Exploitation de l’Open Data avec Python par l’architecte

Démonstration par la pratique des apports potentiels au sein de la conception architecturale

ABSTRACT

Ce mémoire a pour but d'aborder les enjeux de l'appropriation d'un langage de programmation par les architectes afin d'exploiter les données aujourd'hui disponibles sur les plateformes en Open Data, tel que des données concernant la morphologie de l'existant, caractérisant leur besoins énergétiques ou encore leurs matériaux. Plus précisément, ce mémoire se concentrera sur l'emploi du **langage Python** avec lequel je travaille régulièrement, choisi ici pour sa syntaxe claire et simplifiée par rapport à d'autres langages, à travers une **approche pratique constituée de plusieurs scripts** répondant aux principaux enjeux autour de l'appropriation des données ouvertes par les architectes. En quoi un langage comme Python est-il indispensable aujourd'hui pour exploiter les données en Open Data ? De quelle manière peut-on les intégrer dans l'environnement de travail de la conception architecturale ? Quel usage approfondi vis à vis de ces données est-il alors possible de mettre en place grâce à la programmation ?

SOMMAIRE

ABSTRACT **2**

SOMMAIRE **3**

INTRODUCTION **4**

1 Le script : outil d'exploitation privilégié de l'Open Data (Etude de cas des données des « volumes bâtis » de l'Open Data Paris.) **9**

1.1 L'Open Data : entre nomenclature et variables 11

1.1.1 Variables et typologies des valeurs 12

1.1.2 Les métadonnées : clé de compréhension des données 14

1.2 Le langage Python pour s'approprier aisément les données ouvertes 19

1.2.1 Les formats tabulaires 20

1.2.2 Les formats hiérarchisés 21

1.2.3 Les formats géographiques 24

1.3 Aperçu de la souplesse des fonctions de manipulation de données 27

1.3.1 Approche compréhensive des différentes notations grâce à Python 28

1.3.2 De la chaîne de caractère à la valeur numérique 31

2 lolkdedleded **34**

BIBLIOGRAPHIE **34**

ICONOGRAPHIE **36**

INTRODUCTION

Au cours des dernières années, un déploiement prolifique de jeux de données est en train d'avoir lieu sous l'égide de « l'Open Data ».

Le gouvernement définit ce terme comme « l'effort que font les institutions, notamment gouvernementales, qui partagent les données dont elles disposent ».[1] En effet, c'est avant tout une stratégie prônant l'ouverture du plus grand nombre de bases de données au public, les rendant ainsi totalement accessibles. A l'instar des autres mouvements du même type, tel que « l'Open Source », le traitement et la rediffusion des données sont autorisées, voir même encouragées comme c'est le cas par le gouvernement français : **« les données partagées trouvent des réutilisateurs qui les intègrent dans de nouveaux services à forte valeur ajoutée économique ou sociale. »**.

Comme le précise Oracle France,[2] "Une base de données publique contient des données qui sont et doivent être disponibles au public pour des raisons d'intérêt général (service publique, environnemental,…).". Cependant, une donnée accessible en Open Data ne provient pas forcément d'un organisme public, comme la société *Uber*, permettant d'accéder publiquement à des données anonymisées sur ses taxis.[3]

Dès lors, un point essentiel est l'**accès public** (sans authentification ou contrepartie par exemple), indépendamment de sa source. Enfin, Oracle France précise que "Le terme **« publique »** ne doit pas être confondu avec **« libre de droit »**." En effet, les règles relatives à leur réutilisation font l'objet d'une licence publique et universelle (tel que la *Licence Ouverte d'Etalab* pour l'immense majorité des données publiques en France), ne réclament pas ou peu de démarches pour se l'approprier.

[1] *L'ouverture des données publiques. Gouvernement.fr* [en ligne]. [s. d.]. [Consulté le 17 janvier 2021]. Disponible à l'adresse : https://www.gouvernement.fr/action/l-ouverture-des-donnees-publiques.

[2] *Qu'est-ce qu'une base de données publiques ? | Oracle France* [en ligne]. 8 février 2021. Disponible à l'adresse : https://www.oracle.com/fr/database/base-donnees-publique-definition.html.

[3] *Uber Movement : Let's find smarter ways forward, together.* [en ligne]. 8 février 2021. Disponible à l'adresse : https://movement.uber.com/cities?lang=fr-FR.

Ainsi, sur le territoire Français, des dispositifs mis en place par le gouvernement tel qu'« Etalab », chargé de la coordination et la mise en place de l'ouverture de jeux de données (par décret du 30 Octobre 2019) incarnent cette volonté de faciliter la diffusion de données ouvertes, tout en promouvant leur réutilisation.[4] De nombreuses plateformes mises en place par diverses instances opérant dans des domaines très variés ont vu le jour au cours des dernières années, allant d'organismes spécialisés dans les données géographiques comme l'Institut national de l'information géographique et forestière (IGN)[5] jusque dans le domaine des transports comme Ile de France Mobilités,[6] en passant par l'environnement et l'écologie tel que l'ADEME.[7]

Bien que cette nécessité étatique de partager l'information publique ne date pas de l'apparition du Web (comme l'explique la loi Cada de 1978), Ce dernier a permis, au-delà de la dispense de tout intermédiaire (notamment humain) entre le fournisseur et l'utilisateur, d'exploiter de nouvelles formes d'accès et surtout de consommation, en particulier via des scripts ou des algorithme écrits dans un langage de programmation afin d'automatiser la récupération de données depuis les formats de fichiers ouverts.

Ainsi, quiconque cherche à mettre en place un travail d'analyse le plus exhaustif possible d'un contexte donné peut, grâce aux plateformes et moyens cités ci-dessus, disposer très rapidement de données riches et abondantes.

De ce point de vue-là, il paraît extrêmement pertinent pour les métiers issus de l'architecture et de l'urbanisme, et en particulier le métier d'architecte, de se saisir des données issues de l'Open Data afin de renforcer leur compréhension du territoire sur lequel ils construisent, que cela soit par la simple analyse statistique ou bien la récupération d'informations géométriques d'un site.

---

[4] *Etalab - Qui sommes-nous. Le blog d'Etalab* [en ligne]. [s. d.]. [Consulté le 17 janvier 2021]. Disponible à l'adresse : https://www.etalab.gouv.fr/qui-sommes-nous.

[5] *Géoservices | Accéder au téléchargement des données libres IGN* [en ligne]. [s. d.]. [Consulté le 15 septembre 2020]. Disponible à l'adresse : https://geoservices.ign.fr/documentation/diffusion/telechargement-donnees-libres.html.

[6] *Portail Open data Île-de-France Mobilités* [en ligne]. [s. d.]. [Consulté le 17 janvier 2021]. Disponible à l'adresse : https://data.iledefrance-mobilites.fr/pages/home/.

[7] *Portail open data de l'ADEME* [en ligne]. [s. d.]. [Consulté le 17 janvier 2021]. Disponible à l'adresse : https://data.ademe.fr/.

Or, les architectes ont tendance à préférer, de par leur expertise orientée sur la conception, réclamant un esprit de synthèse affûté, les résultats explicites d'analyse de données plutôt que les données en elles-mêmes. De plus, les outils numériques sur lesquels les architectes se forment relèvent très majoritairement des domaines du dessin, de la modélisation ou de la communication plutôt que de l'analyse en elle-même, qui accentue leur besoin de résultats synthétiques « préfabriqués ».

Cependant, il existe depuis les années 2010 un certain essor des travaux de recherche basés sur des données issues en partie ou totalement de l'Open Data, et ce, grâce à un langage de programmation en particulier, dont la simplicité de la syntaxe couplée à une profusion de bibliothèques (comparables à des « plug-in ») spécialisées dans le traitement de données informatiques en ont fait un outil populaire pour la recherche d'aujourd'hui, le Python. A juste titre, ce langage est aujourd'hui très répandu au sein des Systèmes d'Informations Géographiques (SIG) tels que ArcGIS, où ses caractéristiques mentionnés ci-dessus permettent de manière accessible de mener des travaux complexes autour des données géographiques ouvertes, de l'analyse et la datavisualisation[8] à l'entraînement de modèles de prédiction.[9]

Comme l'illustre l'exemple du travail de recherche « CityEngine - Twitter » mené au « Centre for Advanced Spatial Analysis » de Londres[10] , proposant une cartographie urbaine de densité basée sur des Tweets géolocalisés dans cette même ville, un seul et unique script en Python permet à la fois de récupérer les messages sur une plage de 24 heures (via une bibliothèque , nommée « Tweepy » , permettant au code d'interagir avec l'API de Twitter), de les trier et d'en extraire leurs coordonnées et leur horaire de publication, et enfin de fournir ces données directement à l'outil de génération de modèles 3D urbains « CityEngine » (publié par l'ESRI) afin que ce

[8] *Spatial and temporal distribution of service calls using big data tools | ArcGIS for Developers* [en ligne]. [s. d.]. [Consulté le 3 février 2021]. Disponible à l'adresse : https://developers.arcgis.com/python/sample-notebooks/spatial-and-temporal-trends-of-service-calls/.

[9] *Automate Road Surface Investigation Using Deep Learning | ArcGIS for Developers* [en ligne]. [s. d.]. [Consulté le 3 février 2021]. Disponible à l'adresse : https://developers.arcgis.com/python/sample-notebooks/automate-road-surface-investigation-using-deep-learning/.

[10] HÜGEL, Stephan et ROUMPANI, Flora. *CityEngine-Twitter* [logiciel]. [S. l.] : Zenodo, 14 mai 2014. [Consulté le 28 décembre 2020]. DOI 10.5281/ZENODO.9795.

dernier puisse constituer une carte procédurale (animée selon le nombre de tweets sur une plage de 24 heures).

Certaines agences d'architecture telles que MVRDV[11] promeuvent et expérimentent déjà le fait de concevoir à partir d'une masse de données (plus communément appelé *"Design by data"*), et ce depuis les années 2000. *Metacity/Datatown*,[12] projet de recherche datant de 1999 abordait déjà cette question des données pour aborder la conception urbaine, où il y était affirmé à plusieurs reprises : "Datatown is based only upon data".

Une telle étude étant désormais possible sur des données massives privées, ce type d'exploitation peut encore plus aisément être mis en place lorsque les données utilisées sont totalement ouvertes et avec accès illimité. Ainsi, grâce à des données massives accessibles (tant en termes de tarifs qu'en terme de facilité d'extraction) couplées à un langage de programmation comme Python, développer ses propres analyses par exploitation de données brutes est désormais à la portée des chercheurs, sans avoir besoin d'un bagage informatique conséquent.

Dès lors, face à la complexité des enjeux auxquels la conception architecturale fait appel (climatique, socio-économique, écologique ou structurel par exemple), il semble pertinent d'envisager que des architectes se saisissent de ce type d'outil, dans le but de construire, au prisme de leurs propres volontés d'intervention (même complexes), leurs propres modèles de compréhension du territoire. Ce nouveau regard, personnalisé par l'architecte, pourrait alors apporter à ce dernier des éléments susceptibles de le guider de manière bien plus significative, en particulier dans les premières phases d'esquisse, afin d'améliorer la qualité de sa production.

Ainsi, dans quelle mesure l'exploitation de données issues de l'Open Data grâce au langage Python

[11] *MVRDV - NEXT* [en ligne]. 8 février 2021. Disponible à l'adresse : https://www.mvrdv.nl/themes/15/next.
[12] *MVRDV - Metacity / Datatown* [en ligne]. 8 février 2021. Disponible à l'adresse : https://www.mvrdv.nl/projects/147/metacity-%2F-datatown-.

représente-t-elle un avantage certain pour l'architecte ? Après avoir initialement démontré l'intérêt du langage Python dans l'extraction et la manipulation des données issues des plateformes accessibles en Open Data à travers l'élaboration complète d'un script de récolte de données, ce dernier sera complété à travers un aperçu constitué d'exemples clés de la capacité de Python à produire des documents de travail utiles à l'architecte (cartographie, dessin et modélisation). Enfin, ce travail d'exploitation sera abouti en montrant la prodigieuse capacité du langage Python à permettre de manière accessible l'analyse complexe de ces données ainsi que la mise en place d'algorithmes de prédiction.

1 Le script : outil d'exploitation privilégié de l'Open Data (Etude de cas des données des « volumes bâtis » de l'Open Data Paris.)

Tel que le stipule le portail européen de données, au-delà de l'accessibilité en elle-même des données, la question de la lisibilité des structures de données et des formats de fichiers disponibles sur les plateformes relevant de l'Open Data est d'importance cruciale : « On peut utiliser les données car elles sont disponibles sous une forme commune et lisibles par des machines. ».[13] Cet organisme relève également un autre aspect primordial, celui de la facilité du traitement des données par les outils informatiques. En effet, elles ont davantage vocation à faire l'objet de manipulations automatiques (synthèse, tri, etc…) plutôt que d'être simplement lues par un utilisateur humain.

Pour partager des données tabulaires (sous forme de tableur) par exemple, là où un utilisateur humain préfèrera un format Excel (.XLSX) (en y incluant notamment couleurs et styles de polices pour améliorer sa lisibilité), le portail européen des données recommande plutôt d'autres formats comme le .CSV (Comma Separated Values), format ouvert constitué de texte brut séparé par des caractères spéciaux, compatible avec un large panel d'outils logiciels capable d'opérations de traitement.

Face à ce besoin de compréhension et de manipulation de données brutes, les langages de programmation de haut niveau d'abstraction (possédant une syntaxe plus lisible et concise pour l'humain, rendant leur utilisation accessible) et en particulier le Python apparaissent alors comme des outils offrant la souplesse et la puissance nécessaire pour répondre à cette problématique.

Au sein de cette section, le jeu de données « Volumes bâtis » de la plateforme Open Data Paris sera étudié de près en tant qu'exemple type, à travers une approche concrète. **Plus précisément, nous chercherons à**

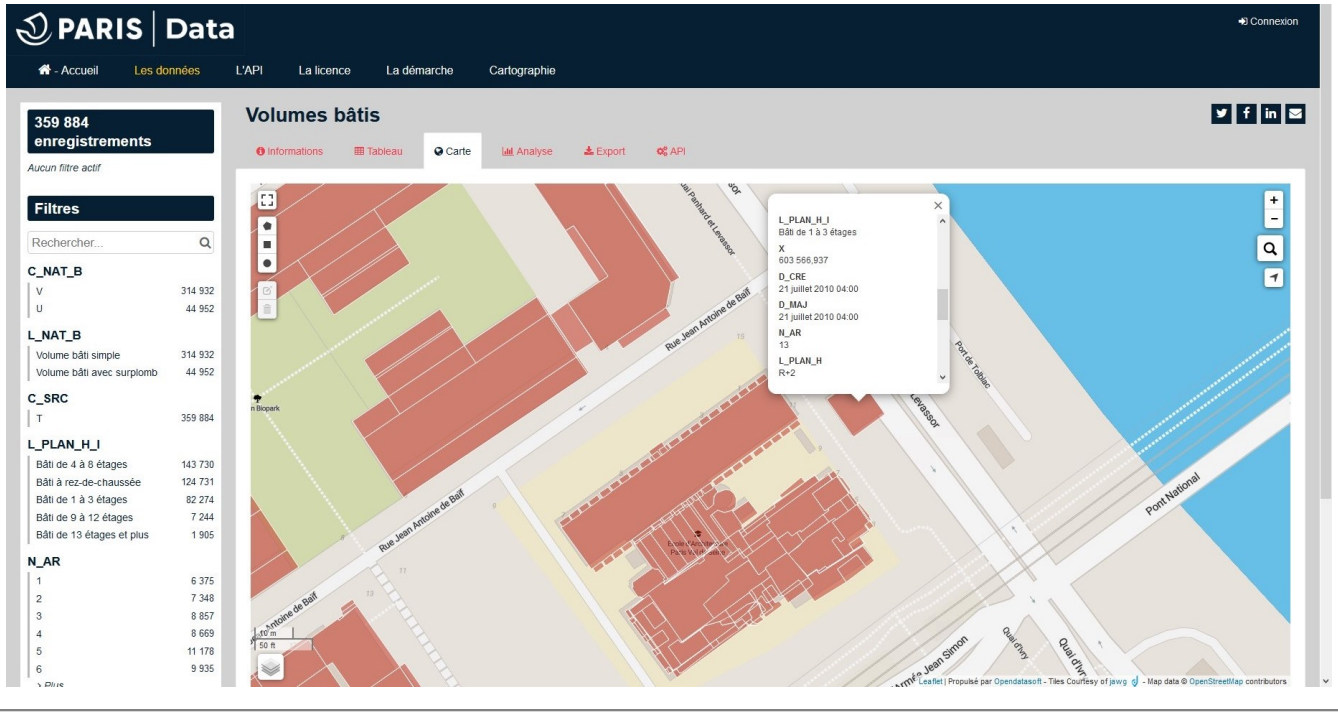
[13] *What is open data ?* [en ligne]. [s. d.]. [Consulté le 28 décembre 2020]. Disponible à l'adresse : https://www.europeandataportal.eu/elearning/en/module1/#/id/co-01.

identifier et extraire toute information relative à la morphologie (emprise et hauteur) à travers un script Python. Ce travail servira également de base pour aborder les concepts plus approfondis des chapitres suivants.

1.1 L'Open Data : entre nomenclature et variables

La plupart des plateformes distribuant des données en Open Data proposant directement en ligne des moyens de prévisualiser un jeu de données, cela semble constituer un moyen pratique de discerner et comprendre son contenu en détail. C'est le cas sur la plateforme Open Data Paris, qui nous permet de prévisualiser le jeu de données des volumes bâtis sous la forme d'un tableau, mais aussi d'une carte, laquelle formera un premier contact avec les données en elle-mêmes.

Fig 1.1 : Prévisualisation cartographique du jeu de données des volumes bâtis.

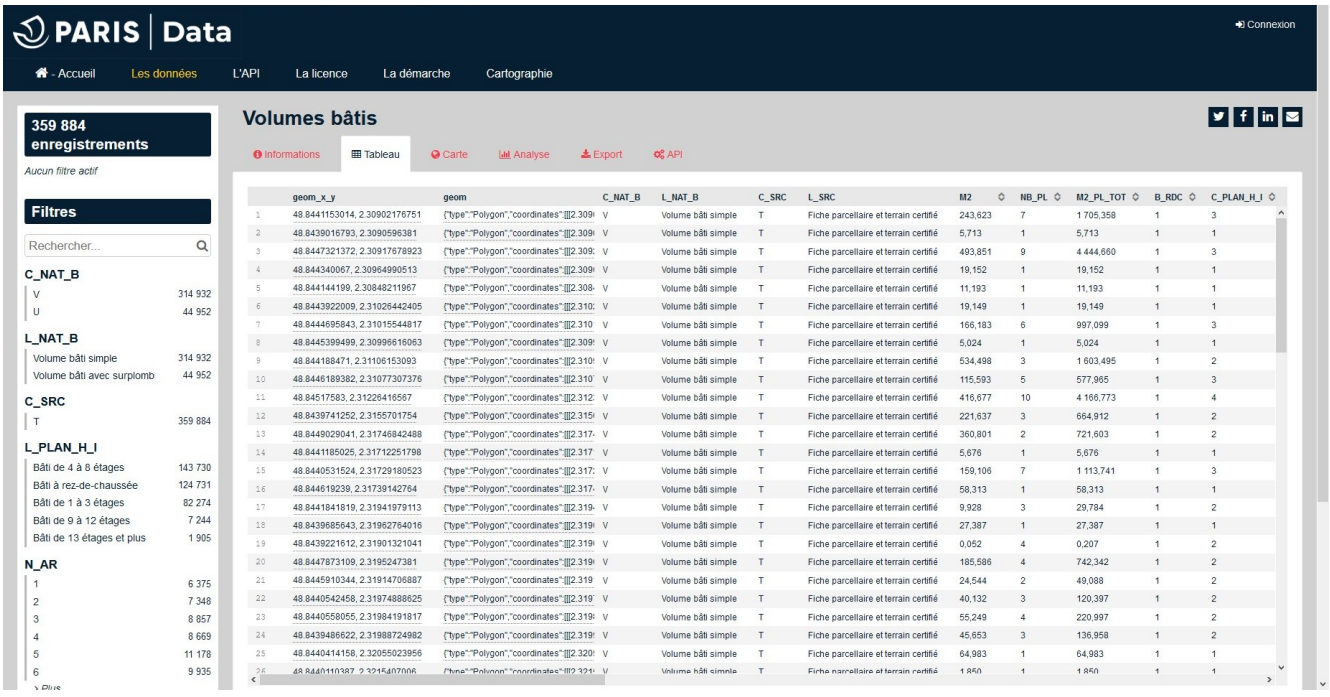


1.1.1 Variables et typologies des valeurs

Le premier constat que l'on peut réaliser après avoir brièvement interagi avec la carte est que le jeu de donnée associe un ensemble de **variables** (dont la dénomination est commune à l'ensemble de ce jeu) et leurs **valeurs** (possédant elles aussi une notation spécifique) avec une **forme géométrique géolocalisée** sur un fond de carte (en l'occurrence, ce sont des **polygones**, formes géométriques les plus à même de représenter l'emprise en plan des différents bâtiments).

Afin de permettre une lecture plus complémentaire, il paraît intéressant de consulter le tableau fin d'avoir une vue plus "centrée" sur les différentes variables et leurs valeurs.

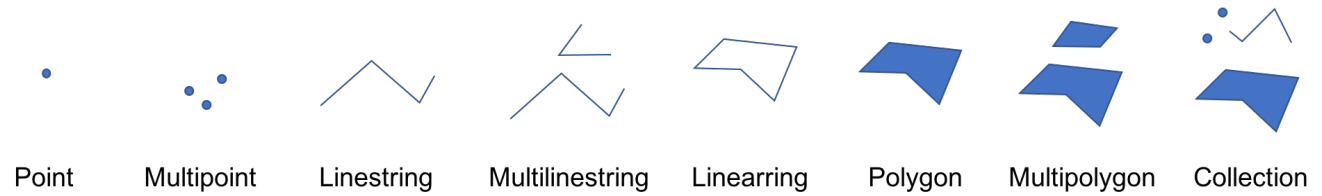
Fig 1.2 : Prévisualisation du jeu de données des volumes bâtis sous forme de tableau.



Chacune d'entre elles est ici représentée par une **colonne**, chaque ligne correspondant à un **volume bâti**.

Tout d'abord, la variable *geom* est celle qui contient les informations géométriques, nous renseignant sur le type de géométrie employée, ainsi que les coordonnées des points qui la définissent. En l'occurence, la typologie géométrique "Polygon" se base sur les types primitifs de références des Systèmes d'Informations Géographiques (SIG), et ses coordonnées sont définies en **latitude/longitude** (ce que confirme la variable *geom_x_y*).

Fig 1.3 : Primitives géométriques des données géoréférencées



Nous pouvons également noter que certaines variables comme *L_NAT_B* ou *L_SRC* sont exprimées sous forme de texte, qualifié alors de **"chaîne de caractères"** ("string" ou "str" en anglais) d'un point de vue informatique. Elles semblent également **catégoriques**, c'est à dire ne pouvant prendre qu'un nombre défini de valeurs possibles. Bien que les noms de ces variables ne soient pas explicites, leurs valeurs permettent d'avoir une première idée de ce qu'elles renseignent.

A l'inverse, d'autres variables comme *B_RDC* ne possèdent ni un nom explicite, ni une valeur permettant de suggérer sa signification, étant catégorique mais notée sous forme d'**entiers**.

Enfin, d'autres variables comme *M2* ou *NB_PL* sont notées **numériquement**, pouvant à priori prendre une infinité de valeurs , respectivement sous la forme de **réels** (ou nombres à virgule, qualifiés de *float* en anglais), et d'**entiers** (*int*). Bien que l'on puisse deviner que *M2* semble représenter la surface d'un volume bâti, cela reste une supposition.

Rappelons également que toutes ces observations sont faites sur un échantillon visible d'un jeu de données massif. Certaines subtilités présentes plus loin dans le tableau peuvent encore échapper à cette lecture préliminaire.

Dès lors, chaque variable possédant sa **propre nomenclature**, et étant plus ou moins explicite dans sa dénomination, une première difficulté de lecture émerge. Heureusement, les jeux de données en Open Data disposent généralement d'informations complémentaires capables de renseigner l'utilisateur sur ces nomenclatures.

1.1.2 Les métadonnées : clé de compréhension des données

Comme c'est le cas ici, la majorité des jeux de données accessibles en Open Data disposent d'un document annexe de référence, dont le but est à minima de fournir à l'utilisateur qui souhaite se saisir des données contenues les explications nécessaires à la compréhension des variables constituant le jeu de données en question. Ce sont **les métadonnées**.

Elles peuvent également contenir des informations complémentaires concernant le fournisseur, la manière dont les données ont été acquises ou encore d'éventuelles limites de précision et recommandations d'utilisation par exemple.

Fig 1.4 : Page descriptive issue des métadonnées

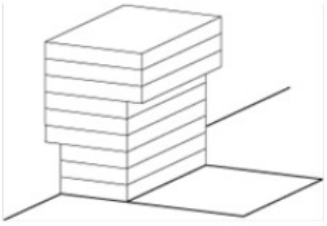
VILLE DE PARIS - DU | Fiche : **VOLUME BATI** | Dernière rédaction : 07/07/2017

VOLUME BATI	**Producteur** : Ville de Paris / Direction de l'Urbanisme Département de la Topographie et de la Documentation Foncière **Actualité** : avril 2017

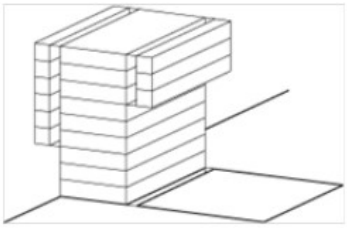
1 DÉFINITION

1.1 Définition de l'objet

Donnée vecteur qui décrit les bâtiments de manière détaillée en différenciant les bâtis en fonction de leur hauteur et des parties en saillie ou en retrait, définissant ainsi des volumes.



Remarques
Les volumes bâtis décrivent les bâtiments tels que représentés sur le plan parcellaire raster géré jusqu'en 2015 par le Service de la Topographie et de la Documentation Foncière (STDF). La couche vecteur des volumes bâtis décrit les parties en saillie sur la voie publique, alors qu'elles ne sont pas figurées sur le plan parcellaire raster.

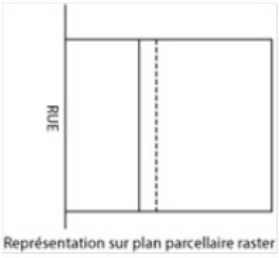


1.2 Étendue géographique

Paris.

1.3 Contraintes géographiques et topologiques

Un volume bâti est formé d'un seul polygone.
Cas des surfaces à trou : actuellement la couche ne gère pas les objets à trou.



Représentation sur plan parcellaire raster

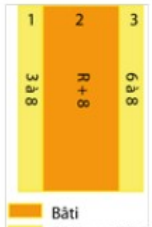
1.4 Identification et clés

Nom informatique de l'objet : VOLUME_BATI

Code court de l'objet : VB

Identification informatique : L'identifiant (N_SQ_VB) est un numéro séquentiel unique.

Clé sémantique : Pas de clé sémantique.



Couche Volume bâti

1.5 Changement d'identifiant et disparition d'objets

Changement de l'identifiant :

Conservation de l'identifiant : l'identifiant ne change pas lors de modifications géométriques des objets

Disparition de l'identifiant : l'identifiant disparait quand le bâtiment est démoli.

1.6 Limite d'usage et remarques

Néant

En l'occurence, la première page nous renseigne de manière plus exhaustive sur la manière dont ont été tracés les différents polygones, à travers quelques schémas, ainsi qu'un paragraphe exprimant la source de ces tracés. Premièrement, ce document explique sa logique de séparer un bâtiment "réel" en plusieurs volumes fictifs, suivant s'ils sont en porte à faux ou non, permettant d'apporter une certaine précision.

Dès lors, les deux informations primordiales associées à chaque polygone sont sa **hauteur**, ainsi que ses différents **intervalles de hauteur** s'il est en porte à faux. cette fiche indique également le contexte géographique, ainsi que les limitations géométriques (empêchants ici de représenter un polygone "évidé", obligeant à le sectionner si l'on veut représenter de manière correcte un patio par exemple).

Fig 1.5 : Table descriptive des variables issue des métadonnées

Nom	Libellé	Type	O	Valeurs possibles
N_SQ_VB	Identifiant séquentiel du volume bâti	N	O	
C_NAT_B	Code nature du volume bâti	C 1	O	V : Volume bâti simple U : Volume bâti avec surplomb
L_NAT_B	Nature du volume bâti	C 30		
C_SRC	Code nature de la source d'information	C 1		F : fiche parcellaire T : fiche parcellaire et terrain certifié C : fiche parcellaire et terrain non certifié
L_SRC	Nature de la source d'information	C 50		
M2	Surface graphique (m²)	N		
NB_PL	Nombre de planchers	N		
M2_PL_TOT	Surface totale de planchers (m²)	N		
H_ET_MAX	Hauteur max (nb étages/sol)	N		
C_PLAN_H_I	Classification du plan Hauteur d'étages	N		1 : Bâti à rez-de-chaussée 2 : Bâti de 1 à 3 étages 3 : Bâti de 4 à 8 étages 4 : Bâti de 9 à 12 étages 5 : Bâti de 13 étages et plus
L_PLAN_H_I	Libellé des classes du plan Hauteur d'étages	C 50		
L_PLAN_H	Description du plan Hauteur d'étages	C 10		Ex : R+2
B_RDC	Présence d'un RDC ? (1=vrai ; 0 = faux)	N		
L_B_U	Détail du volume avec surplomb	C 100		Ex : 1 à 5
X	Coord. X centre du polygone	N		
Y	Coord. Y centre du polygone	N		
N_AR	Numéro d'arrondissement (Bois séparés)	N		
N_QU	Numéro de quartier (Bois séparés)	N		
D_CRE	Date de constitution	D		
D_MAJ	Date de la dernière modification	D		Renseignée à la date de constitution de la donnée en l'absence de modification.

Enfin, la seconde page contient les informations cruciales concernant les données à exploiter.

En effet, le tableau-ci-dessus renseigne sur le **libellé** de chaque variable (son contenu explicite), son **type** (ici, **C*n*** où *n* est un entier signifie que les valeurs sont sous forme textuelle de *n* caractères de long, tandis que **N** désigne simplement des valeurs numériques), ainsi que ses valeurs possibles si ces dernières sont prédéfinies (servant à distinguer les variables **catégoriques**).

Dès lors, il est possible de repérer les deux variables les plus pertinentes si l'on souhaite extraire la hauteur des différents volumes. En l'occurence, ce seront les variables **H_ET_MAX** ainsi que **L_B_U** (pour les volumes en porte à faux), toutes deux exprimées en nombre d'étages. La surface de plancher totale **M2_PL_TOT** est également intéressante à extraire.

Ainsi, les métadonnées offrent les clés de compréhension nécessaires à l'utilisateur afin de comprendre le contenu d'un jeu de données en profondeur. Cette prise de connaissance permet désormais de manipuler les données en elle-mêmes, au sein d'un script en Python.

1.2 Le langage Python pour s'approprier aisément les données ouvertes

Afin d'exploiter ce jeu de données constitué d'objets géolocalisées et leurs données, le langage de programmation **Python** sera exclusivement employé. Comme mentionné dans l'introduction, ce dernier possède toutes les fonctionnalités nécessaires pour manipuler simplement ce type de données.

La plateforme Open Data Paris permettant de définir un périmètre afin de restreindre le jeu de données directement sur la carte, cette fonction sera utilisée afin d'en télécharger un échantillon (en l'occurence localisé autour de l'ENSAPVS).

Fig 1.6 : Définition d'une zone d'extraction de données des volumes bâtis

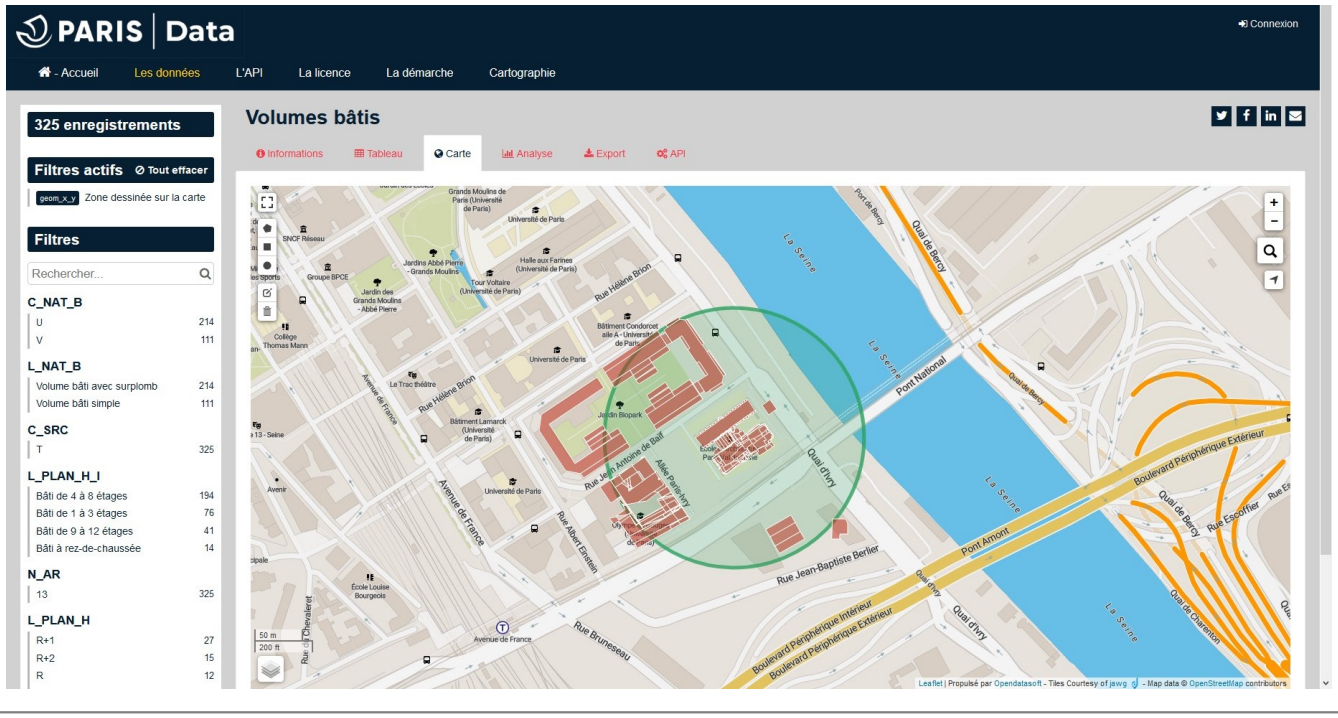
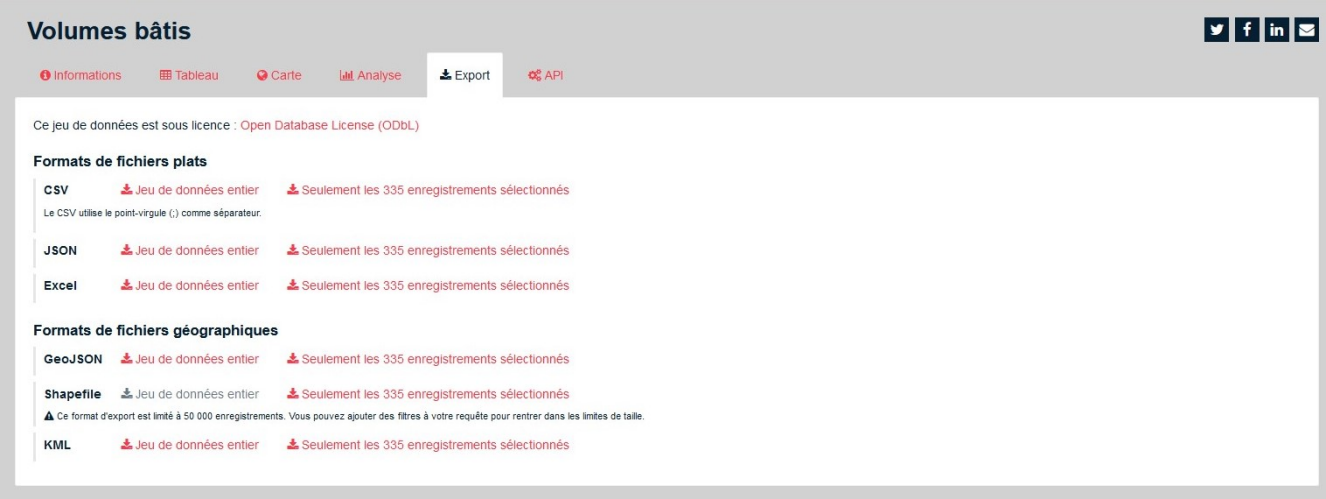


Fig 1.7 : Un choix étendu de formats de fichiers

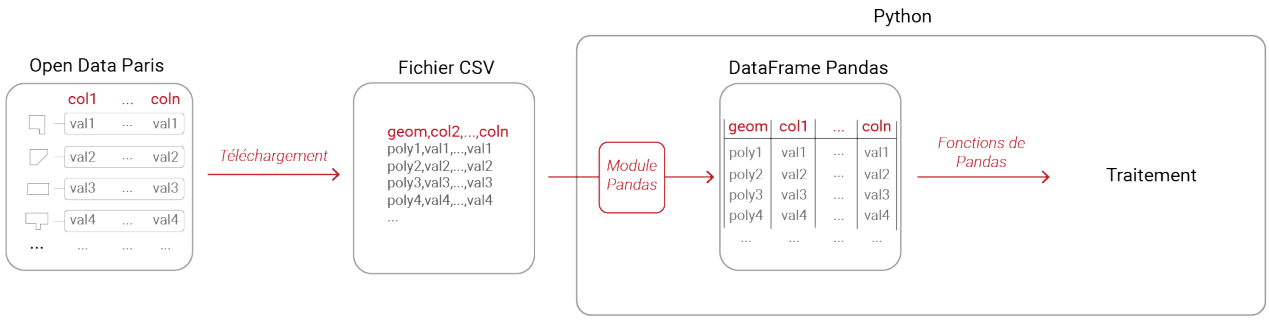


A l'instar d'autres jeux de données, celui des "volumes bâtis" propose **plusieurs formats** lors du téléchargement.

Dès lors, grâce à son large éventail de **bibliothèques** (comparables à des "extensions") le langage Python se révèle ici précieux car il est **capable de manipuler tous les formats de fichiers proposés**. Or, le choix d'un format en particulier pourrait alors être perçu comme un "non-problème", étant donné de telles capacités. Ainsi, une **brève entrevue** sera apportée sur chacun de ces formats grâce à Python, afin de déterminer lequel choisir.

1.2.1 Les formats tabulaires

Fig 1.8 : Chargement du jeu de données sous un format tabulaire dans Python



Chargement du jeu de données sous un format tabulaire dans Python

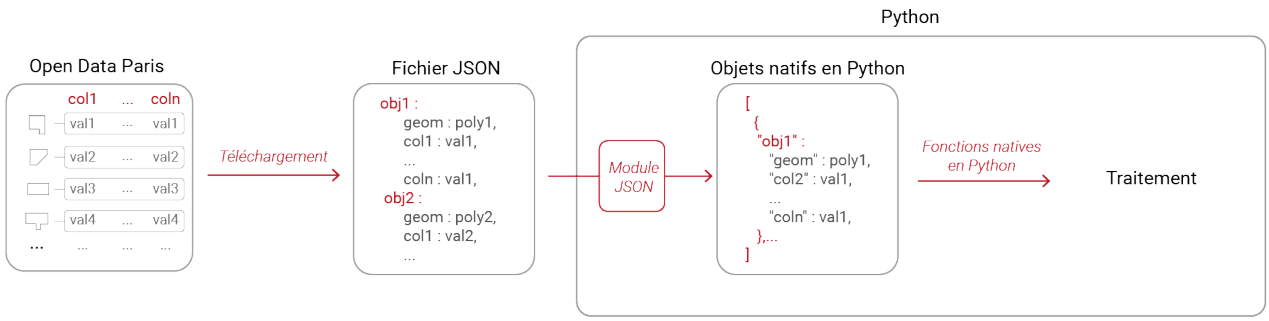
```
# Import de la bibliothèque pandas
import pandas
# Lecture du fichier CSV
data = pandas.read_csv("DONNEES/volumesbatisparis.csv",sep=";")
# Affichage du premier élément de la colonne geom
geom = data["geom"][0]
print(type(geom))
```

```
## <class 'str'>
```

Tout d'abord, les formats conventionnels de type tableur comme **CSV** ou **Excel** représentent des choix inadaptés. En effet, dû au fait que le tableur est limité à **un type de valeur par colonne**, en l'occurence **soit du texte, soit des chiffres**, des variables telles que **GEOM**, d'une structure plus complexe s'intègrent ici très mal. Cela peut être prouvé grâce à la bibliothèque **Pandas**, spécialisée dans la manipulation de formats tableurs, en recréant dans Python un objet nommé *DataFrame*, composé de lignes et de colonnes. En l'occurrence, la variable **GEOM** a perdu sa structure, étant notée telle quelle sous forme de **texte** (*str*).

1.2.2 Les formats hiérarchisés

Fig 1.9 : Chargement du jeu de données sous un format hiérarchisé dans Python



Chargement du jeu de données sous un format hiérarchisé dans Python

```
# Import de la bibliothèque json
import json
# Lecture du fichier JSON
data = json.load(open("DONNEES/volumesbatisparis.json","r"))
# premier objet de la liste de volumes
print(data[22]["fields"]["geom"]["coordinates"])
```

```
## [[[2.382921195270158, 48.82676576669778], [2.382937421271572, 48.826750336247784],
    [2.382934765837276, 48.826749121884085], [2.382917547695962, 48.82676551493904],
    [2.382921195270158, 48.82676576669778]]]
```

Ensuite, le format **JSON** semble adapté à la structure de la variable **GEOM**, permettant ici d'en extraire les sous-informations. Cependant, il est nécessaire au préalable de **lire le fichier en lui même** via un éditeur de texte, afin d'en repérer la hiérarchie. Cette dernière se repose sur la **notation objet**, une syntaxe commune à de nombreux langages de programmation modernes comme le Python. Elle est constituée d'ensembles (listes) de couples **attribut : valeur**, que l'on peut **hiérarchiser** en les imbriquant.

Chargement du jeu de données sous un format hiérarchisé dans Python

```
expl = {
  "datasetid" : "volumesbatisparis",
  "recordid" : "9cb9b7143c595f9cd2b88e4d5bc588112fedc5a8",
  "fields" : {
    "objectid" : 536158,
    "n_sq_pf" : 750031416,
    "d_maj" : "2010-07-21T04 :00 :00+02 :00",
    "l_src" : "Fiche parcellaire et terrain certifié",
    "l_b_u" : "Ra1_et_encorbt_au_3",
    "h_et_max" : 3.0,
    "b_rdc" : 1.0,
    "n_ar" : 13,
    "m2_pl_tot" : 12.7752023,
    "d_cre" : "2010-07-21T04 :00 :00+02 :00",
    "m2" : 4.2584008,
    "l_plan_h_i" : "Bâti de 1 à 3 étages",
    "n_sq_qu" : 750000050,
    "geom_x_y" : [
        48.8272714473,
        2.38477660061
    ],
    "l_nat_b" : "Volume bâti avec surplomb",
    "n_qu" : 50.0,
    "c_plan_h_i" : 2.0,
    "shape_area" : 0.0,
    "c_src" : "T",
    "nb_pl" : 3.0,
    "n_sq_vb" : 750170119,
    "shape_len" : 0.0,
    "c_nat_b" : "U",
    "geom" : {
      "type" : "Polygon",
      "coordinates" : [[
                        [
                            2.384770481308271,48.827286816032526
                        ],
                        [
                            2.384796971933453,48.82726286263405
                        ],
                        [
                            2.384782602464847,48.8272560560162
                        ],
                        [
                            2.384756236578184,48.82728017373901
                        ],
                        [
                            2.384770481308271,48.827286816032526
                        ]
                    ]]
            },
            "n_sq_ar" : 750000013,
            "y" : 125199.7506478,
            "x" : 603542.699385
  }}
```

Dès lors, une fois cette notation assimilée, ce format permet une lecture relativement **aisée** pour l'utilisateur, tout en restant **simple** à lire par la machine. Le code d'extraction découle donc naturellement de la lecture

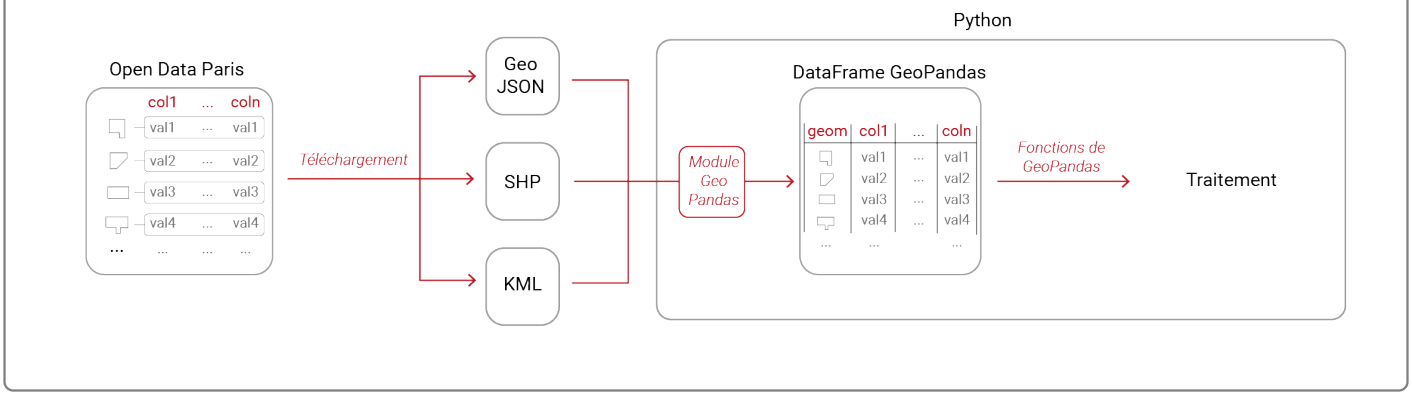
humaine.

1.2.3 Les formats géographiques

Enfin, puisque le jeu de données en question est constitué **d'objets géoréférencés**, les formats **GeoJSON**, **Shapefile** et **KML** semblent être les formats les plus adaptés. Or, chacun de ces formats possède **sa propre notation normée pour les données géographiques**, ce qui nécessite habituellement l'utilisation d'outils spécifiques comme **QGIS** afin de les lire.

Cependant, le langage Python possède une bibliothèque nommée **GeoPandas** spécialisée dans la **manipulation de données géoréférencées**. De manière similaire à la bibliothèque **Pandas** montrée plus haut, elle permet de traduire une collection d'**objets géoréférencés** en un objet *DataFrame* similaire à un tableur, cette fois-ci sans limitations au niveau des types de données. En outre, cette représentation prodigue un grand nombre de **fonctions** appliquables aux lignes et aux colonnes permettant par la suite d'opérer des transformations de manière aisée.

Fig 1.10 : Chargement du jeu de données sous un format géographique dans Python



Chargement du jeu de données sous un format géographique dans Python

```
# Import de géopandas sous l'acronyme gpd
import geopandas as gpd
# Spécification nécessaire à la lecture du KML
gpd.io.file.fiona.drvsupport.supported_drivers['KML'] = 'rw'
# Lecture du fichier KML et du premier objet de la colonne "geom"
data = gpd.read_file("DONNEES/volumesbatisparis.kml", driver='KML')
print(data)
# Lecture du fichier SHP et du premier objet de la colonne "geom"
```

```
##     Name Description                                           geometry
## 0      V              POLYGON ((2.38462 48.82753, 2.38461 48.82753, ...
## 1      U              POLYGON ((2.38475 48.82722, 2.38472 48.82721, ...
## 2      U              POLYGON ((2.38493 48.82732, 2.38490 48.82730, ...
## 3      U              POLYGON ((2.38354 48.82665, 2.38350 48.82664, ...
## 4      U              POLYGON ((2.38304 48.82670, 2.38300 48.82669, ...
## ..   ...          ...                                               ...
## 330    U              POLYGON ((2.38469 48.82720, 2.38467 48.82719, ...
## 331    U              POLYGON ((2.38451 48.82752, 2.38450 48.82752, ...
## 332    U              POLYGON ((2.38453 48.82753, 2.38451 48.82752, ...
...
```

```
data = gpd.read_file("DONNEES/volumesbatisparis.shp")
print(data)
# Lecture du fichier GeoJSON et du premier objet de la colonne "geom"
```

```
##     c_nat_b  ...                                           geometry
## 0         V  ...  POLYGON ((2.38462 48.82753, 2.38463 48.82753, ...
## 1         U  ...  POLYGON ((2.38475 48.82722, 2.38476 48.82721, ...
## 2         U  ...  POLYGON ((2.38493 48.82732, 2.38495 48.82730, ...
## 3         U  ...  POLYGON ((2.38354 48.82665, 2.38323 48.82652, ...
## 4         U  ...  POLYGON ((2.38304 48.82670, 2.38315 48.82659, ...
## ..      ...  ...                                               ...
## 330       U  ...  POLYGON ((2.38469 48.82720, 2.38469 48.82720, ...
## 331       U  ...  POLYGON ((2.38451 48.82752, 2.38452 48.82751, ...
## 332       U  ...  POLYGON ((2.38453 48.82753, 2.38454 48.82752, ...
...
```

```
data = gpd.read_file("DONNEES/volumesbatisparis.geojson")
print(data)
```

```
##      objectid  ...                                           geometry
## 0      536092  ...  POLYGON ((2.38462 48.82753, 2.38463 48.82753, ...
## 1      536130  ...  POLYGON ((2.38475 48.82722, 2.38476 48.82721, ...
## 2      536134  ...  POLYGON ((2.38493 48.82732, 2.38495 48.82730, ...
## 3      536186  ...  POLYGON ((2.38354 48.82665, 2.38323 48.82652, ...
## 4      536243  ...  POLYGON ((2.38304 48.82670, 2.38315 48.82659, ...
## ..        ...  ...                                               ...
## 330    536070  ...  POLYGON ((2.38469 48.82720, 2.38469 48.82720, ...
## 331    536084  ...  POLYGON ((2.38451 48.82752, 2.38452 48.82751, ...
## 332    536086  ...  POLYGON ((2.38453 48.82753, 2.38454 48.82752, ...
...
```

De plus, **les objets géoréférencés** sont automatiquement convertis en formes géométriques, permettant d'y appliquer des manipulations géométriques spécifiques.

Chargement du jeu de données sous un format géographique dans Python

```
# Calcul du centroide pour le premier objet en une seule ligne
print(data["geometry"][0].centroid)


## POINT (2.384586335419862 48.82747966979255)
```

Pour toutes ses qualités, cette approche avec **GeoPandas** sera privilégiée à travers les chapitres suivants. Enfin, quant au format de fichier en lui-même, le **GeoJSON** sera ici arbitrairement choisi.

Ainsi, grâce à une **immense compatibilité** avec la majorité des types de formats courants en Open Data (prodiguée par ses nombreuses bibliothèques), le langage Python constitue d'ores-et-déjà un formidable outil d'extraction, permettant de **s'approprier facilement** des jeux de données à la **structure complexe**. Cependant, son véritable intérêt réside dans son statut de langage de programmation, étant alors capable de traitements automatiques avancés.

1.3 Aperçu de la souplesse des fonctions de manipulation de données

Après la phase d'extraction précédente, il est nécessaire que les données etraites en l'état soient exploitables pour la suite.

Premièrement, le jeu de données sera réduit aux variables identifiées lors de la lecture des métadonnées (à savoir **GEOM**,**M2_PL_TOT**,**H_ET_MAX** ainsi que **L_B_U**), et seront renommées afin d'être plus facilement lisible par la suite.

Chargement du jeu de données sous un format géographique dans Python

```
data = data[["geometry","m2_pl_tot","h_et_max","l_b_u"]]
data = data.rename(columns={
    "m2_pl_tot"  : "surface",
    "h_et_max"  : "hauteur",
    "l_b_u"  : "hauteur_paf"
    })
print(data)
```

```
##                                                geometry  ...          hauteur_paf
## 0    POLYGON ((2.38462 48.82753, 2.38463 48.82753, ...  ...                 None
## 1    POLYGON ((2.38475 48.82722, 2.38476 48.82721, ...  ...                  5a8
## 2    POLYGON ((2.38493 48.82732, 2.38495 48.82730, ...  ...           Ra1_et_3a9
## 3    POLYGON ((2.38354 48.82665, 2.38323 48.82652, ...  ...                  2a8
## 4    POLYGON ((2.38304 48.82670, 2.38315 48.82659, ...  ...                  3a9
## ..                                                 ...  ...                  ...
## 330  POLYGON ((2.38469 48.82720, 2.38469 48.82720, ...  ...                  2a4
## 331  POLYGON ((2.38451 48.82752, 2.38452 48.82751, ...  ...    R_et_encorbt_au_3
## 332  POLYGON ((2.38453 48.82753, 2.38454 48.82752, ...  ...  Ra1_et_encorbt_au_5
...
```

Dès lors, il serait souhaitable d'exprimer les hauteurs en **mètres** plutôt qu'en nombre de niveaux.

Chargement du jeu de données sous un format géographique dans Python

```
h_etage = 3
print(data["hauteur"][0] * h_etage)
```

```
## 9.0
```

Cette opération est triviale pour l'attribut **hauteur**, étant exprimé numériquement. Il suffit alors de définir une **hauteur d'étage type** et d'effectuer une multiplication.

Cependant, l'attribut **hauteur_paf** étant exprimé sous la forme de **texte** (chaînes de caractères) décrivant les

plages de hauteur, il est nécessaire de convertir cette notation numériquement afin d'effectuer cette multiplication.

1.3.1 Approche compréhensive des différentes notations grâce à Python

Heureusement, Python est capable de reconnaître des morceaux de texte au sein de chaînes de caractères, permettant ainsi de les remplacer par leur équivalent numérique.

Chargement du jeu de données sous un format géographique dans Python

```
if "da" in "data" :
  print(1)
else :
  print(0)
```

```
## 1
```

Dès lors, afin de comprendre les différentes manières d'exprimer les hauteurs en porte à faux, il est possible de les énumérer par la **longueur de leur texte** de chacune des notations.

Pour ce faire, un **dictionnaire** vide sera préalablement créé. C'est le type de donnée utilisé en Python afin de représenter la notation objet, en associant un **attribut** (sous forme de texte ou d'entiers) avec une **valeur** (pouvant être de type varié).

Chargement du jeu de données sous un format géographique dans Python

```
# Dictionnaire à deux objets
d = {"A"  : 10, "B"  : 15}
# Récupération de la valeur associée à "A"
print(d["A"])
# Changement de la valeur associée à "B"


## 10


d["B"] = 30
print(d["B"])


## 30
```

Une **boucle** avec **for** servira ensuite à itérer à travers les différentes notations de la variable **l_b_u**.

Chargement du jeu de données sous un format géographique dans Python

```
# Liste de valeurs
l = [0,1,2,"trois"]

for item in l :
  print(item)


## 0
## 1
## 2
## trois
```

Comme le montre le code ci-dessous, pour chaque objet (**for** *objet*) contenu dans la liste l (*in l*), la boucle affecte un nom de variable défini (ici, **item**), permettant d'opérér sur chaque objet individuellement.

Ainsi, en itérant à travers chaque notation contenue dans la colonne **hauteur_paf**, un **exemple de notation** sera affecté au dictionnaire vide créé précédemment en prennant comme attribut sa **longueur**. En réalité, pour chaque longueur, c'est la dernière notation qui sera conservée, chacune d'elle écrasant la précédente. Enfin, la fonction *dropna()* est employée ici pour supprimer les valeurs nulles pour les volumes non-concernés.

Chargement du jeu de données sous un format géographique dans Python

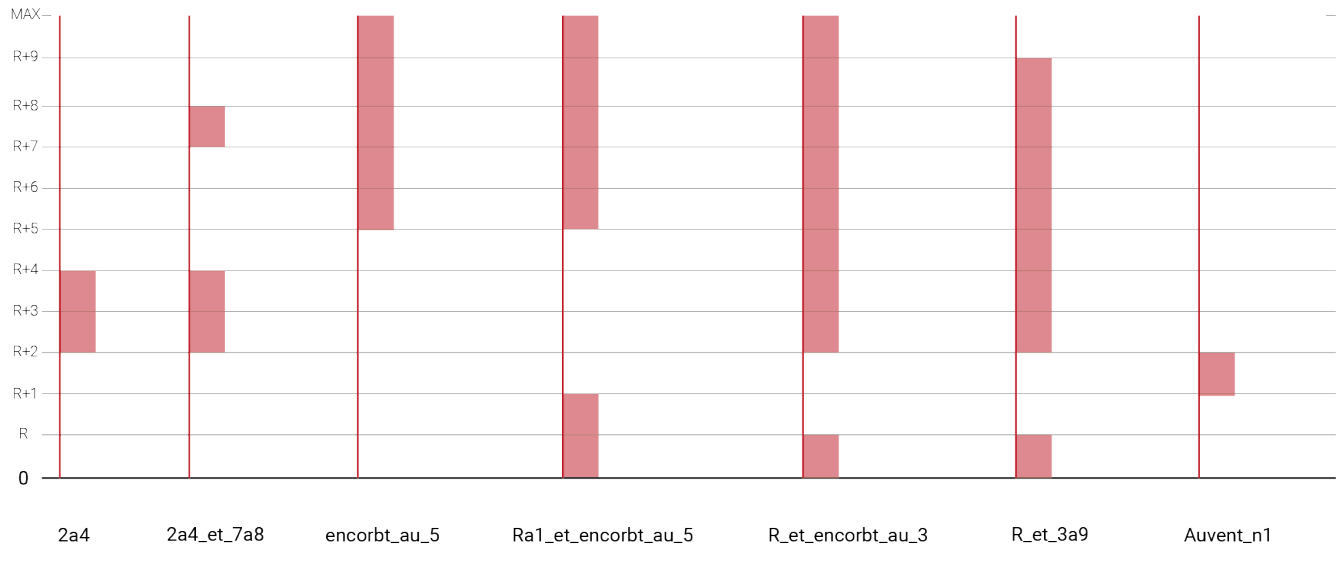
```
# Création d'un dictionnaire vide
notations = {}
# Itération à travers la colonne "l_b_u"
for txt in data["hauteur_paf"].dropna() :
  notations[len(txt)] = txt
print(notations)
```

```
## {3 : '2a4', 10 : '2a4_et_7a8', 12 : 'encorbt_au_5', 19 : 'Ra1_et_encorbt_au_5', 8 : 'R_et_3a9', 9 : '
    Auvent_n1', 4 : '4a11', 17 : 'R_et_encorbt_au_3', 15 : 'Ra1_et_3_et_5a8'}
```

Dès lors, certains éléments constituants peuvent être notés :

- Une plage de hauteur est principalement renseignée par ses **deux niveaux de hauteur** séparées par un *a*
- et est utilisé pour renseigner **plusieurs plages de hauteur**.
- *"encorbt_au_N"* désigne une plage de hauteur du niveau **N** au niveau maximal (exprimé par la variable "hauteur"). S'ils désignent le même niveau, la plage sera du niveau $\mathbf{N^{-1}}$ au niveau maximal.
- *auvent* désigne une plage du niveau **N** à $\mathbf{N^{+1}}$.
- Enfin, la présence de *R* exprimé seul suggère que **R** désigne une plage d'une **seule hauteur d'étage partant du sol**, tandis que **Ra1** désigne **deux hauteurs d'étages partant du sol**.

Fig 1.11 : Aperçu des différentes notations caractérisant les porte à faux



Cette dernière observation est cruciale : si l'on fixe **R = 0**, alors la plage **Ra1** devient **0a1**. Or, si l'on multiplie ces deux bornes par la hauteur d'étage type de 3 mètres, le volume aura une plage de hauteur de **0 à 3m**, tandis qu'il faudrait obtenir **0 à 6m**. Dès lors, il faut **ajouter 1 à tout nombre de niveau sauf R** avant multiplication par la hauteur d'étage type.

1.3.2 De la chaîne de caractère à la valeur numérique

A la lumière de toutes ces subtilités, il est désormais possible de créer une fonction capable de **transformer des chaînes de caractères** en **valeurs numériques**. Le code ci-dessous illustre la transformation d'une notation $\mathbf{N^1aN^2}$. Après s'être assuré que la notation contient bien un *a*, le code sépare les deux bornes à cet endroit, puis les convertit en **entiers**. Enfin, chaque borne est incrémentée de **1** avant multiplication par la hauteur d'étage type, sauf *R* qui prend la valeur 0.

Aperçu des différentes notations caractérisant les porte à faux

```
# Rappel : h_etage = 3 mètres
h_paf =  "Ra8"
if "a" in h_paf :
  # Séparation des deux chiffres au niveau du "a"
  hauteur = h_paf.split("a")
  # Conversion de chaque chiffre en entier, puis multiplication par la hauteur d'étage type
  # Si "R" est présent, il prend la valeur 0
  hauteur = [(int(h)+1)*h_etage if h != "R" else 0 for h in hauteur]

print(hauteur)


## [0, 27]
```

Suivant ce principe, la **fonction** suivante opèrera ce type de transformation suivant les différents notations relevées précédemment sur l'ensemble des volumes. Une **fonction** est un bloc de code que l'on définit **une seule fois** dans un script avec un **nom** et éventuellement des **arguments** (paramètres), et que l'on peut exécuter autant de fois que l'on souhaite par la suite en l'appelant par son nom. Ici, cette fonction sera appelée **texte_to_num**, et prendra pour argument chaque volume, afin de convertir la notation caractérisant son porte à faux en expression numérique. Elle appliquera également la transformation de la variable **hauteur** (à savoir un incrément de 1 suivi d'une multiplication par la hauteur d'étage type).

Aperçu des différentes notations caractérisant les porte à faux

```
# Rappel : h_etage = 3 mètres
def texte_to_num(volume) :
     # Si le volume n'est pas en porte à faux
    if volume["hauteur_paf"] == None :
        # Affecter une valeur nulle
        volume["hauteur_paf"] = None
    else :
        # Liste vide pour contenir les notations numériques
        output = []
        # Séparation des sous-notations au niveau du "_et_"
        # Si absent, la notation sera conservée telle quelle
        intervalles = volume["hauteur_paf"].split("_et_")
        # Itération à travers les sous-notations
        for interv in intervalles :
            # filtrage par syntaxe
            if "encorbt_au_" in interv :
                n = int(interv.strip("encorbt_au_"))
                if n == volume["hauteur"] :
                    output.append([n*h_etage,(n+1)*h_etage])
                else :
                    output.append([(n+1)*h_etage,(volume["hauteur"]+1)*h_etage])
            elif "auvent_n" in interv :
                n = int(interv.strip("auvent_n"))
                output.append([n*h_etage,(n+1)*h_etage])
            elif "a" in interv :
                if "R" in interv :
                    output.append([0,(int(interv.strip("Ra"))+1)*h_etage])
                else :
                    output.append([(int(h)+1)*h_etage for h in interv.split("a")])
            elif interv == "R" :
                output.append([0,h_etage])
            # Remplacement par la notation numérique
            volume["hauteur_paf"] = output

    volume["hauteur"] = (volume["hauteur"] + 1)*h_etage
    return volume
```

Enfin, cette fonction peut-être appliquée automatiquement à l'ensemble du jeu.

Aperçu des différentes notations caractérisant les porte à faux

```
# Application à chaque volume avec .apply()
data = data.apply(texte_to_num,axis=1)
# Affichage des 3 premiers volumes
print(data.head(3))
```

```
##                                             geometry  ...         hauteur_paf
## 0  POLYGON ((2.38462 48.82753, 2.38463 48.82753, ...  ...                None
## 1  POLYGON ((2.38475 48.82722, 2.38476 48.82721, ...  ...          [[18, 27]]
## 2  POLYGON ((2.38493 48.82732, 2.38495 48.82730, ...  ...  [[0, 6], [12, 30]]
##
## [3 rows x 4 columns]
```

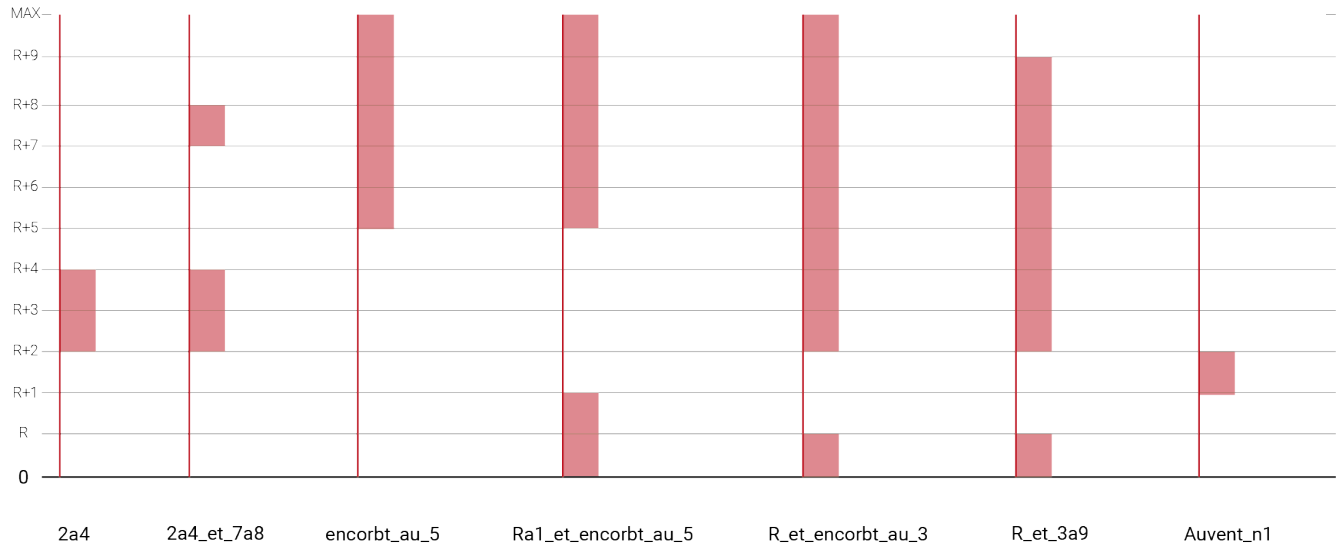
Comme démontré au cours de ce chapitre, le langage de programmation Python possède une souplesse lui permettant d'extraire et de manipuler facilement les formats de données courants en Open Data, et de palier aux

éventuelles difficultés posées par la structure ou encore le formatage des jeux de données, le tout bénéficiant d'une syntaxe claire tout au long du code. **Ainsi, à la fois outil de compréhension et de traitement, il se révèle extrêmement précieux lorsque l'on souhaite exploiter des jeux de données en Open Data.**

Dans le contexte de la profession architecturale, l'obtention de ces données n'a cependant que peu de valeur si l'architecte ne peut l'intégrer dans son environnement de travail, ce à quoi le prochain chapitre est dédié.

2 lolkdedleded

Fig 2.1 : Aperçu des différentes notations caractérisant les porte à faux



BIBLIOGRAPHIE

Automate Road Surface Investigation Using Deep Learning | ArcGIS for Developers [en ligne]. [s. d.]. [Consulté le 3 février 2021]. Disponible à l'adresse : https://developers.arcgis.com/python/sample-notebooks/automate-road-surface-investigation-using-deep-learning/

Etalab - Qui sommes-nous. Le blog d'Etalab [en ligne]. [s. d.]. [Consulté le 17 janvier 2021]. Disponible à l'adresse : https://www.etalab.gouv.fr/qui-sommes-nous

Géoservices | Accéder au téléchargement des données libres IGN [en ligne]. [s. d.]. [Consulté le 15 septembre 2020]. Disponible à l'adresse : https://geoservices.ign.fr/documentation/diffusion/telechargement-donnees-libres.html

L'ouverture des données publiques. Gouvernement.fr [en ligne]. [s. d.]. [Consulté le 17 janvier 2021]. Disponible à l'adresse : https://www.gouvernement.fr/action/l-ouverture-des-donnees-publiques

MVRDV - Metacity / Datatown [en ligne]. 8 février 2021. Disponible à l'adresse : https://www.mvrdv.nl/projects/

147/metacity-%2F-datatown-

MVRDV - NEXT [en ligne]. 8 février 2021. Disponible à l'adresse : https://www.mvrdv.nl/themes/15/next

Portail open data de l'ADEME [en ligne]. [s. d.]. [Consulté le 17 janvier 2021]. Disponible à l'adresse : https://data.ademe.fr/

Portail Open data Île-de-France Mobilités [en ligne]. [s. d.]. [Consulté le 17 janvier 2021]. Disponible à l'adresse : https://data.iledefrance-mobilites.fr/pages/home/

Qu'est-ce qu'une base de données publiques ? | Oracle France [en ligne]. 8 février 2021. Disponible à l'adresse : https://www.oracle.com/fr/database/base-donnees-publique-definition.html

Spatial and temporal distribution of service calls using big data tools | ArcGIS for Developers [en ligne]. [s. d.]. [Consulté le 3 février 2021]. Disponible à l'adresse : https://developers.arcgis.com/python/sample-notebooks/spatial-and-temporal-trends-of-service-calls/

Uber Movement : Let's find smarter ways forward, together. [en ligne]. 8 février 2021. Disponible à l'adresse : https://movement.uber.com/cities?lang=fr-FR

What is open data ? [en ligne]. [s. d.]. [Consulté le 28 décembre 2020]. Disponible à l'adresse : https://www.europeandataportal.eu/elearning/en/module1/#/id/co-01

HÜGEL, Stephan et ROUMPANI, Flora. *CityEngine-Twitter* [logiciel]. [S. l.] : Zenodo, 14 mai 2014. [Consulté le 28 décembre 2020]. DOI 10.5281/ZENODO.9795

VANNIEUWENHUYZE, Aurélien. *Intelligence artificielle vulgarisée : le Machine Learning et le Deep Learning par la pratique*. [S. l.] : [s. n.], 2019. ISBN 978-2-409-02073-5. OCLC : 1127535504

ICONOGRAPHIE

1.1 Prévisualisation cartographique du jeu de données des volumes bâtis. - https ://opendata.paris.fr/ 11

1.2 Prévisualisation du jeu de données des volumes bâtis sous forme de tableau. - https ://opendata.paris.fr/ 12

1.3 Primitives géométriques des données géoréférencées - https://github.com/ClementDelgrange/Cours_programmation_SIG/ 13

1.4 Page descriptive issue des métadonnées - https ://opendata.paris.fr/ 14

1.5 Table descriptive des variables issue des métadonnées - https ://opendata.paris.fr/ 16

1.6 Définition d'une zone d'extraction de données des volumes bâtis - https ://opendata.paris.fr/ . . 19

1.7 Un choix étendu de formats de fichiers - https ://opendata.paris.fr/ 19

1.8 Chargement du jeu de données sous un format tabulaire dans Python - réalisation personnelle . 20

1.9 Chargement du jeu de données sous un format hiérarchisé dans Python - réalisation personnelle 21

1.10 Chargement du jeu de données sous un format géographique dans Python - réalisation personnelle 24

1.11 Aperçu des différentes notations caractérisant les porte à faux - Réalisation personnelle 30

2.1 Aperçu des différentes notations caractérisant les porte à faux - Réalisation personnelle 34