



Rapport E1

MACHINE LEARNING

Exploiter l'intelligence artificielle dans le développement d'applications

Prédiction du prix des logements à Lyon

Sommaire

INTRODUCTION	3
CHAPITRE I	4
Compréhension du besoin client et traduction technique et choix technique du projet	4
1. Compréhension du besoin client	4
2. Traduction technique et choix technique du projet	5
<i>2.1 Qu'est-ce que Machine Learning (ML)</i>	6
<i>2.2 Types de Machine Learning</i>	7
<i>2.3 Comment fonctionne le Machine Learning ?</i>	9
<i>2.4 Choix technique du projet</i>	10
CHAPITRE II	11
Réponse technique mise en œuvre dans projet	11
1. Data preprocessing	11
<i>1.1 Collecte de données</i>	11
<i>1.2 Préparation des données</i>	12
2. Modèle de régression prédictive	39
<i>2.1 Choix d'un modèle</i>	39
<i>2.2 Entraînement du modèle</i>	42
<i>2.3 Evaluation</i>	43
<i>2.4 Réglage des paramètres</i>	46
3. Amélioration du modèle	47
<i>3.1 Régression linéaire à variables multiples</i>	47
<i>3.2 Arbre de décision</i>	50
<i>3.3 Forêt aléatoire</i>	54

CHAPITRE III	59
Bilan et les améliorations du projet	59
1. Prédiction	60
2. Déploiement	61
CONCLUSION	63
Annexe	64
1. Modèle conceptuel de données (MCD)	64
2. Modèle physique de données (MPD)	65

INTRODUCTION

Sur le marché de l'immobilier, on peut constater que la demande de logement augmente et particulièrement dans les grandes villes. Mais pour les agences immobilières il n'est pas simple de comprendre les besoins des clients, chacun ayant différentes attentes. C'est également difficile pour les clients de trouver un logement respectant tous leurs critères tout en restant à un prix abordable. Comprendre le marché du logement et obtenir des informations sur l'évolution des prix est la clé pour prendre une décision d'achat.

C'est comme ça que l'idée de développer une application qui aide les utilisateurs à prévoir les prix des logements en fonction des besoins spécifiques de chaque personne est née. L'objectif de cette application est de pouvoir aider les agences mais également les acheteurs.

CHAPITRE I

Compréhension du besoin client et traduction technique et choix technique du projet

1. COMPREHENSION DU BESOIN CLIENT

Pour ce projet, je n'ai pas eu l'opportunité de réaliser un projet avec un vrai client. Le besoin est donc issu de mon besoin réel d'acheter un appartement à Lyon. Je me suis donc mise à la place de l'utilisateur pour définir les besoins des clients.

Lors de l'achat d'un appartement, la première information que je regarde est le prix, car ce serait pour moi mon premier appartement. Donc, la prédiction du prix d'un appartement est quelque chose qui m'intéresse beaucoup et que je trouve très important dans le processus de prise de décision d'achat d'un appartement.

Mon besoin actuel est d'acheter une maison / un appartement, pas de louer, donc dans ce projet, je me concentre sur l'achat d'un logement. De plus, dans le cadre du projet, je travaille uniquement sur la ville de Lyon, car mon besoin est de trouver un appartement dans cette ville.

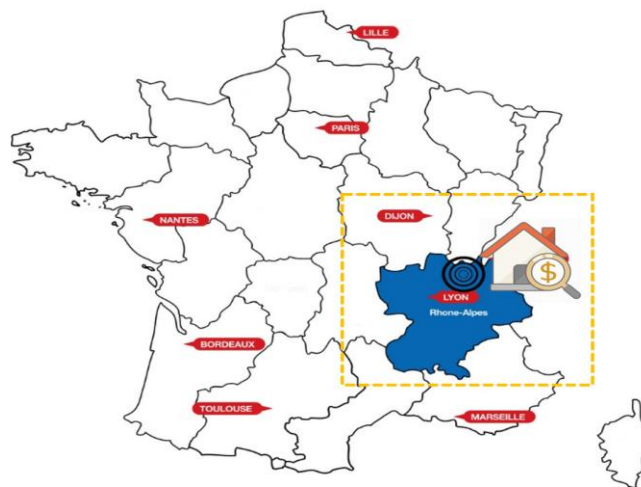


Figure 1 : Portée du projet

2. TRADUCTION TECHNIQUE ET CHOIX TECHNIQUE DU PROJET

Au cours des dernières décennies, l'intelligence artificielle est devenue l'évolution majeur pour la société depuis révolution informatique. Dans le cadre de celle-ci, le machine learning et les techniques associées sont utilisées pour résoudre les problèmes de la vie réelle à la place des humains. Les systèmes d'aide à la décision basés sur la modélisation prédictive sont de plus en plus courants. Les modèles prédictifs sont utilisés pour trouver des indications potentiellement intéressantes dans les données ou pour prédire le résultat d'un événement. Il existe de nombreuses techniques prédictives, allant de techniques simples telles que la régression linéaire aux techniques plus complexes et puissantes telles que les réseaux de neurones artificiels.

Les modèles complexes obtiennent généralement de meilleures performances prédictives, mais sont opaques et ne peuvent donc pas être utilisés pour expliquer les prédictions. Le choix de la conception de la technique prédictive à utiliser devient encore plus difficile car aucune technique ne surpasse les autres sur un large ensemble de problèmes. Il est même difficile de trouver le meilleur paramétrage pour une technique spécifique, car les réglages dépendent également du problème.

Bref, il n'y a pas de solution miracle et donc il n'existe pas de technique optimale qui s'applique à tout type de données et de domaines. Chaque méthode, chaque technique a ses avantages et ses inconvénients.

Avec ma propre compréhension, mes expériences ainsi que les exigences de la formation, j'ai choisi la méthode de Machine Learning pour mon modèle. Dans ce projet, je développerais différents algorithmes supervisés de machine learning telles que : régression linéaire, Arbres de décision, forêt aléatoire et je ferai les comparaisons pour savoir quel modèle est le mieux adapté aux données que j'ai obtenues ainsi qu'aux circonstances spécifiques du projet.

2.1 Qu'est-ce que Machine Learning (ML)

Avant de parler de ML, je veux parler de 3 éléments différents que l'on peut regrouper sous le terme « intelligence artificielle » : Intelligence Artificielle (IA), Machine Learning (ML) et Apprentissage Profond (Deep learning – DL). IA, ML et DL sont souvent confondus car ils partagent un bon nombre de caractéristiques. Pour faire simple, DL est un sous-ensemble de ML, qui est lui-même un sous-ensemble de l'IA (figure 2). Par conséquent, le DL est aussi de l'IA, mais l'IA n'est pas nécessairement du DL. Par définition, tous sont des programmes informatiques qui sont développés pour essayer de reproduire des comportements humains en appliquant des algorithmes.

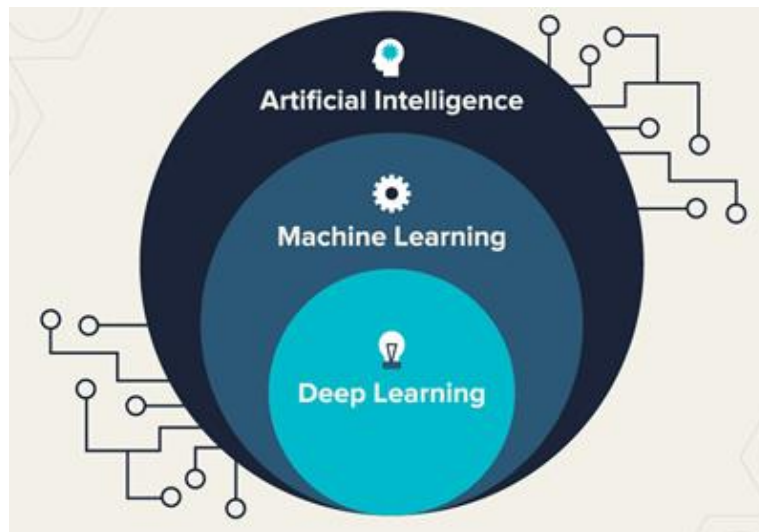


Figure 2 : Différences entre IA, ML et DL

Pour une définition plus détaillée de chaque concept, je voudrais définir ce qui suit :

- **L'intelligence artificielle** est une science comme les mathématiques ou la biologie. Elle étudie les moyens de créer des programmes capables de résoudre des problèmes de manière créative.
- Le **Machine learning** est un sous-ensemble de l'IA qui offre aux systèmes la possibilité d'apprendre et de s'améliorer. En ML, il existe différents algorithmes (par exemple linéaire, arbres de

décision, forêt aléatoire, xgboost...) qui aident à résoudre les problèmes.

- **L'apprentissage profond** ou apprentissage neuronal profond, est un sous-ensemble du ML, qui utilise les réseaux de neurones pour analyser différents facteurs.

En bref, le ML est un programme qui a la capacité d'apprendre grâce aux données. A partir de cet apprentissage, il est possible de découvrir et identifier les tendances et les « patterns » que nous utiliserons pour prédire, classer, analyser, détecter les problèmes, puis comparer et prendre des décisions.

2.2 Types de Machine Learning

Il existe de nombreux types de Machine Learning que nous pouvons utiliser pour notre application. Le type ML est déterminé par de nombreux critères tels que le type et la quantité de données dont nous disposons, qu'allons-nous faire avec le modèle ML, comment allons-nous entraîner notre modèle ML, etc... Les principales catégories de systèmes de ML sont présentées sur la figure 3.

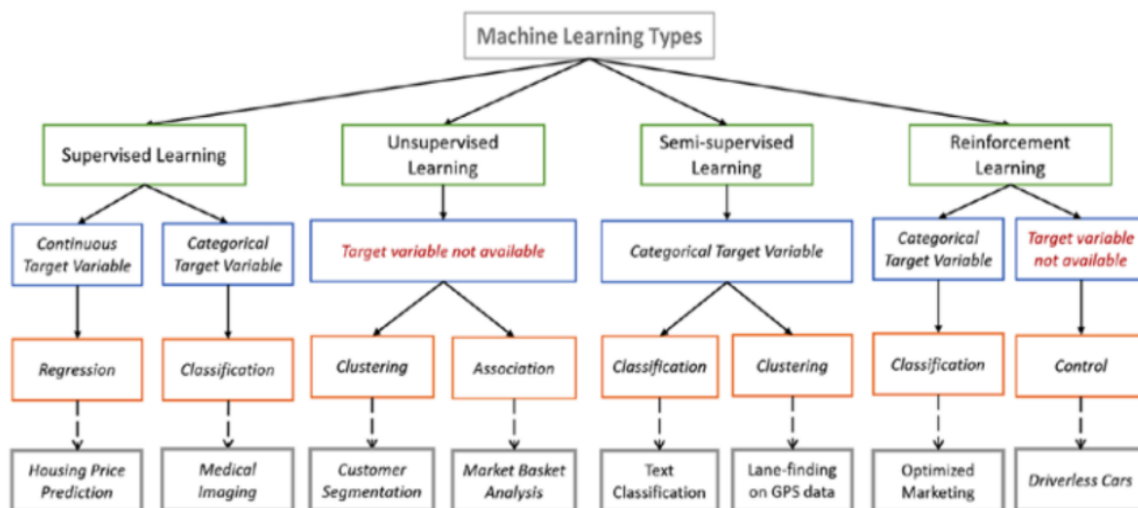


Figure 3 : Catégorisation principale de ML

En regardant le diagramme de la figure 3, nous voyons que le ML est classé en quatre catégories principales : supervisé, semi- supervisé, renfort et non supervisé. Dans chacune des grandes catégories, nous avons d'autres sous-catégories. Dans le cadre de ce rapport, je voudrais juste donner une description des principales catégories.

- **Supervisé** : le modèle est entraîné avec des données étiquetées (labeled) qui permet de modéliser les relations et les dépendances entre la sortie de prédiction cible (target) et les caractéristiques d'entrée. Ce modèle permet de prédire les valeurs de sortie avec de nouvelles données en fonction des relations apprises à partir des ensembles de données précédents. Les principaux types d'algorithme supervisé sont la régression et la classification
- **Non supervisé** : le modèle est entraîné avec des données non étiquetées (unlabeled). Cet algorithme est particulièrement utile dans le cas où on ne sait pas quoi rechercher dans les données. Il n'y a pas de catégories de sortie ou d'étiquettes sur la base desquelles l'algorithme peut essayer de modéliser les relations. Ces algorithmes utilisent des techniques sur les données d'entrée pour rechercher des règles, détecter des modèles, résumer et regrouper les points de données qui aident à obtenir des informations significatives et à mieux décrire les données aux utilisateurs. Le type principal d'algorithme « non supervisé » est le clustering.
- **Semi-supervisé** : L'apprentissage semi-supervisé se situe entre supervisé et non supervisé. Dans de nombreuses situations pratiques, le coût de l'étiquetage est très élevé, car cela nécessite des experts pour le faire. Ainsi, en l'absence d'étiquettes dans la majorité des observations mais présentes dans peu, les algorithmes semi-supervisés sont les meilleurs pour la construction du modèle. Ces méthodes exploitent l'idée que même si les appartenances aux groupes des données non étiquetées sont inconnues, ces données contiennent des informations importantes sur les paramètres du groupe.
- **Renfort** : vise à utiliser les observations recueillies à partir de l'interaction avec l'environnement pour prendre des mesures qui

maximiseraient la récompense ou minimiseraient le risque. L'algorithme d'apprentissage par renforcement (appelé l'agent) apprend continuellement de l'environnement de manière itérative. Dans le processus, l'agent apprend de ses expériences avec l'environnement jusqu'à ce qu'il explore la gamme complète des états possibles.

2.3 Comment fonctionne le Machine Learning ?

Le Machine Learning est formé à l'aide d'un ensemble de données d'apprentissage pour créer un modèle. Lorsque de nouvelles données d'entrée sont introduites dans l'algorithme de ML, il effectue une prédiction sur la base du modèle.

La précision de la prédiction est évaluée et si la précision est acceptable, le ML est déployé. Si la précision n'est pas acceptable, l'algorithme de ML est entraîné à plusieurs reprises avec un ensemble de données d'apprentissage amélioré.

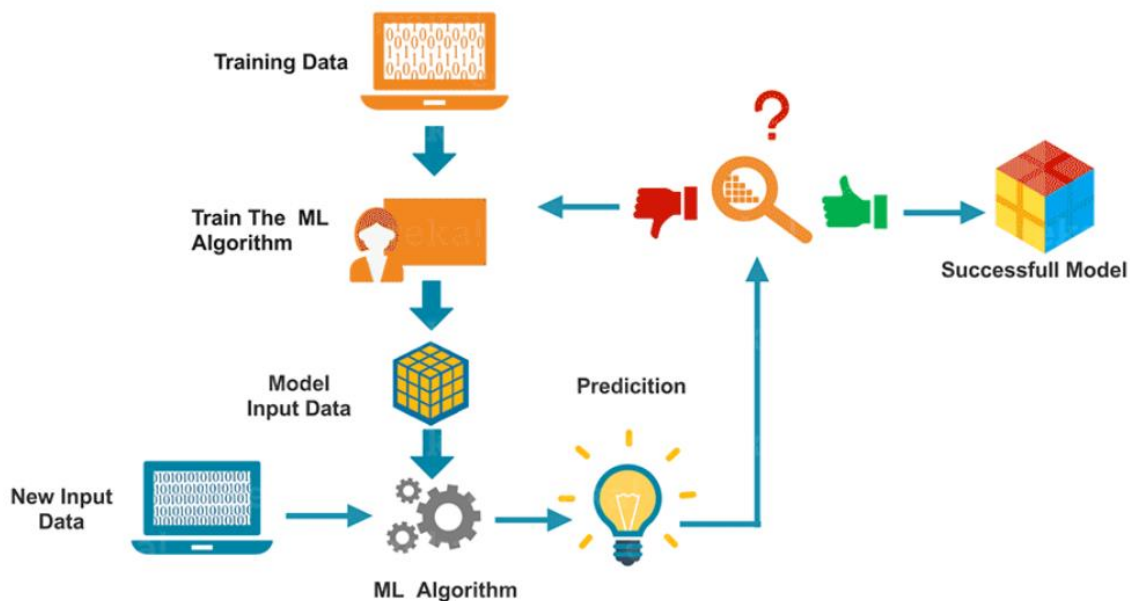


Figure 4 : Processus de Machine Learning

2.4 Choix technique du projet

Le but de ce projet est de créer une application de prédiction des prix de l'immobilier. Pour cette raison, je connaissais déjà la cible à atteindre, le prix du logement, alors j'ai choisi le Machine Learning supervisé.

L'apprentissage supervisé est divisé en deux sous-catégories : classification et régression.

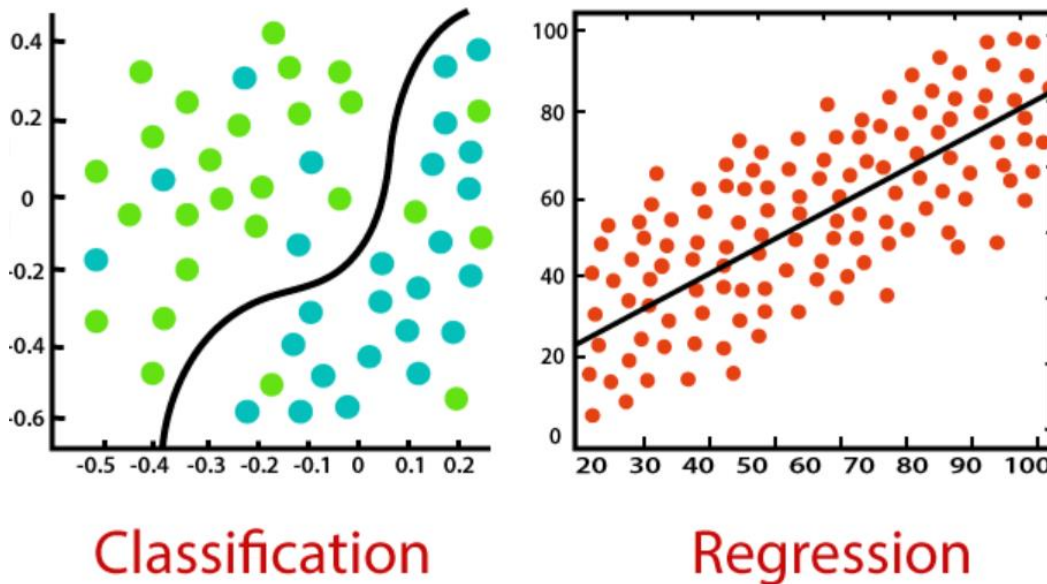


Figure 5 : Classification et régression en Machine Learning

Le travail d'un algorithme de classification est de prendre une valeur d'entrée et de lui attribuer une classe, ou une catégorie, dans laquelle il s'intègre en fonction des données d'entraînement fournies.

La régression est un processus statistique prédictif dans lequel le modèle tente de trouver une relation importante entre les variables dépendantes et indépendantes. Le but de l'algorithme de régression est de prédire un continuum comme les ventes, les revenus, les résultats des tests ou encore les prix des logements. C'est pourquoi dans ce projet, j'ai choisi d'appliquer la technique de la régression.

CHAPITRE II

Réponse technique mise en œuvre dans projet

Pour résoudre un problème de ML, nous avons 7 étapes principales : collecte de données, préparation des données, choix d'un modèle, entraînement du modèle, évaluation, réglage de paramètres et prédiction.

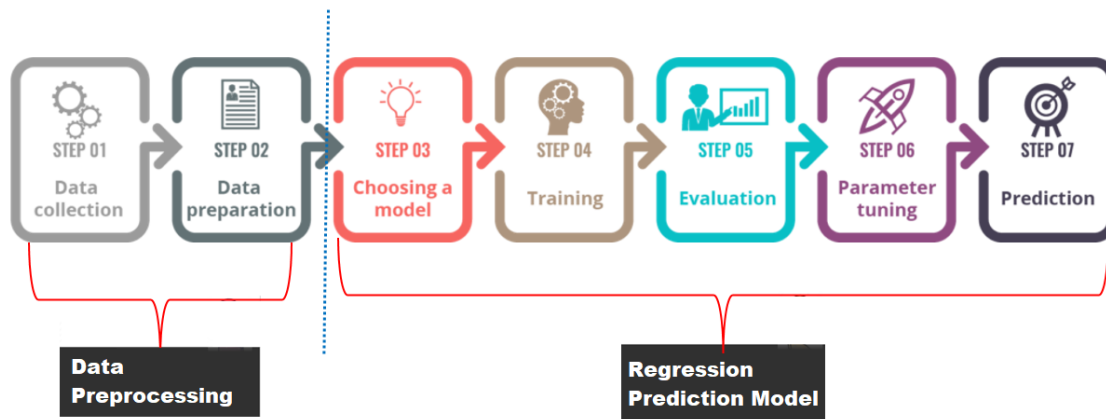


Figure 6 : Sept étapes principales dans ML

Dans le schéma précédent, les deux premières étapes correspondent au processus de préparation des données d'entrée pour le modèle, les 5 étapes suivantes sont le processus de développement du modèle et d'utilisation du modèle pour prédire les prix des logements. Cependant, au cours du processus de mise en œuvre de ces étapes, il y aura des moments où nous devrons revenir aux étapes précédentes pour optimiser le modèle et trouver la meilleure adaptation avec le contexte du projet.

1. DATA PREPROCESSING

1.1 Collecte de données

Les données de ce projet sont des données open source portant sur les valeurs foncières déclarées à l'occasion des mutations immobilières en France. La publication de ces données répond à l'objectif de transparence des marchés fonciers et immobiliers. Je les ai collectées et téléchargées sur le site

du gouvernement français (data.gouv.fr) qui est pilotée, sous l'autorité du Premier ministre, par la mission Etalab, dirigée par Mme Laure Lucchesi.

Voici le lien des données :

<https://www.data.gouv.fr/en/datasets/demandes-de-valeurs-foncières-geolocalisées/>

Les données collectées comprennent 6 fichiers CSV correspondant à 6 années de données de 2014 à 2019 (figure 7). Chaque fichier de données pèse en moyenne 450 Ko, avec 40 champs et deux millions et demi de lignes de données. Dans l'ensemble, il s'agit d'un ensemble de données assez volumineux.







Name ^	Type	Size
 2014	CSV File	426,648 KB
 2015	CSV File	467,696 KB
 2016	CSV File	499,288 KB
 2017	CSV File	572,077 KB
 2018	CSV File	395,921 KB
 2019	CSV File	171,972 KB

Figure 7 : Six fichiers CSV correspondant à 6 années de données
(2014 -2019)

1.2 Préparation des données

La préparation des données est l'une des étapes indispensables de tout cycle de vie de développement de Machine Learning. Dans le monde d'aujourd'hui, les données sont présentes sous une forme structurée ou non structurée. Pour traiter ces données, nous devons passer près de 70 à 80% du temps à les préparer afin de mieux comprendre les données et d'en extraire les informations les plus précieuses. Dans ce projet, j'ai passé aussi la plupart de mon temps sur cette étape.

Afin de préparer au mieux les données, nous devons également passer par de nombreuses étapes comme : sélectionner, découvrir, nettoyer, transformer, enrichir et stocker (figure 8).

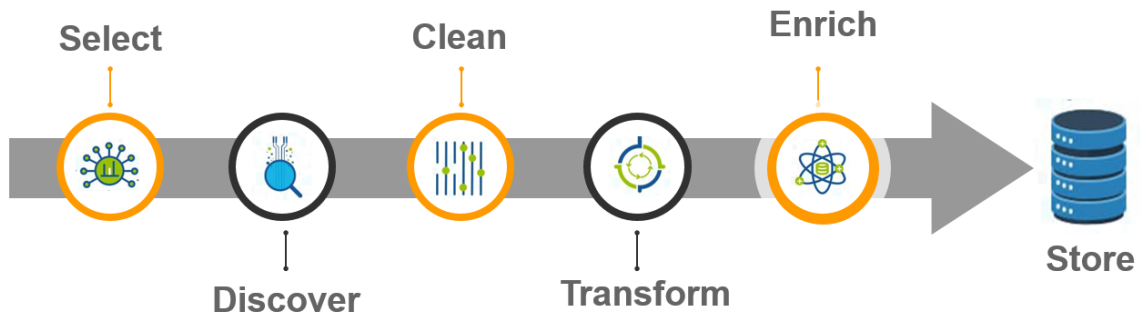


Figure 8 : *Etapes de préparation des données*

- **Sélectionner**

Une fois que nous avons collecté les données nécessaires pour le projet, nous ne pouvons pas les utiliser immédiatement, nous devons sélectionner les éléments de données à partir des données d'origine qui correspondent aux critères et besoins du projet.

Pour ce projet, les données que j'ai rassemblées à l'étape ci-dessus sont les données sur les prix des logements pour toute la France, mais dans le cadre du projet, je ne m'intéresse qu'à la ville de Lyon, donc je ne sélectionnerais que les données relatives à Lyon.

Année	La France (lignes)	Lyon (lignes)	Lyon/France (%)
2014	2516688	17429	0.69
2015	2749830	24622	0.90
2016	2936524	18796	0.64
2017	3361073	22569	0.67
2018	2339002	1617	0.07
2019	1017154	2046	0.20

Figure 9 : *Pourcentage de données de Lyon par rapport à la France de 2014 à 2019*

Pour faire le travail de sélection, j'utilise la librairie Python Pandas. Après avoir sélectionné uniquement les données de Lyon avec le champ correspondant dans le fichier de données « code postal » avec les valeurs de 69001 à 69009, la quantité de données a été considérablement réduite, à moins de 1% des données originales (figure 9).

Cependant, avec seulement 1%, le nombre de lignes de données est toujours élevé, de 2014 à 2017, nous avons en moyenne plus de 20000 lignes de données par an. Pour 2018 et 2019 seuls nous avons environ 2 milles lignes par fichier.

En plus de filtrer les données par ville (filtre 1), sur la base de la demande du client, pour ce projet, je ne m'intéresse qu'aux maisons ou appartements, les autres biens immobiliers tels que terrain, jardin, etc... ne m'intéressent pas. Je ne sélectionnerai donc que les données relatives aux maisons et aux appartements (filtre 2), et je supprimerai les autres données.

De plus, avec la compréhension et l'analyse des besoins des clients, le modèle se construit uniquement sur la vente (filtre 3), toutes les autres actions liées à l'immobilier comme la location, l'adjudication, l'échange, l'expropriation, etc.. ont également été supprimées.

Après avoir appliqué ces deux filtres, les données restantes sont comme suivies (figure 10) :

Année	Lyon	Nb de lignes après appliquer filtration 2 et 3	% données restantes après filtration 2 et 3
2014	17429	9492	54
2015	24622	13575	55
2016	18796	9679	51
2017	22569	10466	46
2018	1617	806	50
2019	2046	1108	54

Figure 10 : Données restantes après filtration 2 et 3 de 2014 à 2019

Il est clair qu'après chaque filtre, la quantité de données est réduite, et cette fois, la quantité de données est réduite de moitié (50%) par rapport à la quantité de données à Lyon (figure 11).

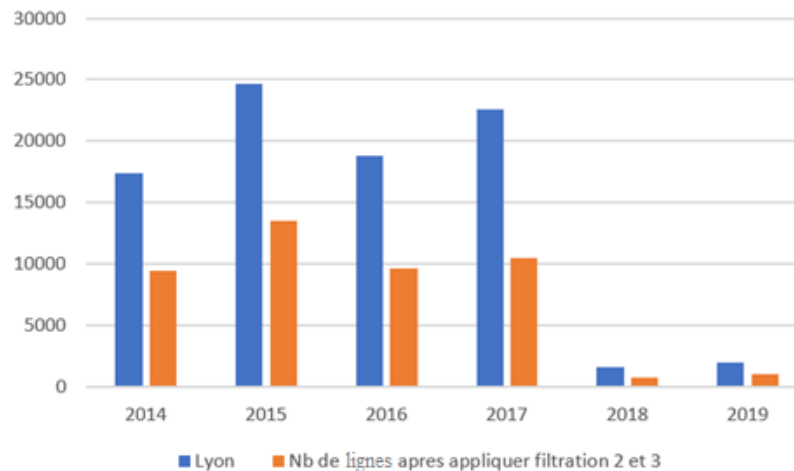


Figure 11 : Graphique nombre de lignes de données avant et après l'application des filtres 2 et 3 de 2014 à 2019

- **Découvrir**

En fonction de l'objectif du projet et des exigences du client, nous avons d'abord sélectionné les bonnes données pour le modèle. Cependant ce n'est pas suffisant et pour améliorer notre sélection, il faut aussi comprendre les données. L'étape suivante est donc l'étape de la découverte des données dans le moindre détail.

Pour une compréhension plus approfondie des données, la méthode la plus intuitive est la visualisation. Cependant, avant de faire la visualisation des données, je vais fusionner les 6 fichiers CSV en un seul fichier CSV en utilisant la méthode « concat() » de la librairie Pandas.

Maintenant, nous n'avons qu'un seul fichier CSV, avec la forme (45126, 40). Autrement dit, nous avons 45126 lignes et 40 colonnes. Quarante colonnes c'est trop pour entraîner un modèle, donc avant de visualiser les données, je vais choisir les propriétés que je trouve les plus significatives, celles qui auront le plus d'impact sur les acheteurs lors d'une vente.

Après avoir sondé les personnes qui ont acheté des maisons et celles qui envisagent d'acheter une maison, j'ai décidé de choisir les 12 colonnes de données suivantes :

1	id_mutation
2	date_mutation
3	nature_mutation
4	valeur_fonciere
5	code_postal
6	code_type_local
7	type_local
8	surface_reelle_bati
9	surface_terrain
10	nombre_pieces_principales
11	longitude
12	latitude

Figure 12 : Les 12 colonnes de données sélectionnées

Maintenant, je vais utiliser Power BI pour visualiser ces données. L'objectif de la visualisation des données est de mieux comprendre l'évolution des prix des logements sur ces 6 années (2014 à 2019). Premièrement, je veux connaître le prix moyen, le prix minimum et le prix le plus élevé des logements sur 6 ans. Le résultat est surprenant et semble être faux car le logement le plus bas est à 1 euro, le plus élevé à 77,6 millions d'euros, la valeur moyenne est également supérieure à celle de l'immobilier lyonnais (près de 910 mille euros pour un logement « moyen »).



Figure 13 : Prix minimum, moyen et maximum de logement sur les 6 années

Pour savoir quelle est la cause d'un tel résultat dans les prix des logements, je dois approfondir ma compréhension des données. Je vais commencer par regarder le prix moyen.

En dessous j'ai 3 graphiques, correspondant au prix moyen du logement par année, par mois et par trimestre. De plus, j'ai également catégorisé les données en maisons et appartements afin de mieux comprendre les données. Le bleu foncé représente les maisons et le bleu plus clair représente les appartements.

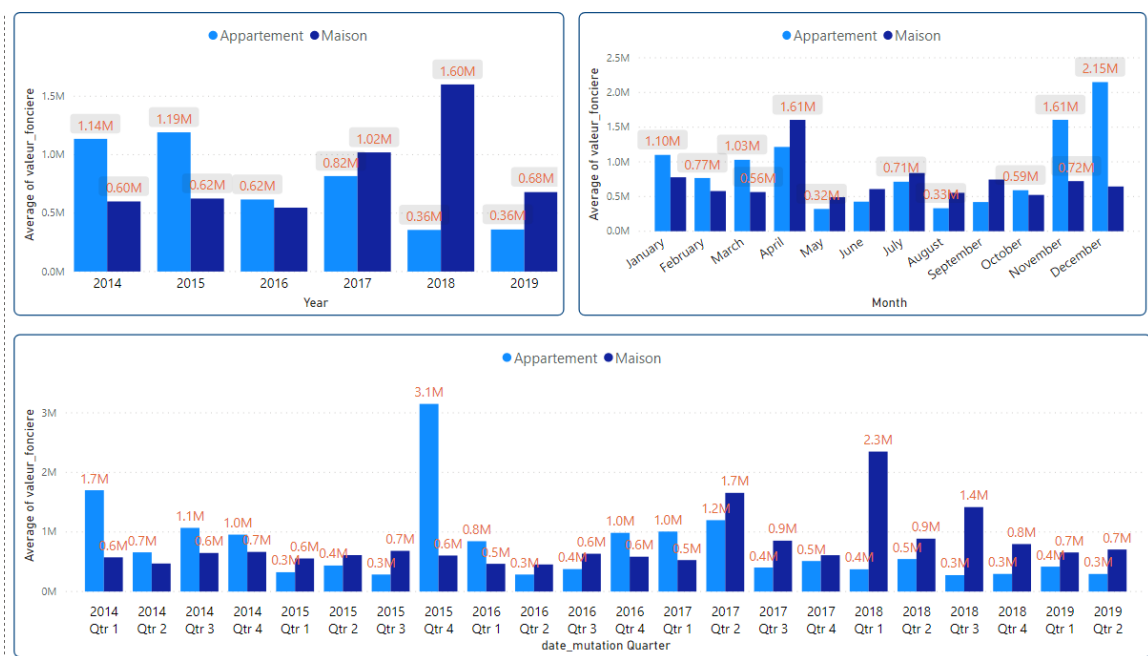


Figure 14 : Prix moyen des logements sur les 6 années

En regardant la figure 14 dans le coin supérieur gauche, nous voyons que les prix des appartements ont tendance à baisser d'année en année, mais les prix des maisons ont tendance à augmenter. Surtout en 2018, le prix moyen d'une maison peut atteindre 1,6 million d'euros pour une maison.

Dans le coin supérieur droit de la figure 14, on voit que le prix moyen des logements dans les derniers mois de l'année et en début d'année est un peu plus élevé, surtout en décembre, avec 2,15 millions d'euros pour un appartement. Les prix moyens des logements au milieu de l'année ont diminué.

Avec le graphique final, on constate une nette différence de prix des logements au quatrième trimestre 2015, le prix moyen d'un appartement est supérieur à 3 millions euros. C'est certainement absurde. Et à cause de ces absurdités, nous avons besoin de l'étape suivante : nettoyer les données.

- **Nettoyer**

Avant de nettoyer les données, nous devons savoir où les nettoyer, je vais donc zoomer plus profondément sur les valeurs aberrantes.

Par exemple, dans l'analyse ci-dessus, au quatrième trimestre 2015, nous avons constaté qu'il y avait une absurdité dans le prix moyen d'un appartement, je vais donc zoomer sur les données du quatrième trimestre 2015.

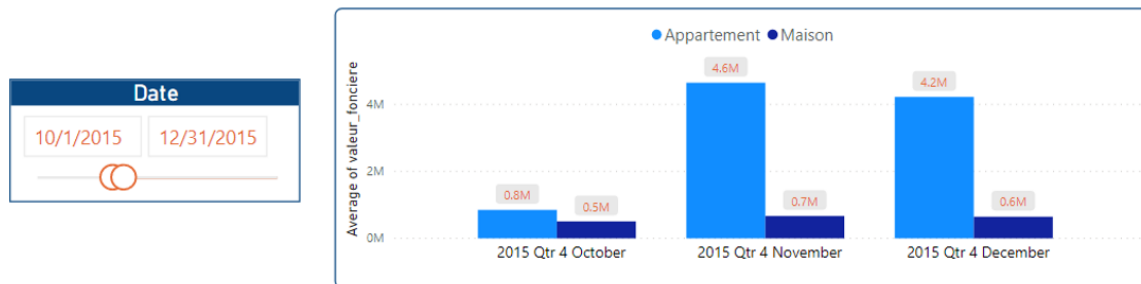


Figure 15 : *Prix moyen des logements sur le quatrième trimestre 2015*

En regardant le graphique de la figure 15, on voit qu'en octobre, le prix moyen des logements est plutôt correct, mais novembre et décembre sont encore élevés, plus de 4 millions d'euros pour un appartement, je vais donc zoomer sur les données du mois novembre et décembre 2015, en ne prenant en compte que les données des appartements.

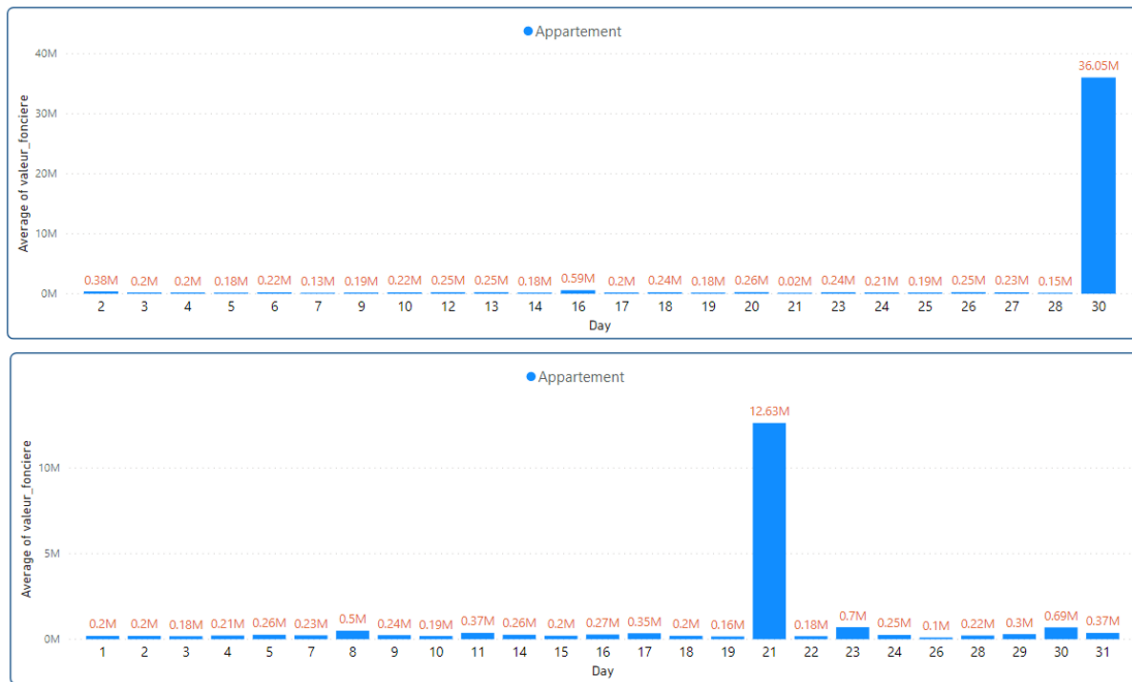


Figure 15 : Prix moyen des logements du mois novembre et décembre 2015

À ce stade, on voit plus clairement que les anomalies dans les données se situent au 30 novembre 2015 et au 21 décembre 2015. Je vais regarder la table de ces deux jours pour voir ce qui cause ces prix anormalement élevés.

id_mutation	date_mutation	nature_mutation	valeur_fonciera	code_postal	code_type_local	type_local	surface_reelle_bati	nombre_pieces_principales	longitude	latitude
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	135	6	4.834851	45.76089
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	83	2	4.834851	45.76089
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	33	1	4.834851	45.76089
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	115	3	4.834851	45.76089
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	115	3	4.834851	45.76089
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	229	0	4.834851	45.76089
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	117	3	4.834851	45.76089
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	83	2	4.834851	45.76089
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	17	1	4.834851	45.76089
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	83	2	4.834851	45.76089
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	117	3	4.834851	45.76089
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	78	2	4.834851	45.76089
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	83	2	4.834851	45.76089
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	115	3	4.834851	45.76089
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	25	1	4.834851	45.76089
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	117	3	4.834851	45.76089
2015-767873	11/30/2015	Vente	77600000	69002	2	Appartement	123	4	4.834851	45.76089

Figure 16 : Table des données du 30 novembre 2015

id_mutation	date_mutation	nature_mutation	valeur_fonciere	code_postal	code_type_local	type_local	surface_reelle_bati	nombre_pieces_principales	longitude	latitude
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	86	4	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	87	4	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	98	5	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	70	3	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	97	5	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	72	3	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	82	4	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	98	5	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	86	4	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	86	4	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	84	4	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	97	5	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	72	3	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	98	5	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	82	4	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	98	5	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	70	3	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	82	4	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	98	5	4.8008	45.766612
2015-768430	12/21/2015	Vente	15804374	69009	2	Appartement	86	4	4.8008	45.766612

Figure 17 : Table des données du 21 décembre 2015

En regardant les deux tables des données du 30/11/2015 et 21/12/2015, je me suis rendu compte qu'il y a de nombreuses transactions différentes, dans une même transaction (même « id_mutation »). C'est-à-dire qu'il y a des personnes ou des entreprises qui ont acheté plusieurs maisons ou appartements à la fois pour investir, et la valeur de la colonne « valeur_fonciere » est la somme de tous les logements de la transaction (exemple : achat d'un immeuble complet comprenant x appartements).

Ces données ne sont pas pertinentes dans la carte du projet. Je ne suis intéressé que par les données relatives aux acheteurs uniques d'un appartement ou d'une maison. Je vais donc supprimer les achats de plusieurs biens en même temps.

Pour supprimer les transactions comportant plus d'un logement, je vais les regrouper en fonction de l'identifiant de transaction, puis je compterai le nombre de transactions avec le même identifiant, puis supprimerai les données avec les identifiants en double. Je fais cela en utilisant la fonction « groupby » de Pandas.

Après avoir supprimé les lignes de données avec des valeurs en double sur l'identifiant (on a en total 12361 lignes), les données restantes sont de 32765 lignes. Après chaque étape de préparation des données pour le modèle, on constate que la quantité de données a grandement diminué. Cependant, ce

processus doit être répété jusqu'à ce que nous ne trouvions plus de points aberrants dans les données.

Après avoir répété à nouveau le processus ci-dessus, en observant les données de plus près, j'ai continué à trouver des anomalies, comme pour le 1er novembre 2018.

id_mutation	date_mutation	nature_mutation	valeur_fonciere	nature_culture	code_postal	type_local	surface_reelle_bati	surface_terrain	nombre_pieces	longitude	latitude
2018-656200	1/11/2018	Vente	21082166	sols	69009	Maison	97	705	3	4.803311	45.78518
2018-656210	1/11/2018	Vente	313550		69004	Appartement	68		3	4.835794	45.77609
2018-656220	1/11/2018	Vente	285000		69005	Appartement	97		5	4.80453	45.75482
2018-656245	1/11/2018	Vente	238500		69004	Appartement	50		2	4.834539	45.77829
2018-656321	1/11/2018	Vente	135750		69009	Appartement	51		2	4.792124	45.76533
2018-656425	1/11/2018	Vente	230000		69002	Appartement	64		3	4.823029	45.74237
2018-656475	1/11/2018	Vente	164000		69002	Appartement	31		1	4.835128	45.76199
2018-656508	1/11/2018	Vente	286350		69005	Appartement	82		3	4.791286	45.75557
2018-657235	1/11/2018	Vente	77000		69008	Appartement	24		1	4.877237	45.72805
2018-657239	1/11/2018	Vente	683000		69008	Appartement	72		3	4.877237	45.72805

Figure 18 : Table des données du 1 novembre 2018

En regardant la figure 18, sur la première ligne de données, la valeur d'une maison à Lyon monte à 21 millions d'euros, ce qui est très élevé donc je veux savoir pourquoi.

En regardant la colonne « surface_terrain », nous pouvons deviner la raison, la surface du terrain de cette maison est de 705 mètres carrés, alors que la surface réelle de la maison n'est que de 97 mètres carrés. Pour les maisons avec une grande surface du terrain, j'ai également choisi de les supprimer de mes données. Je ne garderai que des maisons dont la surface du terrain est inférieure à 400 mètres carrés. J'ai pris une telle décision pour les raisons suivantes :

- Ce n'est pas un besoin du client, car dans ce projet, le client ne s'intéresse qu'aux maisons avec une superficie de terrain de moins de 400 mètres carrés.
- J'ai pris toutes les données et j'ai constaté qu'il y avait 285 logements avec une surface de terrain de plus de 400 mètres carrés, cela ne représente que 0.8% du total des données. En théorie, si nous supprimons moins de 3% des données, cela n'affectera pas le résultat du modèle.

Après avoir nettoyé les données une fois de plus, j'ai revu les données, je n'ai plus trouvé l'anomalie (figure 19).

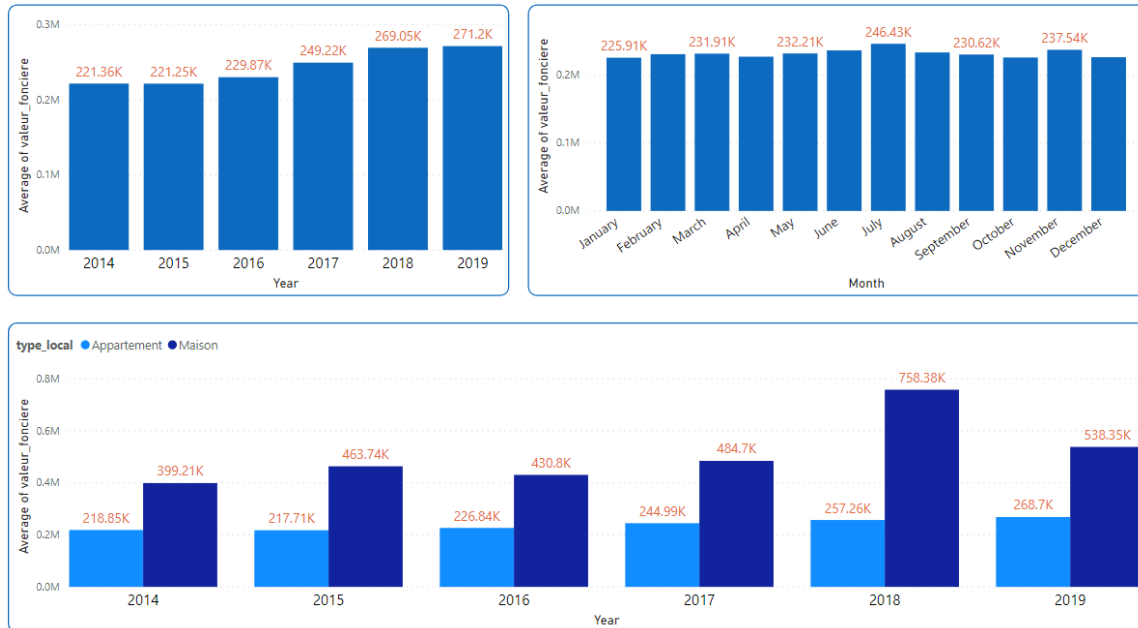


Figure 19 : Graphique du prix moyen des logements de 2014 à 2019 avec des données nettoyées

Cependant, ci-dessus nous voyons juste la valeur moyenne du prix, lors du calcul des valeurs min et max, j'ai toujours trouvé des points déraisonnables. Voir la figure 20.



Figure 20 : Prix min, moyen et max des logements de 2014 à 2019 avec des données nettoyées

La moyenne semble, quant à elle, raisonnable. Cependant, pour les valeurs min et max, je dois trouver l'origine de ces valeurs anormales. Un logement ne peut pas coûter 1 euro et une maison de plus de 21 millions d'euros à Lyon est en effet trop cher.

id_mutation	date_mutation	nature_mutation	valeur_fonciere	code_postal	type_local	surface_reelle_bati	surface_terrain	nombre_pieces_principales	longitude	latitude
2014-703005	2/21/2014	Vente	1	69002	Appartement	50		2	4.83373	45.75957
2014-703611	3/31/2014	Vente	1	69009	Appartement	36		0	4.804817	45.77799
2014-703865	3/19/2014	Vente	1	69002	Appartement	150		5	4.83373	45.75957
2014-704472	6/24/2014	Vente	1	69002	Appartement	133		5	4.830739	45.75471
2014-705436	7/30/2014	Vente	1	69002	Appartement	68		2	4.83373	45.75957
2014-705437	7/29/2014	Vente	1	69002	Appartement	78		4	4.83373	45.75957
2014-705848	7/22/2014	Vente	1	69002	Appartement	180		4	4.83373	45.75957
2014-706207	9/9/2014	Vente	1	69001	Appartement	70		2	4.829217	45.7685
2014-706378	9/22/2014	Vente	1	69009	Appartement	63		2	4.808037	45.77427
2014-712766	11/25/2014	Vente	1	69006	Appartement	16		1	4.856024	45.76771
2014-713191	10/27/2014	Vente	1	69006	Appartement	106		4	4.842963	45.76691
2014-713193	11/12/2014	Vente	1	69006	Appartement	29		2	4.842963	45.76691
2015-763808	4/9/2015	Vente	1	69001	Appartement	108		4	4.834426	45.77115
2015-765993	7/20/2015	Vente	1	69009	Appartement	66		3	4.809667	45.77277

Figure 21 : Extraction de la table des données avec les prix des logements les plus bas

id_mutation	date_mutation	nature_mutation	valeur_fonciere	nature_culture	code_postal	type_local	surface_reelle_bati	surface_terrain	nombre_pieces_principales	longitude	latitude
2017-939514	11/17/2017	L'État futur d'ach.	21018000	sols	69007	Appartement	130	286	4		
2014-705545	8/1/2014	Vente	5256000		69005	Appartement	78		3	4.823028	45.75981
2017-932542	12/21/2017	Vente	3900000		69009	Appartement	74		3	4.800131	45.76945
2015-769439	3/16/2015	Vente	2395150		69006	Appartement	237		4	4.850076	45.77431
2017-929999	9/4/2017	Vente	2350000		69002	Appartement	221		6	4.827699	45.75695
2019-277795	3/6/2019	Vente	2350000	sols	69004	Appartement	75	265	3	4.838295	45.77459
2014-703363	3/17/2014	Vente	2139100		69002	Appartement	308		7	4.828073	45.75754
2014-709002	3/28/2014	Vente	2000000		69006	Appartement	407		9	4.843382	45.77233
2018-657487	11/8/2018	Vente	1985000	sols	69007	Maison	30	276	2	4.840473	45.74123
2015-762521	2/10/2015	Vente	1840000		69002	Appartement	854		0	4.834346	45.76138
2018-656925	4/30/2018	Vente	1800000	sols	69004	Maison	169	396	7	4.821917	45.77597
2018-656510	1/18/2018	Vente	1740000		69002	Appartement	250		7	4.826608	45.75583
2019-278423	5/21/2019	Vente	1738850		69002	Appartement	178		4	4.832153	45.75346
2017-937603	9/6/2017	Vente	1711900		69006	Appartement	330		9	4.843815	45.77055
2017-937761	9/1/2017	Vente	1658360		69006	Appartement	280		7	4.841954	45.76451
2017-936649	7/10/2017	Vente	1632000	sols	69006	Maison	80	159	3	4.843788	45.77277

Figure 22 : Extraction la table des données avec les prix des logements plus hauts

En regardant le tableau de données avec les valeurs maximales et minimales du logement, je ne peux pas expliquer pourquoi elles sont si erronées. Peut-être que pendant le processus de saisie des données des erreurs ont été commises ?

Pour résoudre cette erreur, je vais m'appuyer sur la théorie selon laquelle la suppression de 3% des données n'affectera pas les résultats du modèle. Je vais nettoyer la base pour supprimer environ 3% des valeurs sur les plus élevées et les plus basses.

Après calcul, j'ai trouvé une fourchette de valeur pour savoir quelles données garder ou non. Je vais garder les données qui sont inférieures à 5 millions d'euros et supérieures à 50 mille euros comme prix d'un logement. Lorsque je conserve les données correspondant à cette plage, c'est-à-dire que

j'enlève 513 lignes des données déraisonnables, cela ne supprime que 1,56% des données. Je suis donc toujours sous le seuil des 3%.



Figure 23 : Prix min, moyen et max des logements de 2014 à 2019 avec des données propres

Après avoir appliqué les techniques de nettoyage des données, nous avons enfin de données pertinentes pour le modèle. Je continue de faire quelques visuels sur ces données propres afin de passer en revue toutes les caractéristiques de base des prix des logements lyonnais sur 6 ans (2014 à 2019).

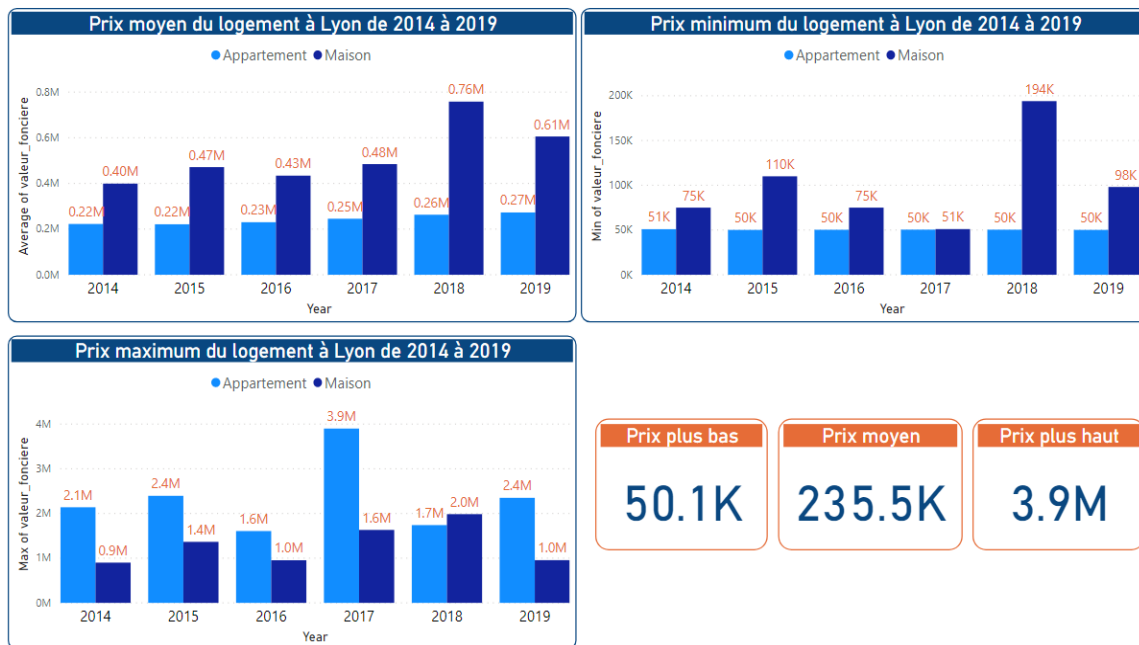


Figure 24 : Graphique le prix moyen, min, max des logements de 2014 à 2019 avec les données nettoyées

Dans le graphique de la figure 24, on voit que le prix moyen d'un appartement évolue peu dans le temps, le prix moyen d'une maison évolue légèrement plus que l'appartement, notamment en 2018, le prix moyen d'une maison est la plus élevée (760 milles euros pour une maison).

Concernant le prix de l'appartement le plus bas, il n'y a pas de changement de 2014 à 2019, mais pour le prix de la maison le plus bas, on assiste à un changement soudain en 2018, puisqu'il faut au moins 200000 euros pour acheter une maison, alors que pour les autres années avec seulement 75 milles euros il était possible d'acheter une maison.

En regardant les logements aux prix les plus élevés, nous avons une image complètement différente. Ici, les prix des appartements et des maisons connaissent de fortes fluctuations. Notamment en 2017, le prix d'un appartement pouvait atteindre près de 4 millions d'euros, alors qu'en 2018 il n'atteignait que près de deux millions d'euros. En 2019, le prix de la maison la plus élevée n'est que de 1 million d'euros.

Avec les deux graphiques ci-dessus sur les prix moyens et les prix les plus bas, on constate que, en général, le prix de la maison est toujours supérieur au prix de l'appartement. Cela signifie que pour acheter une maison, nous avons besoin de plus d'argent que pour acheter un appartement. Cependant, le logement le plus cher est un appartement et non une maison. Toutes ces caractéristiques sont maintenues sur les 6 années.

Ce qui suit est une autre analyse sur les prix des logements à Lyon à travers l'arrondissement, le nombre de pièces principales et le mois (figure 25).

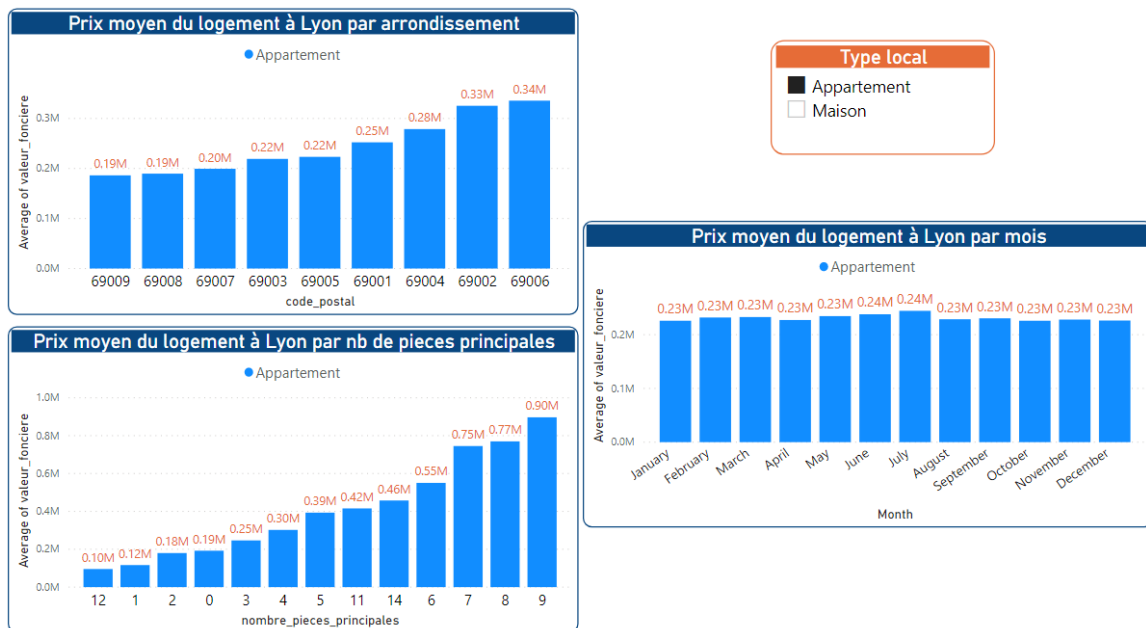


Figure 25 : Graphiques du prix moyen des appartements par arrondissement, du nombre de pièces principales et du mois

Sur la figure 25, on voit que le prix moyen d'un appartement à Lyon 6 est le plus élevé et que Lyon 9 est le plus bas. Le prix moyen d'un appartement dans le centre-ville de Lyon (Lyon 69001) est légèrement supérieur à la moyenne.

En ce qui concerne le nombre de pièces principales, normalement, plus il y a de pièces principales, plus l'appartement est cher. Mais à Lyon, ce n'est pas vrai. Les appartements de 6 à 9 pièces ont des prix élevés, mais pour les appartements de plus de 9 pièces, les prix des appartements sont souvent moins élevés.

En ce qui concerne la période de l'année, avec le graphique mensuel des prix des logements, nous ne voyons pas trop de différence entre les mois de l'année, ce qui signifie que les gens peuvent souhaiter acheter des appartements à tout moment, en toute saison.

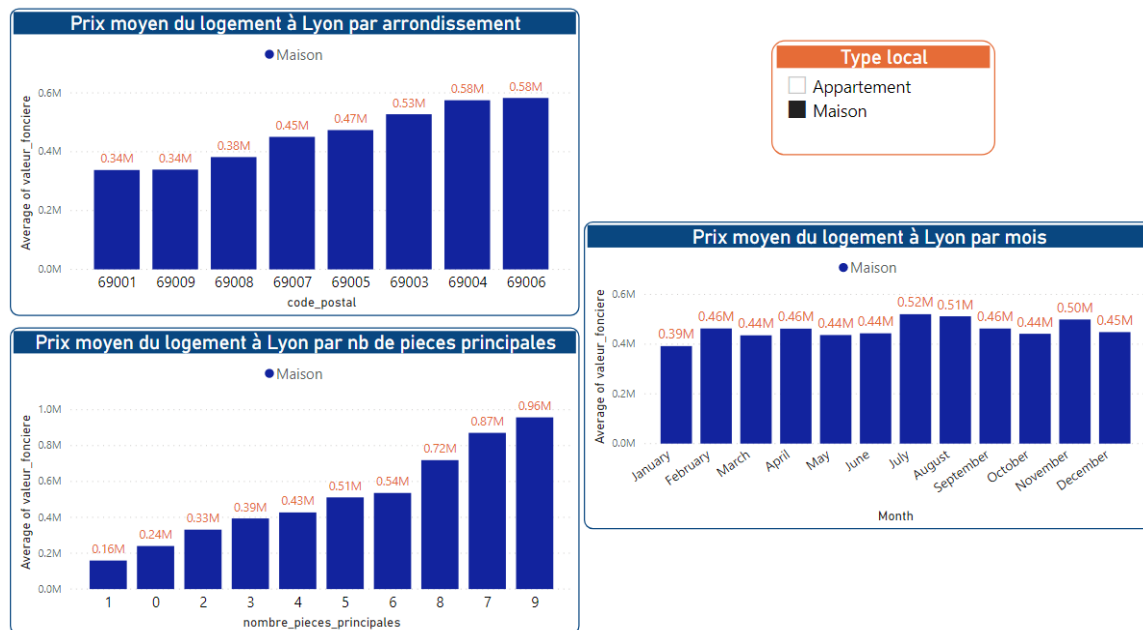


Figure 26 : Graphiques du prix moyen des maisons par arrondissement, du nombre de pièces principales et du mois

Pour le prix moyen des maisons, les prix sont toujours les plus élevés à Lyon 6, et les plus bas à Lyon 9, mais dans le centre de Lyon (Lyon 69001), le prix moyen des logements est aussi plus bas (le même prix qu'à Lyon 9).

Il n'y a pas beaucoup de différence de prix par rapport au mois de l'année. Cette propriété est similaire aux appartements.

Pour visualiser les données, nous avons de nombreuses façons différentes, et voici une autre façon d'afficher le prix moyen du logement à Lyon par arrondissement.

À gauche de la figure 27, nous voyons différentes tailles de bulles qui correspondent aux prix moyens des logements dans différents quartiers de Lyon. Plus la bulle est grande, plus le prix moyen des logements dans ce quartier est élevé. Donc, ici, l'arrondissement 6 et l'arrondissement 2 ont le prix moyen de logement le plus élevé, tandis que le prix moyen le plus bas se situe dans les arrondissements 7, 8 et 9.

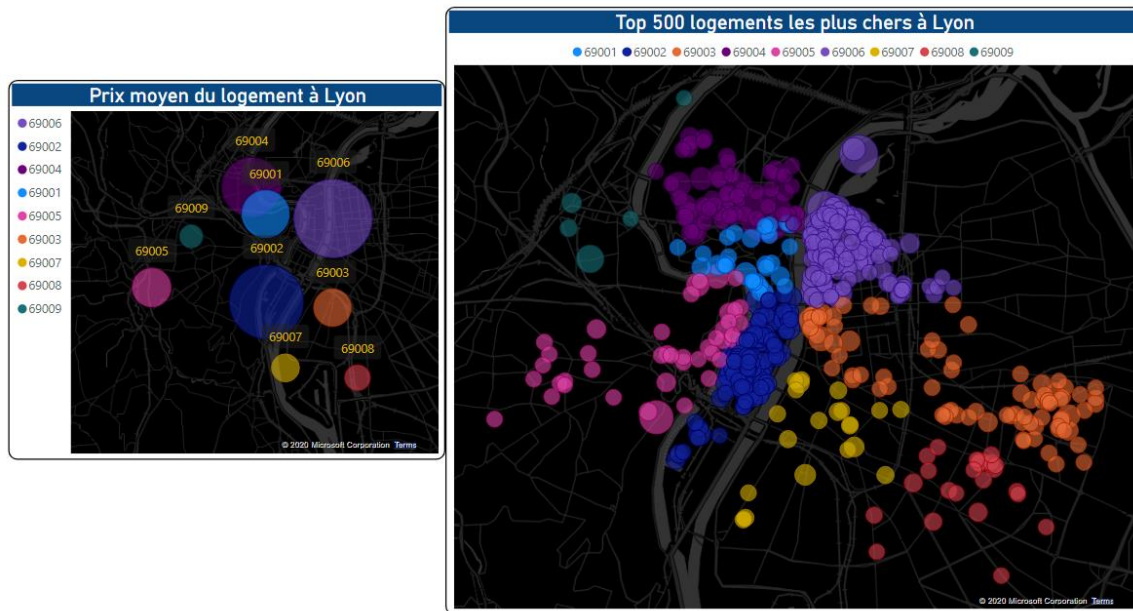


Figure 27 : Carte du prix des logements à Lyon par arrondissement

À droite de la figure 27 se trouve également une carte avec des bulles de différentes couleurs et tailles. Ici, cependant, je ne montre que le top 500 des logements les plus chers à Lyon. De là, je peux voir que les logements à prix élevé sont concentrés dans les arrondissements 2 et 6 où il y a une bulle dense, et sont moins concentrés dans les arrondissements 7, 8 et 9 où il y a une densité de bulles plus faibles.

- **Transformer**

Chaque étape du processus de préparation des données est axée sur l'objectif de fournir aux modèles de données pertinentes pour obtenir de bonnes informations. Les données dans le monde réel peuvent être très compliquées et dans la plupart des cas, un nettoyage des données doit être effectué avant toute analyse de données. Transformer est aussi une sorte de nettoyage.

Le nettoyage des données en général et la transformation en particulier n'est pas une tâche facile. Sans les bonnes techniques, la transformation des données prend du temps et est fastidieuse. Néanmoins, il s'agit d'une étape critique qui garantit une qualité maximale des données et ce afin d'augmenter

la précision des prévisions. Voici quelques techniques clés de la transformation :

- Supprimer les colonnes inutilisées et répétées (1)
- Utilisation des types de données corrects (2)
- Gérer les données manquantes (3)
- Convertir des données catégorielles en données numériques (4)
- Convertir les horodatages (5)

Pour ce projet, j'ai utilisé les techniques 1, 3 et 4.

La technique 1 : pour supprimer les colonnes inutilisées et répétées, nous devons d'abord afficher toutes les colonnes de l'ensemble de données en imprimant seulement 3 lignes de l'ensemble de données en utilisant la fonction « `head()` » de la librairie « Pandas » de Python:

```
raw_data.head(3)
```

	id_mutation	date_mutation	nature_mutation	valeur_fonciere	nature_culture	adresse_nom_voie	code_postal	code_commune	nom_commune
0	2014-702373	1/6/2014	Vente	170000.0	NaN	RUE JACQUES - LOUIS HENON	69004	69384	Lyon 4e Arrondissement
1	2014-702374	1/9/2014	Vente	220000.0	NaN	RUE JACQUARD	69004	69384	Lyon 4e Arrondissement
2	2014-702375	1/3/2014	Vente	164000.0	NaN	RUE GEORGES MARTIN WITKOWSKI	69005	69385	Lyon 5e Arrondissement

code_type_local	type_local	surface_reelle_bati	surface_terrain	nombre_pieces_principales	longitude	latitude
2	Appartement	66	NaN	4	4.816632	45.779399
2	Appartement	60	NaN	3	4.822497	45.776321
2	Appartement	66	NaN	4	4.787471	45.747831

Souvent, nous supprimons les colonnes qui ne semblent pas liées au résultat que nous voulons atteindre. Pour cette raison, je supprimerai la colonne « `id_mutation` ». Nous supprimons aussi les colonnes avec des informations dupliquées. Je supprimerai donc les colonnes « `adresse_nom_voie` », « `code_commune` », « `nom_commune` » car elles sont similaires à la colonne « `code_postal` », et je supprimerai la colonne « `code_type_local` » car elle est similaire à la colonne « `type_local` ».

Pour supprimer les colonnes ci-dessus, j'utilise la fonction « drop() » de Pandas.

```
data = raw_data.drop(['id_mutation', 'adresse_nom_voie', 'code_commune',  
                      'nom_commune', 'code_type_local'], axis = 1)
```

Pour vérifier les colonnes restantes ainsi que pour vérifier que les colonnes voulu sont supprimées, j'utilise la fonction « columns » de Pandas :

```
data.columns  
  
Index(['date_mutation', 'nature_mutation', 'valeur_fonciere', 'nature_culture',  
       'code_postal', 'type_local', 'surface_reelle_bati', 'surface_terrain',  
       'nombre_pieces_principales', 'longitude', 'latitude'],  
      dtype='object')
```

Le nombre de colonnes restantes est de 11 colonnes et le nombre de lignes est de 31924.

```
data.shape  
  
(31924, 11)
```

La technique 3 (gérer les données manquantes) est la résolution de données incomplètes qui peuvent varier en fonction de l'ensemble de données. Si une valeur est manquante, on peut envisager l'imputation, le processus de remplacement de la valeur manquante par un simple espace réservé ou une autre valeur par exemple la valeur moyenne. Si l'ensemble de données est suffisamment volumineux, on peut probablement supprimer les données sans entraîner une perte substantielle de la puissance statistique.

Pour savoir le nombre total de valeurs manquantes par colonne, j'utilise 2 fonctions « isnull() » et « sum() ».

```
data.isnull().sum()

date_mutation          0
nature_mutation        0
valeur_fonciere        0
nature_culture         31551
code_postal            0
type_local             0
surface_reelle_bati    0
surface_terrain        31551
nombre_pieces_principales 0
longitude              79
latitude               79
dtype: int64
```

De là, nous voyons que 4 colonnes manquent de données. Les deux colonnes « nature_culture » et « surface_terrain » ont des données souvent manquantes (31551 lignes), ce qui représente 98,8% des données. Avec une telle quantité d'absence de données, il était très difficile pour moi de trouver une valeur de remplacement, j'ai donc décidé de supprimer ces deux colonnes. Les données de deux colonnes « longitude » et « latitude » manquent 79 fois, soit 0.2% des données, et je peux donc supprimer ces lignes. Pour supprimer toutes les valeurs manquantes, j'ai utilisé la fonction « dropna() » de Pandas :

```
data_no_mv = data.dropna(axis=0)
```

Pour vérifier s'il manque encore des données ou non, j'ai réutilisé la fonction « isnull() » et « sum() » :

```
data_no_mv.isnull().sum()

date_mutation          0
nature_mutation        0
valeur_fonciere        0
code_postal            0
type_local             0
surface_reelle_bati    0
nombre_pieces_principales 0
longitude              0
latitude               0
dtype: int64
```


J'ai donc un nouvel ensemble de données composé de 31845 lignes et 9 colonnes.

```
data_no_mv.shape
```

```
(31845, 9)
```

La technique 4 : convertir des données catégoriques en données numériques. De nombreux modèles de ML nécessitent que les données catégorielles soient au format numérique, ce qui nécessite la conversion de valeurs telles que oui ou non en 1 ou 0, car si on essaie d'entraîner le modèle avec des données catégorielles, on obtiendra immédiatement une erreur.

Les données catégoriques sont simplement des informations agrégées en groupes, par exemple le sexe, le niveau d'éducation, le type de logement...

Dans ce projet, on peut voir que les colonnes « nature_mutation », « code_postal » et « type_local » sont des colonnes de données catégorielles. Pour transformer les 3 colonnes en des données numériques, on a d'abord besoin de connaître la liste des valeurs uniques pour chaque colonne. Pour faire ça, j'ai utilisé la méthode « unique » de Numpy :

```
nature_mutation = np.unique(data_no_mv['nature_mutation'])
nature_mutation|
array(['Vente', "Vente en l'état futur d'achèvement",
       'Vente terrain à bâtir'], dtype=object)
```

Pour la colonne « nature_mutation », nous avons 3 valeurs uniques, correspondant à 3 types de vente : vente, vente en l'état futur d'achèvement et vente terrain à bâtir.

Je fais de même pour les deux autres colonnes, et j'obtiens le résultat suivant :

```
code_postal = np.unique(data_no_mv['code_postal'])
code_postal
```

```
array([69001, 69002, 69003, 69004, 69005, 69006, 69007, 69008, 69009],
      dtype=int64)
```

```
type_local = np.unique(data_no_mv['type_local'])
type_local
```

```
array(['Appartement', 'Maison'], dtype=object)
```

Ainsi pour les codes postaux, nous avons 9 arrondissements différents (de 69001 à 69009), et pour le type de logement nous avons 2 types, qui sont appartement et maison.

Nous pouvons maintenant générer un schéma de « label encoding » pour lier chaque catégorie à une valeur numérique en utilisant les méthodes de la librairie scikit-learn.

```
from sklearn.preprocessing import LabelEncoder
nature = LabelEncoder()
nature_mutation_labels = nature.fit_transform(data_no_mv['nature_mutation'])
nature_mutation_mappings = {index: label for index, label in
                             enumerate(nature.classes_)}
nature_mutation_mappings
```

```
{0: 'Vente',
 1: "Vente en l'état futur d'achèvement",
 2: 'Vente terrain à bâtir'}
```

Par conséquent, un schéma de mapping a été créé dans lequel chaque valeur de catégories est liée à un nombre à l'aide de l'objet LabelEncoder. Les « label » transformées sont stockées dans la variable « nature_mutation_labels ».

Et maintenant, les valeurs « vente », « vente en l'état futur d'achèvement » et « vente terrain à bâtir » sont convertis en valeurs numériques 0, 1 et 2.

J'ai fait la même chose pour les deux autres colonnes : « code_postal » et « type_local » :

```
code_postal_mappings
```

```
{0: 69001,
 1: 69002,
 2: 69003,
 3: 69004,
 4: 69005,
 5: 69006,
 6: 69007,
 7: 69008,
 8: 69009}
```

```
type_local_mappings
```

```
{0: 'Appartement', 1: 'Maison'}
```

Maintenant, je vais ajouter les données converties ci-dessus dans mes données, puis j'affiche les 5 premières lignes de données avec les colonnes de données que je viens d'ajouter :

```
data_no_mv['nature_mutation_label'] = nature_mutation_labels
data_no_mv['code_postal_label'] = code_postal_labels
data_no_mv['type_local_label'] = type_local_labels
data_no_mv[['nature_mutation', 'nature_mutation_label',
            'code_postal', 'code_postal_label',
            'type_local', 'type_local_label']].iloc[0:5]
```

	nature_mutation	nature_mutation_label	code_postal	code_postal_label	type_local	type_local_label
0	Vente	0	69004	3	Appartement	0
1	Vente	0	69004	3	Appartement	0
2	Vente	0	69005	4	Appartement	0
3	Vente	0	69009	8	Appartement	0
4	Vente	0	69004	3	Appartement	0

Cependant, si nous fournissons directement les propriétés « nature_mutation_label », « code_postal_label » et « type_local_label » en tant que fonctionnalité dans le modèle de ML, cela les traitera comme des données numériques continue. Par exemple, pour l'algorithme la valeur 8 (qui correspond à Lyon 69009) est supérieure à 4 (Lyon 69005), mais cela n'a pas de sens car Lyon 9 n'est certainement ni plus grand ni plus petit que Lyon 5, qui sont essentiellement des valeurs ou des genres différents qui ne peuvent

être directement comparés. Par conséquent, nous avons besoin d'une classe de schéma de codage supplémentaire dans laquelle des fonctionnalités factices (colonnes) sont créées pour chaque valeur ou catégorie unique. Le schéma « one-hot encoding » nous aidera à le faire. Le schéma « one-hot encoding » transforme l'attribut en « *m* » entités binaires (colonnes) qui ne peuvent contenir qu'une valeur de 1 ou 0. Chaque observation dans l'entité catégorielle est ainsi convertie en un vecteur de taille « *m* » avec une seule des valeurs comme 1 (l'indiquant comme active).

```
# encode generation labels using one-hot encoding scheme
nature_ohe = OneHotEncoder()
nature_feature_arr = nature_ohe.fit_transform(
    data_no_mv[['nature_mutation']]).toarray()
nature_feature_labels = list(nature.classes_)
nature_features = pd.DataFrame(nature_feature_arr,
                               columns=nature_feature_labels)
nature_features
```

	Vente	Vente en l'état futur d'achèvement	Vente terrain à bâtir
0	1.0	0.0	0.0
1	1.0	0.0	0.0
2	1.0	0.0	0.0
3	1.0	0.0	0.0
4	1.0	0.0	0.0

On peut voir que la colonne « nature_mutation » a 3 valeurs uniques et elle est divisée en 3 colonnes. La colonne « type_local » a 2 valeurs uniques, elle est donc séparée en 2 colonnes. L'état actif d'une catégorie est indiqué par la valeur 1.

	Appartement	Maison
0	1.0	0.0
1	1.0	0.0
2	1.0	0.0
3	1.0	0.0
4	1.0	0.0

Après avoir terminé « one-hot encoding », je vais concaténer tous les attributs ci-dessus ensemble dans une donnée finale.

```
housing_df_ohe = pd.concat([data_no_mv, nature_features, code_postal_features, type_local_features], axis=1)
housing_df_ohe.head()
```

	date_mutation	nature_mutation	valeur_fonciere	code_postal	type_local	surface_reelle_bati	nombre_pieces_principales	longitude	latitude
0	1/6/2014	Vente	170000.0	69004.0	Appartement	66.0	4.0	4.816632	45.779399
1	1/9/2014	Vente	220000.0	69004.0	Appartement	60.0	3.0	4.822497	45.776321
2	1/3/2014	Vente	164000.0	69005.0	Appartement	66.0	4.0	4.787471	45.747831
3	1/3/2014	Vente	126000.0	69009.0	Appartement	60.0	2.0	4.801490	45.770254
4	1/13/2014	Vente	155120.0	69004.0	Appartement	40.0	2.0	4.835525	45.777301

nature_mutation_label	...	69002	69003	69004	69005	69006	69007	69008	69009	Appartement	Maison
0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0
0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0

On peut également appliquer facilement le schéma « one-hot-encoding » en utilisant la fonction « to_dummies () » de Pandas.

```
nature_onehot_features = pd.get_dummies(data_no_mv['nature_mutation'])
nature_onehot_features
```

	Vente	Vente en l'état futur d'achèvement	Vente terrain à bâtir
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0

Utiliser la fonction « `to_dummies()` » de Pandas est similaire à utiliser « `OneHotEncoder` » de `sklearn`, sauf que dans le cas « `to_dummies()` », lorsqu'elle est appliquée sur un classificateur avec m étiquettes séparées, nous pouvons obtenir « $m - 1$ » objet binaire en définissant le paramètre « `drop_first = True` » (par défaut à `False`).

Il est extrêmement important de supprimer un des « dummies » car avec « $m - 1$ » dummies, on peut prédire complètement l'information finale, car un seul et unique dummy peut être à 1 à la fois. Donc si tous les « $m - 1$ » dummies sont à 0 cela veut dire que le dummy m est à 1. Nous pouvons donc deviner la valeur de cette colonne.

De plus, si nous gardons toutes les « encoded features », nous faisons apprendre à notre modèle un poids supplémentaire qui n'est pas vraiment nécessaire. Cela consomme de la puissance et du temps de calcul. Cela donne également un objectif d'optimisation qui pourrait ne pas être très raisonnable, puisque l'algorithme va chercher à optimiser des données inutiles.

Pour ces raisons, pour chaque donnée « one hot encoder », je supprime une « encoded feature » :

```
code_postal_onehot_features = pd.get_dummies(data_no_mv['code_postal'], drop_first = True)
code_postal_onehot_features
```

	69002	69003	69004	69005	69006	69007	69008	69009
0	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0
2	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	1
4	0	0	1	0	0	0	0	0
...
31919	0	1	0	0	0	0	0	0
31920	0	0	0	0	0	0	1	0
31921	0	0	0	0	0	1	0	0

• Enrichir

Comme toutes les autres techniques de préparation des données, l'enrichissement des données est aussi une importante technique qui contribue à améliorer la qualité des données.

Il existe deux façons principales d'enrichir les données :

- Collecter plus de données à partir d'autres sources
- À partir des données d'origine, procéder à des calculs pour ajouter de nouvelles données

Dans ce projet, n'ayant pas d'autre sources à ma disposition, je choisirai la deuxième option, j'ajouterai une colonne de données sur le prix par mètre carré « prix/m² » en divisant le prix du logement par la superficie réelle (valeur_fonciere/surface_reelle_bati).

```
data_with_dummies['prix/m2'] = data_with_dummies.valeur_fonciere/data_with_dummies.surface_reelle_bati
data_with_dummies.head()
```

ati	nombre_pieces_principales	longitude	latitude	nature_mutation_label	...	69002	69003	69004	69005	69006	69007	69008	69009	Maison	prix/m2
66	4	4.816632	45.779399	0 ...		0	0	1	0	0	0	0	0	0	2575.757576
60	3	4.822497	45.776321	0 ...		0	0	1	0	0	0	0	0	0	3666.666667
66	4	4.787471	45.747831	0 ...		0	0	0	1	0	0	0	0	0	2484.848485
60	2	4.801490	45.770254	0 ...		0	0	0	0	0	0	0	1	0	2100.000000
40	2	4.835525	45.777301	0 ...		0	0	1	0	0	0	0	0	0	3878.000000

- **Stocker**

Après avoir terminé toutes les étapes de préparation des données, nous pouvons stocker les données de différentes manières, aussi complexes que la création d'une base de données ou plus simplement en les enregistrant sous la forme de fichiers CSV.

Dans ce projet, je vais enregistrer dans des fichiers csv en utilisant la méthode « to_csv » de Pandas.

```
data_with_dummies.to_csv('prix_logement_lyon_netoye.csv')
```

2. MODELE DE REGRESSION PREDICTIVE

Comme mentionné précédemment, pour résoudre un problème de ML, nous avons 7 étapes principales, dont les 5 dernières sont le développement d'un modèle de données qui utilisent les données que j'ai préparé dans les deux étapes précédentes.

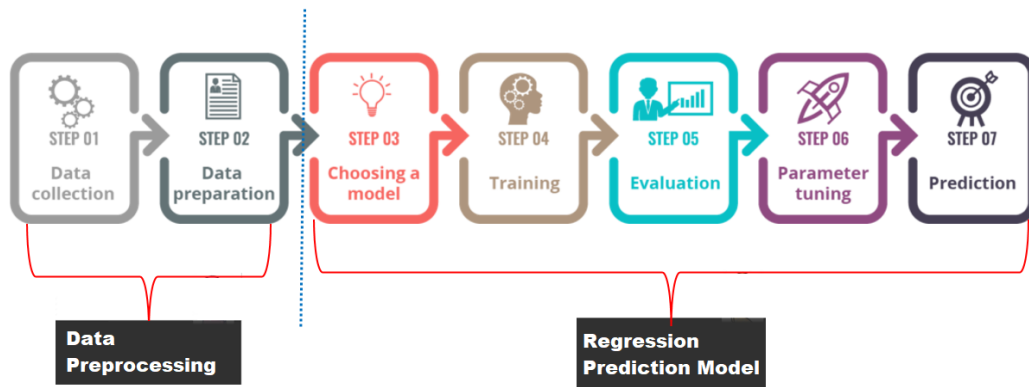


Figure 28 : Rappel sept étapes principales de ML

2.1 Choix d'un modèle

Avant de choisir un modèle pour le projet, je veux savoir comment les propriétés sont liées au prix du logement dans les données, car le prix du logement est la cible que je devrai prédire dans ce projet.

Pour ce faire, il y a un bon moyen de vérifier rapidement les corrélations entre les colonnes. Il existe de nombreuses façons de connaître les relations

entre les propriétés dans un ensemble de données. La visualisation de la matrice de corrélation sous forme de carte thermique (« heat-map ») est la façon la plus courante. Dans la « heat-map », plus la valeur est proche de 1 ou -1 et plus les deux propriétés sont liées. J'utilise la librairie « seaborn » pour créer la « heat-map ».

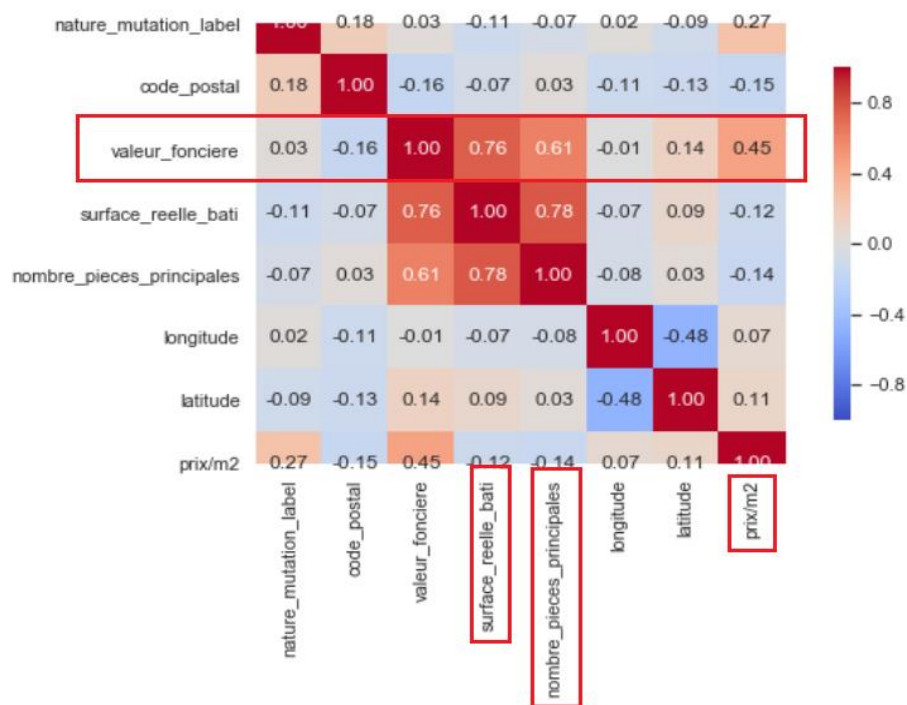


Figure 29 : Heat-map corrélations entre les colonnes

En regardant la figure 29, nous voyons que les 3 propriétés qui influencent le plus le prix du logement sont « surface_reelle_bati » (0.76), suivies de « nombre_pieces_principales » (0.61) et de « prix/m2 » (0.45).

À partir de là, je décide de créer d'abord un modèle simple, avec un seul facteur indépendant : « surface_reelle_bati ». L'algorithme que j'ai choisi pour ce modèle est également simple : la régression linéaire.

La raison de ce choix est que l'analyse de régression est l'une des méthodes de prédiction les plus utilisées. La régression linéaire est probablement la méthode de ML la plus fondamentale et un point de départ pour développer un modèle de prédiction. De plus, la régression linéaire est

une approximation linéaire d'une relation causale entre deux ou plusieurs variables. Les modèles de régression linéaire sont très importants, car ils constituent l'un des moyens les plus courants de faire des inférences et des prédictions.

La relation entre les variables y et x est représentée par cette équation :

$$\hat{y}_i = b_0 + b_1 x_i$$

À l'intérieur :

- \hat{y}_i : variable dépendante que nous essayons de prédire ou d'estimer
- x_i : variable indépendante que nous utilisons pour faire des prédictions
- b_0 : constante - également connu sous le nom d'interception y . Si x est égal à 0, y serait égal à b_0
- b_1 : coefficient - est la pente de la droite de régression - elle représente l'effet de x sur y (figure 30)

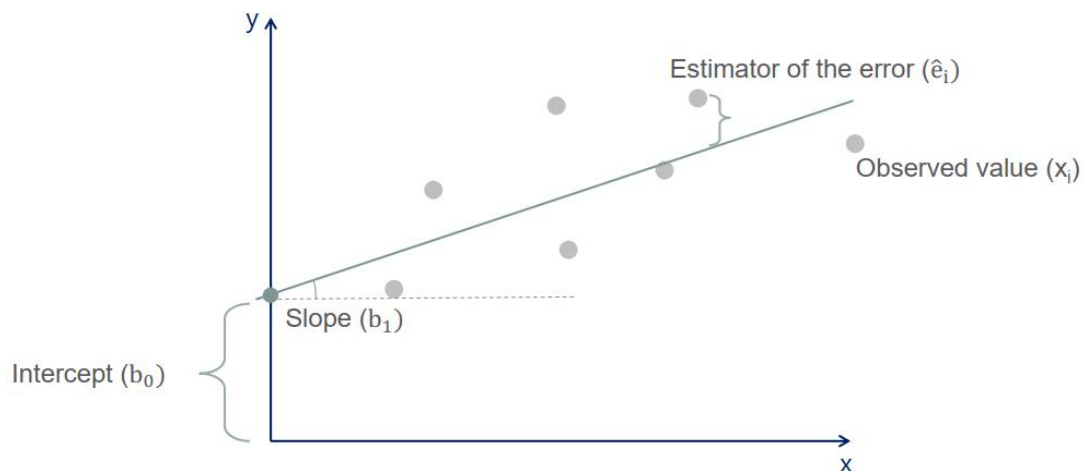


Figure 30 : Représentation géométrique de la régression linéaire

Comme mentionné ci-dessus, la cible (variable dépendante) de ce modèle linéaire à une variable est « valeur_fonciere » et la variable indépendante du modèle est « surface_reelle_bati ».

```
targets = data_lyon['valeur_fonciere']  
inputs = data_lyon['surface_reelle_bati']
```

Mais afin d'alimenter « inputs » à « sklearn », il doit s'agir d'un tableau 2D (une matrice), mais quand je vérifie la taille de « inputs », il n'est pas en 2D, je dois donc le transformer en 2D en utilisant reshape (-1,1).

```
inputs.shape  
(31845,)
```

```
inputs = inputs.values.reshape(-1,1)
```

```
inputs.shape  
(31845, 1)
```

2.2 Entraînement du modèle

Avant d'entraîner le modèle, je diviserai les données en deux parties, l'une pour l'entraînement du modèle (80%) et l'autre pour les tests (20%). Pour ce faire, je dois importer la fonction « train_test_split » de la librairie « sklearn » de Python.

```
# Import the module for the split  
from sklearn.model_selection import train_test_split  
x_train, x_test, y_train, y_test = train_test_split(inputs, targets, test_size=0.2, random_state=42)
```

Dans cette fonction, j'utilise le paramètre « random_state » pour m'assurer que chaque fois que j'exécute le modèle, j'obtiens le même ensemble de « train » et de « test ».

Je vérifie que les données ont été correctement divisées avec le taux train : test (80 :20) et que le format de données est bien une matrice 2D :

```
x_train.shape  
(25476, 1)
```

```
x_test.shape  
(6369, 1)
```

Une fois que les données d'entrée ont la taille et le format correct, je procède à la création du modèle en utilisant la fonction « LinearRegression » de « sklearn » :

```
from sklearn.linear_model import LinearRegression
```

Puis je crée un objet de régression linéaire, et j'utilise la fonction « fit () » pour « fit » un modèle linéaire.

```
reg = LinearRegression()  
reg.fit(x_train,y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Pour vérifier les sorties de la régression, je vais les stocker dans « y_hat » car c'est le nom théorique des prédictions :

```
y_hat = reg.predict(x_train)
```

2.3 Evaluation

La façon la plus simple d'évaluer un modèle est de comparer y_train (target) et y_hat (prediction). Pour le faire, je vais les tracer sur un graphique de type « scatter plot ». Plus les points sont proches de la bissectrice (45°), meilleure est la prédiction.

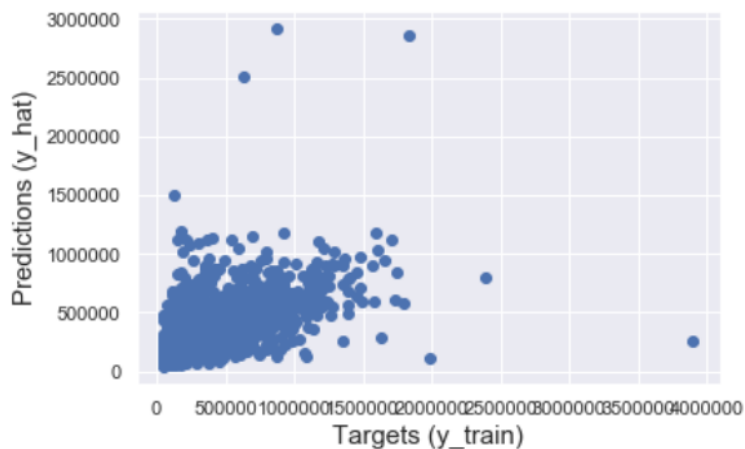


Figure 31 : Comparaison de la prédiction et de la « target »

En regardant la figure 31, nous voyons que le graphique suit approximativement la ligne à 45 degrés. Cependant, pour en savoir plus, il faut calculer le score du modèle (R-squared) :

```
reg.score(x_train,y_train)
```

0.5657010865899784

Pour mieux comprendre « R-squared », nous devons comprendre trois coefficients : la somme des carrés totale (SST), la somme des carrés de régression (SSR) et la somme des carrés d'erreur (SSE) (figure 32).

« R-squared » est calculé par le calcul SSR / SST . Cette valeur varie de 0 à 1.

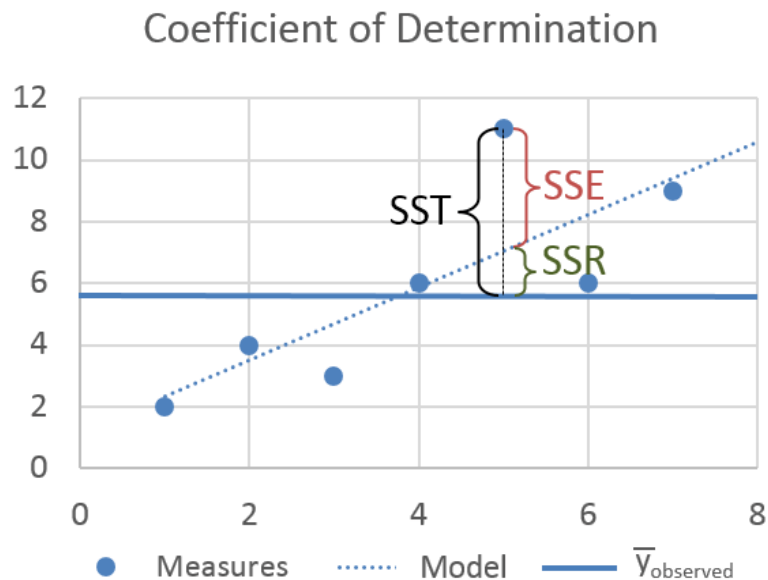


Figure 32 : Représentation géométrique de SST, SSE et SSR

Pour faire simple, « R-squared » est la qualité de la prédiction. Plus la valeur est proche de 1, plus le modèle est meilleur car les résultats de prédiction du modèle sont plus précis. Dans ce cas, nous obtenons la valeur 0,56, ce qui signifie que si nous utilisons ce modèle pour prédire les prix des logements, 56% des prédictions seront correctes.

Récupération des coefficients et de l'interception de la régression :

```
reg.intercept_  
12750.716545042262
```

```
reg.coef_  
array([3332.38845954])
```

On obtient l'équation de régression : $y_{\text{hat}} = 12750.72 + 3332.39x$

Avec des données « train », on peut calculer les valeurs de « y_{hat} » et on peut donc montrer cette ligne sur le graphique comme suit (figure 33 - ligne jaune).

```
plt.scatter(x_train,y_train)  
yhat = 12750.72 + 3332.39*x_train  
fig = plt.plot(x_train, yhat, lw=4, c='orange', label='regression line')  
plt.xlabel('surface réelle bati (m2)',size=15)  
plt.ylabel('Predictions du prix du logement (euros)',size=15)
```

```
Text(0, 0.5, 'Predictions du prix du logement (euros)')
```

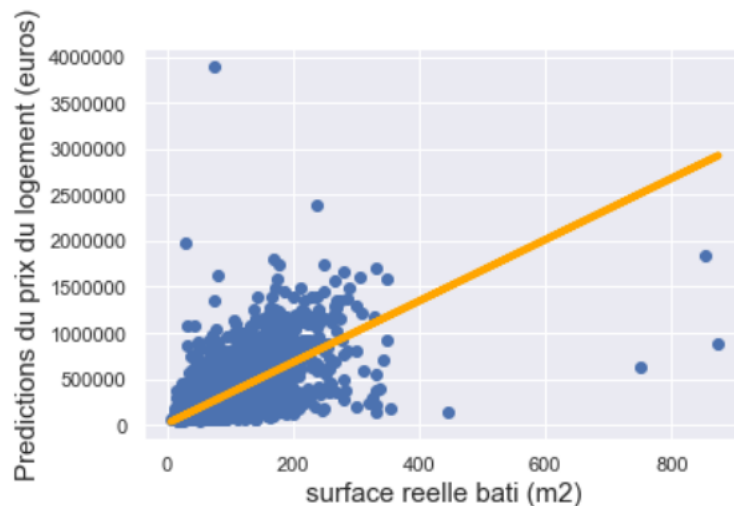


Figure 33 : Ligne de régression

Je vais essayer de prédire les prix des logements à Lyon en utilisant ce modèle avec la précision de prédiction est 0,56. L'input du modèle est « surface_reelle_bati ».

252682.69 prediction du prix du logement
= 252682.69 euros

2.4 Réglage des paramètres

Nous pouvons passer d'un paramètre à une fonction. Dans ce cas, un paramètre est un argument de la fonction qui peut avoir une valeur parmi une plage de valeurs. En ML, le modèle spécifique que nous utilisons est la fonction et elle nécessite des paramètres pour effectuer une prédiction sur de nouvelles données.

Il s'agit d'un processus qui nécessite beaucoup de temps, une compréhension approfondie de chaque paramètre de chaque modèle ainsi que la patience de tester les paramètres avec des valeurs différentes.

PAGE 46

3. AMELIORATION DU MODELE

L'optimisation ou l'amélioration de modèles est un travail important et essentiel pour parvenir à un modèle efficace. C'est aussi un travail qui demande beaucoup d'efforts, de temps et de patiences.

Pourquoi est-ce que je dis ça ? Nous avons beaucoup de façons pour améliorer un modèle : enrichir les données, nettoyer les données plus soigneusement, régler les paramètres du modèle, essayer avec un autre modèle ou algorithme ... et il est très difficile de savoir quelle voie est la meilleure. Nous devons prendre chacune des options une par une et essayer encore et encore. Parfois, nous essayons de nombreuses façons, mais toutes ne fonctionnent pas mieux que le premier résultat.

En général, nous pouvons dire que plus nous avons d'expérience et moins le processus d'optimisation du modèle nécessitera de temps et d'efforts.

Dans ce projet, j'ai également appliqué mon expérience acquise dans l'étape de préparation des données. J'ai essayé de nettoyer en profondeur les données, ainsi que d'enrichir les données, avant de créer le modèle de régression linéaire à une variable. J'aurais pu commencer directement à construire le modèle, mais avec mon expérience, je sais que si je le fais, cela sera une perte de temps car les résultats que j'obtiendrai ne fonctionneront évidemment pas correctement.

Cependant, je ne m'arrête pas là, je vais tester d'autres méthodes pour améliorer le modèle. Je sélectionne donc des modèles avec différents algorithmes pour optimiser le modèle tel que régression linéaire à variables multiples, arbres de décision, forêt aléatoire...

3.1 Régression linéaire à variables multiples

Dans la section ci-dessus, j'ai utilisé la régression linéaire simple ou la régression linéaire à variable unique qui est le cas le plus simple de régression linéaire avec une seule variable indépendante. Dans cette section, pour améliorer le modèle, j'utiliserai une régression linéaire multivariée, c'est-à-dire que l'entrée du modèle comprendra de nombreuses variables dépendantes.

Cependant, le choix de la variable indépendante du modèle n'est pas simple. Nous avons besoin de techniques pour savoir quelles propriétés sont nécessaires pour le modèle et celles à supprimer pour éviter les interférences.

Dans la partie ci-dessus, j'ai utilisé une « heat-map » pour trouver la propriété qui affecte le plus le prix du logement, c'est « surface_reelle_bati », et ensuite, je trouve également deux autres propriétés qui affectent le prix du logement, se sont « nombre_pieces_principales » et « prix/m2 ».

Pour être sûr, j'appliquerai une autre technique pour découvrir quelles propriétés ont vraiment un sens pour le modèle. C'est la méthode de calcul de la « p-value ».

La valeur P est un test statistique qui détermine la probabilité de résultats extrêmes du test d'hypothèse statistique, en prenant l'hypothèse nulle pour être correcte.

La valeur P fournit la probabilité du test d'hypothèse, donc dans un modèle de régression, la valeur P pour chaque variable indépendante teste l'hypothèse nulle selon laquelle il n'y a « pas de corrélation » entre la variable indépendante et la variable dépendante, cela aide également à déterminer que la relation observée dans l'échantillon existe également dans des données plus volumineuses.

Ainsi, si la valeur P est inférieure au niveau de signification (généralement 0,05), cela veut dire que la variable indépendante testée est liée à la variable dépendante et on peut donc utiliser cette variable indépendante pour construire notre modèle.

Pour calculer la valeur P, j'utilise la fonction « f_regression » de « sklearn ».

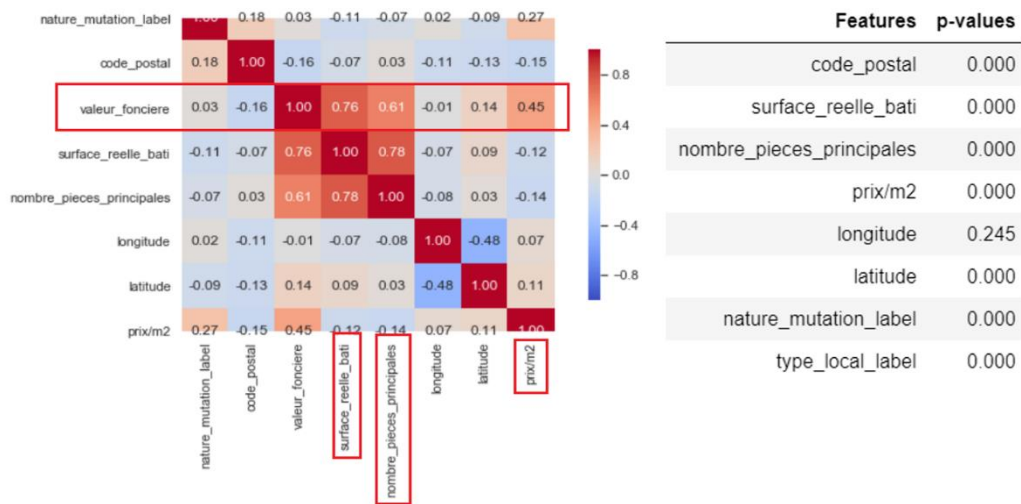


Figure 34 : Comparaison entre « heat-map » et « p-value »

En regardant la figure 34, avec la valeur p, nous voyons que la plupart des attributs sont significatifs pour le modèle, seule la propriété « longitude » est insignifiante car la valeur de « longitude » est supérieure à 0,05 et on doit l'enlever. Cependant, avec la carte thermique, nous voyons plus clairement l'influence des propriétés sur les prix des logements, je vais donc d'abord effectuer le modèle avec la méthode de régression linéaire multivariée avec les variables indépendantes « surface_reelle_bati » et « nombre_pieces_principales ». Puis, je vais ajouter « prix/m2 », puis j'expérimenterai d'autres variables.

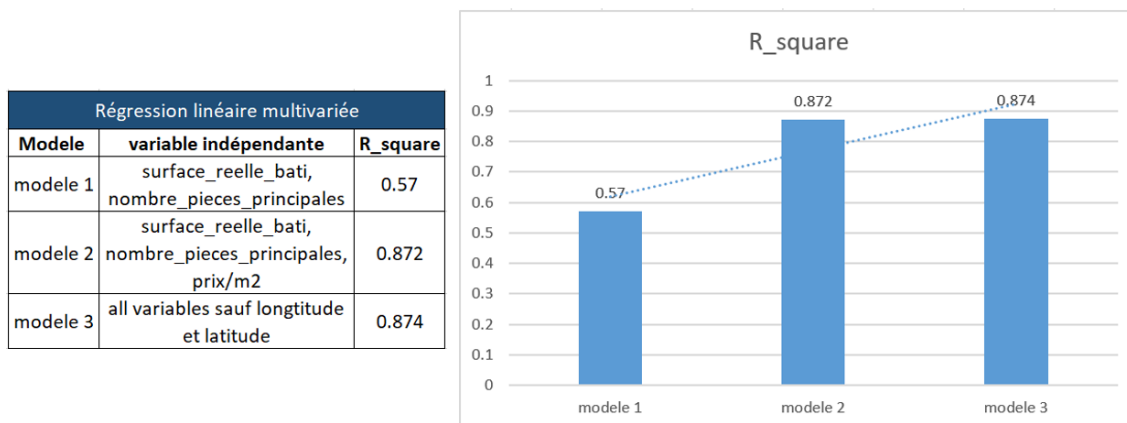


Figure 35 : Divers essais avec des modèles de régression linéaire multivariée

En regardant la figure 35, nous voyons que le modèle de régression linéaire multivariée donne de meilleurs résultats que le modèle de régression linéaire simple. Dans le modèle de régression linéaire simple, les résultats prédits du modèle ne sont que de 0,56, mais avec les modèles de régression linéaire multivariée, nous avons des résultats supérieurs : spécifiquement 0,57, 0,872, 0,874. Grâce à cela, nous voyons également qu'avec le même algorithme (régression linéaire multivariée), plus le nombre de variables indépendantes est grand, plus les résultats de prédiction sont précis. Dans le premier modèle, avec deux variables indépendantes, le résultat est de 0,57 (pas beaucoup plus élevé que dans le modèle de régression linéaire simple). Mais pour le 2ème modèle avec 3 variables indépendantes, les résultats subissent un changement clair (0,872). Lorsque je suis passé avec encore plus de variables indépendantes, j'ai obtenu un résultat légèrement supérieur à 0,874.

Donc, avec le même algorithme, en ne modifiant que le nombre de variables indépendantes, nous avons des résultats très différents. Donc, si on utilise des algorithmes différents, à quoi ressemblera le résultat du modèle ? Maintenant, je vais donc réaliser une modélisation avec d'autres algorithmes tels que les arbres de décision et la forêt aléatoire.

3.2 Arbre de décision

Pour réaliser une analyse de décision, un arbre de décision peut être utilisé pour représenter visuellement et explicitement les décisions et la prise de décision. Comme son nom l'indique, il utilise un modèle de décisions en forme d'arbre. Bien qu'il s'agisse d'un outil couramment utilisé dans l'exploration de données pour élaborer une stratégie pour atteindre un objectif particulier, il est également être utilisé en ML.

Pour mieux comprendre, considérons un exemple très basique qui utilise un ensemble de données issu du naufrage du Titanic pour prédire si un passager survivra ou non. Le modèle ci-dessous utilise 3 caractéristiques / attributs / colonnes de l'ensemble de données, à savoir le sexe, l'âge et la famille (nombre de conjoint et/ ou d'enfants).

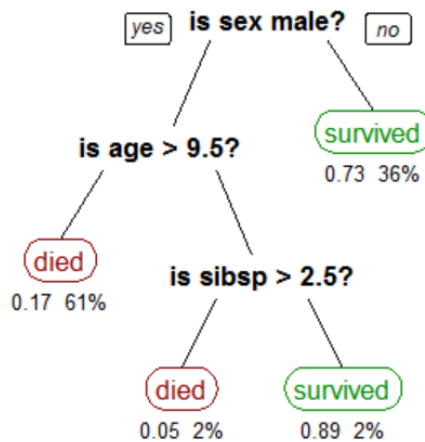


Figure 36 : Exemple d'un arbre de décision

Un arbre de décision est dessiné à l'envers avec sa racine en haut. Dans l'image 36, le texte en gras en noir représente une condition / nœud interne, sur la base duquel l'arbre se divise en branches / bords. La fin de la branche qui ne se sépare plus est la décision / feuille, dans ce cas, si le passager est décédé ou a survécu, il est représenté respectivement en texte rouge ou vert.

Avec un vrai jeu de données, il y aura beaucoup plus d'attributs et donc beaucoup plus de branches, mais même avec beaucoup de données cet algorithme reste simple. Mais c'est important que les propriétés soient clairement présentées et que les relations peuvent être facilement visualisées. Cette méthodologie est plus communément connue sous le nom d'arbre de décision d'apprentissage à partir de données et l'arbre ci-dessus est appelé arbre de classification car l'objectif est de classer le passager comme ayant survécu ou décédé. Les arbres de régression sont représentés de la même manière, mais ils prédisent des valeurs continues comme le prix d'une maison dans le cas de ce projet.

Pour appliquer cet algorithme, j'utilise également la librairie « sklearn » de python, et les étapes sont les mêmes que pour la régression linéaire. Le résultat que j'obtiens est le suivant :

Arbre de décision		
Modele	Propriétés	R_square
modele 1	surface_reelle_bati	0.63
modele 2	surface_reelle_bati, nombre_pieces_principales	0.7
modele 3	surface_reelle_bati, nombre_pieces_principales, prix/m2	1
modele 4	all variables sauf longitude et latitude	1

Figure 37 : Divers essais avec des modèles d'arbre de décision

Les résultats sont nettement meilleurs que ceux de la régression linéaire, mais pour les modèles 3 et 4, le résultat est égal à 1, ce qui me fait douter. Car si un modèle est trop adapté aux données, il est contre-productif ! Ce phénomène dans ML est appelé « overfitting », ce que nous devons toujours éviter lors de la création de modèles.

Il existe de nombreuses façons de détecter et de gérer ce phénomène. Dans ce projet, j'ai choisi une méthode simple qui est de supprimer un attribut que j'ai trouvé dupliqué par rapport à d'autres propriétés. Je choisis de supprimer « prix/m2 » car si nous analysons attentivement les propriétés, si nous connaissons le prix d'un mètre carré du logement alors évidemment avec la superficie du logement, nous calculerons le prix du logement et nous possédons déjà la surface et le prix.

Après avoir supprimé la propriété « prix/m2 », le résultat obtenu est le suivant :

Arbre de décision		
Modele	Propriétés	R_square
modele 1	surface_reelle_bati	0.63
modele 2	surface_reelle_bati, nombre_pieces_principales	0.7
modele 3	surface_reelle_bati, nombre_pieces_principales, prix/m2	1
modele 4	all variables sauf longitude, latitude et prix/m2	0.84

Figure 38 : Résultat des modèles d'arbre de décision après supprimant la propriété « prix/m2 »

Lorsque je supprime l'attribut « prix/m2 », le modèle 3 devient le modèle 2 et le résultat du modèle 4 passe à 0,84.

Ce qui est drôle car la propriété que je viens de supprimer du modèle pour cause de « overfitting » est la propriété que j'ai ajouté dans l'étape précédente pour enrichir mes données. Ainsi, quand nous décidons d'un changement pour le modèle, nous devons réfléchir attentivement pour éviter de perdre du temps. En outre, cela confirme en plus que le processus de recherche d'un modèle optimal est extrêmement ardu, nécessitant de nombreuses méthodes et techniques différentes à tester.

À ce stade, j'ai également retravaillé le modèle avec la méthode de régression linéaire, et le résultat final que j'ai obtenu est le suivant :

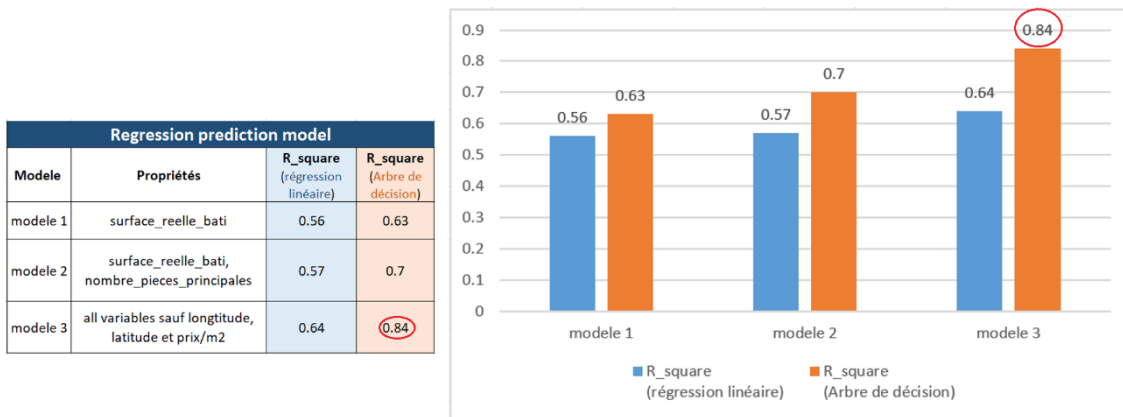


Figure 39 : Comparaison modèles d'arbre de décision et régression linéaire

En regardant la figure 39, nous voyons qu'avec le même jeu de données d'entrée, le modèle d'arbre de décision donne toujours de meilleurs résultats que le modèle de régression linéaire. Et (jusqu'à présent), après une série de tests, le meilleur modèle est l'arbre de décision (modèle 3) avec $R_square = 0.84$.

L'arbre de décision est un algorithme facile à comprendre et à interpréter, par conséquent, un seul arbre peut ne pas être suffisant pour que le modèle en apprenne les caractéristiques. D'autre part, Random Forest est également un algorithme basé sur un arbre qui utilise les caractéristiques de qualités de plusieurs arbres de décision pour prendre des décisions. Je vais donc construire le modèle avec l'algorithme de forêt aléatoire, pour comparer avec l'arbre de décision afin de vérifier si c'est vraiment le meilleur choix ?

3.3 Forêt aléatoire

La forêt aléatoire est un algorithme de classification composé de nombreux arbres de décision. Lors de la construction de chaque arbre, ces derniers sont construits aléatoirement tout en étant cloisonnés les uns par rapport aux autres. Chaque arbre individuel de la forêt aléatoire génère une prédiction de classe et la classe avec le plus de votes devient la prédiction de notre modèle.

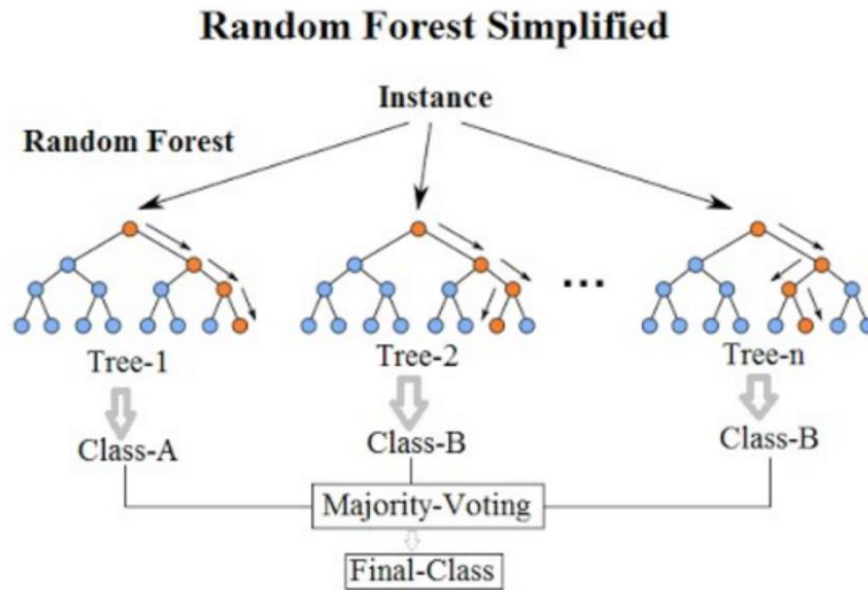


Figure 40 : La forêt aléatoire est un algorithme de classification composé de nombreux arbres de décision

Pour modéliser la forêt aléatoire, j'ai utilisé la fonction « RandomForestRegressor » de « sklearn » :

```
from sklearn.ensemble import RandomForestRegressor
```

Et j'ai appliqué cet algorithme avec les mêmes données que j'ai utilisé pour les algorithmes ci-dessus, j'ai obtenu le résultat :

Forêt aléatoire		
Modele	variable indépendante	R_square
modele 1	surface_reelle_bati	0.63
modele 2	surface_reelle_bati, nombre_pieces_principales	0.69
modele 3	all variables sauf longitude, latitude et prix/m2	0.82

Figure 41 : Résultat des modèles de la forêt aléatoire

On voit que le modèle est toujours meilleur avec plus de variables indépendantes. Cependant, dans cet algorithme, je vais essayer de modifier les paramètres pour voir si le résultat change.

Avec les résultats obtenus ci-dessus, j'ai défini les paramètres du modèle avec 10 arbres (« `n_estimators` » est le nombre d'arbres dans la forêt) comme suit :

```
forest = RandomForestRegressor(n_estimators = 10, random_state = 0)
```

Cependant, nous avons également de nombreux autres paramètres :

```
(bootstrap=True, criterion='mse', max_depth=None,  
max_features='auto', max_leaf_nodes=None,  
min_impurity_decrease=0.0, min_impurity_split=None,  
min_samples_leaf=1, min_samples_split=2,  
min_weight_fraction_leaf=0.0, n_estimators=10,  
n_jobs=None, oob_score=False, random_state=0, verbose=0,  
warm_start=False)
```

paramètres
de la forêt aléatoires

Avec mon expérience, il y a 3 paramètres principaux qui peuvent être réglés pour améliorer la puissance prédictive du modèle : « `max_features` », « `n_estimators` » et « `min_sample_leaf` ». Je vais commencer avec « `max_features` ».

« **Max_features** » est le nombre maximum d'attributs que Random Forest va utiliser pour créer un arbre individuel et il existe plusieurs options disponibles dans Python (default = "auto").

- *auto*: `max_features = n_features`
- *sqrt*: `max_features = SQRT (n_features)`
- *log2*: `max_features = Log2 (n_features)`
- *int*: `max_features = max_features`
- *float*: `max_features = int(max_features * n_features)`

L'augmentation de « `max_features` » améliore généralement les performances du modèle car à chaque nœud, nous avons maintenant un plus grand nombre d'options à considérer. Par contre, si nous augmentons « `max_features` », la vitesse de l'algorithme va diminuer.

Forêt aléatoire					
Modele	variable indépendante	max_feature			
		Auto	1	2	3...10
modele 1	surface_reelle_bati	0.63	0.63		
modele 2	surface_reelle_bati, nombre_pieces_principales	0.69	0.69	0.69	
modele 3	all variables sauf longitude, latitude et prix/m2	0.82	0.82	0.82	0.82

Figure 42 : Résultat des modèles de la forêt aléatoire avec plusieurs tests du paramètre « max_features »

Dans la figure 42, nous voyons qu'avec des changements dans le paramètre « max_features », le résultat du modèle ne change pas. Ainsi dans ce projet, le paramètre « max_features » n'a pas d'influence sur le résultat du modèle.

Je vais maintenant essayer de régler d'autres paramètres :

« **N_estimators** » est le nombre d'arbres dans la forêt. Le plus élevé est ce paramètre, le meilleur le résultat sera mais le temps d'exécution du programme sera également plus élevé. Nous choisirons donc une valeur aussi élevée que le processeur peut le supporter, car cela rend les prédictions plus solides et plus stables. Dans Python, ce paramètre a une valeur par défaut de 100.

« **Min_sample_leaf** » est le nombre minimum de « sample » requis pour chaque nœud final d'un arbre de décision. Plus la valeur de ce paramètre est basse et plus le modèle incorporera de bruit parasite dans les données d'entraînement. En général, je préfère une taille de feuille minimale de plus de 50. Cependant, on peut essayer plusieurs tailles de feuille pour trouver la plus optimale.

Forêt aléatoire													
Modele	variable indépendante	max_feature				n_estimators			min_samples_leaf				
		Auto	1	2	3...10	5	10, 15, 20	100, 150, 200	1	2	3	5	10
modele 1	surface_reelle_bati	0.63	0.63			0.63	0.63	0.63	0.63	0.62	0.62	0.61	0.61
modele 2	surface_reelle_bati, nombre_pieces_principales	0.69	0.69	0.69		0.68	0.69	0.69	0.69	0.68	0.67	0.65	0.64
modele 3	all variables sauf longitude, latitude et prix/m2	0.82	0.82	0.82	0.82	0.81	0.82	0.82	0.82	0.79	0.77	0.75	0.72

Figure 43 : Résultat des modèles de la forêt aléatoire après réglage des paramètres

En regardant la figure 43, nous voyons qu'il n'y a pas beaucoup de changement lorsque je règle les paramètres « max_features » et « n_estimators ». En particulier pour le paramètre « min_sample_leaf », plus j'augmente la valeur de ce paramètre, moins le résultat est bon.

En bref, pour ce projet, la première option avec « max_features » = auto, « n_estimators » = 10 et « min_sample_leaf » = 1 est toujours la plus optimale.

CHAPITRE III

Bilan et les améliorations du projet

Dans ce projet, j'ai construit de nombreux modèles différents en utilisant différents algorithmes (régression linéaire, arbre de décision, forêt aléatoire) ainsi que des techniques différentes de ML.

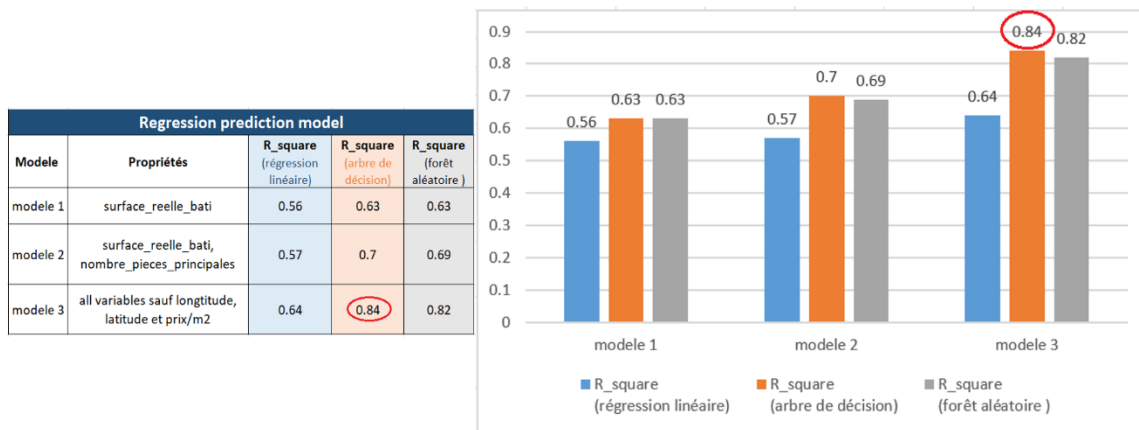


Figure 44 : Résultat de 3 modèles différents de régression prédictive

En regardant la figure 44, nous voyons que le modèle d'arbre de décision donne les meilleurs résultats. Par conséquent, je vais choisir ce modèle pour effectuer des exemples de prédictions des prix immobiliers. Je créerai ensuite une application qui permettra aux utilisateurs d'utiliser facilement mon modèle. Enfin, il y aura forcément des choses à améliorer sur ce projet à l'avenir.

1. PREDICTION

Afin de prédire les prix des logements, j'utilise la fonction « predict() » avec des données de test que j'avais séparé au début (20% des données).

surface_bati	nb_pieces	vente_en_futur	vente_terrain	Lyon_2	Lyon_3	Lyon_4	Lyon_5	Lyon_6	Lyon_7	Lyon_8	Lyon_9	Maison	Prix_logement	Prix_logement_prediction
49	2	0	0	0	0	0	0	0	0	0	1	0	125000.0	151896.5
55	3	0	0	1	0	0	0	0	0	0	0	0	185000.0	284325.0
105	5	0	0	0	0	1	0	0	0	0	0	0	458830.0	403056.2
48	2	0	0	0	1	0	0	0	0	0	0	0	152000.0	168741.2
130	5	0	0	0	1	0	0	0	0	0	0	0	467000.0	539000.0
...
30	1	0	0	0	0	0	0	1	0	0	0	0	155110.0	135762.5
93	3	0	0	0	0	0	1	0	0	0	0	0	320000.0	387500.0
82	4	0	0	0	1	0	0	0	0	0	0	0	280200.0	276351.3
121	4	0	0	0	0	0	0	1	0	0	0	0	482000.0	521241.7
62	2	0	0	0	0	0	0	0	0	0	0	0	240000.0	271220.0

Figure 45 : Résultat de la prédiction du prix du logement avec des données de test

En regardant la figure 45, nous pouvons voir la différence entre les prix réels et les prix prédictifs des logements, mais la différence n'est pas trop grande. Par exemple, avec la première ligne, le prix réel d'appartement avec 49 m2, 2 pièces à Lyon 9 vendu est 125000 euros et le prix prédit pour ce logement est 151896.5 euros.

Je vais prédire combien coûterait un appartement que je souhaiterais acheter dans un proche avenir.

L'appartement que je souhaite **acheter** est un **appartement** de **100** m2, avec **3** pièces principales, à **Lyon 7**

```
dream_appartement = tree.predict([[100, 3, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]])
dream_appartement[0]
```

302200.0 prédiction du prix
d'appartement

Je peux donc savoir que l'appartement que je recherche coûtera environ 302200 euros.

2. DEPLOIEMENT

Pour permettre aux utilisateurs d'utiliser plus facilement mon modèle de prédiction de prix de logements à Lyon, j'ai créé une application « housing price prediction » en utilisant le framework « Flask » de Python et Heroku qui est une plate-forme cloud pour créer, distribuer, surveiller et étendre des applications.

Mon application comporte deux pages principales : accueil et prédiction.

- **Accueil** : Pour que les utilisateurs remplissent les informations du logement qu'ils souhaitent acheter, dans laquelle, pour les paramètres avec la réponse oui ou non, l'utilisateur entrera 1 ou 0 à la place.
- **Prediction** : Pour présenter le résultat de la prédiction du prix du logement correspondant aux critères choisis sur la page d'accueil.

The figure shows two screenshots of a web application titled "Housing Price Prediction" with a "DREAM" logo. Both screenshots show a form titled "Please enter parameters of your dream house below:" with the following fields:

- Surface réelle bâtiment:
- Nombre de pieces principales:
- Vous voulez un logement en future (oui/non):
- Vous voulez acheter une terrain (oui/non):
- Lyon 2 (oui/non - 1/0):
- Lyon 3 (oui/non - 1/0):
- Lyon 4 (oui/non - 1/0):
- Lyon 5 (oui/non - 1/0):
- Lyon 6 (oui/non - 1/0):
- Lyon 7 (oui/non - 1/0):
- Lyon 8 (oui/non - 1/0):
- Lyon 9 (oui/non - 1/0):
- Vous voulez une Maison (oui/non - 1/0):

A "Predict" button is at the bottom of the form. In the right screenshot, the following values are entered:

- Surface réelle bâtiment: 100
- Nombre de pieces principales: 3
- Vous voulez un logement en future (oui/non): 0
- Vous voulez acheter une terrain (oui/non): 0
- Lyon 2 (oui/non - 1/0): 0
- Lyon 3 (oui/non - 1/0): 0
- Lyon 4 (oui/non - 1/0): 0
- Lyon 5 (oui/non - 1/0): 0
- Lyon 6 (oui/non - 1/0): 0
- Lyon 7 (oui/non - 1/0): 1
- Lyon 8 (oui/non - 1/0): 0
- Lyon 9 (oui/non - 1/0): 0
- Vous voulez une Maison (oui/non - 1/0): 0

Figure 46 : La page d'accueil de l'application



Thank You! Here is the Information You Gave for your dream house in Lyon:

- Surface réelle bâtiment: 100 (m2)
- Nombre de pieces principales: 3
- Vous voulez un logement en future: 0
- Vous voulez acheter une terrain: 0
- Vous voulez habiter à Lyon 2: 0
- Vous voulez habiter à Lyon 3: 0
- Vous voulez habiter à Lyon 4: 0
- Vous voulez habiter à Lyon 5: 0
- Vous voulez habiter à Lyon 6: 0
- Vous voulez habiter à Lyon 7: 1
- Vous voulez habiter à Lyon 8: 0
- Vous voulez habiter à Lyon 9: 0
- Vous voulez une masion: 0

>> Your Predicted Housing Price is: 302200 £

[Back to your parameters of your dream house](#)

Figure 47 : La page « prédiction » de l'application

Depuis la page prédiction, vous pouvez retourner sur la page d'accueil pour faire une nouvelle prédiction.

Vous pouvez dès à présent cliquer le lien ci-dessous pour utiliser mon application :

<https://housing-price-prediction-app.herokuapp.com/>

CONCLUSION

Comme mentionné plusieurs fois dans ce rapport, la mise en œuvre d'un problème de ML est extrêmement compliquée et demande beaucoup de temps et d'efforts. Plus nous passons de temps, plus nous trouvons de choses à améliorer.

Dans le cadre de ce projet, j'ai développé avec succès de nombreux modèles différents et appliqué de nombreuses techniques différentes pour essayer de trouver le meilleur modèle. En fin de compte, j'ai trouvé que le modèle avec l'arbre de décision était le meilleur. J'ai également créé une application pour faciliter l'utilisation de mon modèle par les utilisateurs.

Cependant, avec plus de temps, j'aurai aimé pouvoir tester le modèle avec plus de sources de données. J'aurai créé moi-même une base de données complète avec plus de variables indépendantes. En fait, j'avais une idée pour une telle base de données, j'avais imaginé les besoins réels des clients (plus complexe), et à partir de là, j'ai construit un modèle conceptuel de données (MCD), puis à partir de ce MCD, j'aurai rédigé un modèle physique de données (MPD), à partir de ce MPD, j'aurai pu créer une base de données complète pour le projet. Si vous souhaitez plus d'informations à ce sujet, les schémas de MCD et MPD sont en annexe de ce rapport.

De plus, j'aurai pu expérimenter plus d'algorithmes de ML tels que le xgboost, la régression logistique... Et, bien évidemment, j'aurai eu plus de temps pour tester différents réglages pour les paramètres du modèle...

Beaucoup de travail et d'idées si j'avais eu plus de temps !

Annexe

1. MODELE CONCEPTUEL DE DONNEES (MCD)

Le modèle conceptuel de données sert à conceptualiser l'application ! Il met en évidence deux éléments : les entités et les associations.

L'entité est un objet que l'on souhaite modéliser, par exemple un logement, une commune, une agence, un acheteur... comme sur la figure A1. Chaque entité possède des attributs : le logement possède par exemple une surface réelle, un nombre de pièces principales, une valeur foncière... (figure A1).

Les attributs des entités possèdent des types standardisés pour les décrire, par exemple : int, float, date, text, boolean...

Les associations illustrent le lien entre les entités : les relations sémantiques. Par exemple, un logement détermine un seul type de logement mais un type de logement peut déterminer plusieurs logements différents (figure A1).

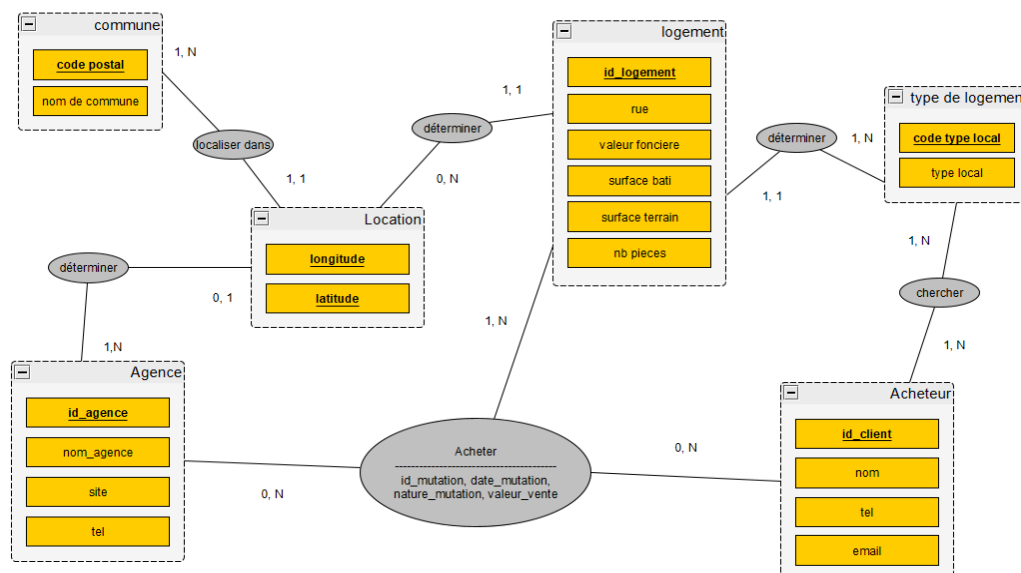


Figure A1 : Modèle conceptuel de données (MCD)

2. MODELE PHYSIQUE DE DONNEES (MPD)

L'étape de création du MPD est presque une formalité comparée à la création du MCD. En s'appuyant sur des règles, on peut faire évoluer la modélisation de haut niveau pour la transformer en un schéma plus proche des contraintes des logiciels de bases de données.

Concrètement, cette étape permet de construire la structure finale de la base de données avec les différents liens entre les éléments qui la composent.

Quand on passe du MCD au MPD, les entités se transforment en tables, les propriétés se transforment en champs (ou attributs) et toute entité doit être représentée par une table.

Cependant, on a encore quelques règles importantes à suivre pour passer du MCD au MPD :

Règle 1 : Dans le cas d'entités reliées par des associations de type « 1 : n », chaque table possède sa propre clef primaire, mais la clef primaire de l'entité côté 0,n (ou 1,n) migre vers la table côté 0,1 (ou 1,1) et devient une clef étrangère (index secondaire).

Par exemple dans le MCD, entre l'entité « agence » et l'entité « location », on voit les associations de type « 1 : n », et donc dans le MPD, la clef de l'entité « agence » est migré dans la table « location » comme clef étrangère.

Règle 2 : Dans le cas d'entités reliées par des associations de type « n:n », une table intermédiaire dite table de jointure, doit être créée, et doit posséder comme clef primaire une conjonction des clefs primaires des deux tables pour lesquelles elle sert de jointure.

Par exemple dans le MCD, on voit qu'entre l'entité « type de logement » et l'entité « acheteur », il y a des associations de type « n:n », donc dans le MPD (figure A2), on peut trouver une table de jointure « logement recherché » avec les clefs primaires des deux tables « type de logement » et « acheteur ».

Règle 3 : Les propriétés se trouvant au milieu d'une relation génèrent une nouvelle table ou glissent vers la table adéquate en fonction des cardinalités de la relation

Par exemple, dans le MCD, entre 3 entités « logement », « agence » et « acheteur » on peut générer une nouvelle table « mutation » dans le MPD avec les clefs primaires des trois tables.

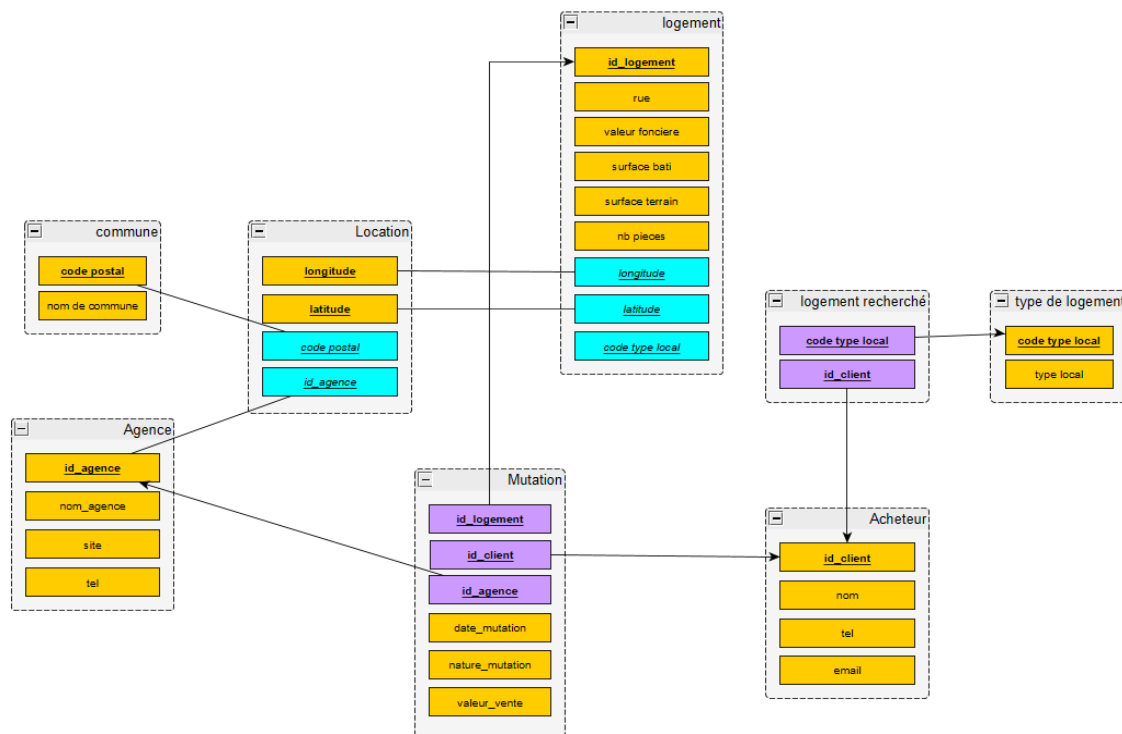


Figure A2 : Modèle physique de données (MPD)