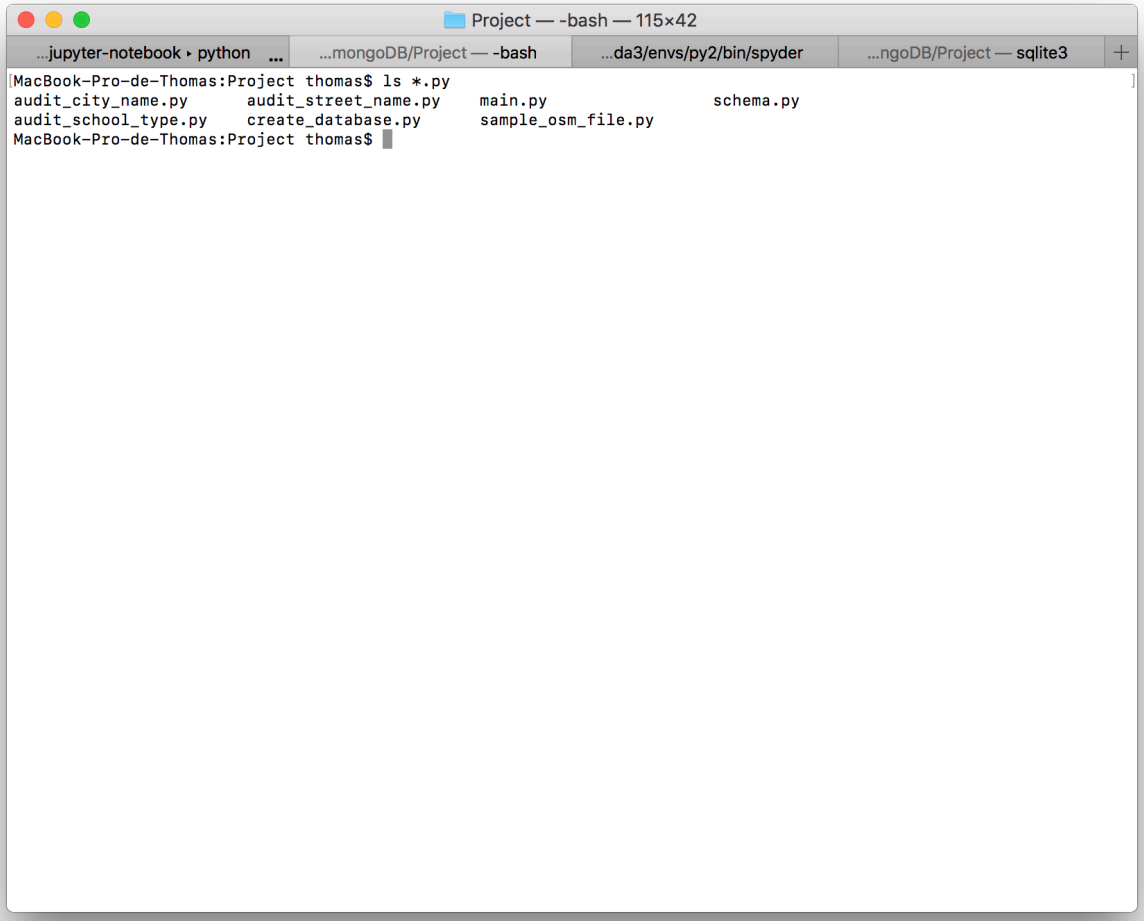
Toulouse.osm file size is 862 Mo.

For validation purpose, I included in the files provided for project review a reduced version of this file (Toulouse_small_sample.osm is only 8.7 Mo)

Data wrangling

I programmatically cleaned, wrote the node, node tags, way, way tags and way nodes data in csv and stored the result in a database with a python program.
The python program that executes these tasks is split into several scripts:

- sample the data for validation purpose: sample_osm_file.py
- audit data script: audit_street_name.py, audit_city_name.py, audit_school_type.py
- schema validation before data is written in csv: schema.py
- SQL database creation: create_database.py
- main script to execute all the tasks (except sampling): main.py



Data sampling

In order to validate the scripts while I was writing them, I sampled the Toulouse.osm file into 2 files:

- Toulouse_small_sample.osm —> sampled every 100 nodes
- Toulouse_medium_sample.osm —> sampled every 10 nodes

These files allowed a convenient and fast validation of the different functions of the python program.

Data audit

The data found in OpenStreetMap is human edited and contain plenty inconsistencies in the way the fields are written. The audit part of the program allows to clean some of the inconsistencies without pretending to remove them all.

Three tag fields have been cleaned:
- street types
- city names
- school types

Street types have been cleaned with the audit and mapping technic proposed in the module and adapted to French street types.

For city names, I converted all the names in the same case (first letter of the word uppercase and others in lower case), then replaced '-' by ' '. Few inconsistencies remained that I cleaned with the mapping technic.
School types were not too 'dirty' but I wanted to make the field more homogeneous. The French school system is organized the following way:
- maternelle for kids between 3 and 5 years old
- élémentaire for kids between 6 and 10 years old
- collège for kids between 11 and 14
- lychee for kids between 15 and 17 years old

Maternelle and élémentaire schools are part of the primary school (primaire)
Collège and lycée are part of the secondary school (secondaire)
To make the data more homogeneous, I simply replaced
- primaire by maternelle; élémentaire
- secondaire by collège; lycée

A further improvement would be to split the tags into elementary ones to ease the SQL analysis later on. For instance school tags like 'maternelle; élémentaire' would be split into two tags: 'maternelle' and 'élémentaire'.
I tried to apply such improvement by appending more data to the tag dictionary. Unfortunately, python dictionary does not allow multiple identical keys. Another way around is to be found …

As much as possible, auditing and cleaning is performed in an individual script that is then called by the main script.

Schema validation

Schema validation before data is written in csv is done with cerberus library and a pre-defined schema for each file (node, node_tags, way, way_tags and way_nodes). This part has been provided by Udacity and is only used with the small sample size file due to long computation time.

SQL database creation

Once data is imported from ism file, cleaned and written in the csv files, the SQL database can be created. I chose to perform this part with the python API and sqlite3 library. I also performed this part directly in sqlite3 but found that it more convenient to do it programatically.

Each csv file is imported into a table with the following schema:

Node:

	Type	Primary key	Foreign key
id	Integer	Yes	
lat	Float		
lon	Float		
user	String		
uid	Integer		
version	String		
changeset	Integer		
timestamp	String		

Node_tags:

	Type	Primary key	Foreign key
id	Integer		Yes (node id)
key	String		
value	String		
type	String		

Way:

	Type	Primary key	Foreign key
id	Integer	Yes	
user	String		
uid	Integer		
version	String		
changeset	Integer		
timestamp	String		

Way_tags:

	Type	Primary key	Foreign key
id	Integer		Yes (way id)
key	String		
value	String		
type	String		

Way_nodes:

	Type	Primary key	Foreign key
id	Integer		Yes (way id)
node_id	Integer		Yes (node id)
position	Integer		

SQL queries

SQL queries are also performed programmatically with a Jupyter notebook to keep the interactivity and perform quick plotting thanks to Matplotlib.

Following queries are done to explore the Toulouse openstreetmap database:

- unique users and contribution
- number of nodes and ways
- list of amenities
- restaurants in Toulouse
- bicycle and car parks
- school types
- population by city in Toulouse area

Unique users and contribution

Based on node table, the number of unique users is 1500.

```
In [217]: QUERY='''SELECT count(sub_query.uid) as unique_user
          FROM
          (SELECT uid
          FROM NODE
          GROUP BY uid) as sub_query
          '''
          df=db_sql_query(db_name,QUERY)
          df.head()
```

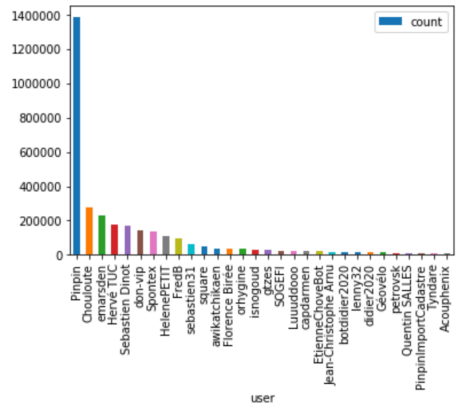
Out[217]:

	unique_user
0	1500

Contribution of top contributing users is the following. One can see that user Pinpin is by far the top contributor of openstreetmap in Toulouse area.

```
In [218]: QUERY='''SELECT uid, user, count(uid) as count
          FROM NODE
          GROUP BY uid
          ORDER BY count DESC
          '''
          df=db_sql_query(db_name,QUERY)
          df[df['count']>10000].plot(kind='bar',x='user',y='count')
```

Out[218]: <matplotlib.axes._subplots.AxesSubplot at 0x118524cd0>



Number of nodes and ways

In [216]:
```
QUERY='''SELECT count(id) as nb_nodes
FROM NODE
'''
df1=db_sql_query(db_name,QUERY)

QUERY='''SELECT count(id) as nb_ways
FROM WAY
'''
df2=db_sql_query(db_name,QUERY)

df=pd.concat([df1, df2], axis=1)
df
```

Out[216]:

	nb_nodes	nb_ways
0	3495082	26220

List of amenities

In order to study some amenities, first step is to list the most recurrent ones.

In [149]:
```
QUERY='''SELECT *
FROM
    (SELECT *, count(value) as count
    FROM NODE_TAG
    WHERE key='amenity'
    GROUP BY value
    ORDER BY count DESC) as subquery
WHERE count>100
'''
df=db_sql_query(db_name,QUERY)
df
```

Out[149]:

	id	key	value	type	count
0	5917722323	amenity	recycling	regular	1304
1	5917663734	amenity	bench	regular	1106
2	5919134804	amenity	restaurant	regular	904
3	5909794812	amenity	waste_basket	regular	594
4	5913575574	amenity	post_box	regular	565
5	5915915480	amenity	bicycle_parking	regular	514
6	5792706053	amenity	fast_food	regular	380
7	5844045418	amenity	parking	regular	337
8	5907965112	amenity	bank	regular	296
9	5839550737	amenity	bicycle_rental	regular	285
10	5893977287	amenity	pharmacy	regular	285
11	5904951540	amenity	drinking_water	regular	230
12	5903168807	amenity	school	regular	218
13	5214564447	amenity	kindergarten	regular	202
14	5839563185	amenity	bar	regular	183
15	5801777333	amenity	parking_space	regular	162
16	5884556170	amenity	parking_entrance	regular	158
17	5818189269	amenity	cafe	regular	147
18	5848360775	amenity	toilets	regular	140
19	5903910580	amenity	atm	regular	117
20	5858829013	amenity	doctors	regular	102

City furniture like recycling bins and benches are the top amenities but I would rather study restaurant, parking and schools …

Restaurants in Toulouse

First I list and order restaurant by cuisine type. It looks like most restaurants are either local, Italian or asian cuisine.

In [18]:
```
QUERY='''SELECT *
FROM
    (SELECT cuisine, count(cuisine) as count
    FROM
        (SELECT id
        FROM NODE_TAG
        WHERE key='amenity'
        AND value='restaurant') as subquery_id
    LEFT JOIN
        (SELECT id, value as cuisine
        FROM NODE_TAG
        WHERE key='cuisine') as subquery_cuisine
    ON subquery_id.id=subquery_cuisine.id
    GROUP BY cuisine
    ORDER BY count(cuisine) DESC) as subquery_count
WHERE count>=3
'''
df=db_sql_query(db_name,QUERY)
df
```

Out[18]:

	cuisine	count
0	french	90
1	pizza	65
2	regional	47
3	asian	32
4	japanese	27
5	italian	26
6	indian	22
7	chinese	19
8	vietnamese	17
9	burger	7
10	pancake	7
11	seafood	7
12	lebanese	6
13	spanish	6
14	steak_house	6
15	international	5
16	thai	5
17	couscous	4
18	kebab	4
19	mexican	4
20	moroccan	4
21	american	3
22	brasserie	3
23	crepe	3
24	italian;pizza	3
25	sandwich	3
26	sushi	3

One can see that some restaurants have mixed cuisine types (‘italian;pizza’ for instance). It would be interesting to split this information as suggested in the wrangling part to have a more accurate analysis.

Then I extract more information about restaurant with regional cuisine.

In [12]:
```
QUERY='''SELECT subquery_id.id, name, cuisine, phone, website, lat, lon
FROM
    (SELECT id
    FROM NODE_TAG
    WHERE key='amenity'
    AND value='restaurant') as subquery_id
LEFT JOIN
    (SELECT id, value as name
    FROM NODE_TAG
    WHERE key='name') as subquery_name
ON subquery_id.id=subquery_name.id
LEFT JOIN
    (SELECT id, value as cuisine
    FROM NODE_TAG
    WHERE key='cuisine') as subquery_cuisine
ON subquery_id.id=subquery_cuisine.id
LEFT JOIN
    (SELECT id, value as phone
    FROM NODE_TAG
    WHERE key='phone') as subquery_phone
ON subquery_id.id=subquery_phone.id
LEFT JOIN
    (SELECT id, value as website
    FROM NODE_TAG
    WHERE key='website') as subquery_website
ON subquery_id.id=subquery_website.id
JOIN NODE
ON subquery_id.id=NODE.id
WHERE cuisine='regional'
'''
df=db_sql_query(db_name,QUERY)
df
```

Out[12]:

	id	name	cuisine	phone	website	lat	lon
0	259131130	L'Écluse de Castanet	regional	None	None	43.523510	1.519007
1	450161376	La Cave au Cassoulet	regional	+33 5 61 13 60 30	None	43.602051	1.441060
2	520124626	Le Bon Vivre	regional	None	None	43.605308	1.447199
3	615204677	Buffalo Grill	regional	None	None	43.553785	1.483223
4	630662464	L'Oncle Pom	regional	+33 5 61 54 39 86	http://www.lonclepom.com/	43.606196	1.455216
5	766485053	Le p'tit resto	regional	+33 5 61 73 66 74	None	43.518524	1.506090
6	769244212	L'Accessoire	regional	+33 5 61 42 79 05	http://restaurant-laccessoire.over-blog.com/	43.596443	1.432838
7	772900237	Mamie et les Ours	regional	+33561215533	None	43.609194	1.438997
8	773250510	Bois et Charbon	regional	+33 5 61 63 61 21	http://www.boisetcharbon.net/	43.604766	1.454725

Bicycle and car parks

First I list the different parking types for both bicycles and cars.
This query highlights the fact that the tag fields can still be improved because one bicycle parking type is ‘yes’ which does not make much sense …

In [13]:
```
QUERY='''SELECT value as bicycle_parking_type
FROM NODE_TAG
WHERE key='bicycle_parking'
GROUP BY bicycle_parking_type
'''
df1=db_sql_query(db_name,QUERY)

QUERY='''SELECT value as car_parking_type
FROM NODE_TAG
WHERE key='parking'
GROUP BY car_parking_type
'''
df2=db_sql_query(db_name,QUERY)

df=pd.concat([df1, df2], axis=1)
df
```

Out[13]:

	bicycle_parking_type	car_parking_type
0	anchors	carpool
1	bollard	disabled
2	building	multi-storey
3	lockers	surface
4	rack	underground
5	shed	NaN
6	stands	NaN
7	wall_loops	NaN
8	wide_stands	NaN
9	yes	NaN

Then I compare the parking capacity for bicycles and cars.

In [14]:
```
QUERY='''SELECT sum(capacity) as bicycle_parking_capacity
FROM
    (SELECT id
    FROM NODE_TAG
    WHERE value='bicycle_parking') as subquery_id
LEFT JOIN
    (SELECT id, value as capacity
    FROM NODE_TAG
    WHERE key='capacity') as subquery_capacity
ON subquery_id.id=subquery_capacity.id
'''
df1=db_sql_query(db_name,QUERY)

QUERY='''SELECT sum(capacity) as car_parking_capacity
FROM
    (SELECT id
    FROM NODE_TAG
    WHERE value='parking') as subquery_id
LEFT JOIN
    (SELECT id, value as capacity
    FROM NODE_TAG
    WHERE key='capacity') as subquery_capacity
ON subquery_id.id=subquery_capacity.id
'''
df2=db_sql_query(db_name,QUERY)

df=pd.concat([df1, df2], axis=1)
df
```

Out[14]:

	bicycle_parking_capacity	car_parking_capacity
0	4014	8116

It’s a shame that Toulouse metropolitan area still has twice as much car park compared to bicycle park.

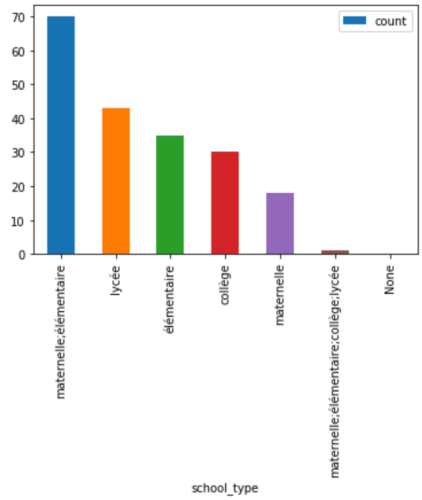
School types

My kids are still young and I'm very much interested by the way Toulouse metropolitan area is equipped with teaching facilities. I would like to know how many schools there is in the area and how they are split in the different types:

- maternelle for kids between 3 and 5 years old
- élémentaire for kids between 6 and 10 years old
- collège for kids between 11 and 14
- lychee for kids between 15 and 17 years old

In [13]:
```
QUERY='''SELECT school_type, count(school_type) as count
FROM
    (SELECT id
    FROM NODE_TAG
    WHERE value='school') as subquery_id
LEFT JOIN
    (SELECT id, value as school_type
    FROM NODE_TAG
    WHERE key='school:FR') as subquery_school_type
ON subquery_id.id=subquery_school_type.id
GROUP BY school_type
ORDER BY count DESC
'''
df=db_sql_query(db_name,QUERY)
df.plot(kind='bar',x='school_type',y='count')
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x118495b50>
```



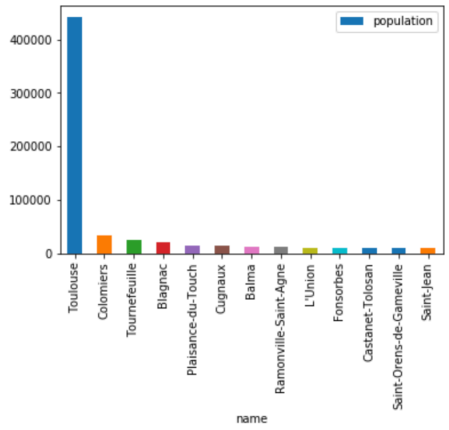
Analysis suggest that there plenty more primary schools (maternelle and elementary) than secondary schools (collège and lycée). Given that there are roughly the same number of students in each class of age, I suspect the maternelle and élémentaire schools to be smaller than collège and lycée. Unfortunately number of students per school is not included in the database to back the hypothesis.

Population by city in Toulouse area

I make a final query to get the population of each city of the Toulouse area extracted in the openstreetmap file.

In [16]:
```
QUERY='''SELECT name, population
FROM
    (SELECT id
    FROM NODE_TAG
    WHERE key='population'
    AND type='regular') as subquery_id
LEFT JOIN
    (SELECT id, value as name
    FROM NODE_TAG
    WHERE key='name') as subquery_name
ON subquery_id.id=subquery_name.id
LEFT JOIN
    (SELECT id, CAST(value AS INT) as population
    FROM NODE_TAG
    WHERE key='population'
    AND type='regular') as subquery_population
ON subquery_id.id=subquery_population.id
ORDER BY population DESC
'''
df=db_sql_query(db_name,QUERY)
df[df['population']>10000].plot(kind='bar',x='name',y='population')
```

Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1185bdf50>
```



Of course Toulouse is the largest city in the area.

And the total population of the Toulouse metropolitan area is 861.043.

In [22]:
```
QUERY='''SELECT sum(population) as total_population
FROM
    (SELECT id
    FROM NODE_TAG
    WHERE key='population'
    AND type='regular') as subquery_id
LEFT JOIN
    (SELECT id, CAST(value AS INT) as population
    FROM NODE_TAG
    WHERE key='population'
    AND type='regular') as subquery_population
ON subquery_id.id=subquery_population.id
'''
df=db_sql_query(db_name,QUERY)
df.head()
```

Out[22]:

	total_population
0	861043