

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	5
1 Le script : outil d'exploitation privilégié de l'Open Data (Etude de cas des données des « volumes bâtis » de l'Open Data Paris.)	10
1.1 L'Open Data : entre nomenclature et variables	12
1.1.1 Variables et typologies des valeurs	12
1.1.2 Les métadonnées : clé de compréhension des données	15
1.2 Le langage Python pour extraire et comprendre la structure des données	19
1.2.1 Des données constituées d'objets hiérarchisés	20
1.2.2 L'hétérogénéité : caractéristique intrinsèque aux données ouvertes	22
1.3 Aperçu de la souplesse des fonctions de manipulation de données	25
1.3.1 Approche compréhensive des différentes notations grâce à Python	25
1.3.2 De la chaîne de caractère à la valeur numérique	27
2 Intégrer des données ouvertes au sein du “workflow” de l'architecte	30
2.1 Production de documents synthétiques interactifs	32
2.1.1 Visualisation statistique des données	32
2.1.2 Cartographier de manière interactive	36
2.2 Génération automatique de documents techniques.	39
2.2.1 Fichier CAD vectorisé et hiérarchisé	39

2.2.2	Construction d'un modèle 3D	42
3	Exploiter l'Open Data pour entraîner des modèles de prédiction	48
3.1	Problématique du modèle de prédiction	50
3.1.1	Du contexte à la démarche	50
3.1.2	Approche des objectifs du modèle	52
3.2	L'agrégation pour compiler son propre jeu de données	53
3.2.1	Une multiplicité des sources à exploiter	53
3.2.2	Le recouplement géométrique	61
3.3	Résultats et bilan technique	62
3.3.1	Entraînement du modèle	62
3.3.2	Aperçu des limites actuelles	62
CONCLUSION		63
BIBLIOGRAPHIE		66
ICONOGRAPHIE		69
ANNEXE : UN MÉMOIRE RÉDIGÉ EN R MARKDOWN		71

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'éologie 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.

⁴Etalab - 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>.

⁵Géoservices / 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>.

⁶Portail 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/>.

⁷Portail 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 datavisualisation⁸ 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 Londres¹⁰, 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/>.

¹⁰HÜGEL, Stephan et ROUMPANI, Flora. *CityEngine-Twitter* [logiciel]. [S. I.] : 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 MVRDV¹¹ 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

¹¹MVRDV - NEXT [en ligne]. 8 février 2021. Disponible à l'adresse : <https://www.mvrdv.nl/themes/15/next>.

¹²MVRDV - 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'algorithme 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 tableau) 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**

¹³What 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.

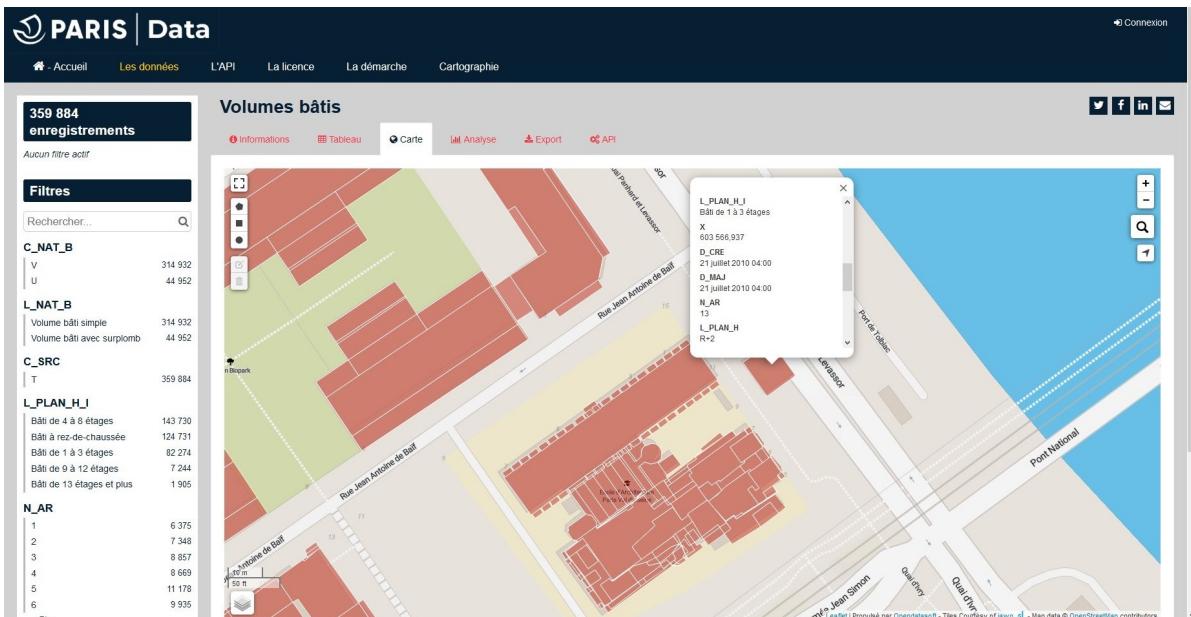


FIG. 1.1 : Prévisualisation cartographique du jeu de données des volumes bâtis.

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.



The screenshot shows a data visualization interface for 'Volumes bâtis'. At the top, there's a navigation bar with links for Accueil, Les données, L'API, La licence, La démarche, and Cartographie. On the right, there are social media sharing icons. The main area has a title '359 884 enregistrements' and a 'Filtres' sidebar with dropdown menus for C_NAT_B (V, U), L_NAT_B (Volume bâti simple, Volume bâti avec surpoids), C_SRC (T), and L_PLAN_H_I (Bâtis de 4 à 6 étages, Bâtis à rez-de-chaussée, Bâtis de 1 à 3 étages, Bâtis de 9 à 12 étages, Bâtis de 13 étages et plus). The main table has columns: geom_x_y, geom, C_NAT_B, L_NAT_B, C_SRC, L_SRC, M2, O, NB_PL, M2_PL_TOT, B_RDC, and C_PLAN_H_I. The table contains 28 rows of data, each representing a building volume record with its coordinates and various metadata fields.

geom_x_y	geom	C_NAT_B	L_NAT_B	C_SRC	L_SRC	M2	O	NB_PL	M2_PL_TOT	B_RDC	C_PLAN_H_I
1	48.8441153014, 2.30902176751					243.623	7	1 705,358	1	3	
2	48.8439010793, 2.3090596381					5.713	1	5 713	1	1	
3	48.8447321372, 2.30917878923					493.851	9	4 444,680	1	3	
4	48.844340087, 2.309944990513					19.152	1	19.152	1	1	
5	48.8441441199, 2.3084211987					11.193	1	11.193	1	1	
6	48.8443922009, 2.31026442405					19.149	1	19.149	1	1	
7	48.8444695843, 2.31016544817					166.183	6	997.099	1	3	
8	48.8445399499, 2.3099661606					5.024	1	5 024	1	1	
9	48.8441884471, 2.31106153093					534.498	3	1 603,495	1	2	
10	48.8446188382, 2.31077307376					115.593	5	577.965	1	3	
11	48.84517583, 2.31220416567					416.677	10	4 166,773	1	4	
12	48.8439741252, 2.31557071754					221.637	3	684.912	1	2	
13	48.8440020041, 2.31746842486					360.801	2	721.803	1	2	
14	48.8441185025, 2.31712251798					5.676	1	5 676	1	1	
15	48.8440531524, 2.31729160523					159.106	7	1 113,741	1	3	
16	48.844619239, 2.31739142764					58.313	1	58.313	1	1	
17	48.8441841819, 2.31941979113					9.928	3	29.784	1	2	
18	48.8439885043, 2.31962764016					27.387	1	27.387	1	1	
19	48.8432216162, 2.31901321041					0.052	4	0.207	1	2	
20	48.8447873109, 2.3195247381					185.586	4	742.342	1	2	
21	48.8445910344, 2.31914706887					24.544	2	49.088	1	2	
22	48.8440542456, 2.31974488625					40.132	3	120.397	1	2	
23	48.844056005, 2.31984191817					55.249	4	220.997	1	2	
24	48.843948622, 2.3198874982					45.653	3	136.958	1	2	
25	48.844041158, 2.32055023956					64.983	1	64.983	1	1	
26	48.8441110387, 2.3144077000					1 840	1	1 840	1	1	

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'occurrence, 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*).

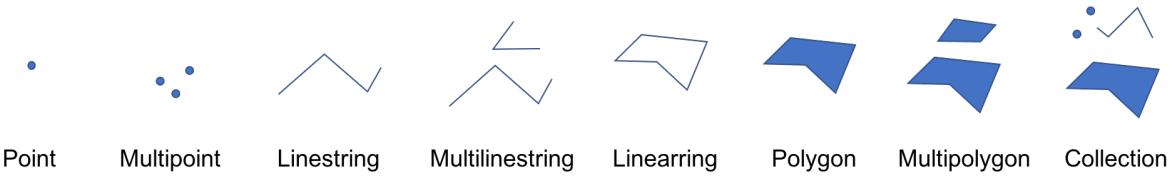


FIG. 1.3 : Primitives géométriques des données géoréférencées

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 dé-

nomination, 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**.

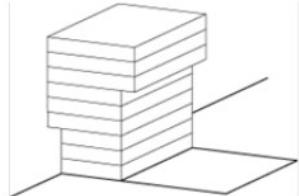
Elles peuvent également contenir des informations complémentaires concernant le fournisseur, la manière dont les données ont été acquises ou encore d'éventuelles limites de précision et recommandations d'utilisation par exemple.

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

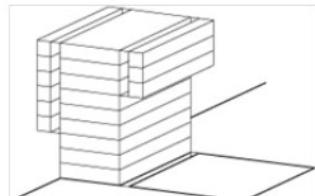
Donnée vecteur qui décrit les bâtiments de manière détaillée en différenciant les bâtis en fonction de leur hauteur et des parties en saillie ou en retrait, définissant ainsi des volumes.



Remarques

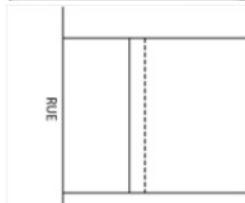
Les volumes bâtis décrivent les bâtiments tels que représentés sur le plan parcellaire raster géré jusqu'en 2015 par le Service de la Topographie et de la Documentation Foncière (STDF).

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

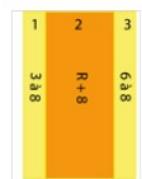
Paris.



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.



1.4 Identification et clés

Nom informatique de l'objet : VOLUME_BATI

Code court de l'objet : VB

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

Clé sémantique : Pas de clé sémantique.

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 disparaît quand le bâtiment est démolie.

1.6 Limite d'usage et remarques

Néant

FIG. 1.4 : Page descriptive issue des métadonnées

En l'occurrence, 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).

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 (m ²)	N		
NB_PL	Nombre de planchers	N		
M2_PL_TOT	Surface totale de planchers (m ²)	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.

FIG. 1.5 : Table descriptive des variables issue des métadonnées

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, **Cn** où n est un entier signifie que les valeurs possibles sont sous la forme d'une chaîne de caractères, contenant n caractères, tandis que **N** désigne simplement des données numériques), ainsi que ses valeurs possibles (servant à distinguer les variables **catégoriques**).

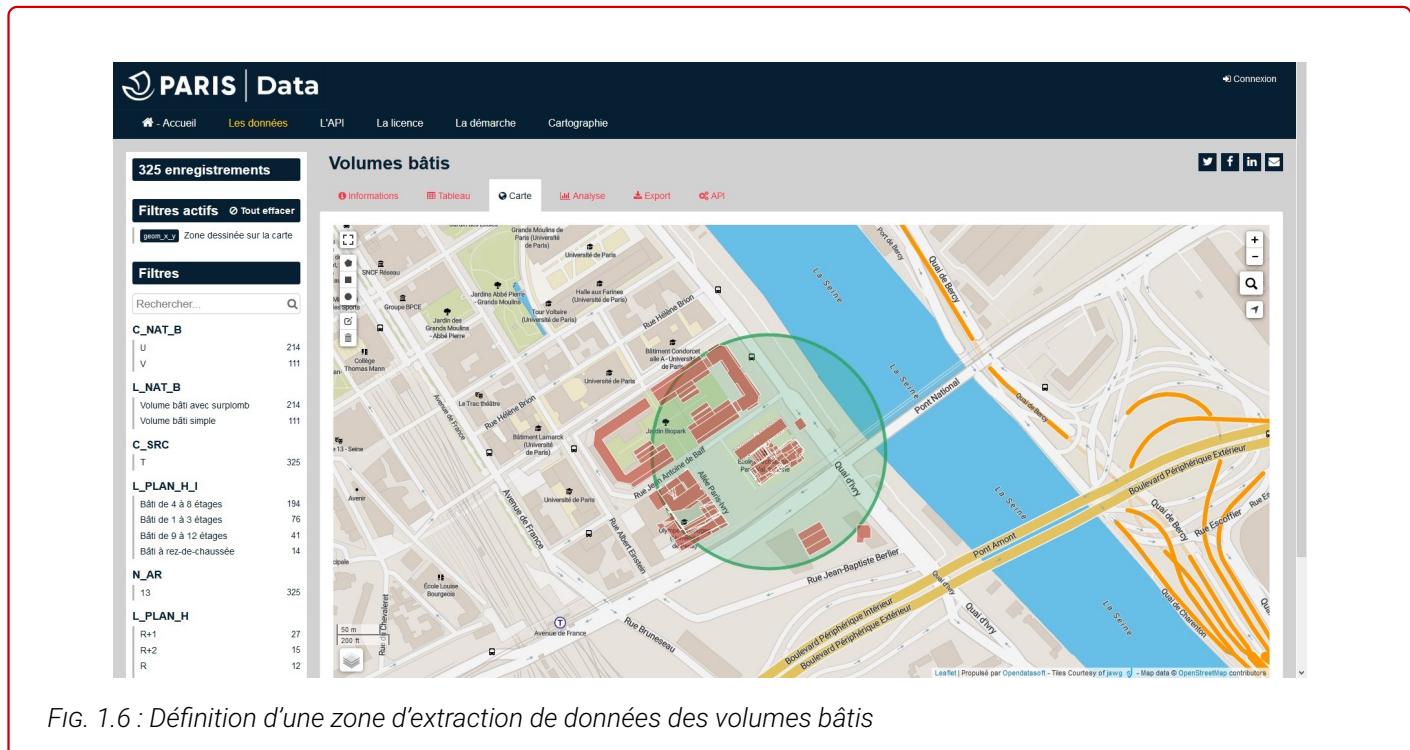
Dès lors, il est possible de repérer les deux variables les plus pertinentes si l'on souhaite extraire la hauteur des différents volumes. En l'occurrence, ce seront les variables **H_ET_MAX** ainsi que **L_B_U** (pour les volumes en porte à faux), toutes deux exprimées en nombre d'étages. La surface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

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'occurrence localisé autour de l'ENSAAPVS).



1.2.1 Des données constituées d'objets hiérarchisés

A l'instar d'autres jeux de données complexes, le jeu des "volumes bâtis". Dès lors, à la lumière de la lecture des métadonnées, le **format de fichier** doit être choisi judicieusement en fonction du contenu du jeu de données. En l'occurrence.

Ainsi, il est plus judicieux de s'orienter vers un format capable de représenter cette hiérarchie dans sa structure.

Le format JSON (JavaScript Object Notation) est un des formats d'échanges hiérarchisés les plus courant (au côté du XML). Ce dernier peut se décomposer en deux éléments primaires : le **dictionnaire**, un ensemble formé de couples **attribut : valeur** (aussi appelés "objets"), et la **liste**, une collection d'objets.¹⁴

Définition d'une zone d'extraction de données des volumes bâtis

```
# exemple de structure d'un script json
js = [ # début de la liste
    { # dictionnaire 1
        "attribut" : "valeur", # couple attribut : valeur
        "attribut2" : [0,1,2,3,4] # une liste peut être une valeur
    },
    { # dictionnaire 2
        "attribut" : { "attribut3" : "valeur3" }, # un dictionnaire contenant lui même des
            # objets peut être une valeur
        "attribut2" : 99
    }
] # fin de la liste
```

Ainsi, ces deux constituantes suffisent à représenter des hiérarchies pouvant être complexes, tout en restant facile à lire et écrire pour les humains.

¹⁴JSON [en ligne]. [s. d.]. [Consulté le 3 février 2021]. Disponible à l'adresse : <https://www.json.org/json-fr.html>.

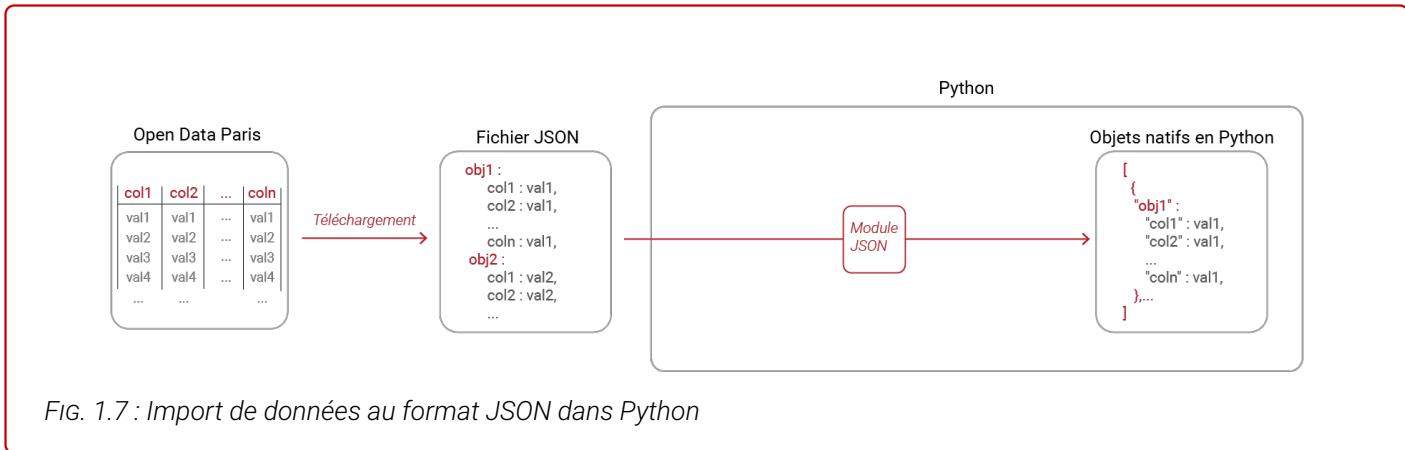


FIG. 1.7 : Import de données au format JSON dans Python

Or, la liste et le dictionnaire sont tous deux très courants au sein des langages de programmation modernes. Ainsi, l'import au sein de Python est extrêmement limpide. La syntaxe afin d'accéder à une valeur donnée s'effectue simplement en saisissant l'**index de l'objet souhaité** pour les listes et le **nom des attributs** en partant de la base de l'arborescence (de gauche à droite) pour les dictionnaires.

Import de données au format JSON dans Python

```
# 1er objet de la liste (dictionnaire 1), puis "attribut2"
print(js[0]["attribut2"])

## [0, 1, 2, 3, 4]
```

Import de données au format JSON dans Python

```
# 2e objet de la liste (dictionnaire 1), puis "attribut2" (dictionnaire 3), puis "attribut3"
print(js[1]["attribut"]["attribut3"])

## valeur3
```

La méthodologie est identique lors de l'import d'un fichier JSON externe, qui sera en l'occurrence le même jeu de données téléchargé depuis le site Open Data Paris, mais au format .JSON.

```
import json
data = json.load(open("DONNEES/volumesbatisparis.json", "r"))
# premier objet de la liste de volumes
print(data[1])

## {'datasetid' : 'volumesbatisparis', 'recordid' : 'c30d8245b2d55a552f4c03a4785bfa19c9d47c73
    , 'fields' : {'objectid' : 536130, 'n_sq_pf' : 750031416, 'd_maj' : '2010-07-21T04
    :00 :00+02 :00', 'l_src' : 'Fiche parcellaire et terrain certifié', 'l_b_u' : '5a8', 'h_et_max' : 8.0, 'b_rdc' : 0.0, 'n_ar' : 13, 'm2_pl_tot' : 9.8429349, 'd_cre' : '2010-07-21
    T04 :00 :00+02 :00', 'm2' : 2.4607337, 'l_plan_h_i' : 'Bâti de 4 à 8 étages', 'n_sq_qu' :
    750000050, 'geom_x_y' : [48.8272100647, 2.38474285824], 'l_nat_b' : 'Volume bâti avec
    surplomb', 'n_qu' : 50.0, 'c_plan_h_i' : 3.0, 'shape_area' : 0.0, 'c_src' : 'T', 'nb_pl' :
    4.0, 'n_sq_vb' : 750169701, 'shape_len' : 0.0, 'c_nat_b' : 'U', 'geom' : {'type' : 'Polygon',
    'coordinates' : [[[2.384753219690729, 48.82722001182662], [2.384761632106771,
    48.82721264787777], [2.384739182508318, 48.827203171721116], [2.3847246454617173,
    48.82720248715756], [2.384721921583155, 48.82720510470582], [2.384753219690729,
    48.82722001182662]]]}, 'n_sq_ar' : 750000013, 'y' : 125193.0588305, 'x' : 603540.4980132},
    'geometry' : {'type' : 'Point', 'coordinates' : [2.38474285824, 48.8272100647]}, 'record_timestamp' : '2020-12-24T11 :41 :01.963+01 :00'}
```

1.2.2 L'hétérogénéité : caractéristique intrinsèque aux données ouvertes

Cette lecture d'un des volumes du jeu de données au format JSON permet de repérer ses différents niveaux hiérarchiques, tout en observant les informations supplémentaires dont on dispose alors.

Tout d'abord, les variables du jeu de données d'origine sont ici présentes sous l'attribut "**fields**", et sera à référencer lors de l'extraction. En son sein, la variable **geom** est ici complètement représentée, contenant elle-même un dictionnaire avec ses deux valeurs (type et coordonnées). Nous pouvons également noter que les coordonnées sont constituées d'une **liste contenant des listes à deux valeurs**, représentant tout simplement une collection de points (définis par x et y). Cependant, elle est elle-même contenue dans une liste englobante superflue, que nous allons supprimer par la suite.

Ainsi, il est possible d'extraire les variables initialement souhaitées pour chacun des volumes avec une **boucle**, qui itérera à travers la liste des volumes. Les résultats obtenus seront ajoutés dans un nouveau dictionnaire (avec des attributs nommés plus explicitement), lui-même ajouté à une liste vide.

Import de données au format JSON dans Python

```
liste = [] # liste vide
for volume in data :
    liste.append({
        # Le [0] à la fin permet de supprimer la liste englobante superflue
        "coords" : volume["fields"]["geom"]["coordinates"][0],
        "surface" : volume["fields"]["m2_pl_tot"],
        "hauteur" : volume["fields"]["h_et_max"],
        "hauteur_paf" : volume["fields"]["l_b_u"]
    })
## Error in py_call_impl(callable, dots$args, dots$keywords) : KeyError : 'l_b_u'
##
## Detailed traceback :
##   File "<string>", line 7, in <module>
```

Cependant, nous obtenons l'erreur *KeyError : "l_b_u"*, signifiant que l'attribut "l_b_u" n'est pas défini pour certains objets. En effet, là où dans un tableur, un attribut non-défini se traduit par une valeur nulle ou une case vide, les structures hiérarchisées comme le JSON permettent à chaque objet de posséder certains attributs qui lui sont propres. Ainsi, les volumes n'étant pas en porte à faux ne possèdent pas cet attribut.

D'un point de vue plus général, cette hétérogénéité est très courante dans les jeux de données ouvertes. Au-delà d'être causée quand **certaines enregistrements** possèdent **des attributs qui leurs sont propres** sans toutefois justifier la création d'un jeu de données supplémentaire (comme les volumes en porte à faux ici, difficilement dissociables des autres), elle peut être tout simplement provoquée par **certaines valeurs manquantes** dans différents enregistrements (dans le cas d'une enquête aussi complexe et étendue que celle des hauteurs des bâtiments de Paris, cela est compréhensible). Enfin, comme c'est le cas ici, cette hétérogénéité se révèle pendant l'extraction, car pas toujours mentionnée dans les métadonnées.

En conséquences, les fonctions de gestion d'erreur **try** et **except** seront utilisées afin de détecter l'absence de l'attribut "**l_b_u**".

Import de données au format JSON dans Python

```
liste = [] # liste vide
for volume in data :
    try :
        liste.append({
            "coords" : volume["fields"]["geom"]["coordinates"][0],
            "surface" : volume["fields"]["m2_pl_tot"],
            "hauteur" : volume["fields"]["h_et_max"],
            "hauteur_paf" : volume["fields"]["l_b_u"]
        })
    except KeyError :
        liste.append({
            "coords" : volume["fields"]["geom"]["coordinates"][0],
            "surface" : volume["fields"]["m2_pl_tot"],
            "hauteur" : volume["fields"]["h_et_max"]
        })
print(liste[1])

## {'coords' : [[2.384753219690729, 48.82722001182662], [2.384761632106771,
48.82721264787777], [2.384739182508318, 48.827203171721116], [2.3847246454617173,
48.82720248715756], [2.384721921583155, 48.82720510470582], [2.384753219690729,
48.82722001182662]], 'surface' : 9.8429349, 'hauteur' : 8.0, 'hauteur_paf' : '5a8'}
```



```
print(liste[7])

## {'coords' : [[2.38351222440612, 48.826747420640146], [2.383527749099538,
48.8267326493293], [2.383165045737739, 48.82659869275863], [2.383149529111281,
48.82661346765992], [2.38351222440612, 48.826747420640146]], 'surface' : 121.4416013, 'hauteur' : 1.0}
```

Ainsi, le langage Python a permis de mettre en lumière à la fois l'importance de la hiérarchisation dans un jeu de données complexe, dont une certaine hétérogénéité persiste (sans être forcément repérable en amont de la manipulation) réclamant une souplesse de la part des outils d'extraction à cet égard.

1.3 Aperçu de la souplesse des fonctions de manipulation de données

Après la phase d'extraction précédente, il est nécessaire que les données extraites en l'état soient exploitables pour la suite. En l'occurrence, il serait souhaitable d'exprimer les hauteurs en **mètres** plutôt qu'en nombre de niveaux.

Import de données au format JSON dans Python

```
h_etage = 3
print(liste[0][ "hauteur" ] * 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 d'une chaîne de caractères (texte décrivant les plages de hauteur), il est nécessaire de convertir cette notation numériquement.

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.

Import de données au format JSON 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 **longueur**.

Import de données au format JSON dans Python

```
notations = {}
for volume in liste :
    try :
        notations[len(volume["hauteur_paf"])] = volume["hauteur_paf"]
    except KeyError :
        pass

print(notations.values())

## dict_values(['2a4', '2a4_et_7a8', 'encorbt_au_5', 'Ra1_et_encorbt_au_5', 'R_et_3a9', 'Auvent_n1', '4a11', 'R_et_encorbt_au_3', '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 **N⁻¹** au niveau maximal.
- **auvent** désigne une plage du niveau **N** à **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**.

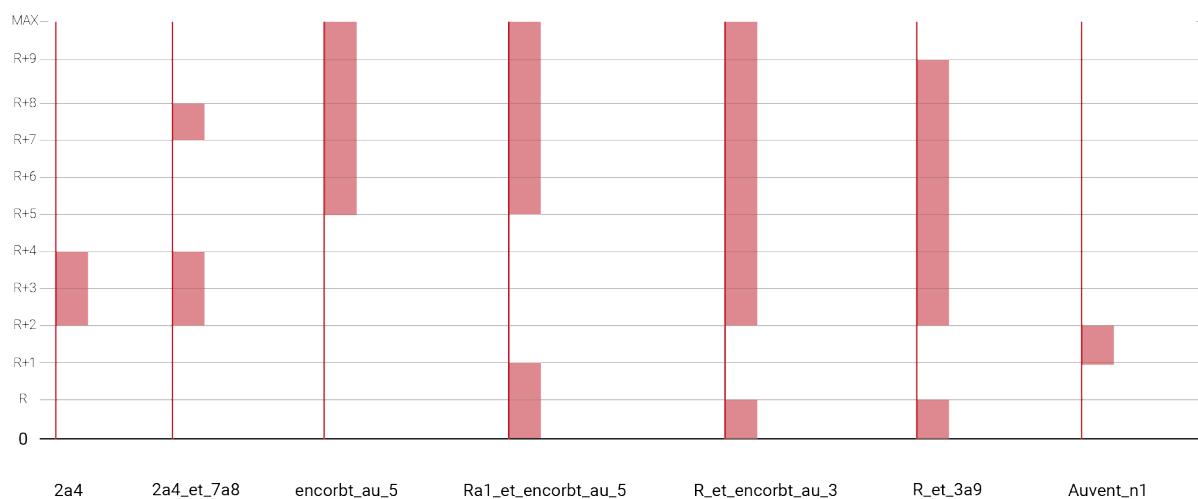


FIG. 1.8 : Aperçu des différentes notations caractérisant les porte à faux

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 **N¹aN²**. 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 portes à faux

```
# Rappel : h_etage = 3 mètres
h_paf = "Ra8"
if "a" in h_paf :
    hauteur = h_paf.split("a")
    hauteur = [(int(h)+1)*h_etage if h != "R" else 0 for h in hauteur]

print(hauteur)

## [0, 27]
```

Suivant ce principe, le bloc de code suivant opère ce type de transformation suivant les différents notations relevées précédemment. L'incrément de **1** sera également appliqué à l'attribut **hauteur**

Aperçu des différentes notations caractérisant les porte à faux

```
# Rappel : h_etage = 3 mètres
for volume in liste:
    intervalles = []
    try :
        intervs = volume["hauteur_paf"].split("_et_")
        for interv in intervs :
            if "encorbt_au_" in interv :
                n = int(interv.strip("encorbt_au_"))
                if n == volume["hauteur"] :
                    intervalles.append([n*h_etage,(n+1)*h_etage])
                else :
                    intervalles.append([(n+1)*h_etage,(volume["hauteur"]+1)*h_etage])
            elif "a" in interv :
                if "R" in interv :
                    intervalles.append([0,(int(interv.strip("Ra"))+1)*h_etage])
                else :
                    intervalles.append([(int(h)+1)*h_etage for h in interv.split("a")])
            elif interv == "R" :
                intervalles.append([0,h_etage])
        volume["hauteur_paf"] = intervalles
    except KeyError :
        pass

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

print(liste[1])

## {'coords' : [[2.384753219690729, 48.82722001182662], [2.384761632106771,
48.82721264787777], [2.384739182508318, 48.827203171721116], [2.3847246454617173,
48.82720248715756], [2.384721921583155, 48.82720510470582], [2.384753219690729,
48.82722001182662]], 'surface' : 9.8429349, 'hauteur' : 27.0, 'hauteur_paf' : [[18, 27]]}
```

Enfin, ce jeu de données apurées peut être sauvegardé au format JSON, afin de pouvoir le réutiliser facilement dans le chapitre suivant.

Aperçu des différentes notations caractérisant les porte à faux

```
json.dump(liste,open("DONNEES/liste_apuree.json","w"))
```

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 sans réelle difficulté aux éventuelles subtilités présentes dans des jeux de données hétérogènes, le tout bénéficiant d'une syntaxe claire tout au long du code. **A 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 Intégrer des données ouvertes au sein du ``workflow'' de l'architecte

La production de documents synthétiques, en particulier les éléments graphiques faisant partie intégrante du « workflow » de l'architecte, il est primordial de s'y intéresser au sein de ce mémoire.

En effet, c'est via ce type de document que l'architecte est capable de non seulement communiquer sa production ou encore sa démarche de conception, mais également de se documenter au cours de sa démarche.

Or, il s'avère que le langage Python regorge de bibliothèques spécialisées dans la datavisualisation tel que *Matplotlib*¹⁵ ou encore *Seaborn*,¹⁶ comme le montre la plateforme *The Python Graph Gallery*¹⁷ offrant énormément de possibilités de représentation, du graphique statistique au modèle 3D.

Ce chapitre présentera ainsi plusieurs variantes d'exploitation du script obtenu à la fin du chapitre précédent dans le but d'obtenir des documents synthétiques à partir des données des bâtiments obtenues. Elles seront respectivement consacrées à l'élaboration de graphiques statistiques, puis d'une carte interactive, d'un dessin vectorisé avec gestion des calques, puis aboutir à un modèle 3D du contexte bâti.

¹⁵ *Matplotlib : Python plotting* [en ligne]. [s. d.]. [Consulté le 16 janvier 2021]. Disponible à l'adresse : <https://matplotlib.org/>.

¹⁶ *Seaborn : statistical data visualization* [en ligne]. [s. d.]. [Consulté le 28 janvier 2021]. Disponible à l'adresse : <https://seaborn.pydata.org/>.

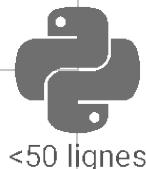
¹⁷ *The Python Graph Gallery : Visualizing data with Python* [en ligne]. [s. d.]. [Consulté le 28 janvier 2021]. Disponible à l'adresse : <https://python-graph-gallery.com/>.

```

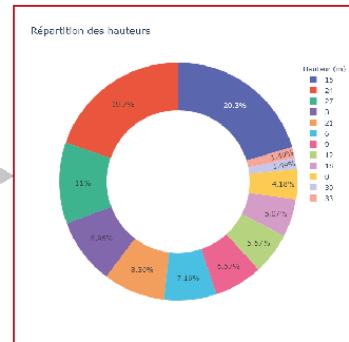
[{"h": 9.0, "interv": [15, 24], "nb": 10, "nb_percent": 10.0, "order": 1, "type": "h"}, {"h": 18.0, "interv": [15, 24], "nb": 10, "nb_percent": 10.0, "order": 2, "type": "h"}, {"h": 27.0, "interv": [15, 24], "nb": 10, "nb_percent": 10.0, "order": 3, "type": "h"}, {"h": 36.0, "interv": [15, 24], "nb": 10, "nb_percent": 10.0, "order": 4, "type": "h"}, {"h": 45.0, "interv": [15, 24], "nb": 10, "nb_percent": 10.0, "order": 5, "type": "h"}, {"h": 54.0, "interv": [15, 24], "nb": 10, "nb_percent": 10.0, "order": 6, "type": "h"}, {"h": 63.0, "interv": [15, 24], "nb": 10, "nb_percent": 10.0, "order": 7, "type": "h"}, {"h": 72.0, "interv": [15, 24], "nb": 10, "nb_percent": 10.0, "order": 8, "type": "h"}, {"h": 81.0, "interv": [15, 24], "nb": 10, "nb_percent": 10.0, "order": 9, "type": "h"}, {"h": 90.0, "interv": [15, 24], "nb": 10, "nb_percent": 10.0, "order": 10, "type": "h"}, {"h": 9.0, "interv": [0, 15], "nb": 10, "nb_percent": 10.0, "order": 11, "type": "h"}, {"h": 18.0, "interv": [0, 15], "nb": 10, "nb_percent": 10.0, "order": 12, "type": "h"}, {"h": 27.0, "interv": [0, 15], "nb": 10, "nb_percent": 10.0, "order": 13, "type": "h"}, {"h": 36.0, "interv": [0, 15], "nb": 10, "nb_percent": 10.0, "order": 14, "type": "h"}, {"h": 45.0, "interv": [0, 15], "nb": 10, "nb_percent": 10.0, "order": 15, "type": "h"}, {"h": 54.0, "interv": [0, 15], "nb": 10, "nb_percent": 10.0, "order": 16, "type": "h"}, {"h": 63.0, "interv": [0, 15], "nb": 10, "nb_percent": 10.0, "order": 17, "type": "h"}, {"h": 72.0, "interv": [0, 15], "nb": 10, "nb_percent": 10.0, "order": 18, "type": "h"}, {"h": 81.0, "interv": [0, 15], "nb": 10, "nb_percent": 10.0, "order": 19, "type": "h"}, {"h": 90.0, "interv": [0, 15], "nb": 10, "nb_percent": 10.0, "order": 20, "type": "h"}]

```

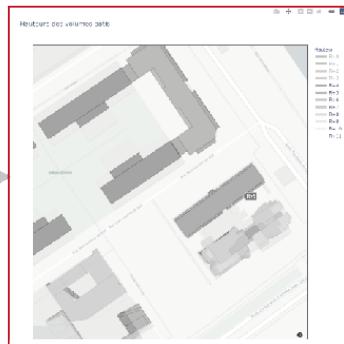
Données épurées brutes



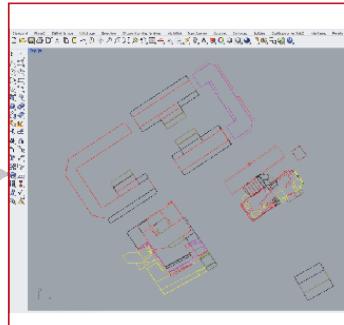
<50 lignes



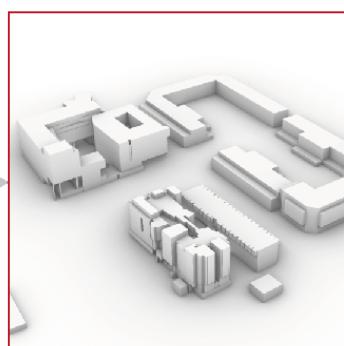
Graphiques p.32



Carte interactive p.32



Dessin vectoriel p.32



Modèle 3D p.32

FIG. 2.1 : Moyens d'export et de synthèse des données récoltées

2.1 Production de documents synthétiques interactifs

Cette section présentera deux méthodes afin de visualiser les données extraites. Au sein de cette section, la bibliothèque *Plotly* sera employée, possédant davantage d'options graphiques ainsi que de possibilités d'export (dont des options d'interactivité) que son concurrent plus répandu *Matplotlib* mentionné précédemment.

2.1.1 Visualisation statistique des données

Lorsque l'on aborde la question de la synthèse de données quelles qu'elles soient, la représentation statistique par des graphiques semble représenter l'approche la plus intuitive et la plus directe.

Moyens d'export et de synthèse des données récoltées

```
import json
import plotly.graph_objects as go
volumes = json.load(open("DONNEES/liste_apuree.json", "r"))
```

Le premier graphique créé sera un histogramme de **répartition du nombre de volumes par hauteur**. Le code ci-dessus permet d'importer la bibliothèque *Plotly* (par l'intermédiaire d'un de ses « sous-module » nommé «graph_objects», que nous appellerons ici avec «go» tout au long du script), et charge également la liste des volumes enregistrée à la fin du chapitre 1.

Moyens d'export et de synthèse des données récoltées

```
n_par_hauteur = {}
for volume in volumes :
    try :
        n_par_hauteur[volume["hauteur"]] += 1
    except KeyError :
        n_par_hauteur[volume["hauteur"]] = 1

print(n_par_hauteur)

## {12.0 : 22, 27.0 : 66, 30.0 : 37, 21.0 : 17, 6.0 : 30, 24.0 : 28, 15.0 : 19, 18.0 : 68, 9.0 : 24,
3.0 : 14, 33.0 : 5, 36.0 : 5}
```

Cette liste des volumes est ensuite analysée dans le code ci-dessus à travers une boucle. Le but est en effet d'en récupérer le **nombre de volumes par hauteur**. La fonction **try/except** permet d'inclure une nouvelle hauteur dans le dictionnaire **n_par_hauteur** si elle n'existe pas encore. Si elle existe déjà, sa valeur est incrémentée.

Enfin, quelques lignes de codes permettent à la fois la construction d'un graphique, ainsi que son export. La **liste des différentes hauteurs** (soit celle des *attributs* du dictionnaire **n_par_hauteur**) représentera l'axe x, tandis que les différents **nombres de volumes** associés formeront les valeurs à renseigner pour l'axe y. Quelques paramètres graphiques servent à définir un titre général, des libellés pour les deux axes ainsi qu'une résolution d'export. Ici, deux exports possibles seront montrés, à savoir un export **statique** sous forme d'image au format PNG, ainsi qu'un export **interactif** au sein d'une page web au format HTML.

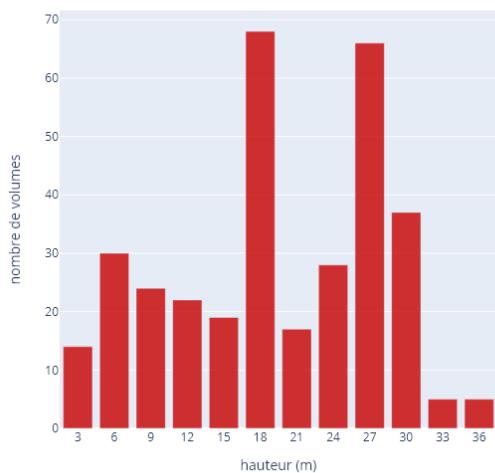
Moyens d'export et de synthèse des données récoltées

```
# traçage du graphique
fig = go.Figure([go.Bar(x=list(n_par_hauteur.keys()), y=list(n_par_hauteur.values()), opacity=0.8, marker_color='rgb(200,0,0)')])
fig.update_xaxes(categoryorder='category ascending', tickvals=sorted(list(n_par_hauteur.keys())))

fig.update_layout(title="Répartition des hauteurs", xaxis_title="hauteur (m)", yaxis_title="nombre de volumes", width=600, height=600)

fig.write_image("OUTPUT/graphique_hauteurs.png")
fig.write_html("OUTPUT/graphique_hauteurs.html")
```

Répartition des hauteurs



Répartition des hauteurs

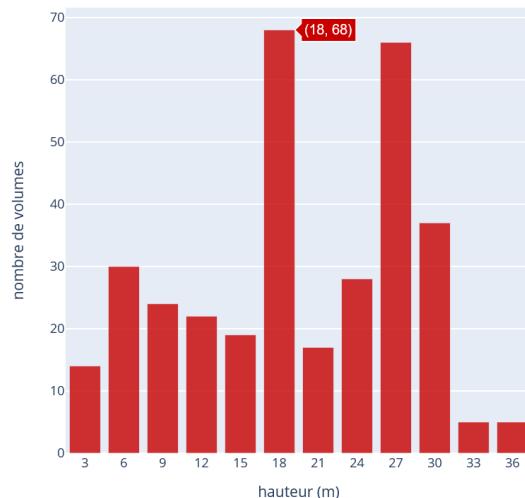


FIG. 2.2 : Nombre de volumes par hauteur

Il est également possible de créer de la même manière une multitude d'autres graphiques que *Plotly* permet de construire. Un second graphique plus complet peut être construit, en s'intéressant cette fois-ci à une répartition **des volumes suivant leur hauteur, leur surface et s'ils sont en porte à faux**. Ici, le sous-module express de *Plotly* sera employé avec *Pandas*, permettant une mise en forme plus condensée et une meilleur gestion des données. Il sera possible de réutiliser directement les éléments de la liste des volumes. Cependant, un attribut devra être ajouté à chaque volume, spécifiant s'il est en porte à faux ou pas, afin de pouvoir être traité dans *Plotly*.

Nombre de volumes par hauteur

```
import plotly.express as px
import pandas

for volume in volumes :
    try :
        _ = volume[ "hauteur_paf" ]
        volume[ "is_paf" ] = 1
    except KeyError :
        volume[ "is_paf" ] = 0

df = pandas.DataFrame(volumes).drop(columns=[ "coords", "hauteur_paf" ])
fig2 = px.parallel_coordinates(df, color="hauteur", color_continuous_scale=px.colors.
    sequential.amp)
fig2.update_layout(font={"size" :20},width=1800,height=800)

fig2.write_image( "OUTPUT/graphique_repart.png" )
fig2.write_html( "OUTPUT/graphique_repart.html" )
```

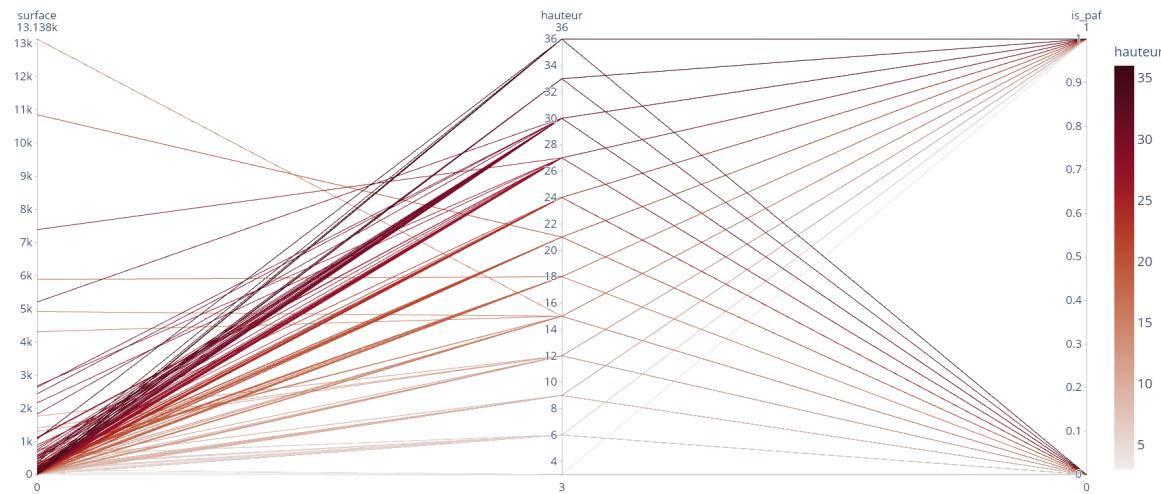


FIG. 2.3 : Répartition des volumes selon leurs caractéristiques

En l'occurrence, le jeu de données extrait contient davantage de petites surfaces ainsi qu'un nombre équilibré de volumes en porte à faux. Cependant, cette souplesse dans la création de représentations graphiques que permet *Plotly* rend également possible de dessiner les **emprises des volumes en eux-mêmes**, qui sera également augmentée avec de l'interactivité.

2.1.2 Cartographier de manière interactive

Cette sous-section a pour but de présenter une fonctionnalité particulièrement utile pour la profession architecturale. En effet, *Plotly* est capable de créer de représenter des **formes géométriques sur un fond de carte** (d'une manière similaire aux solutions de SIG sans avoir besoin de convertir les coordonnées en latitude/longitude), avec laquelle il est possible d'**interagir**, à travers notamment des fonctionnalités telles que le **zoom**, l'**affichage/masquage** d'éléments légendés ainsi que l'affichage de caractéristiques au **survol avec le curseur**.

Ainsi, l'objectif sera ici de générer une **carte interactive des volumes par hauteur**.

Répartition des volumes selon leurs caractéristiques

```
import json
import plotly.graph_objects as go
volumes = json.load(open("DONNEES/liste_apuree.json", "r"))
```

Après avoir chargé le jeu de données de base, la première étape est de **grouper les volumes par hauteur**. Ceci sera effectué comme dans la section précédente, à l'exception que les **volumes ainsi que leurs caractéristiques** seront ajoutés à une liste par hauteur au lieu d'être simplement comptés. Le tout sera trié selon la hauteur par **ordre croissant**.

Répartition des volumes selon leurs caractéristiques

```
volumes_par_hauteur = {}
for volume in volumes :
    try :
        volumes_par_hauteur[volume["hauteur"]].append(volume)
    except KeyError :
        volumes_par_hauteur[volume["hauteur"]] = [volume]

volumes_par_hauteur = dict(sorted(volumes_par_hauteur.items()))
```

Ensuite, une **boucle** permet d'itérer à travers chaque **hauteur et ses volumes** afin de les tracer. Une **couleur** sera préalablement attribuée pour chacune d'entre elles, calculées selon une échelle de gris en RGB (plus le volume est **haut**, plus il sera **clair**). La seule subtilité ici est de devoir séparer **les listes de coordonnées latitude/longitude en deux listes séparées**, et ainsi avoir une liste pour les valeurs de **latitude** et une autre pour

les valeurs de **longitude**, contenant des valeurs **nulles** pour séparer les volumes lors du traçage.

Répartition des volumes selon leurs caractéristiques

```
traces = []
for h,volumes in volumes_par_hauteur.items():
    couleur = "rgb(" + ",".join([str(h/max(volumes_par_hauteur.keys())*255)]*3) + ")"
    X = []
    Y = []
    for vol in volumes:
        for coords in vol["coords"]:
            X.append(coords[0]) # longitude
            Y.append(coords[1]) # latitude
        X.append(None) # Séparations entre chaque volume
        Y.append(None)
    # Traçage des volumes
    traces.append(go.Scattermapbox(
        name=str(h) + "m",
        mode="lines",
        line = {"width" : 0.5, "color" : couleur},
        lon=X,
        lat=Y,
        opacity=1.0,
        hoverinfo="name",
        fill="toself"))
```

Enfin, la carte complète (avec chaque couche de volumes pour chaque hauteur) est créée à partir de la liste `traces`, en configurant le titre, la légende ainsi que le style de fond de carte. Le tout est sauvegardé au format .HTML, permettant de l'ouvrir dans un navigateur afin de permettre l'interactivité et la **navigation libre** dans le fond de carte.

Répartition des volumes selon leurs caractéristiques

```
fig = go.Figure(traces)
fig.update_layout(title="Hauteurs des volumes bâtis", legend_title="Hauteur", autosize=True,
                  mapbox = {'style':'carto-positron', 'center': {'lon': 2.3848515, 'lat': 48.8272092}, "zoom": 16.6})
# Export sous forme d'une page web interactive

fig.write_html("OUTPUT/carte_des_hauteurs.html")
```

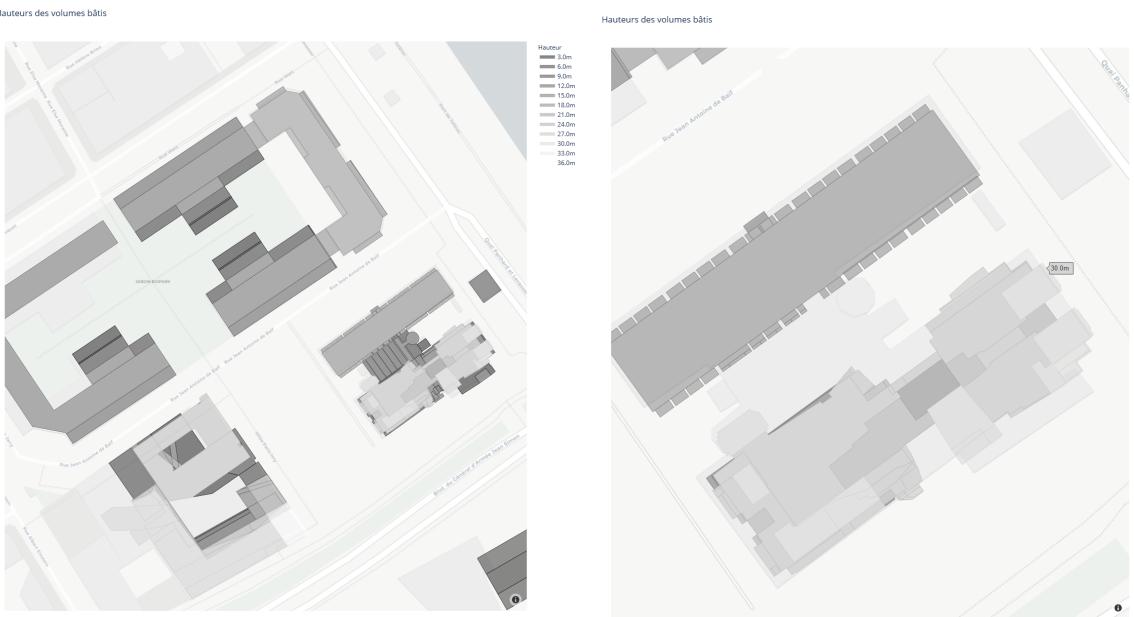


FIG. 2.4 : Carte des volumes par hauteur avec de l'interactivité

Dès lors, une telle capacité à cartographier avec souplesse des données brutes présente un intérêt certain au sein de la pratique architecturale.

Ainsi, ces aperçus attestent de la capacité du langage Python à produire facilement de multiples représentations graphiques synthétiques à partir de données brutes, du graphique statistique aux cartes interactives, renforçant ainsi la pertinence de son utilisation dans le cadre de l'exploitation de données issues de l'Open Data pour les architectes.

2.2 Génération automatique de documents techniques.

Les capacités de synthèse graphique et de cartographie des données montrées précédemment sont certes intéressantes, mais le domaine de la conception architecturale est surtout concerné par la production de dessins, modèles 3D et autres représentations techniques à l'échelle dans le cadre d'un projet. Cette section permettra de répondre à cet enjeu grâce à Python.

2.2.1 Fichier CAD vectorisé et hiérarchisé

Le premier aperçu livré dans cette section, sera de **produire un document vectorisé au format .DXF** (format d'échange de dessin vectorisé similaire au .DWG, répandu en CAO), où seront tracés les **différents volumes sous forme de polylignes**. Chacun d'entre eux sera également classé dans un **calque correspondant à sa hauteur**, avec pour chaque calque une couleur différente. Pour ce faire, le module **ezdxf** sera employé. Ce dernier permet la plupart des fonctions de dessin vectoriel que propose d'autres outils de CAD tels qu'AutoCAD, dont la création de calques entre autres.

Un second module nommé **utm** sera également utilisé pour manipuler les coordonnées géographiques. En effet, il est ici indispensable de convertir les **coordonnées géodésiques** exprimées en degrés de latitude/longitude en **coordonnées cartésiennes** exprimées en unités de grandeur terrestres. La notation UTM (*Universal Transverse Mercator*) étant exprimée en **mètres** et dont le calcul prend en compte la rotundité de la Terre, elle se révèle donc essentielle pour pouvoir dessiner dans un repère. (Une petite subtilité sera d'inverser l'ordre des degrés de latitude/longitude dans le code, afin que l'axe x corresponde à la **longitude**, et l'axe y à la **latitude**).

Carte des volumes par hauteur avec de l'interactivité

```
import json
import ezdxf
import utm
volumes = json.load(open("DONNEES/liste_apuree.json", "r"))
```

Après l'import des modules nécessaires et le chargement du jeu de données de base, un nouveau dessin est initialisé. Ensuite, **un calque par hauteur** sera créé en amont des tracés des volumes, de sorte à ce qu'ils apparaissent dans un ordre croissant au sein du futur fichier. Tout comme dans la carte interactive présentée dans la session 2.1.2, une **teinte de couleur proportionnelle à chaque hauteur** sera créée (en l'occurrence, sur du rouge).

Carte des volumes par hauteur avec de l'interactivité

```
# initialisation d'un nouveau dessin
doc = ezdxf.new(dxfversion='R2010')
msp = doc.modelspace()
# boucle sur une ligne pour récupérer les hauteurs
hauteurs = [volume["hauteur"] for volume in volumes]
# permet de supprimer les doublons et de trier par ordre croissant
hauteurs = sorted(list(set(hauteurs)))
for h in hauteurs:
    calque = doc.layers.new(str(h) + "m")
    calque.rgb = (255*(h/hauteurs[-1]), 0, 0)
```

Ceci effectué, une boucle itérera à travers chaque volume. Pour chacun d'entre eux, leurs **coordonnées seront préalablement converties** des degrés de latitude/longitude en mètres grâce au module *utm*. Une **polyligne** peut ainsi être tracée à partir des points aux coordonnées converties pour représenter l'emprise de chaque volume, puis affectée au calque correspondant à la hauteur du volume. Enfin, le document est exporté au format .DXF.

Carte des volumes par hauteur avec de l'interactivité

```
for volume in volumes:
    coords_cart = []
    # Conversion des coordonnées géodésiques (lat/lon) en cartésiennes (x/y)
    for coords in volume["coords"]:
        c = utm.from_latlon(coords[1], coords[0])
        coords_cart.append((c[0], c[1]))

    calque = str(volume["hauteur"]) + "m"
    msp.add_polyline2d(coords_cart, dxftattribs={'layer': calque})

doc.saveas("OUTPUT/plan_bati.dxf")
```



FIG. 2.5 : Fichier vectoriel organisé des emprises des volumes

Le document fourni en sortie peut ensuite être importé dans n'importe quel logiciel supportant le format DXF, ce qui est le cas pour la majorité des outils de dessin vectoriel des agences d'architecture. Ce document est d'autant plus exploitable qu'il reste **géoréférencé** (les volumes conservent leurs coordonnées UTM dans le dessin), et **à l'échelle** (puisque exprimés en mètres).

De plus, les options de **personnalisation** des calques permettent d'organiser les données dont on dispose en amont d'une exploitation "manuelle".

Ainsi, le langage Python prouve également son efficacité lorsqu'il est question de générer des documents vectoriels graphiques, pleinement exploitables comme support de travail.

2.2.2 Construction d'un modèle 3D

Cette sous-section aura pour but de clore le chapitre en exploitant à son plein potentiel le jeu de donné extrait. En effet, puisque l'on dispose à la fois d'**emprises géométriques traçables** ainsi que de **hauteurs**, construire un **modèle 3D** automatiquement se présente comme un aboutissement certain, d'autant plus qu'il représente tout naturellement une forte utilité pour l'architecte.

Bien que Python possède des bibliothèques permettant de manipuler de la 3D telles que **PyOpenGL**¹⁸ ou **Open3D**,¹⁹ ces dernières sont plutôt orientées pour effectuer des rendu en images de synthèses ou de la reconstruction sur des maillages. Or, ces fonctionnalités sont assez éloignées de l'usage souhaité, où l'on cherche plutôt à **construire un modèle 3D** à partir de données brutes puis les exporter dans des formats possédant des **capacités d'organisation du modèle** (comme des calques par exemple) et en évitant le **maillage**, afin de les rendre compatibles avec l'usage qu'en feraient un architecte. Ce n'est pas le cas ici, les bibliothèques citées privilégiant les formats d'échange maillés comme le format OBJ.

De plus, même si ce retard a tendance à diminuer avec leur évolution, la quasi-totalité des bibliothèques orientées 3D pour Python sont basées sur des versions codées dans des langages de programmation plus complexes comme le C++ pour des raisons de performances, ce qui fait qu'il persiste toujours un "retard" entre le moteur natif et son intégration destinée à Python, comme le présente plus en détail l'ouvrage *Python & OpenGL for Scientific Visualization* (P.ROUGIER,2018).²⁰

C'est pourquoi la bibliothèque **RhinolInside**²¹ sera ici utilisée. C'est un outil Open Source développé sous l'initiative de la société Mcneel, intégré au logiciel de modélisation 3D **Rhinoceros** depuis la version 7. Il a en effet pour but de **connecter Rhinoceros à un autre programme exécuté en parallèle**, afin de mettre en place un moyen d'échange direct de données.

¹⁸PyOpenGL – The Python OpenGL Binding [en ligne]. 24 janvier 2021. Disponible à l'adresse : <http://pyopengl.sourceforge.net/>.

¹⁹ZHOU, Qian-Yi, PARK, Jaesik et KOLTUN, Vladlen. Open3D : A Modern Library for 3D Data Processing. *arXiv:1801.09847*. 2018.

²⁰P.ROUGIER, Nicolas. *Python & OpenGL for Scientific Visualization*. Bordeaux : [s. n.], 2018. Disponible à l'adresse : <https://www.labri.fr/perso/nrougier/python-opengl/>.

²¹Rhino - Rhino. Inside [en ligne]. 25 janvier 2021. Disponible à l'adresse : <https://www.rhino3d.com/features/rhino-inside/>.

La version Python de cette bibliothèque permet en l'occurrence de connecter une instance de Rhinoceros sans interface graphique à un script (mais permettant tout de même toute opération possible manuellement dans le logiciel). Autrement dit, il sera possible directement dans un même script de créer un fichier au format natif de Rhinoceros (.3DM), puis y dessiner et extruder des tracés pour enfin le sauvegarder sur son disque dur.

Fichier vectoriel organisé des emprises des volumes

```
import json
import utm
volumes = json.load(open("DONNEES/liste_apuree.json", "r"))
# chargement d'une instance de RhinoInside
import rhinoinside
from pathlib import Path
rhino_path = Path("C:/Program Files/Rhino 7/System")
rhinocore_path = Path("C:/Program Files/Rhino 7/System/RhinoCore.dll")
rhinoinside.load(str(rhino_path))
import System
import Rhino
```

La séquence d'import est un peu plus conséquente, puisqu'il faut ici spécifier le chemin d'installation de Rhino

7. Le module *utm* sera également utilisé ici.

Fichier vectoriel organisé des emprises des volumes

```
# Création du document
DOC = Rhino.RhinoDoc.Create("")
DOC.ModelUnitSystem = Rhino.UnitSystem.Meters
# Création d'un calque principal
calque_bati = Rhino.DocObjects.Layer()
calque_bati.Color = System.Drawing.Color.FromArgb(255, 0, 0, 0)
calque_bati.Name = 'batiments'
DOC.Layers.Add(calque_bati)
# Définition de ce calque comme actuel

c_actuel = DOC.Layers.FindByFullPath("calque_bati", -1)
DOC.Layers.SetCurrentLayerIndex(c_actuel, False)
```

Une première étape consiste à créer une **nouvelle instance d'un document rhino**, spécifier ses unités (ici, en accord avec les unités des coordonnées UTM, soit le *mètre*), ainsi que créer un calque dans lequel seront dessinés les volumes. Ce dernier se verra attribuer un **nom** et une **couleur**.

Fichier vectoriel organisé des emprises des volumes

```
for volume in volumes :
    pts = System.Collections.Generic.List[Rhino.Geometry.Point3d]()
    # Conversion des coordonnées géodésiques (lat/lon) en cartésiennes (x/y)
    for coords in volume["coords"] :
        c = utm.from_latlon(coords[1],coords[0])
        pts.Add(Rhino.Geometry.Point3d(c[0],c[1],0.0))

    poly = Rhino.Geometry.Polyline(pts)
    try :
        # Si le volume est en porte à faux
        for intervalle in volume['hauteur_paf'] :
            poly.SetAllZ(float(intervalle[0]))
            polyz = poly.ToPolylineCurve()
            h_cible = float(intervalle[1])-float(intervalle[0])
            extr = Rhino.Geometry.Extrusion.Create(polyz,-1*h_cible,True)
            DOC.Objects.AddExtrusion(extr)
    except KeyError :
        # Si le volume repose au rdc
        polyz = poly.ToPolylineCurve()
        extr = Rhino.Geometry.Extrusion.Create(polyz,-1*float(volume["hauteur"]),True)
        DOC.Objects.AddExtrusion(extr)

success = DOC.SaveAs('OUTPUT/modele.3dm')
```

La boucle ci-dessus effectue deux opérations distinctes sur chaque volume.

La première étape consiste à **convertir les coordonnées** en latitude/longitude en coordonnées *utm* afin de pouvoir les exploiter dans le repère de Rhino, puis de les grouper au sein d'une liste (nommée *pts*) afin d'en générer une **polyligne**.

Ensuite, si le volume est en **porte à faux**, les différents **intervalles de hauteur** qu'il occupe dans son emprise en plan (exemple : 6 à 15 mètres) sont récupérés. Pour chacun d'entre eux, la polyligne créée précédemment sera déplacée en hauteur pour atteindre la borne "basse" de l'intervalle. Une **extrusion** est alors créée depuis cette base pour atteindre la hauteur définie par la borne "haute" de l'intervalle, et sera enfin écrite dans le fichier Rhinoceros. En revanche, en cas d'un volume simple reposant au rez-de-chaussée, une seule extrusion sera créée afin d'atteindre la hauteur définie par l'attribut *hauteur*, puis sera écrite dans le fichier.

Enfin, le fichier est sauvegardé au format 3DM, que l'on peut ensuite visualiser et parcourir directement dans Rhinoceros.

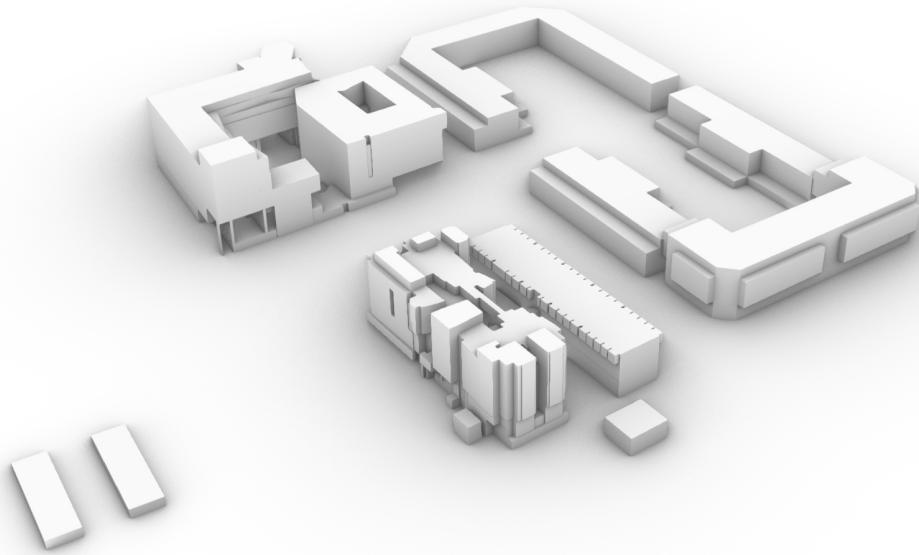


FIG. 2.6 : Modèle 3D généré grâce aux données de hauteur des volumes

Le résultat révèle à la fois une **précision** et un **degré de complexité** certains du jeu de données initial, jusque-là non-visualisables.

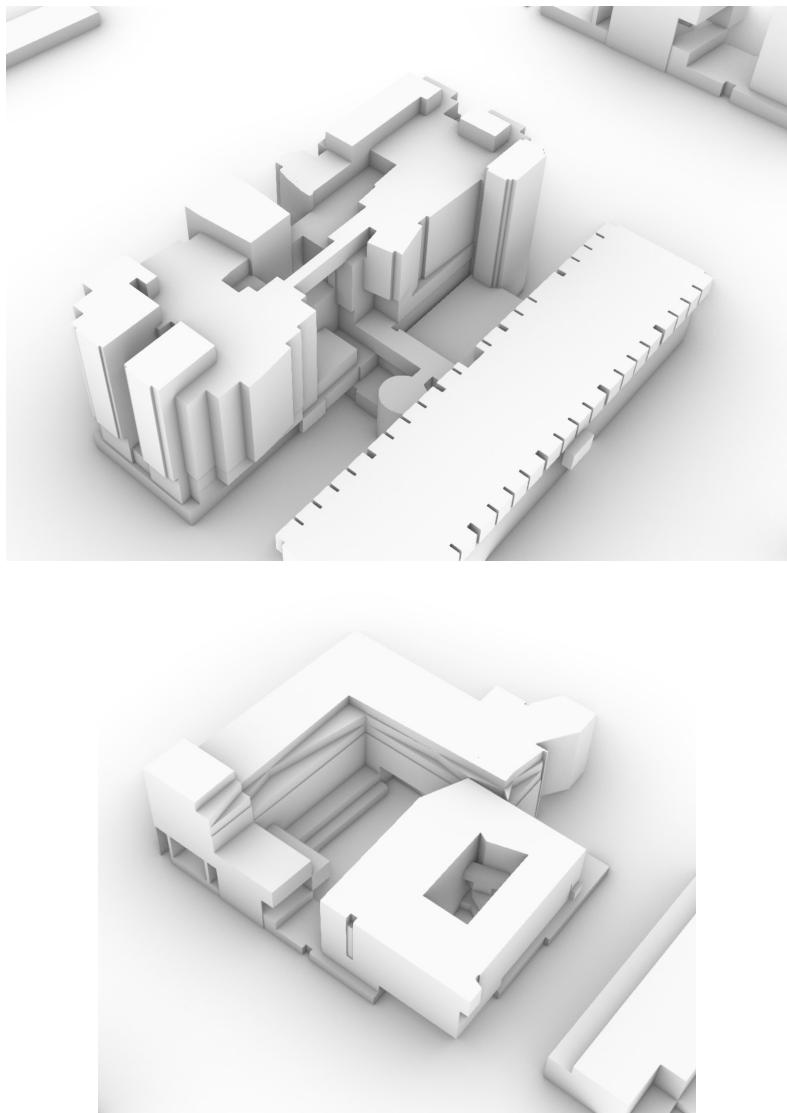


FIG. 2.7 : Une précision et une richesse rendue visible grâce à la modélisation

Bien que la syntaxe de **Rhinoinside** soit plus ardue que celle de la section précédente (dû au fait qu'il est nécessaire de s'appuyer sur de la documentation écrite pour le langage de programmation natif de Rhinoceros, le C#), un architecte possédant une bonne connaissance du logiciel Rhinoceros, et surtout de la manière logique de résoudre un problème de modélisation dans cet outil précis (surtout vis à vis des commandes) peut établir une stratégie viable.

Cette solution a donc permis d'exploiter les données collectées à leur plein potentiel.

Ainsi, au-delà de permettre une simple visualisation statistique des jeux de données en Open Data, le langage Python apporte à l'architecte des solutions techniques variées afin de convertir les jeux de données bruts en fichiers synthétiques dans des formats pleinement exploitables dans son environnement de travail, de la visualisation des données interactives au modèle 3D.

En outre, toutes ces solutions permettent d'obtenir chacun de ces éléments en **un temps d'exécution très faible (quelques minutes d'exécution tout au plus)**, permettant un **gain de temps considérable** une fois qu'ils ont été développés, qui ne fera que **s'accroître** à mesure qu'on le **réutilise**. De plus, les scripts présentés au cours de ce chapitre démontrent de par leur **longueur maîtrisée** (50 lignes maximum) que ce genre de manipulations complexes peuvent être mises en place de manière clarifiée, en prolongement des concepts basiques abordés dans le chapitre 1.

Jusqu'à lors, l'exercice a été de manipuler une infime portion d'un jeu de données ciblé autour d'un site, dans une optique « traditionnelle » de se renseigner sur un contexte immédiat. Le prochain chapitre sera dédié à une approche plus approfondie de l'utilisation de données ouvertes, recélant cependant un intérêt certain pour la conception architecturale.

3 Exploiter l'Open Data pour entraîner des modèles de prédiction

A l'instar du jeu de données étudié au cours des chapitres précédents dont un échantillon spécifique a été prélevé, la **massivité d'un jeu de données** est une caractéristique courante en Open Data. Dès lors, au vu de leur grande taille, certains jeux peuvent être considérés comme suffisamment **représentatifs** vis à vis de ce qu'ils recensent. Autrement dit, c'est une véritable source de connaissances numériques.

Or, le domaine de **l'Intelligence Artificielle**, et tout particulièrement le **Machine Learning** se reposent sur des données disponibles en grande quantité pour pouvoir entraîner des algorithmes. En effet, la précision d'un algorithme étant lié à sa capacité à **généraliser** les connaissances qu'on lui fournit lors d'une session d'entraînement, il convient de disposer de données suffisamment **variées et exhaustives**, tout en étant **en grand nombre** afin de permettre un apprentissage optimal.

Il s'avère également que le langage de programmation Python figure parmi les langages les mieux équipés pour le Machine Learning, disposant de bibliothèques comme **Scikit-Learn**²² proposant une approche simplifiée à travers des algorithmes primaires, jusqu'aux plateformes majeures du secteur comme **TensorFlow**²³ ou **PyTorch**²⁴ majoritairement basées sur ce langage proposant des fonctionnalités plus approfondies.

Un enjeu de taille apparaît alors, celui d'être en capacité d'entraîner un modèle de prédiction exploitable dans la pratique professionnelle de l'architecte basé sur des données accessibles en Open Data

Ce chapitre sera justement centré sur un algorithme que j'ai développé au cours du Semestre 9 dans le cadre de mon Projet de Fin d'Études, **capable d'approcher quantitativement des gisements de matériaux sur**

²²Scikit-learn : machine learning in Python [en ligne]. [s. d.]. [Consulté le 16 janvier 2021]. Disponible à l'adresse : <https://scikit-learn.org/stable/>.

²³TensorFlow [en ligne]. [s. d.]. [Consulté le 20 janvier 2021]. Disponible à l'adresse : <https://www.tensorflow.org/>.

²⁴PyTorch [en ligne]. [s. d.]. [Consulté le 20 janvier 2021]. Disponible à l'adresse : <https://pytorch.org/>.

l'existant, en particulier sur la phase initiale concernant **l'état des lieux des données** ainsi que leur **exploitation avec le langage Python**.

3.1 Problématique du modèle de prédiction

Le développement de cet outil ayant émergé selon un besoin précis dans le cadre d'une intervention architecturale, cette première section aura pour but de présenter le cadre de la démarche ainsi que les données mobilisées.

3.1.1 Du contexte à la démarche

La volonté de mettre en place un algorithme prend ses racines sur une problématique contextuelle liée au site d'intervention ciblé.

Le site en question est celui des anciens entrepôts du BHV (Bazar de l'Hotel de Ville) à Ivry sur Seine. Cet immense terrain laissé à l'état de friche depuis 2013 est situé au sein de la **ZAC Ivry-Confluences**, ancien territoire fortement industrialisé **en renconversion** depuis une dizaine d'années.



FIG. 3.1 : Photographie aérienne de la ZAC Ivry Confluences, contexte du site choisi

Bien que cette reconversion aie pour volonté de conserver un maximum de bâti industriel existant, elle sera malgré tout à l'origine de nombreuses **démolitions**, notamment à cause de la vétusté d'une partie de ce patrimoine et le nouveau tissu urbain induisant une multitude de nouvelles voies de circulation découplant les grands îlots existants.

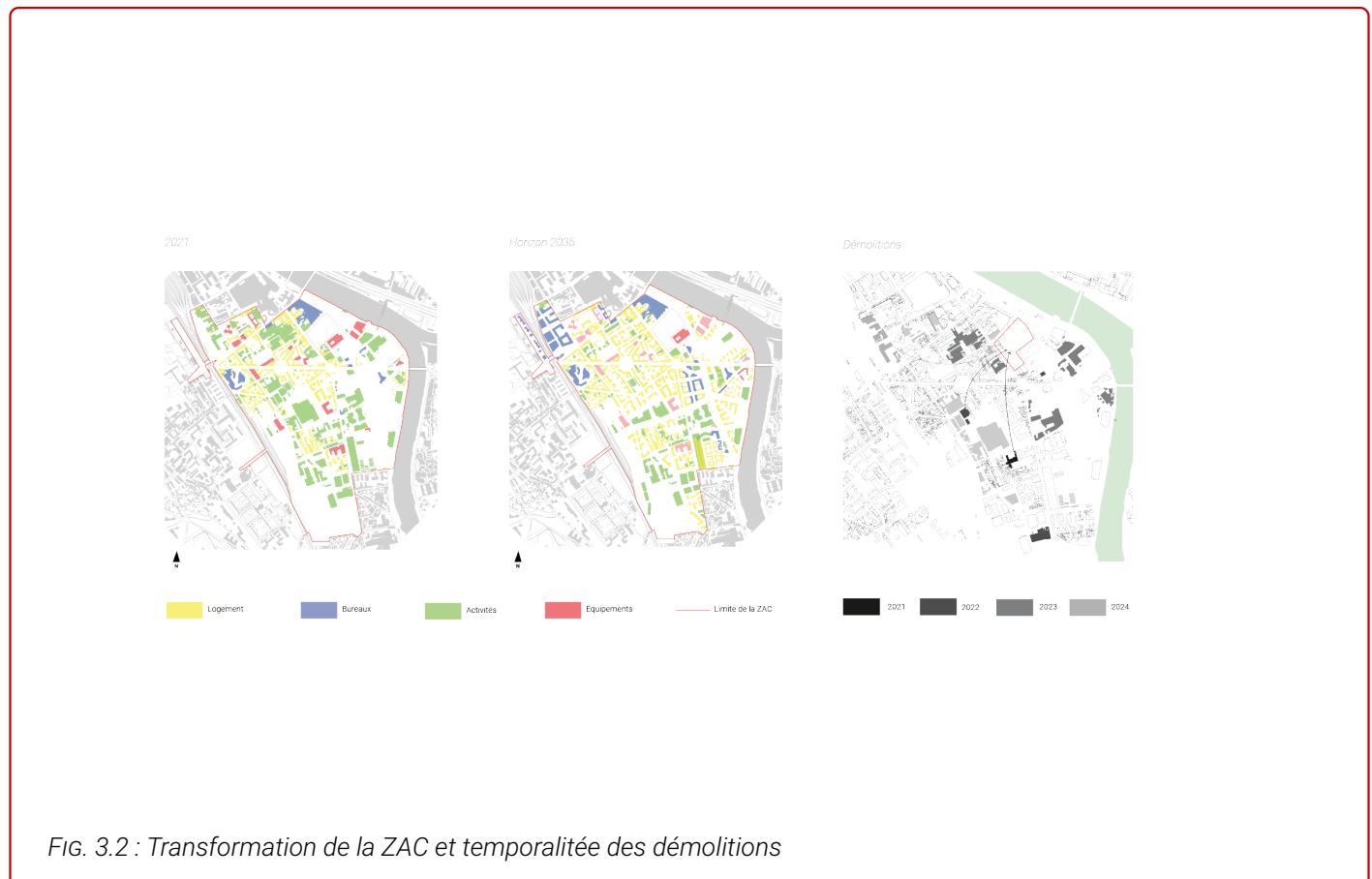


FIG. 3.2 : Transformation de la ZAC et temporalité des démolitions

Dès lors, il existe un véritable enjeu autour de la **valorisation** de ces gisements de matériaux de réemploi issus de la démolition, dont j'ai alors immédiatement envisagé de me saisir comme ressources de construction. Inévitamment, il est nécessaire de les approcher à la fois de manière **qualitative et quantitative** afin de pouvoir

les exploiter pleinement au sein d'un projet.

3.1.2 Approche des objectifs du modèle

Après avoir identifié la **chronologie la plus cohérente** des bâtiments à démolir grâce aux documents rendus disponibles par les collectivités locales, il restait alors deux obstacles de taille :

- * A cause de la quantité importante de bâtiments identifiés, ainsi que l'inaccessibilité de la plupart de ces gisements, l'approche **qualitative** paraît alors complexe à mettre en oeuvre.
- Bien que l'on puisse disposer d'**informations morphologiques basiques** concernant les bâtiments existants (en partie grâce au cadastre vectorisé), il est impossible d'en extraire de quelconques mesures plus précises essentielles à une estimation **quantitative** de matériaux à récupérer, telles que la **surface vitrée** ou encore **l'épaisseur de façade**.

Ainsi, un **algorithme de prédiction** capable de **d'approcher le principal potentiel de matériaux à réemployer** d'un bâtiment grâce à une **approximation de leur nature et leur quantité** se révèlerait ici particulièrement utile, entraîné à partir d'un jeu de données sur des bâtiments existants dans un contexte géographique proche (la région Ile de France par exemple).

Enfin, quant aux données à fournir en entrée de l'algorithme concernant le bâti à démolir, le point de départ sera la **morphologie**. Toute information complémentaire susceptible d'être utile lors de la prédiction (c'est à dire liée directement ou indirectement à la nature ou la quantité de matériaux à réemployer) doit être **facilement obtenable** à cette échelle (idéalement grâce à un jeu de données capable d'être intégré à la représentation 3D).

3.2 L'agrégation pour compiler son propre jeu de données

Dès lors, il paraît compliqué (même impossible) de trouver un jeu de données avec des critères aussi précis que ceux évoqués précédemment. Il est donc nécessaire d'engager une démarche d'**agrégation**, consistant à **regrouper plusieurs jeux** de données entre eux afin de mutualiser leurs informations en un seul et unique jeu.

Cette section sera justement dédiée à l'**énumération** des différentes sources de données accessibles en Open Data qu'il sera nécessaire d'exploiter afin de former le jeu de données souhaité.

3.2.1 Une multiplicité des sources à exploiter

Tout d'abord, des jeux de données **contenant des relevés d'ordre géométrique sur l'existant** suffisamment précis ont dû être identifié. Dès lors, je me suis intéressé de près à la plateforme Open Data de l'**APUR** (Atelier parisien d'urbanisme),²⁵ contenant des jeux de données dans un contexte géographique proche, à savoir le Grand Paris.

J'ai alors identifié le jeu de données intitulé "**BESOIN THEORIQUE CHAUFFAGE ET TYPOLOGIE AU BATI**" comme le plus apte à fournir des informations morphologiques. Bien que l'objectif principal de ce jeu soit de fournir des estimations de besoins énergétiques par bâtiment sur l'existant (conformément à sa dénomination), il inclue également les **données volumiques et surfaciques** utilisées pour le calcul de ces estimations. Comme pour le jeu de données des volumes bâtis de l'Open Data Paris, les **polygones géolocalisés** décrivant l'emprise au sol exprimée en latitude/longitude de chaque bâtiment existant sont inclus, la plateforme de l'APUR permettant une prévisualisation sous forme de carte.

²⁵Atelier Parisien d'Urbanisme [en ligne]. [s. d.]. [Consulté le 4 février 2021]. Disponible à l'adresse : <https://opendata.apur.org/>.

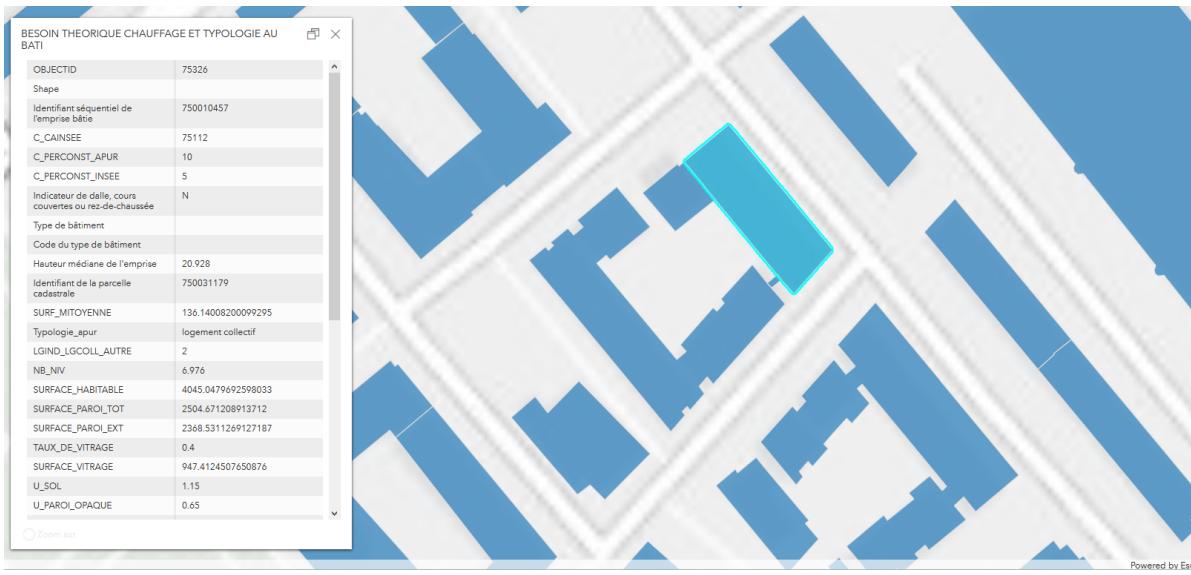


FIG. 3.3 : Prévisualisation des données sous forme de carte

NOM DES VARIABLES	DESCRIPTIF DES VARIABLES	VALEURS POSSIBLES
N_SQ_EB	Identifiant informatique séquentiel de l'entreprise	
N_SQ_EB_ORI	Identifiant informatique séquentiel d'origine de l'entreprise	
C_PERCONST_APUR	Période de construction (dé découpage APUR en 11 classes)	1 : Avant 1800 2 : 1801-1850 3 : 1851-1914 5 : 1915-1939 6 : 1940-1967 7 : 1968-1975 8 : 1976-1981 9 : 1982-1989 10 : 1990-1999 11 : 2000-2007 12 : Après 2008 99 : Non daté
C_PERCONST_INSEE	Période de construction (dé coupe INSEE en 7 classes)	1 : Avant 1949 2 : 1949-1974 3 : 1975-1981 4 : 1982-1989 5 : 1990-1998 6 : 1999-2006 7 : Après 2006 99 : Non daté
B_DALLE	Indicateur de dalle, cours couvertes ou rez-de-chaussée	O (Oui), N (Non)
DUR	Type de bâtiment	Bati dur Bati léger
DUR_CODE	Code du type de bâtiment	01 : Bati dur 02 : Bati léger
H_MEDIANE	Hauteur médiane de l'entreprise bâtie (mètres)	
SURF_MITOYENNE	Surface de mur mitoyenne (m ²)	
Typologie_apur	Nature du bâtiment / occupation	Logement individuel Logement collectif Bâtiment mixte Bâtiment tertiaire ou industriel Non déterminée
LGIND_LGCOLL_AUTRE	Nature du bâtiment simplifiée selon 3 classes	1 : logement individuel 2 : logement collectif / bâtiment mixte 3 : bâtiment tertiaire ou industriel
NB_NIV	Nombre de niveaux/ étages du bâtiment	
SURFACE_HABITABLE	Surface habitable estimée du bâtiment (m ²)	
SURFACE_PAROI_TOT	Surface totale de murs (m ²)	
SURFACE_PAROI_EXT	Surface de murs non mitoyens (m ²)	
TAUX_DE_VITRAGE	Taux de vitrage du bâtiment	
SURFACE_VITRAGE	Surface de vitrage du bâtiment (m ²)	
U_SOL	coefficient de transmission thermique du plancher (W/m ² /K)	
U_PAROI_OPAQUE	coefficient de transmission thermique des murs (W/m ² /K)	

FIG. 3.4 : Des variables fournissant des informations précieuses sur l'existant

Cependant, quelques opérations géométriques sont nécessaires à partir des différentes variables afin d'extraire deux données quantitatives, l'**épaisseur de la façade** ainsi que la **surface de toiture**, pouvant être mises en oeuvre de manière triviale grâce à Python au moment de l'extraction.

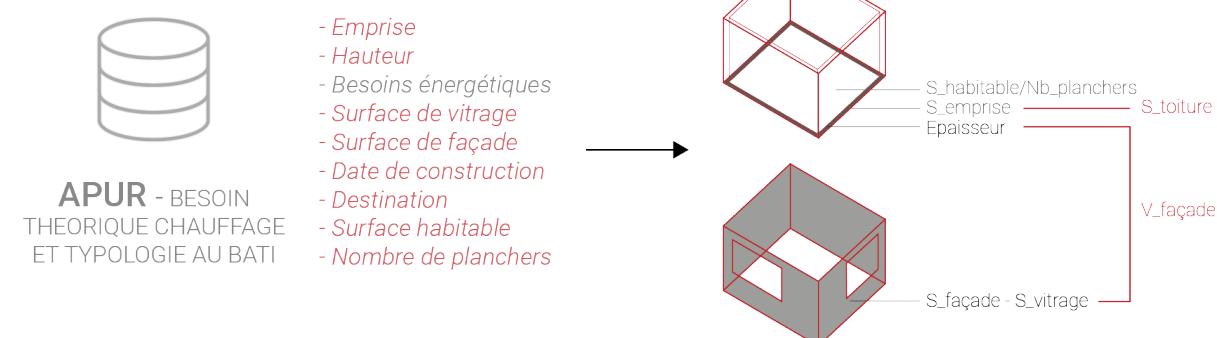


FIG. 3.5 : Opérations géométriques à effectuer à partir des données brutes

Ce jeu de données contenant également des informations sur la **destination** ainsi que l'**époque de construction**, cela pourrait également constituer deux variables à fournir en entrée de l'algorithme, afin d'affiner la prédiction, puisqu'elles auront un lien avec le type de matériau employé entre autres.

Dès lors, un second jeu de données doit être capable de fournir des données concernant la **nature des matériaux** d'un bâtiment, l'idéal étant qu'il provienne également de l'APUR afin de faciliter son recouplement via un identifiant propre à chaque bâtiment, commun aux deux jeux de données. Or, un tel jeu n'existe actuellement pas dans la base de l'APUR.

Cependant, un autre jeu de données contenant des informations sur le bâti existe sur une autre plateforme en Open Data mise en place par l'**IGN**, intitulé la **BD TOPO**. Cette dernière est une base de données vectorielle à l'échelle nationale constituée de plusieurs **couches de données cartographiées et géolocalisées sur de multiples domaines**, tels que les tracés des limites administratives, les infrastructures de transports, l'hydrographie et plus particulièrement **le bâti**. La plateforme d'IGN ne proposant pas de moyens de prévisualisation, les métadonnées seront consultées dans un premier temps.

8.2 BATIMENT

Attributs : Cleabs | Nature | Usage 1 | Usage 2 | Construction légère | Etat de l'objet bati | *Date de création* | *Date de modification* | *Date d'apparition* | *Date de confirmation* | Sources | Identifiants sources | Précision planimétrique | Précision altimétrique | Nombre de logements | Nombre d'étages | Matériaux des murs | Matériaux de la toiture | Hauteur | Altitude minimale sol | Altitude minimale toit | Altitude maximale toit | Altitude maximale sol | Origine du bâtiment | Appariement fichiers fonciers | Géométrie

Les attributs communs à plusieurs thèmes (en italique) ne sont pas décrits dans ce paragraphe (voir 5. Attributs communs à plusieurs thèmes).

Type de géométrie : MultiPolygone 3D

FIG. 3.6 : Extrait des métadonnées : attributs pour chaque bâtiment dans la BD TOPO

Dès lors, ces données contiennent des attributs qui complètent à merveille les données géométriques provenant de l'APUR, à savoir les **matériaux des murs** ainsi que les **matériaux de toiture**.

Matériaux des murs

Format PostgreSQL	Format Shapefile
materiaux_des_murs	MAT_MURS

Type : Texte

Définition : Code indiquant les matériaux des murs du bâtiment, issu des informations contenues dans les fichiers fonciers.

Spécifications de saisie : Code sur 2 caractères issu de la variable **Dmatgm** des fichiers fonciers : Liste des codes.

Valeur particulière : A Mayotte, les bâtiments légers en tôle seront codés avec la valeur 'TL'.

Contrainte sur l'attribut : Valeur non obligatoire.

Matériaux de la toiture

Format PostgreSQL	Format Shapefile
materiaux_de_la_toiture	MAT_TOITS

Type : Texte

Définition : Code indiquant les matériaux de la toiture du bâtiment, issu des informations contenues dans les fichiers fonciers.

Spécifications de saisie : Code sur 2 caractères issu de la variable **Dmatto** des fichiers fonciers : Liste des codes.

Valeur particulière : A Mayotte, les bâtiments légers avec en tôle seront codés avec la valeur 'TL'.

Contrainte sur l'attribut : Valeur non obligatoire.

FIG. 3.7 : Extrait des métadonnées : détail des attributs concernant les matériaux

code	valeur
0	INDETERMINE
1	PIERRE
2	MEULIERE
3	BETON
4	BRIQUES
5	AGGLOMERE
6	BOIS
9	AUTRES

code	valeur
0	INDETERMINE
1	TUILLES
2	ARDOISES
3	ZINC ALUMINIUM
4	BETON
9	AUTRES

FIG. 3.8 : Extraits des tables des matériaux pour les murs (gauche) et les toitures (droite)

La plateforme d'IGN ne proposant pas de moyens de prévisualiser les données sous la forme d'une carte, un petit script Python sera ici utilisé afin de visualiser les données associées à un bâtiment en l'exportant au format JSON pour plus de clarté. Ceci sera réalisé grâce au module **GeoPandas**, spécialisé dans la manipulation de données géographiques, à partir du fichier des bâtiments au format "shapefile" (SHP) contenu dans la base de données une fois téléchargée.

Extraits des tables des matériaux pour les murs (gauche) et les toitures (droite)

```
import geopandas as gpd

data = gpd.read_file("F :\BD_TOPO_IDF\BATI\BATIMENT.shp")

data.head(1).to_file("F :\BD_TOPO_IDF\BATI\BATIMENT.json", driver="GeoJSON")
```

```

{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "ID": "BATIMENT000000245492229",
        "NATURE": "Indifférenciée",
        "USAGE1": "Résidentiel",
        "USAGE2": "Annexe",
        "LEGER": "Non",
        "ETAT": "En service",
        "DATE_CREAT": "2010-11-25 12:44:09",
        "DATE_MAJ": "2019-07-01 10:40:31",
        "DATE_APP": "1920-01-01",
        "DATE_CONF": null,
        "SOURCE": null,
        "ID_SOURCE": null,
        "PREC_PLANI": 3.0,
        "PREC_ALTI": 2.5,
        "NB_LOGTS": 1.0,
        "NB_ETAGES": 2.0,
        "MAT_MURS": "50",
        "MAT_TOITS": "10",
        "HAUTEUR": 4.9,
        "Z_MIN_SOL": 54.0,
        "Z_MIN_TOIT": 58.9,
        "Z_MAX_TOIT": null,
        "Z_MAX_SOL": null,
        "ORIGIN_BAT": "Cadastre",
        "APP_FF": "C 0.6"
      },
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [
              [
                [
                  663956.5,
                  6869331.2,
                  58.9
                ],
                [
                  663956.9,
                  6869331.7,
                  58.9
                ],
                [
                  663963.3,
                  6869325.7,
                  58.9
                ],
                [
                  663962.4,
                  6869324.8,
                  58.9
                ],
                [
                  663958.9,
                  6869320.9,
                  58.9
                ],
                [
                  663952.5,
                  6869326.9,
                  58.9
                ],
                [
                  663956.5,
                  6869331.2,
                  58.9
                ]
              ]
            ]
          ]
        ]
      }
    }
  ]
}

```

FIG. 3.9 : Visualisation d'un bâtiment et de ses données dans la BD TOPO d'IGN

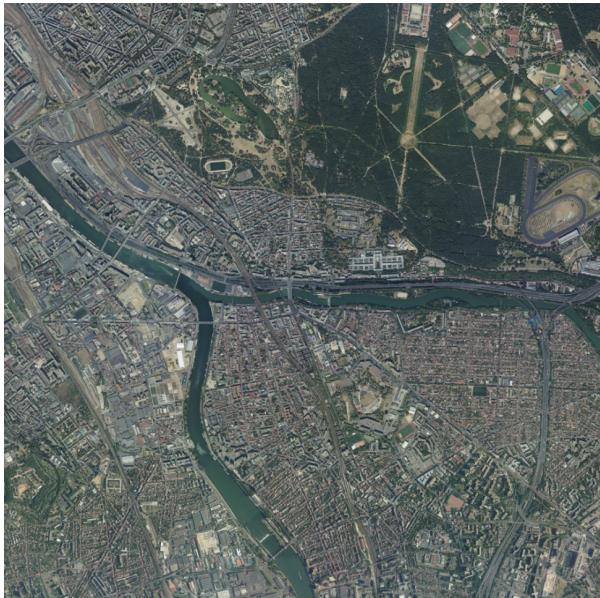
Dès lors, le jeu actuel ne possédant aucun attribut à caractère identifiant en commun avec le jeu de données de l'APUR, les **coordonnées terrestres** de l'emprise représentent **le seul moyen** envisageable de les recouper, de sorte à ce que les attributs concernant les matériaux dans la *BD TOPO* soient ajoutés au jeu de l'APUR. En l'occurrence, les coordonnées ci-dessus sont exprimées selon la notation **UTM**, que l'on peut aisément manipuler grâce à Python comme l'ont montré les chapitres précédents.

Enfin, l'idéal serait de pouvoir fournir des **photographies** de chaque bâtiment à l'algorithme, dans le but d'améliorer grandement sa capacité à identifier les matériaux sur l'existant. Certains critères doivent alors absolument être remplis :

- Ces images doivent être **disponibles pour une grande majorité des bâtiments** et être facilement **accessibles**
- Les photographies doivent être prises du **même point de vue**
- De préférence, chaque bâtiment sujet doit pouvoir être **isolé** (sur un fond blanc par exemple) afin d'améliorer sa détection.

Il semble alors qu'actuellement, le type d'imagerie remplies ces trois critères soit l'**imagerie aérienne**. Ainsi, le jeu de données **BD ORTHO HR** provenant du même organisme (IGN) sera ici exploité. C'est une base de données constituée d'un quadrillage de photographies aériennes en haute définition (de forme carrée de 25000 pixels de côté), dont chaque carreau représente 5 kilomètres de côté. Chacun d'entre eux possède un fichier annexe où l'on retrouve les **coordonnées terrestres aux 4 extrémités de l'image** (en notations *UTM*).

Ceci permettra ainsi d'isoler pour chaque bâtiment (dont on dispose des coordonnées de l'emprise) une **photographie découpée** à partir de ces photographies.



```
!table
!version 300
!charset WindowsLatin1

Definition Table
File "94-2018-0655-6860-LA93-0M20-E080.jp2"
Type "RASTER"
(655000.00,6860000.00) (0,0) Label "Pt 1",
(660000.00,6860000.00) (25000,0) Label "Pt 2",
(660000.00,6855000.00) (25000,25000) Label "Pt 3",
(655000.00,6855000.00) (0,25000) Label "Pt 4"
CoordSys Earth Projection 2003, 33, 7, 3, 46.5, 44,
49.00000000001, 700000, 6600000, -792421,
5278231, 3520778, 9741029
Units "m"
```

FIG. 3.10 : Extrait d'un carreau de la BD ORTHO (gauche) et ses données de géolocalisation (droite)

Ainsi, dans l'objectif de créer un jeu de données regroupant suffisamment d'informations concernant les caractéristiques géométriques des bâtiments existants et la nature de leurs principaux matériaux, il s'avère nécessaire de **recouper plusieurs jeux de données** produits par des **organismes différents** uniquement grâce à leur **géolocalisation**. De plus, il existe une certaine hétérogénéité des **types de données** à manipuler, puisque des **images** viendront compléter des données **alphanumérique** (tel que les dates ou les surfaces).

L'enjeu du langage Python est donc de permettre ce **recouplement géométrique** entre les différents bâtiments.

3.2.2 Le recouplement géométrique

3.3 Résultats et bilan technique

3.3.1 Entraînement du modèle

3.3.2 Aperçu des limites actuelles

CONCLUSION

Grâce à l'approche pratique proposée au sein de ce mémoire, nous avons pu démontrer que le langage de programmation Python constitue un outil souple et performant permettant de d'accompagner l'architecte tout au long de son exploitation des données issues de l'Open Data.

En effet, permettant en premier lieu de rendre compréhensible et manipulable aux plus novices les structures de données couramment rencontrées, en particulier les données hiérarchisées hétérogènes, le langage est doté d'outils de production de documents synthétiques tout à fait capables de convertir les données extraites en formats de fichiers courants pour l'architecte, élément primordial au sein du « workflow » de l'architecte. Ainsi, au-delà des graphiques statistiques, Python permet la génération de cartographies interactives, mais surtout des dessins techniques hiérarchisés et même des modèles 3D de manière automatique et personnalisée, représentant un gain de temps considérable.

Enfin, l'architecte devient capable grâce à Python de mener ses propres analyses de données afin de les mettre à profit dans son activité de conception, grâce à une capacité d'identification de corrélations entre différentes données agrégées, ou encore celle de pouvoir prédire l'impact de son intervention grâce au Machine Learning.

De plus, épaulé par divers modules permettant tous ces usages de manière simplifiée tout au long du processus d'exploitation des données ouvertes, le Python acquiert un véritable caractère universel, autant dans le sens où il est capable de se suffire à lui-même que pour qualifier sa compatibilité extraordinaire avec d'autres outils et services (comme Rhinoceros).

Au-delà du cadre de l'exploitation des données issues de l'Open Data, ce langage représente un véritable pivot afin d'accompagner les agences vers les nouveaux outils plus intelligents.

Premièrement, l'expérience acquise durant la manipulation des jeux de données ouverts (en particulier sur les notions des structures de données) sera extrêmement précieuse lorsqu'il s'agira d'exploiter des jeux de données plus directement liés à la pratique architecturale en elle-même, comme depuis un parc de «Smart Buildings» qu'il faudra surveiller et analyser par exemple.

Au-delà de cet aspect, des solutions logicielles émergentes tel que «Générative Design in Revit» publiée par Autodesk,²⁶ désormais intégrée à Revit 2021 proposent déjà à l'architecte de travailler aux côtés d'algorithmes évolutifs et prédictifs, que ce soit de manière simplifiée par interface graphique ou bien personnalisable grâce au langage de programmation visuelle Dynamo. La collaboration entre des agences d'architecture et des solutions intégrant de l'Intelligence Artificielle commence également à se développer, tel que l'agence *Valode&Pistre* et son partenariat avec «SpaceMaker AI»,²⁷ société norvégienne spécialisée dans la conception urbaine générative.

Ainsi, de nouvelles solutions sont développées chaque année, mais il existera toujours un besoin de **programmer des outils** pour différents types de projet et d'applications au sein de ces derniers, telle que l'insertion urbaine, la conception de l'enveloppe ou encore la question des usages.

Toutes ces opportunités nouvelles requièrent une certaine familiarité avec le fonctionnement algorithmique, et surtout une assimilation de sa manière de fonctionner afin de l'intégrer efficacement à son «workflow»

Dès lors, de par son statut de langage de programmation lui conférant un caractère universel quant aux notions de bases informatiques et surtout algorithmiques (même avec une syntaxe simplifiée), le Python se

²⁶ *Generative Design in Revit now available*. Revit [en ligne]. 8 avril 2020. [Consulté le 17 janvier 2021]. Disponible à l'adresse : <https://blogs.autodesk.com/revit/2020/04/08/generative-design-in-revit/>. Section : What's New.

²⁷ *Valode&Pistre se lance dans l'aventure de l'IA* [en ligne]. 8 février 2021. Disponible à l'adresse : <https://www.cahiers-techniques-batiment.fr/article/valode-pistre-se-lance-dans-l-aventure-de-l-ia.45391>.

présente comme un atout « futur-proof » face à ces nouvelles interactions architecte-machine. De plus, l'adoption massive du Python par les architectes permettrait d'amorcer une solution efficace afin de lutter contre la crainte de la montée en puissance de l'Intelligence Artificielle au sein des métiers du bâtiment, par une atténuation considérable de l'effet « Boite noire » qu'elle suscite. Ceci faciliterait donc grandement la transition du corps architectural dans la numérisation du domaine du bâtiment.

Au-delà de cette appropriation, une certaine motivation de créer des outils logiciels par les architectes pour les architectes intégrant ces nouvelles compétences pourrait émerger, permettant alors de réaffirmer la place du métier d'architecte au sein d'un écosystème de plus en plus technocentré.

BIBLIOGRAPHIE

Atelier Parisien d'Urbanisme [en ligne]. [s. d.]. [Consulté le 4 février 2021]. Disponible à l'adresse : <https://openda.ta.apur.org/>

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/>

Daring Fireball : Markdown Syntax Documentation [en ligne]. [s. d.]. [Consulté le 5 février 2021]. Disponible à l'adresse : <https://daringfireball.net/projects/markdown/syntax>

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>

Generative Design in Revit now available. Revit [en ligne]. 8 avril 2020. [Consulté le 17 janvier 2021]. Disponible à l'adresse : <https://blogs.autodesk.com/revit/2020/04/08/generative-design-in-revit/>. Section : What's New 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>

JSON [en ligne]. [s. d.]. [Consulté le 3 février 2021]. Disponible à l'adresse : <https://www.json.org/json-fr.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>

Matplotlib : Python plotting [en ligne]. [s. d.]. [Consulté le 16 janvier 2021]. Disponible à l'adresse : <https://matplotlib.org/>

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/>

PyOpenGL – The Python OpenGL Binding [en ligne]. 24 janvier 2021. Disponible à l'adresse : <http://pyopengl.sourceforge.net/>

PyTorch [en ligne]. [s. d.]. [Consulté le 20 janvier 2021]. Disponible à l'adresse : <https://pytorch.org/>

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>

Rhino - Rhino.Inside [en ligne]. 25 janvier 2021. Disponible à l'adresse : <https://www.rhino3d.com/features/rhino-inside/>

scikit-learn : machine learning in Python [en ligne]. [s. d.]. [Consulté le 16 janvier 2021]. Disponible à l'adresse : <https://scikit-learn.org/stable/>

Seaborn : statistical data visualization [en ligne]. [s. d.]. [Consulté le 28 janvier 2021]. Disponible à l'adresse : <https://seaborn.pydata.org/>

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/>

TensorFlow [en ligne]. [s. d.]. [Consulté le 20 janvier 2021]. Disponible à l'adresse : <https://www.tensorflow.org/The-Python-Graph-Gallery-Visualizing-data-with-Python> [en ligne]. [s. d.]. [Consulté le 28 janvier 2021]. Disponible à l'adresse : <https://python-graph-gallery.com/>

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>

Valode&Pistre se lance dans l'aventure de l'IA [en ligne]. 8 février 2021. Disponible à l'adresse : <https://www.caheurs-techniques-batiment.fr/article/valode-pistre-se-lance-dans-l-aventure-de-l-ia.45391>

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. I.] : Zenodo, 14 mai 2014. [Consulté le 28 dé-

cembre 2020]. DOI 10.5281/ZENODO.9795

P.ROUGIER, Nicolas. *Python & OpenGL for Scientific Visualization*. Bordeaux : [s. n.], 2018. Disponible à l'adresse :
<https://www.labri.fr/perso/nrougier/python-opengl/>

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

ZHOU, Qian-Yi, PARK, Jaesik et KOLTUN, Vladlen. Open3D : A Modern Library for 3D Data Processing.
arXiv :1801.09847. 2018

ICONOGRAPHIE

1.1	Prévisualisation cartographique du jeu de données des volumes bâtis. - https://opendata.paris.fr/	12
1.2	Prévisualisation du jeu de données des volumes bâtis sous forme de tableau. - https://opendata.paris.fr/	13
1.3	Primitives géométriques des données géoréférencées - https://github.com/ClementDelgrange/Cours_programmation_SIG/	14
1.4	Page descriptive issue des métadonnées - https://opendata.paris.fr/	16
1.5	Table descriptive des variables issue des métadonnées - https://opendata.paris.fr/	17
1.6	Définition d'une zone d'extraction de données des volumes bâtis - https://opendata.paris.fr/	19
1.7	Import de données au format JSON dans Python - Réalisation personnelle	21
1.8	Aperçu des différentes notations caractérisant les portes à faux - Réalisation personnelle	26
2.1	Moyens d'export et de synthèse des données récoltées - Réalisation personnelle	31
2.2	Nombre de volumes par hauteur - Réalisation personnelle	34
2.3	Répartition des volumes selon leurs caractéristiques - Réalisation personnelle	35
2.4	Carte des volumes par hauteur avec de l'interactivité - Réalisation personnelle	38
2.5	Fichier vectoriel organisé des emprises des volumes - Réalisation personnelle	41
2.6	Modèle 3D généré grâce aux données de hauteur des volumes - Réalisation personnelle	45
2.7	Une précision et une richesse rendue visible grâce à la modélisation - Réalisation personnelle	46
3.1	Photographie aérienne de la ZAC Ivry Confluences, contexte du site choisi - Google Earth	50
3.2	Transformation de la ZAC et temporalité des démolitions - Réalisation personnelle	51
3.3	Prévisualisation des données sous forme de carte - https://www.apur.org/open_data/	54
3.4	Des variables fournissant des informations précieuses sur l'existant - https://www.apur.org/open_data/BESOIN_THEORIQUE_CHAUFFAGE_TYP0_BATI_OD.pdf	55

3.5 Opérations géométriques à effectuer à partir des données brutes - Réalisation personnelle	56
3.6 Extrait des métadonnées : attributs pour chaque bâtiment dans la BD TOPO - https://qlf-geoservices.ign.fr/ressources_documentaires/Espace_documentaire/BASES_VECTORIELLES/BDTOPO/DC_BDTOPO_3-0.pdf	57
3.7 Extrait des métadonnées : détail des attributs concernant les matériaux - https://qlf-geoservices.ign.fr/ressources_documentaires/Espace_documentaire/BASES_VECTORIELLES/BDTOPO/DC_BDTOPO_3-0.pdf	57
3.8 Extraits des tables des matériaux pour les murs (gauche) et les toitures (droite) - http://piece-jointe-carto.developpement-durable.gouv.fr/NAT004/DTerNP/html3/annexes/	58
3.9 Visualisation d'un bâtiment et de ses données dans la BD TOPO d'IGN - Réalisation personnelle	59
3.10 Extrait d'un carreau de la BD ORTHO (gauche) et ses données de géolocalisation (droite) - https://geoservices.ign.fr/	61
3.11 Un langage à la fois complet et souple à la syntaxe composée exclusivement de caractères spéciaux - https://markdown-it.github.io/	72
3.12 Outils et formats intermédiaires utilisés pour générer un fichier PDF - https://markdown-it.github.io/	74

ANNEXE : UN MÉMOIRE RÉDIGÉ EN R MARKDOWN

Ce mémoire a été rédigé dans son intégralité en utilisant le langage **R markdown**, sous la forme d'un seul script.

Ce langage est basé en grande partie sur la syntaxe **Markdown**,²⁸ dont le but est de prodiguer un moyen de rédiger des documents textuels de manière **simplifiée** et **souple**, exportable dans une multitude de supports, du format HTML (pour un affichage web) au format PDF paginé pour impression.

²⁸Daring Fireball : Markdown Syntax Documentation [en ligne]. [s. d.]. [Consulté le 5 février 2021]. Disponible à l'adresse : <https://daringfireball.net/projects/markdown/syntax>.

[clear](#) [permalink](#)

```

## Code

Inline `code`

Indented code

    // Some comments
    line 1 of code
    line 2 of code
    line 3 of code

Block code "fences"
```
Sample text here...
```

Syntax highlighting

```js
var foo = function (bar) {
 return bar++;
};

console.log(foo(5));
```

## Tables

Option	Description
data	path to data files to supply the data that will be passed into templates.
engine	engine to be used for processing templates. Handlebars is the default.
ext	extension to be used for dest files.

Right aligned columns

Option	Description
data	path to data files to supply the data that will be passed into templates.
engine	engine to be used for processing templates. Handlebars is the default.
ext	extension to be used for dest files.

## Links

[link text](http://dev.nodeca.com)

[link with title](http://nodeca.github.io/pica/demo/ "title text!")

Autoconverted link https://github.com/nodeca/pica (enable linkify to see)

## Images

![Minion](https://octodex.github.com/images/minion.png)
![Stormtroopocat](https://octodex.github.com/images/stormtroopocat.jpg "The Stormtroopocat")

Like links, Images also have a footnote style syntax

![Alt text][id]

With a reference later in the document defining the URL location:

```

[html](#) [source](#) [debug](#)

Code

Inline `code`

Indented code

```
// Some comments
line 1 of code
line 2 of code
line 3 of code
```

Block code "fences"

```
Sample text here...
```

Syntax highlighting

```
var foo = function (bar) {
    return bar++;
};

console.log(foo(5));
```

Tables

| Option | Description |
|--------|---|
| data | path to data files to supply the data that will be passed into templates. |
| engine | engine to be used for processing templates. Handlebars is the default. |
| ext | extension to be used for dest files. |

Right aligned columns

| Option | Description |
|--------|---|
| data | path to data files to supply the data that will be passed into templates. |
| engine | engine to be used for processing templates. Handlebars is the default. |
| ext | extension to be used for dest files. |

Links

[link text](#)

[link with title](#)

Autoconverted link <https://github.com/nodeca/pica> (enable linkify to see)

Images

FIG. 3.11 : Un langage à la fois complet et souple à la syntaxe composée exclusivement de caractères spéciaux

Dans le cadre du mémoire, je me suis plutôt orienté sur son dérivé augmenté par le langage de programmation

R (d'où le nom de *R markdown*) pour les raisons suivantes :

- Là où le *Markdown* permet uniquement d'afficher du code, il est possible ici **d'exécuter du code directement**

au sein du document. Le tout reste hautement configurable, permettant l'affichage des données en sortie (output), ainsi que de gérer la mise en forme graphique.

- Grâce à un autre logiciel nommé **LaTeX**, il est possible d'y intégrer facilement des **références bibliographiques**, via un fichier .bib contenant les différents ouvrages et leurs informations, en pouvant spécifier la norme des références. La gestion des **images et figurés** est également facilitée, avec une numérotation ainsi qu'une iconographie générée automatiquement tout en restant paramétrable.
- L'insertion de caractères relatifs **aux notations mathématiques** est également facilitée, pouvant s'avérer extrêmement utile.
- Il sera possible **avec peu de changements** à apporter au script d'exporter facilement mon mémoire vers d'autres formats (HTML/DOCX/etc..) à l'avenir.

Cependant, bien que cette solution se soit révélée d'une praticité redoutable dans le cadre de mon mémoire, il me paraît important de souligner les quelques difficultés rencontrées.

- Là où le *Markdown* ne requiert le plus souvent aucune installation tierce pour fonctionner (étant la plupart du temps intégré directement aux éditeurs de texte), la liste de programmes à installer est ici assez conséquente. En plus du langage **R**, ce sont au total **trois solutions logicielles** qui sont réclamées afin de pouvoir générer un fichier PDF (avec LaTeX pesant notamment entre 7 et 8Go).

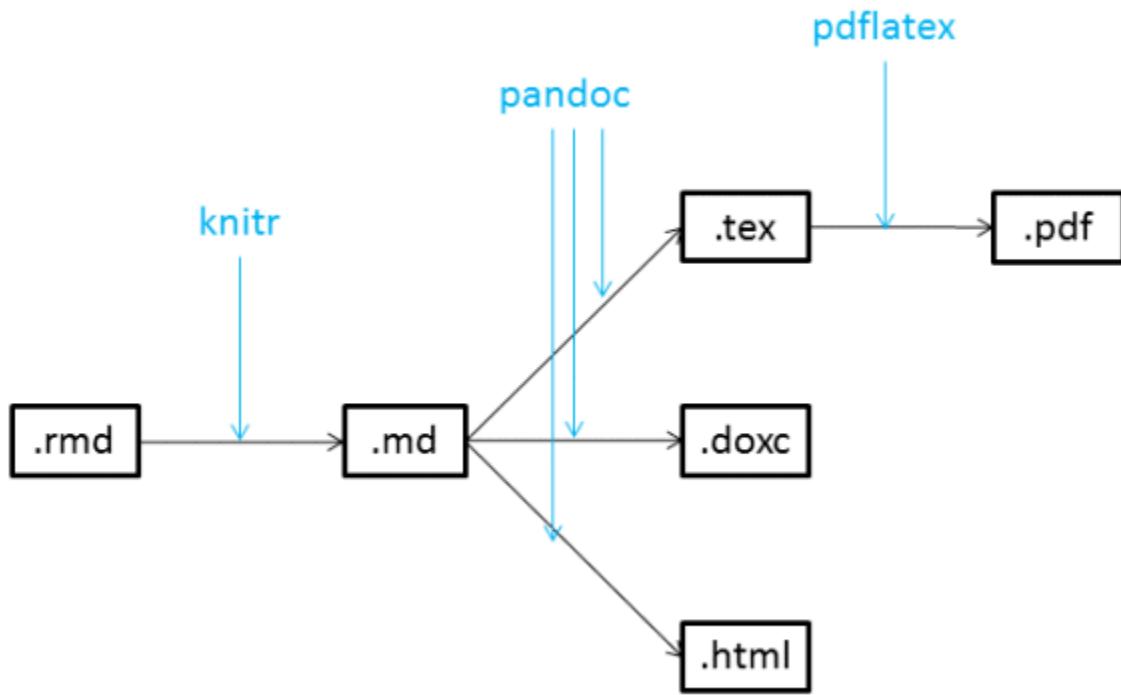


FIG. 3.12 : Outils et formats intermédiaires utilisés pour générer un fichier PDF

- L'utilisation de **LaTeX**, **Pandocs** et **Knitr** en plus du *R markdown* permet une large palette de paramétrage, que l'on paye parfois au prix **d'incompatibilités entre certaines options présentes sur plusieurs de ces programmes à la fois.**
- Enfin, le point le plus frustrant a été **le manque de documentation officielle sur certaines fonctionnalités**. Bien que l'on puisse y pallier grâce aux forums entre autres, cela représente l'antithèse de la simplicité du *Markdown*, dont la documentation de qualité foisonne sur internet.

Malgré ces défauts, cela reste un moyen efficace de rédiger un document de recherche tel qu'un mémoire, et encore davantage pour des documents plus conséquents comme une thèse. Cependant, je souhaiterais explorer d'autres outils similaires moins répandus mais plus minimalistes dans leur approche comme le **Groff**, afin de déterminer quelle solution est à la fois complète, simple et souple.