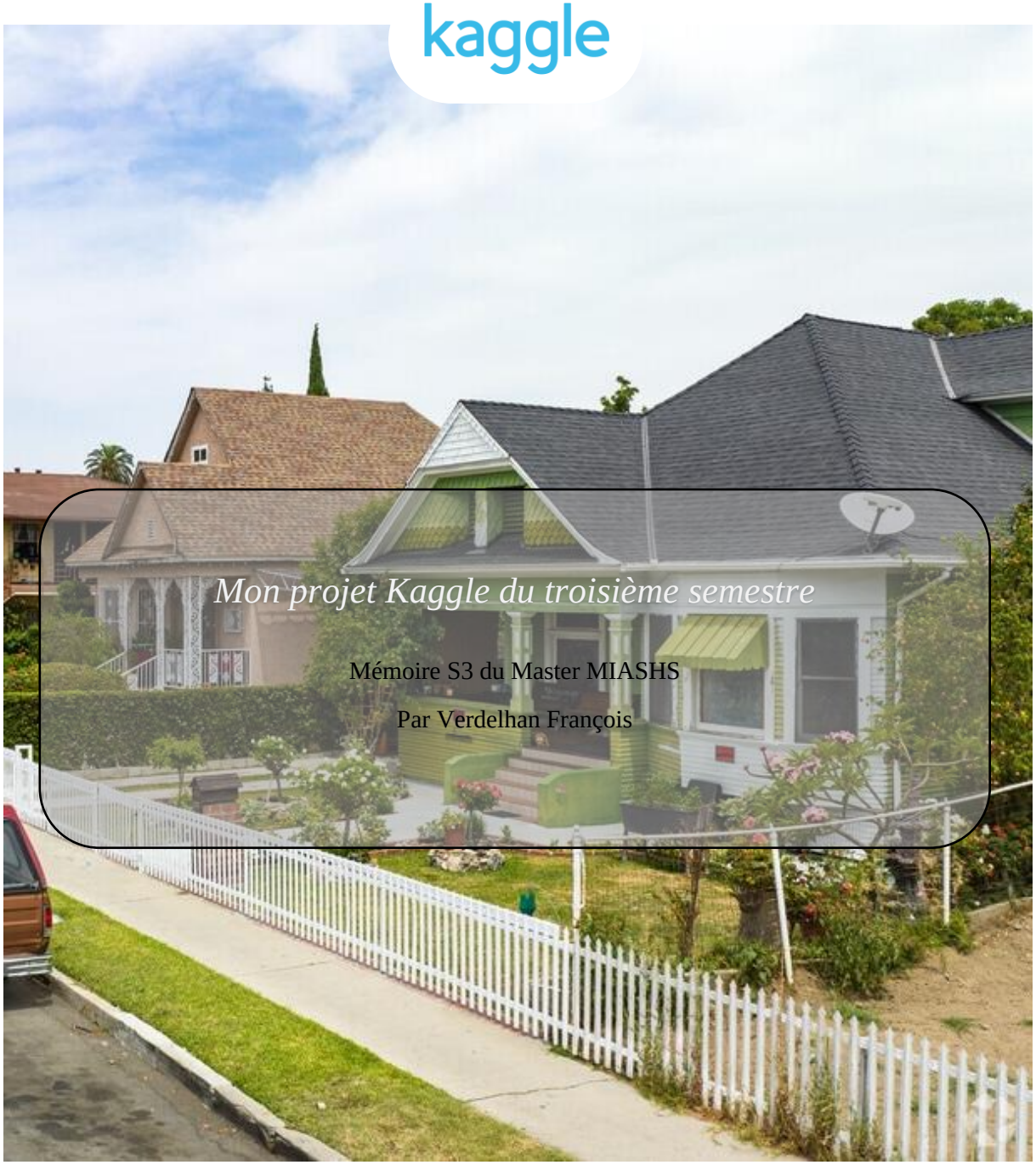
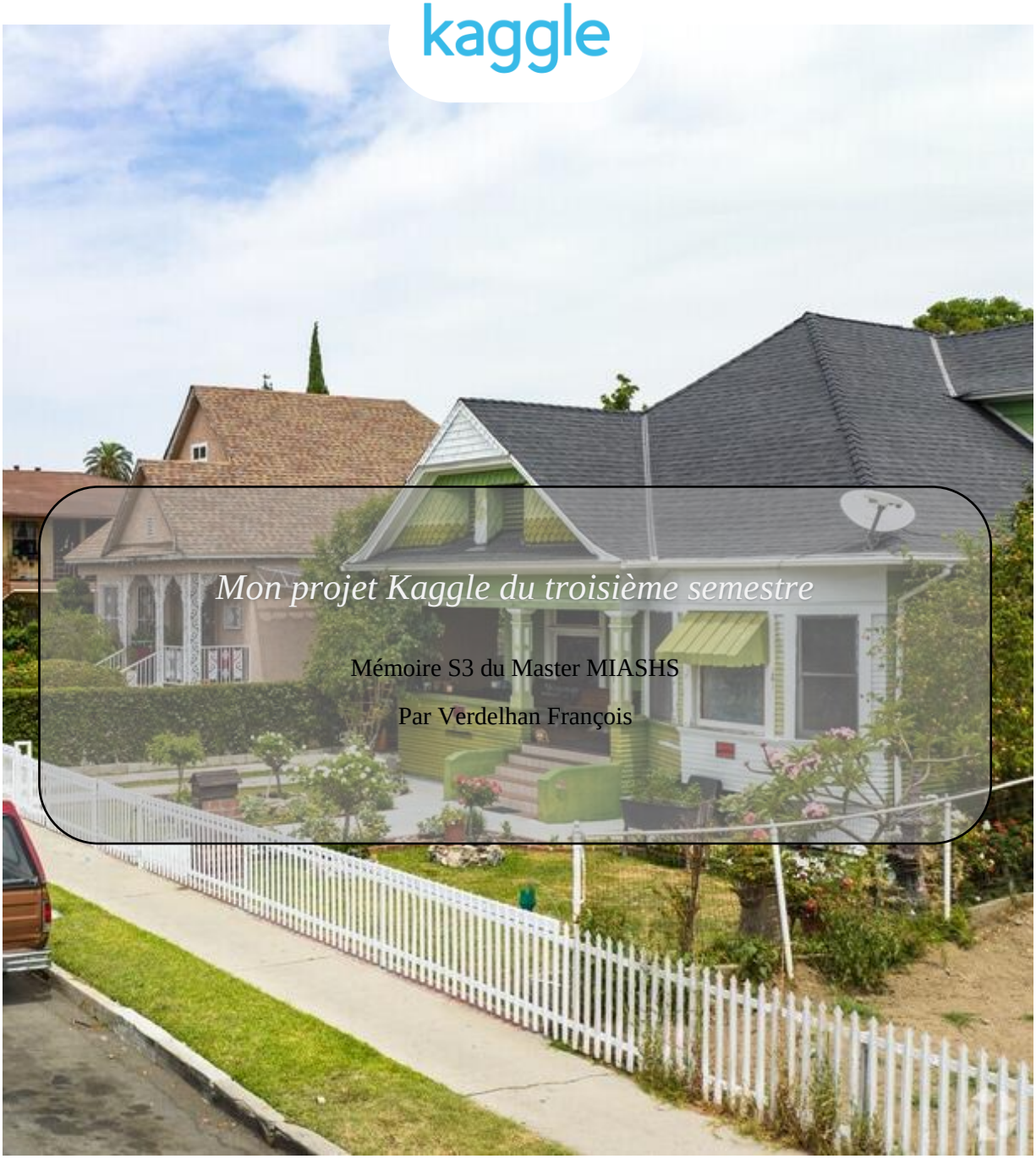


Mon projet Kaggle du troisième semestre

Mémoire S3 du Master MIASHS

Par Verdelhan François



Mon projet Kaggle du troisième semestre

Mémoire S3 du Master MIASHS

Par Verdelhan François

Remerciements

Je remercie les personnes qui m'ont aidé à me relire et qui m'ont aidé durant ce projet Kaggle. Je pense surtout à ma copine et à mes parents pour la relecture. Un grand merci également à Yoann Debain et Quentin Loeb pour l'aide qu'ils ont su m'apporter lors de mes questions.

Je remercie aussi l'équipe enseignante qui s'est montrée tolérante et qui a su me donner la possibilité de réaliser ce projet, afin de pouvoir valider mon troisième semestre de master 2 MIASHS.

Sommaire

Introduction

Je réalise actuellement ma seconde année de master MIASHS à l'Université Paul Valéry à Montpellier.

En fin d'année de M1, l'ambiance au sein de mon entreprise s'est fortement dégradée avec l'arrivée d'un nouveau manager, et les missions s'éloignant de plus en plus de ce qui était demandé en master MIASHS. Un accord a été trouvé pour réaliser une rupture de contrat avec Atlantic.

Durant le semestre qui a suivi, j'ai donc eu pour mission de trouver une nouvelle alternance. J'ai pendant longtemps prospecté pour des alternances et eu plusieurs entretiens avec différentes sociétés et entreprises, Engie, Keolis, RTE, Volvo, K-net. Malgré ces entretiens, aucun ne s'est soldé par une embauche en tant qu'alternant. Voyant le temps se réduire avant la fin de mon semestre, l'équipe enseignante a mis en place pour les personnes n'ayant pas trouvé d'entreprise un projet afin de pouvoir réaliser le mémoire du semestre 3.

I : Mon projet : *House Prices - Advanced Regression Techniques*

Lien : https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques

A : Kaggle

Kaggle est une plateforme en ligne créée pour les passionnés de données et de machine learning, lancée en 2010. Elle réunit de multiples défis ou compétitions proposés par des entreprises ou des communautés, permettant à quiconque de participer à ces projets, qu'ils soient rémunérés ou non. Chaque participant propose un ou plusieurs modèles pour répondre aux exigences du projet. Kaggle met vraiment en avant le partage et l'échange d'informations et d'aide, se révélant être un outil incontournable pour progresser et évoluer dans le domaine de la data et du machine learning.

B : Sélection du projet

Après une recherche sur le site Kaggle, j'ai décidé de porter mon choix sur un projet de prévision de prix de maisons en Californie. J'ai fait ce choix car c'était un domaine que je n'avais pas beaucoup exploré et que j'aimais particulièrement. Surtout avec les connaissances accumulées en M1 et en M2, je savais que de nombreuses méthodes étaient à ma disposition et j'avais envie de les mettre en pratique. Ce projet m'a donné cette opportunité.

De plus, c'était un projet très parlant pour moi où j'arrivais à identifier tout de suite dans quel cadre de la vie réelle un tel projet pourrait être mis en place, que ce soit pour des assurances, des constructeurs de maisons ou n'importe quelle entreprise dans le monde de l'immobilier. Le projet était complet, il comportait de nombreuses variables, tant quantitatives que qualitatives, et je disposais d'une base de données assez grande.

L'avantage de partir sur un projet que je n'avais jamais réalisé, c'est que c'était pour moi l'occasion de renforcer mes connaissances et mes potentiels atouts lors d'un entretien futur pour les prochaines recherches de stage que j'aurais, ou même pour plus tard dans ma vie professionnelle.

C : Les données

Pour ce projet, j'étais donc totalement libre dans le choix des outils, des techniques d'approche et des réalisations afin de répondre au mieux à la problématique posée. J'ai donc dû m'imposer un plan et créer une structure dans ma recherche. Dans la section qui suit, je vais vous présenter les données que j'avais à ma disposition ainsi que la base de données de manière plus générale, et vous présenter le nettoyage de ces données.

La base qui m'était donnée est une base contenant plus de 1459 individus et 79 variables, à la fois quantitatives et qualitatives. J'avais aussi à ma disposition un autre fichier contenant cette fois-ci une centaine d'individus sur lesquels je devais prédire le prix des maisons. Ces prédictions seraient ensuite soumises à Kaggle pour que la plateforme puisse évaluer mon modèle.

Dans un premier temps, je me suis occupé du premier fichier. J'ai d'abord traité mes données pour les mettre au propre, car même si le fichier Kaggle était assez propre, il y avait quelques individus nécessitant un retraitement et certaines variables avaient aussi besoin de remaniement. Par exemple, dans mon jeu de données, il y avait une variable "barrière" indiquant si une maison est entourée de barrières ou non. Pour les maisons sans barrières, c'était indiqué "NA" non pas parce que la donnée était manquante, mais parce qu'il n'y avait pas de barrière. Dans ce genre de cas, j'ai remplacé "NA" par "no fence" pour "pas de barrière". J'ai donc remanié plusieurs données, que ce soit les formats de dates et de mois ou les cas comme la variable "barrière". Après ce traitement de la base, je me suis retrouvé avec 9 individus que j'ai préféré mettre de côté, car ils manquaient trop d'informations.

Ensuite, j'ai pris ce fichier et j'en ai fait une copie. Sur cette copie, j'ai centré et réduit mes données quantitatives, car certains de mes modèles ultérieurs auraient besoin de données standardisées.

Pour finir, j'ai divisé ces fichiers en deux parties : la première serait mon fichier de train, qui représente 80% du total, et la dernière serait mon fichier de test, qui représenterait 20% de ce dernier.

D : Analyse des variables

Tout d'abord, pour avoir un point de vue global sur l'ensemble de mon jeu de données, j'ai décidé de réaliser une table de corrélations des variables quantitatives et une table de corrélations des variables qualitatives.

Voici, par exemple, ci-dessous la table de corrélation des 32 variables quantitatives :

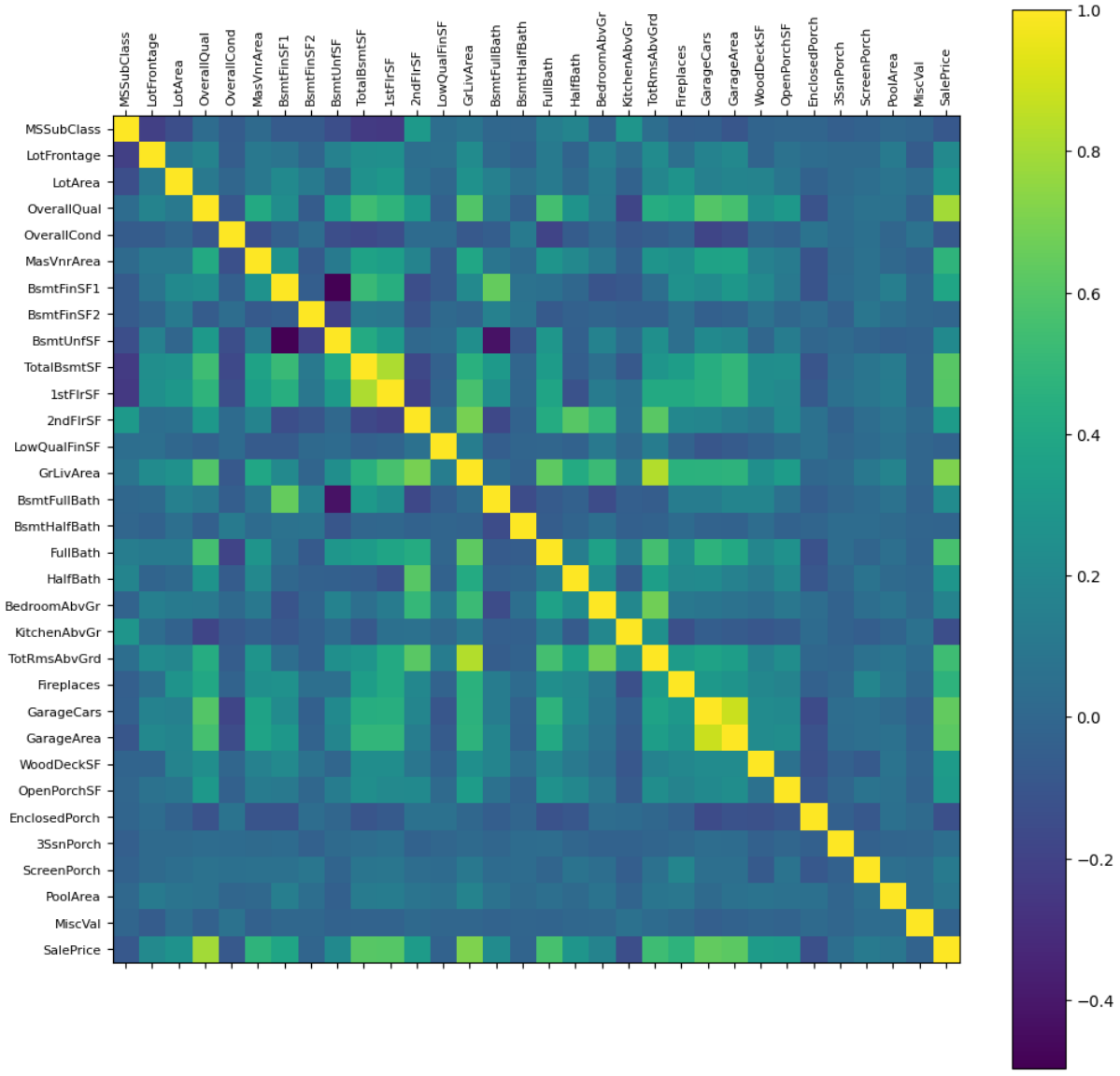


Figure 1 : Table de corrélation des variables quantitatives

Cette table, ainsi que celle des variables qualitatives, vont me servir à identifier les variables qui semblent être corrélées avec la variable « SalePrice ». Par exemple, dans notre exemple plus haut, on peut voir certaines variables qui semblent être corrélées avec cette variable, comme « OverallQual ».

Je vais aussi essayer de mettre en place une Analyse Factorielle des Données Mixtes (AFDM) sur mes variables afin de voir si j'arrive à identifier un pattern ou des éléments supplémentaires. Cependant, après la réalisation de cette AFDM, rien ne semble être particulièrement concluant. En effet, comme on pouvait s'y attendre, il est compliqué de représenter ces variables sur seulement 2 dimensions, et on arrive seulement à capturer environ 6% de l'information.

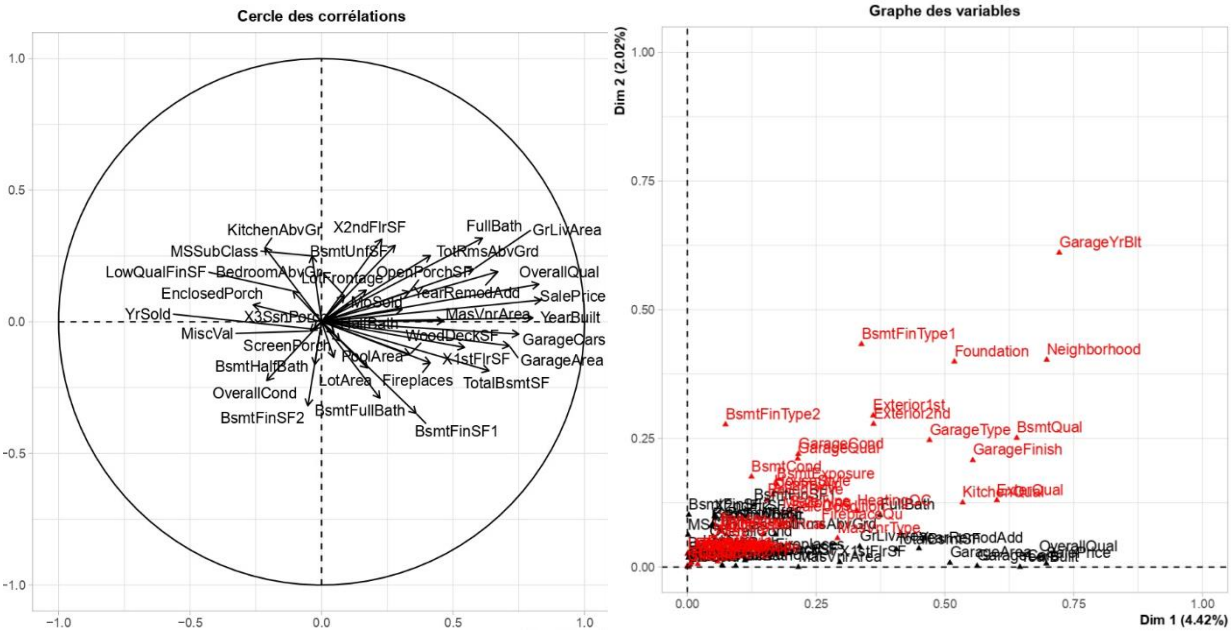


Figure 2 : AFDM

E : Mes modèles

1 : Modèle de référence

Dans un premier temps, j'ai créé un modèle de référence, un modèle simple, presque "stupide", mais qui me permettrait de situer mes modèles plus complexes par rapport à celui-ci. Ce modèle de base a été réalisé de la façon la plus simple possible : j'ai simplement pris la moyenne des prix des maisons de ma base de données et appliqué cette moyenne à l'ensemble des maisons du fichier de soumission Kaggle. Après avoir soumis ce fichier, j'ai obtenu un score de référence de 0.42. Ce modèle me sert donc de point de comparaison pour mes modèles ultérieurs.

Le but de ce modèle est assez simple : il me donne un indicateur rapide de l'efficacité et de la pertinence de mes modèles plus élaborés. Si un modèle obtient un score beaucoup plus faible, il est considéré comme meilleur et plus pertinent. À l'inverse, si un modèle a un score bien plus élevé, cela signifie qu'il est soit inefficace dans ce type de situation, soit mal défini en amont.

Le dernier avantage de ce modèle de référence était simplement de me familiariser avec le processus de soumission d'un fichier sur Kaggle, car c'était ma première soumission. Cela m'a permis de comprendre les étapes nécessaires pour soumettre un modèle, de m'assurer que le format de mon fichier était correct et de m'habituer à l'interface de Kaggle.

2 : L'ANCOVA

Ma première idée était d'utiliser l'un des premiers modèles que nous avons étudiés, à savoir un modèle de régression linéaire. Cependant, dans notre cas actuel, bien que nous disposions et utilisions des variables quantitatives pour notre modèle de régression linéaire, nous avons également des variables qualitatives qui peuvent être intégrées dans un modèle ANOVA. Ainsi, pour combiner ces deux types de variables, nous avons recours à un modèle ANCOVA.

Pour que notre modèle fonctionne de manière optimale, il est préférable de standardiser notre jeu de données. C'est pourquoi nous utilisons notre second jeu de données, qui est standardisé. Nous procéderons également à une sélection de variables, car un excès de variables peut nuire à la performance

de ce type de modèle. C'est la raison pour laquelle mon objectif initial était de choisir les variables présentant une corrélation d'au moins 0.6, soit 60%, avec la variable cible 'SalePrice' qu'elle soit positive ou bien négative. Pour ce faire, je me base sur le tableau des corrélations mentionné précédemment.

Ce qui correspond à ces variables :

'OverallQual', 'TotalBsmtSF', '1stFlrSF', 'GrLivArea', 'GarageCars', 'GarageArea'

J'entraîne donc mon modèle et réalise ensuite une comparaison avec les 20 % de données de test restantes afin de voir la correspondance entre les prédictions et les valeurs réelles. Voici le graphique représentant les résultats de mon modèle.

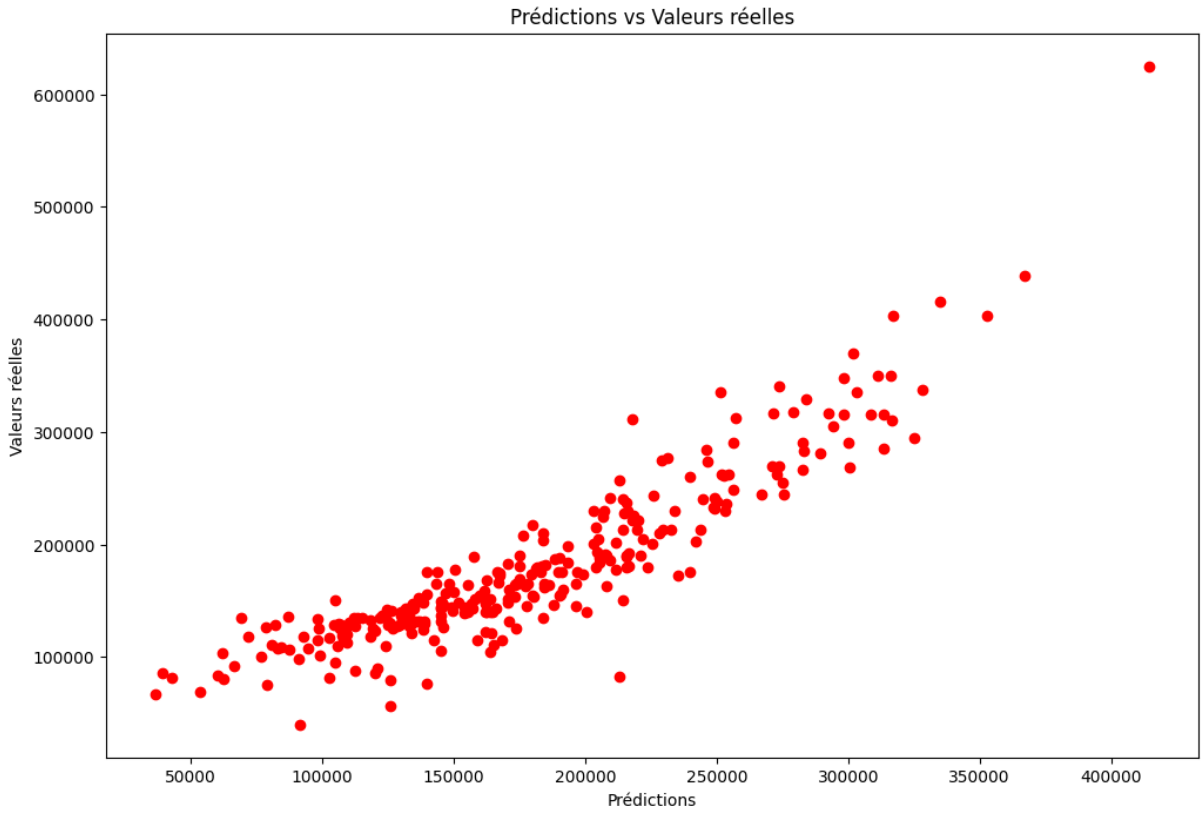


Figure 3 : Prédiction VS Réalité du modèle ANCOVA

On semble obtenir une bonne distribution, en effet, cela semble suivre une ligne diagonale entre les prédictions et les valeurs réelles, ce qui est très encourageant pour la suite.

On peut cependant noter quelques éléments qui semblent être importants à prendre en compte. Tout d'abord, on peut aussi voir que la dispersion semble s'accentuer avec les prix élevés des maisons. Ça peut être dû à plusieurs raisons :

- Premièrement, ça peut être dû à un manque d'individus de ce style-là, autrement dit, un manque de maisons chères.
- Ensuite, peut-être que les critères changent en fonction d'un certain seuil de richesse d'une maison.
- Il peut aussi évidemment y avoir des valeurs aberrantes, même si nous avons essayé de les évincer le plus possible.

Avec ce modèle on obtient un score de 0.22 ce qui est une belle avancée par rapport à notre modèle simple.

Aller plus loin avec les modèles ANCOVA

Nous avons également eu l'occasion de constater qu'il était possible d'aller plus loin dans la sélection des variables pour ce type de modèle. J'ai donc tenté de mettre cela en œuvre.

Au lieu de choisir manuellement, grâce à la table des corrélations, les variables qui semblaient le mieux indiquer les réponses à notre problème, nous avons la possibilité de laisser notre modèle choisir lui-même, ce qui, à mon avis, est préférable.

Pour ce faire, il y a 2 méthodes, en réalité 3, qui existent. Tout d'abord, nous ne donnons à notre modèle aucune variable, puis il va les ajouter une par une, pour voir celle qui semble améliorer la précision de notre modèle.

Il existe aussi l'inverse, qui est le Backward, c'est-à-dire qu'il commence avec toutes les variables, puis il les retire une à une, et à chaque fois, il retire celle qui apporte le moins au modèle.

Un autre modèle vient ensuite, la méthode qui est un mélange des 2 précédentes, c'est-à-dire qu'il va partir de rien et, au fur et à mesure, il va soit ajouter une variable, soit enlever une variable déjà sélectionnée, en fonction de celle qui semble améliorer le mieux le modèle en place.

Cette façon de faire semble idéale, mais déjà elle prend du temps et pas mal de ressources de calcul. Afin d'accélérer le processus, car j'avais beaucoup de modèles à tester et pas seulement des Ancova, je décide de réaliser un modèle de Forward sur le critère BIC. Pour ce faire, je décide d'utiliser le critère de sélection BIC, car c'est un critère qui favorise la sélection de variables, et donc la diminution du nombre de variables utilisées pour la suite.

Avec ce modèle, j'obtiens un score de 0.20, ce qui est une avancée notable. Nous le verrons par la suite, ce n'est pas le mieux que nous pouvons faire, mais avec ce type de modèle, donc les Ancova, cela semble déjà être un très beau score et une belle réalisation.

3 : Random Forest

Ensuite, j'ai décidé d'utiliser un modèle de Random Forest. Qu'est-ce que c'est ? Il s'agit d'un algorithme d'apprentissage supervisé, employé tant pour la classification que pour la régression. Cette dernière peut être de nature binaire, quantitative ou qualitative.

Lors de la création d'un Random Forest, l'algorithme construit un ensemble de plusieurs arbres de décision, d'où son nom de "forêt". Pour la phase de prédiction, notre algorithme regroupe les résultats de tous ces arbres. L'avantage de cette méthode réside dans sa précision, sa robustesse, sa simplicité, ainsi que sa capacité à gérer de grandes quantités de données et un grand nombre de variables, en plus de ça polyvalence.

J'ai donc mis en place deux algorithmes de Random Forest, en utilisant notre jeu de données avec les données non standardisées. En effet, l'un des autres avantages de l'utilisation d'un algorithme de Random Forest est sa résistance aux différentes échelles des variables.

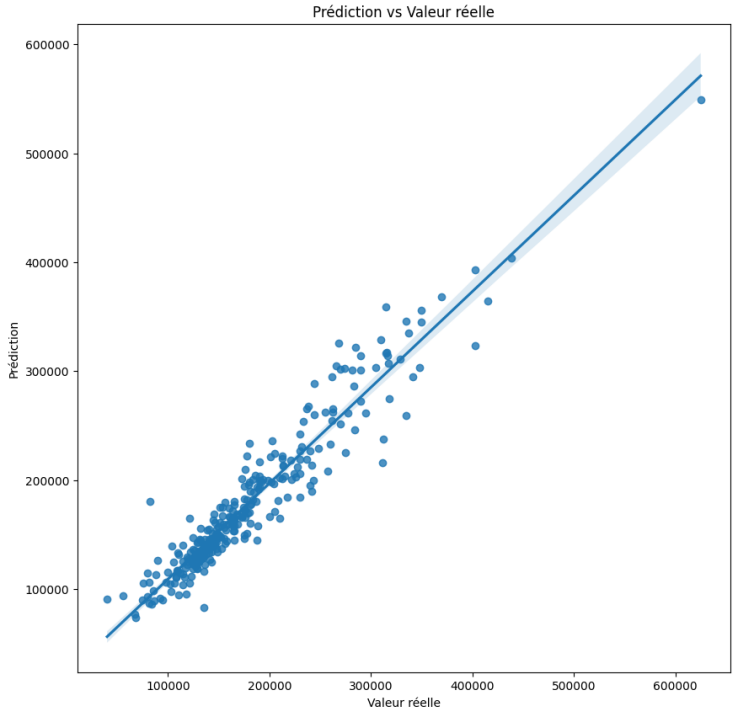


Figure 4 : Prédictions VS Réelle random forest

Voici ci-dessus un autre graphique comparatif entre les prédictions du Random Forest et les valeurs réelles de notre jeu de données. On constate cette fois-ci que la prédiction semble meilleure par rapport à notre graphique précédent, notamment grâce à une courbe qui paraît être plus centrée.

En analysant les résidus, on peut observer certaines caractéristiques. Tout d'abord, notre modèle ne semble pas suivre une loi normale. Cependant, dans le cas d'une Random Forest, cela ne pose pas de problème. En effet, avec les Random Forest, on n'établit pas l'hypothèse de distribution, comme la normalité des données en entrée.

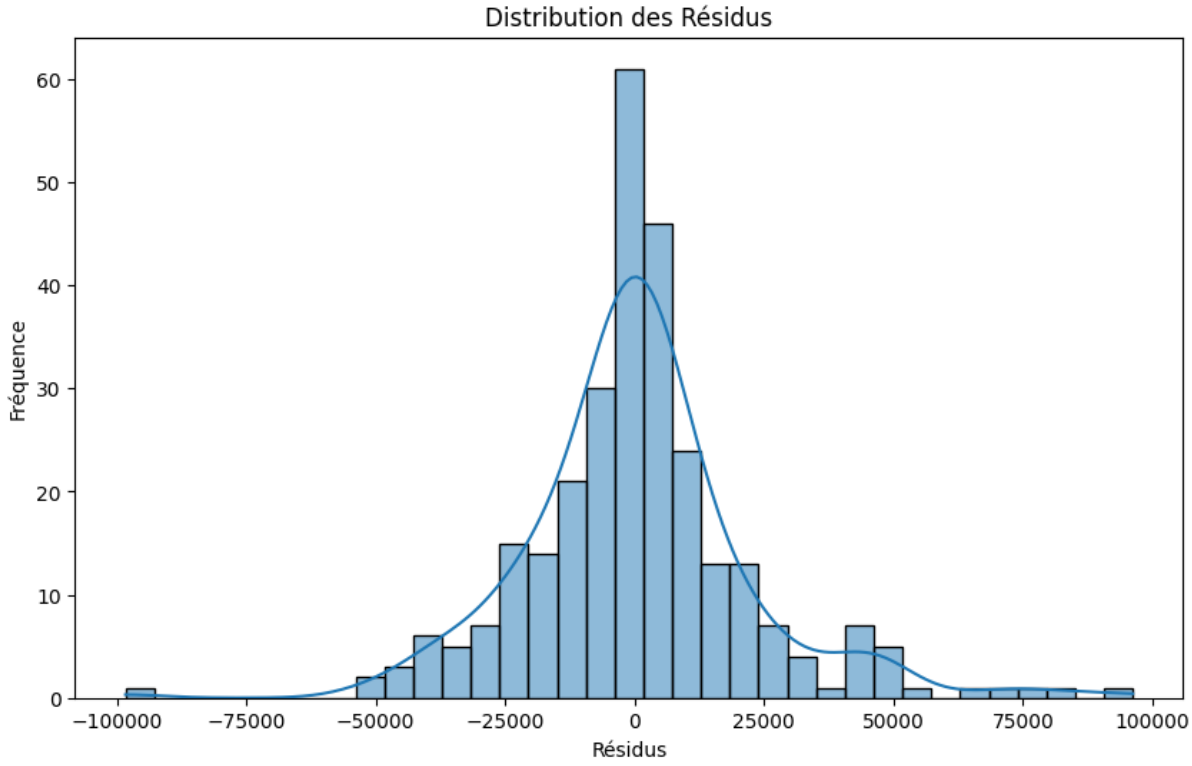


Figure 5 : Distribution des résidus du random forest

Il est intéressant de noter que la dispersion observée sur la courbe concernant les habitations à coûts élevés se maintient dans nos prédictions. Cela est bien plus évident sur le graphique ci-dessous qui représente la dispersion des résidus. Il semble donc que nous ayons réussi à capturer la tendance générale et une bonne partie de l'information, mais une quantité significative de variance subsiste. Cette variance peut être attribuée au bruit dans les données ou indiquer que notre modèle pourrait être amélioré pour capturer davantage d'informations. Il est important de rappeler qu'il est toujours possible, et même probable, que le nombre d'individus dans la zone à coûts élevés, étant peut-être trop faible, rend difficile pour notre modèle la capture d'informations précises pour cerner ces individus.

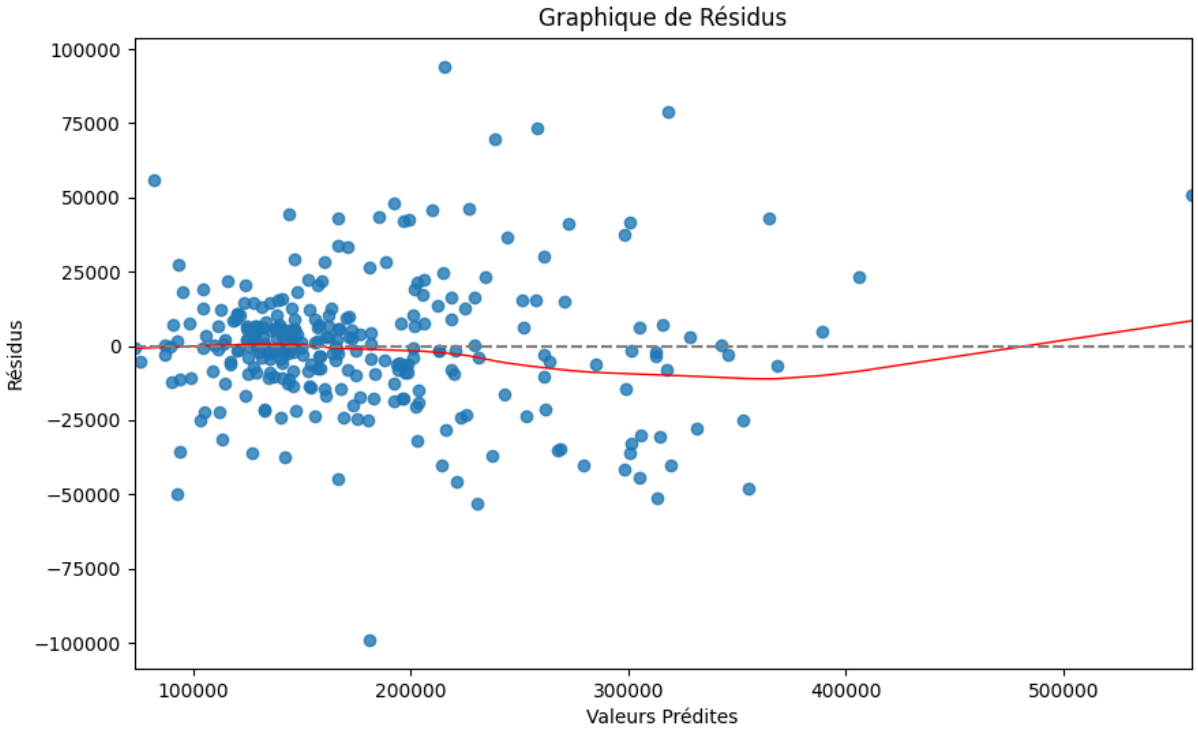


Figure 6 : Variance des résidus random forest

Avec ce modèle, nous obtenons un score de 0.149131, ce qui représente une réelle progression et constitue un très bon score.

Aller plus loin avec les random Forest

Pour l'instant, notre Random Forest a été réalisé avec des paramètres par défaut. C'est pourquoi, dans un second temps, je vais mettre en place un autre algorithme capable de tester plusieurs combinaisons de paramètres. Ce processus impliquera la modification du nombre d'estimateurs (n_estimators) et de la profondeur maximale (max_depth) pour voir si nous avons la possibilité d'améliorer notre Random Forest.

Pour ce faire, je vais mettre en place la méthode "Grid Search". Cette méthode consiste à définir une grille de valeurs possibles pour chaque hyperparamètre de notre modèle, puis à tester toutes les combinaisons possibles avec ces valeurs. Ensuite, le meilleur résultat est choisi et nous conservons les paramètres de ce dernier. Bien que cette méthode soit coûteuse en temps et en ressources, elle nous permet d'améliorer notre modèle. Après un certain temps de travail, les meilleurs paramètres identifiés sont : 'regressor__max_depth': 20, 'regressor__n_estimators': 200.

Cela signifie concrètement que la profondeur maximale (regressor__max_depth) de chaque arbre doit être de 20 nœuds. Cela détermine en quelque sorte la complexité de chacun des arbres. De plus, "regressor__n_estimators" égal à 200 indique que notre modèle va utiliser 200 arbres distincts.

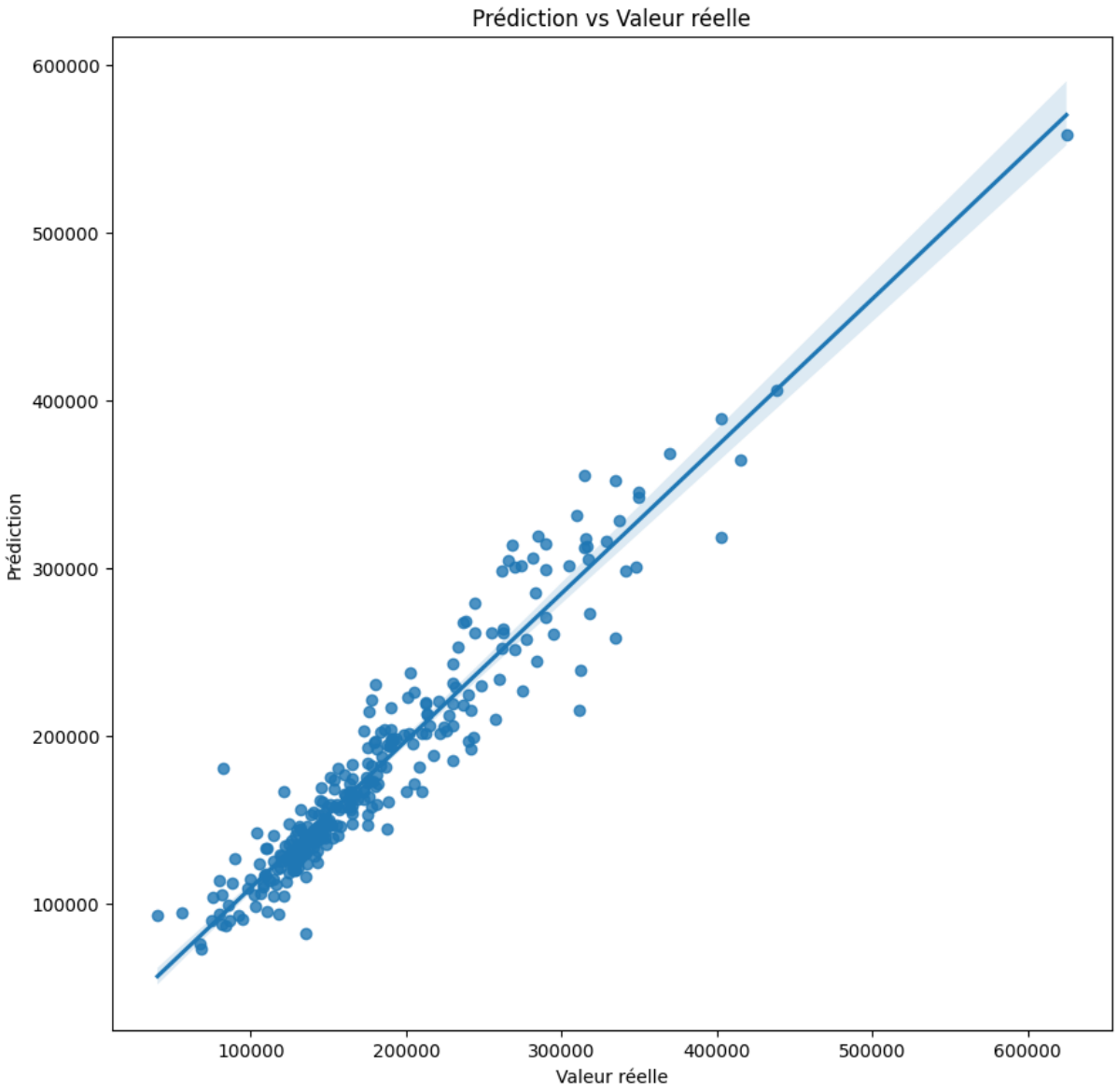


Figure 7 : Prédictions VS Réelle random forest avec sélection des paramètres

Voici ci-dessous le graphique comparant les prédictions et les valeurs réelles avec ces nouveaux paramètres choisis. On observe que la dispersion semble légèrement réduite et que notre modèle paraît un peu plus précis. Effectivement, lors de notre soumission sur Kaggle avec ces paramètres sélectionnés,

nous obtenons un score de 0.14823. Cela représente une amélioration notable et constitue notre deuxième meilleur score durant ce projet.

4 : Modèle Adaboost

Un algorithme d'Adaboost est aussi un type d'ensemble learning, tout comme le Random Forest, mais avec une approche différente. Le Random Forest crée plusieurs arbres qui tentent tous de répondre à la question posée et utilise la moyenne ou la majorité des réponses pour apporter la meilleure solution au problème. Tous ces arbres sont indépendants et chacun essaie de répondre au problème, ce qui les rend plus complexes que dans un modèle Adaboost. De plus, grâce à leur indépendance, l'entraînement de ce modèle est effectué en parallèle, ce qui le rend plus rapide.

L'algorithme d'Adaboost se distingue par une approche différente. En effet, chaque arbre dans Adaboost essaie de corriger les erreurs de l'arbre précédent. Ensuite, à la fin du processus, chaque arbre est pondéré en fonction de sa précision. La prédiction finale est alors réalisée grâce à une combinaison pondérée de ces arbres. L'avantage de l'Adaboost est qu'il met l'accent sur les individus mal classés et se concentre davantage sur les cas difficiles. Ainsi, avec un peu de chance, nos maisons en zone riche pourraient être plus facilement classées grâce à ce modèle. Cependant, il présente aussi des inconvénients : du fait de sa nature séquentielle, il prend plus de temps à être entraîné. De plus, il peut être sujet au surajustement, surtout si les données sont bruyantes ou trop complexes.

Dans le cas d'un modèle Adaboost, les arbres utilisés sont plus simples et sont parfois appelés "stumps" ou arbres nains. En effet, dans ce contexte, un arbre ne cherche pas toujours à répondre à l'ensemble de la question. C'est pourquoi, dans nos paramètres par défaut pour Adaboost, nos arbres ont seulement une profondeur de 5 couches. Cette simplicité permet à chaque arbre de se concentrer sur un aspect spécifique du problème, facilitant ainsi la correction des erreurs par les arbres suivants dans la séquence.

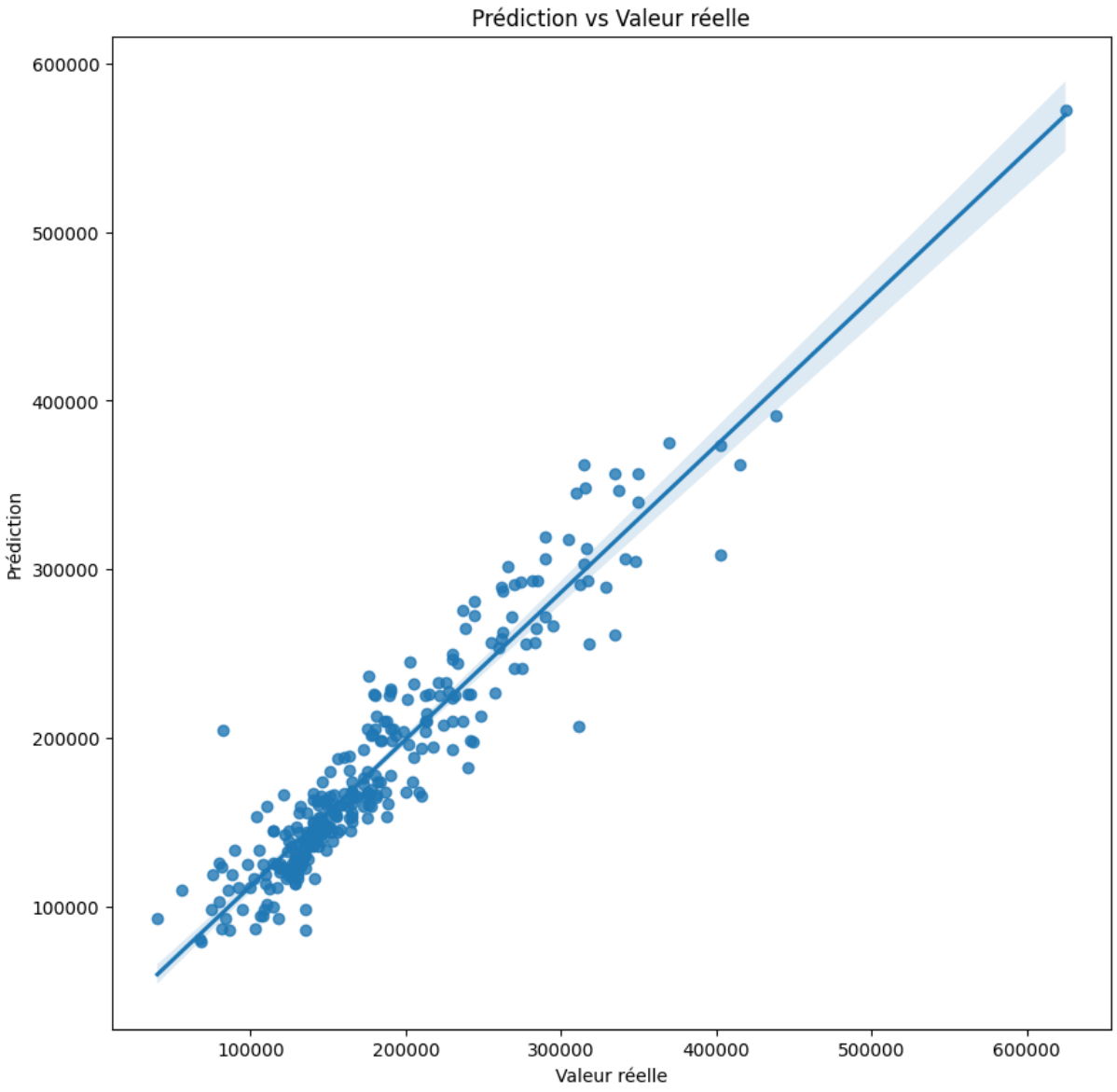


Figure 8 : Prédictions VS Réelle Adaboost sans sélection des paramètres

Déjà, nous pouvons constater que tout fonctionne bien : les individus se trouvent toujours proches de la ligne entre la prédiction et la valeur réelle, et de plus, la variance dans la zone riche semble avoir diminué.

Avec ce modèle, j'obtiens un score de 0.16668. C'est un bon résultat, mais moins bon que celui obtenu précédemment avec un Random Forest classique. On peut en conclure que ce modèle d'Adaboost capture peut-être un peu mieux l'information des mesures en zones riches. Cependant, de manière générale, il semble capturer un peu moins bien l'information, probablement à cause du nombre de nœuds limité à 5 pour les arbres, ce qui reste trop faible pour capturer suffisamment d'informations. C'est pourquoi, comme précédemment, nous allons essayer de l'améliorer.

C'est donc en utilisant la même méthode que celle employée pour le Random Forest, c'est-à-dire la méthode de Grid Search, que nous allons tenter de trouver les meilleurs hyperparamètres pour notre modèle Adaboost. L'objectif est de réaliser cette optimisation sans pour autant tomber dans le surajustement. En effet, si nous réalisons un surajustement sur notre fichier de test, lors de notre soumission sur Kaggle, le score pourrait être bien pire. Cela signifierait que nous aurions seulement appris à coller aux données spécifiques du jeu de test sans véritablement comprendre le comportement général de ces dernières. Il est crucial de trouver un équilibre pour que le modèle reste généralisable à de nouvelles données.

Après avoir exécuté notre algorithme Adaboost avec la méthode Grid Search, celui-ci a déterminé, parmi les options que nous lui avons fournies, que la meilleure configuration pour notre projet est une profondeur d'arbre (max_depth) de 20 avec un nombre d'estimateurs (n_estimators) de 150. En effet, après avoir utilisé ce modèle et soumis nos résultats sur Kaggle, nous avons obtenu un score de 0.14542. Ce score représente notre meilleur résultat durant toute la durée de notre projet.

À ce moment-là, je ne suis pas encore conscient que ce sera mon meilleur résultat. C'est pourquoi je vais vous exposer la suite de ma démarche et de mes recherches. Cette poursuite de l'exploration et de l'optimisation des modèles est essentielle pour comprendre en profondeur les possibilités et les limites de notre approche, et pour éventuellement découvrir des améliorations inattendues.

5 : Modèle XGBoost

Le modèle XGBoost est similaire au modèle AdaBoost dans le sens où ces deux types de modèles tentent de corriger les erreurs de l'arbre précédent. Toutefois, la différence réside dans la manière de corriger ces erreurs. AdaBoost se focalise sur la classification des observations difficiles en augmentant leur poids dans les données d'entraînement, tandis que XGBoost vise à minimiser la fonction de perte globale, en prenant en compte à la fois les erreurs de classification et la complexité du modèle.

XGBoost offre une grande flexibilité grâce à de nombreux hyperparamètres ajustables. Il inclut les hyperparamètres des modèles précédents, mais permet également l'accès à de nouveaux hyperparamètres. Conçu pour être hautement efficace et performant sur de grands ensembles de données

et pour les problèmes complexes, XGBoost pourrait ne pas être le plus adapté dans notre cas, étant donné que notre problème actuel ne semble pas extrêmement complexe.

J'ai d'abord réalisé un modèle classique et obtenu un score de 0.17082, ce qui est correct, mais supérieur à ce que nous avons déjà obtenu par le passé. Voici ci-dessous le graphique comparant nos prédictions aux réponses attendues sur notre base de test.

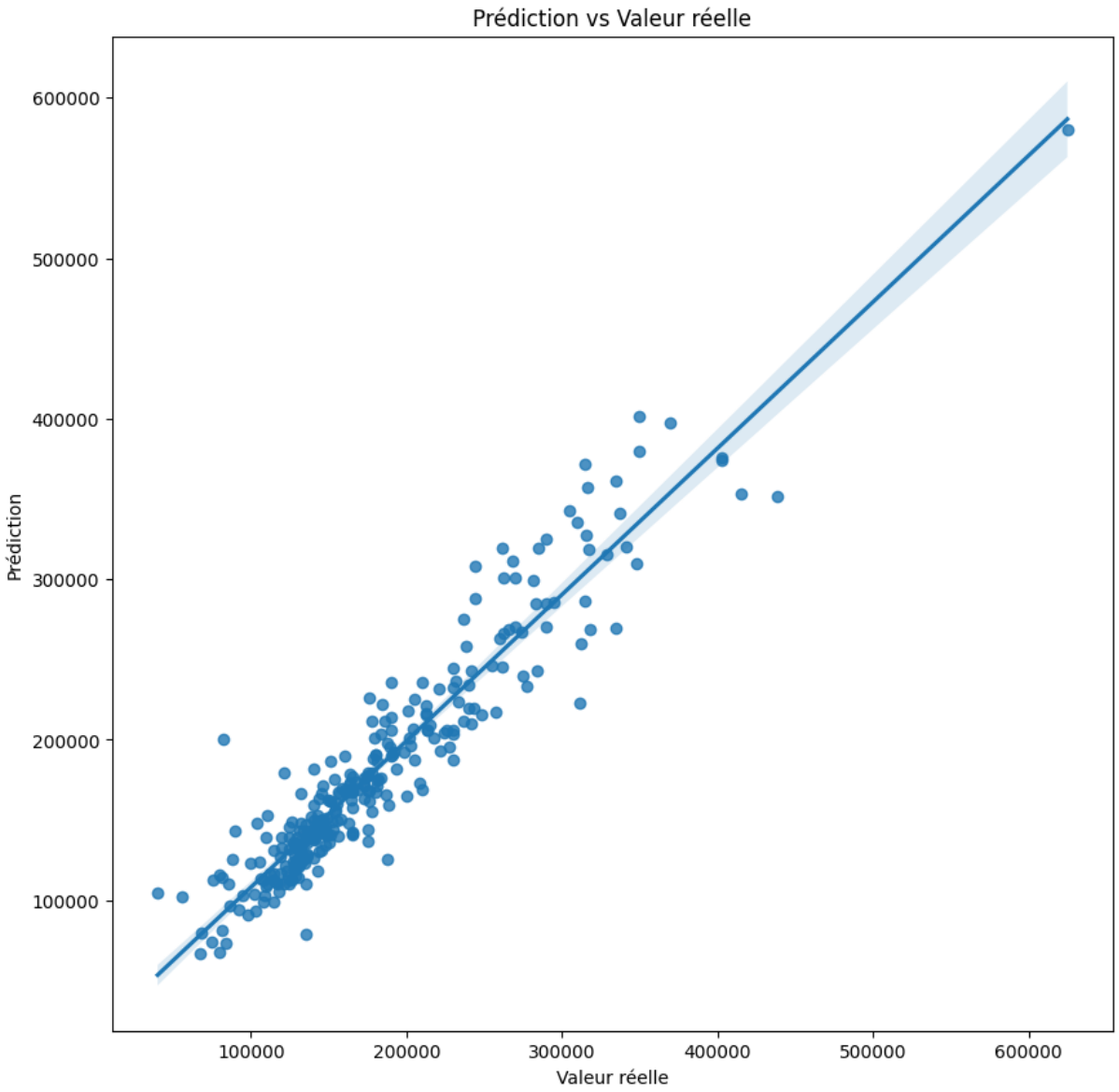


Figure 9 : Prédictions VS Réelle XGBoost sans sélection des paramètres

Aller plus loin avec la méthode XGBoost

Comme à notre habitude, on cherche à aller plus loin. C'est pourquoi nous allons une fois de plus utiliser la méthode de grid search.

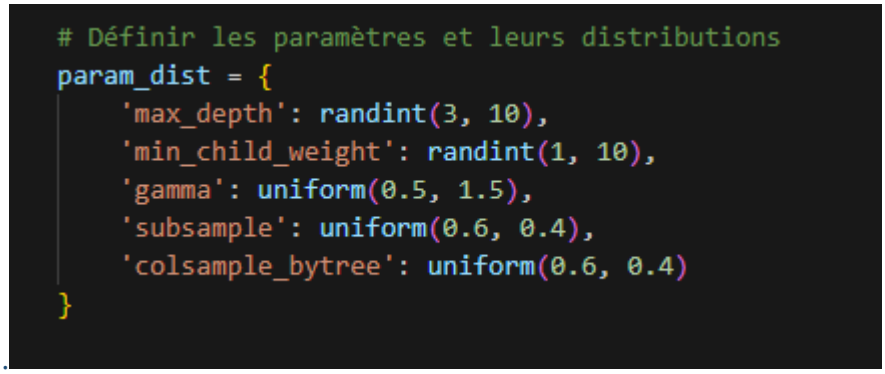


Figure 10 : Liste des hypers paramètres

Voici ci-dessus une liste des paramètres que nous allons tenter de définir. Après un certain temps de calcul, voici les paramètres qui sont estimés comme étant les meilleurs.

- 'colsample_bytree': 0.7923984158587971
- 'gamma': 0.5494350733734006
- 'max_depth': 9
- 'min_child_weight': 2
- 'subsample': 0.761728176289353

Voici une explication simple de ce que représentent les paramètres dans notre modèle :

'colsample_bytree' : Ce paramètre détermine quelle proportion des variables est utilisée pour chaque arbre, ce qui rend chaque arbre un peu différent.

'gamma' : Il sert à contrôler quand arrêter de diviser un nœud dans l'arbre.

'max_depth' : Il fixe la profondeur maximale que chaque arbre peut atteindre.

'min_child_weight' : Ce paramètre aide à décider si un nœud de l'arbre doit être divisé en fonction du poids des observations qu'il contient, empêchant la création de modèles trop spécifiques sur de petits groupes de données.

'subsample' : Il indique la part des données utilisées pour construire chaque arbre, permettant au modèle de ne pas toujours voir toutes les données à chaque fois, ce qui contribue réduis le surajustement.

En utilisant ce modèle, nous obtenons un score de 0.16475, ce qui est une amélioration. Nous observons également que la dispersion des résidus semble se réduire, ce qui est positif car cela signifie que nous avons capturé une plus grande partie de l'information. À mon avis, le score est moins bon ici parce que le modèle semble moins adapté ce genre de régression dans un cadre aussi simple.

6 : Modèle des KNN

Ici nous allons attaquer une nouvelle méthode qui n'appartient plus au domaine des Random Forest, nous allons utiliser l'algorithme des KNN.

Notre prochain modèle est celui des plus proches voisins, également appelé KNN (K-Nearest Neighbors). C'est un modèle basé sur l'idée d'attribuer une valeur à un individu en fonction de ses plus proches voisins. Autrement dit, il positionne notre individu dans un espace selon ses caractéristiques et identifie quels sont les individus les plus proches de lui, c'est-à-dire ceux qui lui ressemblent le plus. Sur cette base, le modèle détermine la valeur à attribuer à notre individu, en prenant en compte les valeurs des voisins similaires. Cette méthode repose donc sur l'hypothèse que des individus similaires auront des valeurs ou des caractéristiques similaires.

Ci-dessous se trouve un schéma, trouvé sur Google Images, qui illustre bien le fonctionnement d'un modèle KNN. Sur ce schéma, un nouvel individu est représenté en bleu, et les individus déjà connus sont en gris. Ici, nous nous trouvons dans un espace à deux dimensions pour simplifier l'exemple, mais en réalité, le nombre de dimensions peut être bien plus élevé. Dans cet exemple, nous avons choisi K = 5, c'est-à-dire que nous considérons les 5 voisins les plus proches de notre nouvel individu pour estimer sa valeur.

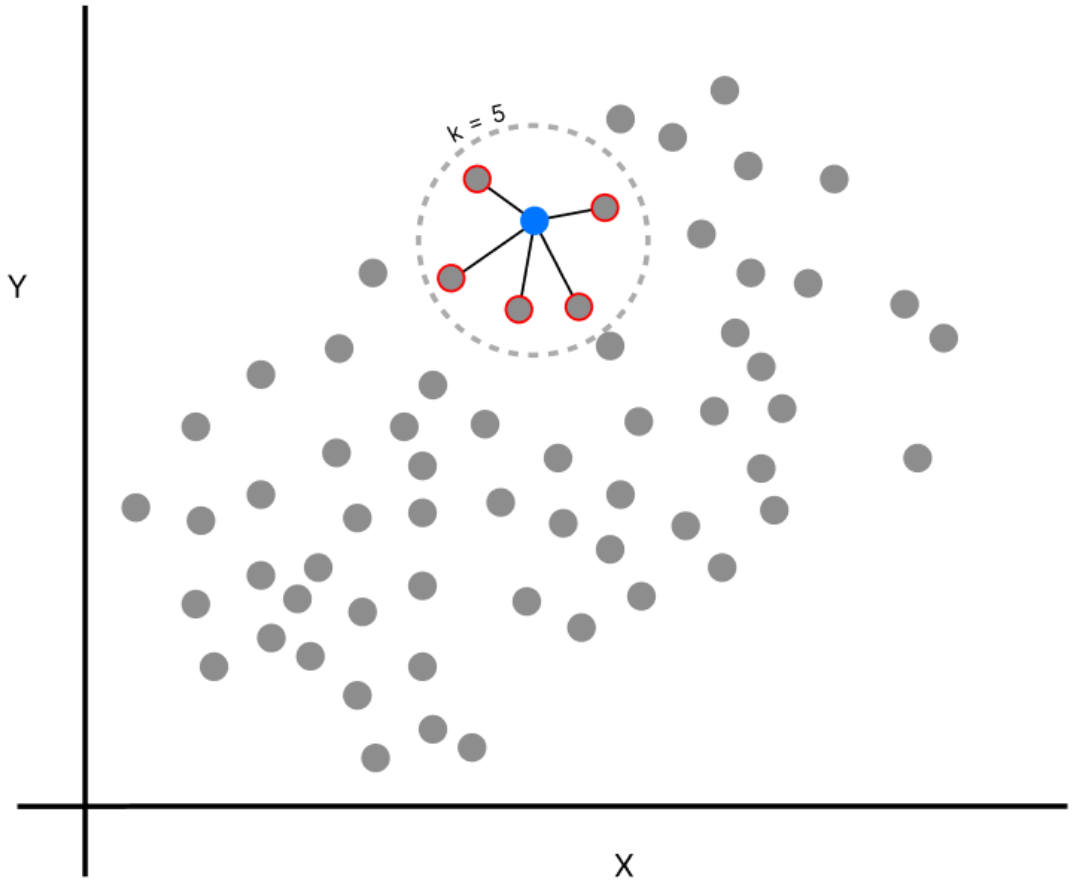


Figure 11 : Exemple du fonctionnement des KNN

Ce type d'algorithme, le KNN, se distingue par sa simplicité et son efficacité. Cependant, il présente plusieurs inconvénients. Tout d'abord, il est généralement plus lent que certains autres types de modèles. En effet, pour chaque prédiction, il doit rechercher les voisins les plus proches, ce qui peut être coûteux en termes de calculs, surtout avec de grands jeux de données. De plus, il est très sensible aux échelles des données et aux données aberrantes. Par conséquent, dans notre cas, il sera nécessaire d'utiliser une base de données qui a été standardisée.

En outre, le KNN est sensible à la dimensionnalité des données (phénomène connu sous le nom de malédiction de la dimensionnalité). Autrement dit, plus il y a de dimensions prises en compte, moins l'algorithme est efficace pour les prédictions. Pour pallier ce problème, nous pourrions, comme dans le cas de la régression linéaire, sélectionner certaines dimensions pertinentes et éviter d'inclure toutes les

dimensions disponibles. Cette approche permet de réduire la complexité du modèle tout en maintenant l'efficacité de la prédiction.

Cet algorithme est plus simple à concevoir que de nombreux autres ; cependant, il s'avère plus difficile à mettre en œuvre et à optimiser en termes de performance, notamment en comparaison avec le Random Forest.

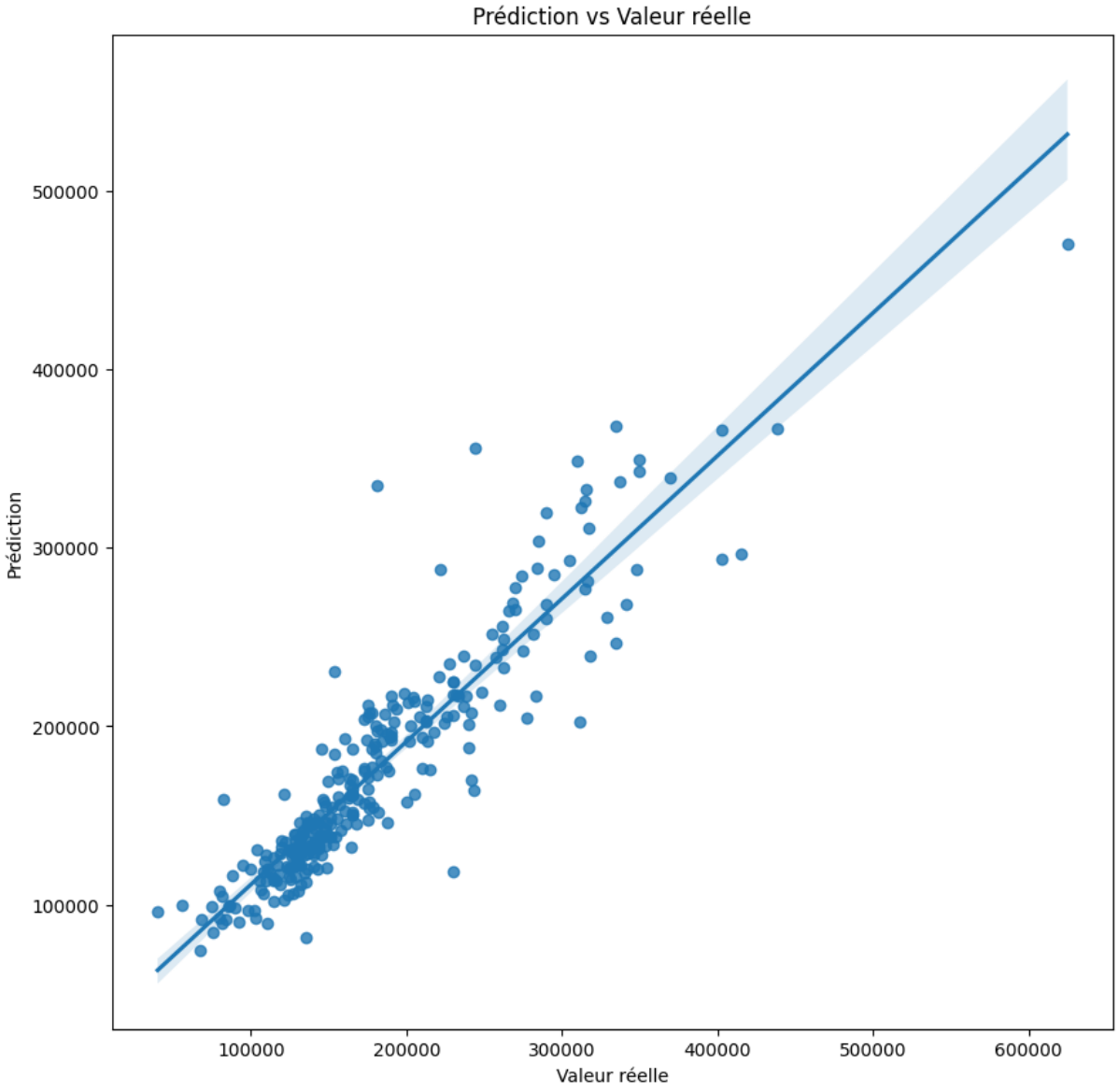


Figure 12: Prédictions VS Réelle KNN sans sélection des paramètres

Ci-dessus et comme d'habitude un graphique représentant les valeurs prédites et les valeurs réelles.

Pour approfondir notre analyse, il aurait été potentiellement intéressant de visualiser le fonctionnement du réseau KNN avec notre jeu de données. Cela impliquerait de réduire l'espace de données, initialement en plusieurs dimensions, à un espace à deux dimensions, grâce à une Analyse Factorielle des Correspondances Multiples (AFCM), comme nous l'avons fait précédemment. Autrement dit, il s'agirait de transformer cet espace de huit dimensions en un espace à deux dimensions, afin de tenter de comprendre comment notre modèle se comporte avec notre jeu de données.

Avec le modèle KNN, nous obtenons un score de 0.17841, ce qui est très satisfaisant. En effet, il est raisonnable de supposer que si deux maisons sont extrêmement similaires, leurs prix le seront également. C'est pour cette raison que ce modèle me semble pertinent.

Aller plus loin avec les KNN

Dans le cas du KNN, il existe plusieurs types d'hyperparamètres que nous pourrions ajuster, tels que les métriques de distance, les poids des voisins, l'algorithme de recherche des voisins, et le nombre de voisins, pour n'en citer que quelques-uns. Cependant, par souci de simplicité et pour gagner du temps, nous allons nous concentrer uniquement sur le nombre de voisins. Cette métrique est cruciale car si nous choisissons un nombre trop élevé de voisins, nous risquons de lisser excessivement les réponses, aboutissant ainsi à une moyenne sans grande spécificité. À l'inverse, si nous sélectionnons un nombre trop restreint de voisins, comme se baser uniquement sur le voisin le plus proche, notre modèle sera très sensible au bruit. Trouver le bon équilibre pour le nombre de voisins est donc essentiel pour optimiser la performance de notre modèle KNN.

Après un certain temps de recherche, nous avons trouvé que le nombre optimal de voisins pour évaluer le prix d'un individu est de 13. Avec ce réglage, on obtient un score de 0.17771, ce qui est une amélioration, même si je soupçonne que, en passant plus de temps et en modifiant d'autres hyperparamètres, ainsi qu'en sélectionnant d'autres dimensions, nous aurions pu améliorer davantage ce score. Cependant, j'ai l'impression qu'un algorithme de Random Forest serait probablement plus performant dans ce type de projet. Les Random Forests, en effet, ont tendance à être plus robustes face à la diversité des données, et peuvent souvent fournir de meilleurs résultats, en particulier dans des cas où les données présentent de nombreuses variables et interactions.

7 : Réseaux de neurones

Maintenant, je vais vous présenter la méthode qui m'a posé le plus de difficultés et celle qui a donné les moins bons résultats. Au moment de sa mise en œuvre, je l'ignorais, mais j'étais en plein processus d'apprentissage et d'expérimentation. Je vais donc vous expliquer le fonctionnement et l'application des réseaux de neurones.

Dans un premier temps, je me suis énormément renseigné sur le fonctionnement et la pertinence d'un réseau de neurones dans le cadre de ce projet. Il est apparu assez rapidement que cela ne serait probablement pas le modèle le plus efficace pour la tâche demandée. En effet, un réseau de neurones excelle davantage dans les tâches complexes et là où il est difficile de modéliser quelque chose avec les méthodes plus traditionnelles que nous avons vues précédemment.

Tout d'abord, avant d'aller plus loin, laissez-moi vous présenter ce qu'est un réseau de neurones. Un réseau de neurones est un modèle informatique inspiré du fonctionnement du cerveau humain. Il est composé de neurones qui sont, en réalité, des points clés à travers lesquels l'information va passer, être traitée, puis transmise au neurone suivant. Ces neurones sont répartis en couches, et chaque couche transmet l'information séquentiellement à la suivante. Le fonctionnement peut paraître complexe, mais l'essentiel à retenir est que les réseaux de neurones sont capables d'apprendre et de s'adapter en ajustant les liens entre chaque neurone pendant leur apprentissage. L'application d'un tel modèle est très vaste : il peut être utilisé pour de la régression, qu'elle soit qualitative ou quantitative, la reconnaissance d'images, la traduction de langues, etc. Ci-dessous, vous trouverez un schéma illustrant de manière simplifiée ce qu'est un réseau de neurones.

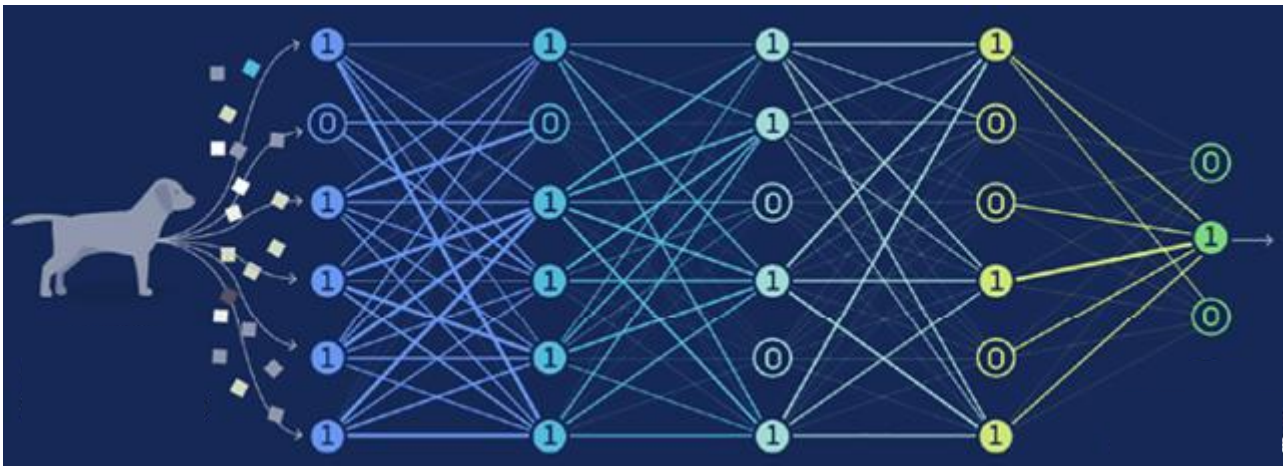


Figure 13 : Schéma d'un réseau de neurone qui décompose une image de chien

Tout d'abord, j'ai commencé par standardiser ma base de données, car cela peut contribuer à améliorer les performances de notre réseau de neurones. Ensuite, j'ai créé un premier réseau de neurones en utilisant l'architecture suivante, grâce à la bibliothèque Python Keras :

```
1. model = Sequential()
2. model.add(Dense(128, input_dim=input_dim, activation='relu'))
3. model.add(Dense(64, activation='relu'))
```

4. model.add(Dense(1))

La toute première ligne indique que notre réseau de neurones sera constitué d'un empilement séquentiel de couches.

Ensuite, nous ajoutons notre première couche, qui est une couche « dense » avec 128 neurones. Cette couche prend en entrée le nombre de colonnes de notre jeu de données, autrement dit toutes nos variables. La fonction d'activation pour cette couche de neurones sera de type ReLU.

Par la suite, nous ajoutons une deuxième couche aussi cachée, qui est également une couche « dense ». Cette fois-ci, elle comprend 64 neurones et utilise toujours une fonction d'activation de type ReLU.

Pour finir, nous ajoutons une dernière couche qui réunit toutes les informations de la couche précédente en un seul neurone. Cette couche est destinée à la prédiction de notre valeur unique. Ici, il n'y a pas de fonction d'activation prédéfinie, ce qui signifie qu'elle va utiliser par défaut une activation linéaire. C'est ainsi que nous obtenons notre prédiction de sortie.

Pour résumer, notre modèle prend en entrée toutes les caractéristiques du nouvel individu et injecte ces caractéristiques dans deux couches séquentielles cachées de neurones. Finalement, il nous fournit une prédiction avec la dernière couche.

Ici, nous utilisons la fonction d'activation ReLU car elle est à la fois plus rapide et assez simple. De manière générale, c'est l'une des fonctions les plus efficaces pour les cas de régression.

```
model.compile(optimizer='RMSprop', loss='mean_squared_error', metrics=['mae'])
```

Cette ligne concerne la configuration de l'entraînement du modèle. Tout d'abord, la partie indiquée comme 'optimiseur' représente l'algorithme utilisé pour définir les poids des neurones. Ensuite, la section 'loss' concerne la fonction de perte, utilisée pour l'optimisation du modèle au fil du temps. Elle nous permet de mesurer à quel point les prédictions sont éloignées des valeurs réelles. Enfin, la partie 'métrique' sert à évaluer directement le modèle lui-même. Pour simplifier, la fonction de perte 'loss' sert à former le modèle, la partie 'métrique' sert à en évaluer les performances.

C'est un modèle assez simple et basique, qui n'est pas une architecture spécifiquement conçue pour la régression et trouvée sur internet. Ainsi, bien qu'il ne soit probablement pas optimal, il représente déjà un bon début dans l'avancement de notre projet.

Notre modèle sera entraîné sur 30 époques, avec un 'batch size' de 16. Autrement dit, le modèle va diviser l'ensemble du jeu de données en échantillons de 16 individus. Il passera ensuite ces échantillons à travers tout le réseau pour son entraînement. Il traitera l'intégralité de ces échantillons dans notre réseau et répétera cette même opération 30 fois pour son entraînement.

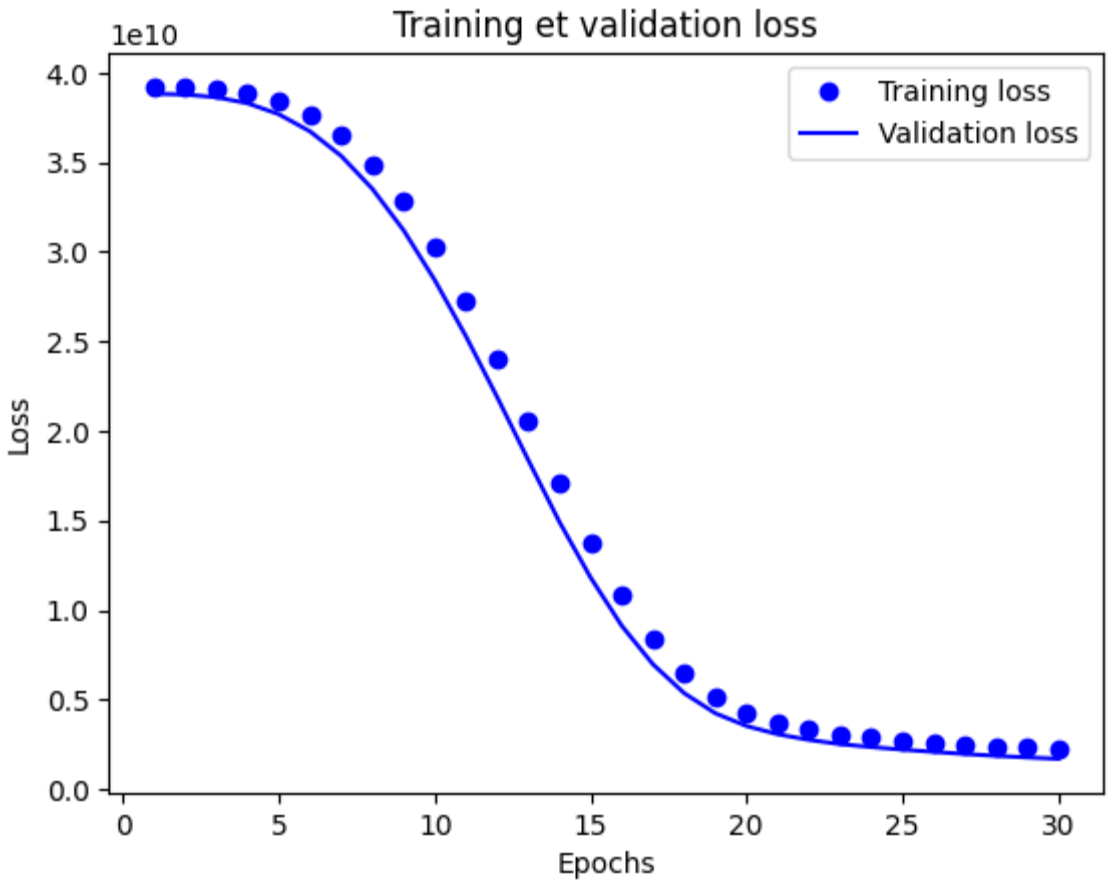


Figure 14 : Courbe d'apprentissage de notre réseau

Ci-dessus, on observe l'évolution de l'apprentissage de notre modèle. La 'train loss', c'est-à-dire les points sur le graphique, représente l'apprentissage de notre réseau de neurones. On note une diminution significative, ce qui est un bon signe car cela indique que notre modèle apprend effectivement à partir du jeu de données d'entraînement. Passée l'époque 20, on constate que l'apprentissage se stabilise et converge, ce qui signifie qu'il ne bénéficie plus autant de l'entraînement qu'auparavant. La ligne en dessous représente la 'validation loss'. Elle indique que notre modèle généralise bien et qu'il n'y a pas de surajustement ('overfitting').

Notre tout premier réseau de neurones affiche un score de 2.16158, ce qui est relativement moyen. Cela représente une erreur moyenne d'environ 25 000 dollars sur l'estimation du prix d'une maison.

Aller plus loin avec les réseaux de neurones

Comme précédemment, nous allons essayer d'approfondir notre travail avec ce type de modèle. Il existe de nombreux paramètres qui peuvent être modifiés, par exemple : la fonction d'activation, le nombre de neurones, le nombre de couches. Nous pouvons également ajuster les fonctions qui facilitent l'apprentissage et l'entraînement de notre modèle. De plus, il est possible de modifier le nombre d'époques d'entraînement et le nombre d'individus dans chaque lot ('batch size'). En résumé, il y a une multitude de paramètres modifiables, mais se concentrer sur tous en même temps serait une perte de temps.

On part du principe que notre architecture n'est pas optimale, car il existe sûrement sur internet des architectures bien plus performantes, mais qu'elle n'est pas trop complexe et devrait réussir à cerner l'information lui permettant de répondre au mieux à notre projet. De plus, on part aussi du postulat que la fonction d'activation ReLU semble être la plus performante par rapport à toutes les autres fonctions d'activation. Il en va de même pour les fonctions qui facilitent l'apprentissage et l'entraînement de notre modèle. C'est pourquoi ici, on va simplement se concentrer sur le nombre d'époques d'entraînement et la taille de l'échantillon. Rien qu'en variant ces deux hyperparamètres, il est sûrement possible d'améliorer grandement notre modèle.

Cette partie va me prendre énormément de temps, car avant d'arriver aux conclusions déjà établies plus haut, je vais réaliser des tests, me renseigner. Et ne serait-ce que modifier ces deux hyperparamètres, implique des essais et du temps. Après plusieurs tentatives, la version qui me semble être la plus intéressante est une version de notre modèle avec 20 époques et un 'batch size' de 32. Avec cette configuration, on obtient un score de 1.0052, ce qui est certes inférieur en termes de résultats, même comparé au pire autres types de nos modèles, et cela représente une erreur moyenne de 17 000 dollars sur les variations du prix d'une maison.

Ces résultats peuvent certes sembler décevants, mais il est important de rappeler que l'utilisation des réseaux de neurones est extrêmement complexe. Nous nous sommes concentrés uniquement sur la modification de deux hyperparamètres, et notre base peut certainement être améliorée. Il est également crucial de noter que ce genre d'application n'est pas où les réseaux de neurones excellent, et pourtant, malgré mon manque de connaissances, le manque d'expérimentation, et le désavantage auquel d'un réseau de neurones dans ce type d'évaluation, je reste assez impressionné par sa performance. Je suis également très intrigué par ce domaine qui, jusqu'à présent, me semblait difficilement accessible. Je réalise à quel point, avec de l'investissement et du temps, ce domaine semble être capable de tout réaliser.

Pour l'heure, c'est la fin de la partie concernant tous les modèles que j'ai eu l'occasion de mettre en place. Dans la section ci-dessous, je vais rapidement aborder les bibliothèques qui m'ont aidé lors de la mise en place de ces différents modèles.

F : Librairies complémentaires

J'ai réalisé ce projet avec bon nombre de librairies complémentaires, même si j'en ai utilisé plus je vais voici celles qui m'ont le plus servi :

- pandas
- numpy
- seaborn
- matplotlib.pyplot
- sklearn

J'aimerais surtout me concentrer sur Scikit-learn, qui m'a vraiment fait gagner beaucoup de temps grâce aux méthodes suivantes :

GridSearch : Cette méthode, elle nous servit dans presque toutes les sections « Aller plus loin ». C'est cette commande qui m'a permis de sélectionner en grande partie mes meilleurs hyperparamètres.

OneHotEncoder : Grâce à cette commande, j'ai pu créer des colonnes binaires pour chaque variable catégorielle, ce qui a permis de mieux faire fonctionner nos modèles.

SimpleImputer : Et cette commande m'a permis de gérer les valeurs manquantes dans mon jeu de données.

G : Mes projets et perspectives

Actuellement, je suis toujours en recherche de stage. Par exemple, le jeudi 24/01/2024, j'ai eu l'occasion de passer deux entretiens d'embauche : un pour un stage à la métropole de Lyon et un autre au CNAM. Le premier concerne du data mining et le second, pour être data analyst. Je continue donc activement ma recherche de stage. Quant à mes projets futurs, le monde de la data m'intéresse énormément. J'aimerais rejoindre, si possible, une grande entreprise où je pourrais réaliser des prédictions, comme je viens de le faire durant ce projet. La partie graphique et la mise en forme des données sont intéressantes mais assez limitées. C'est pourquoi réaliser seulement des dashboards sur Power BI ou tout autre outil similaire m'intéresse peu. Je trouve que dans ces domaines, l'utilité de notre travail est très éphémère. De plus, la courbe d'apprentissage me semble bien plus limitée que dans d'autres domaines, tels que la prédiction ou la classification dans les réseaux de neurones.

Le domaine que j'aimerais approfondir concerne tout ce qui touche à l'analyse textuelle et à l'analyse de sentiments. C'est un domaine que je n'ai pas encore eu l'occasion d'explorer en profondeur et qui m'intéresse grandement. Si possible, j'aimerais beaucoup combiner mes connaissances en informatique et statistique avec le domaine de la sociologie pour réaliser des études comportementales. Cela me permettrait de mettre des mots sur les comportements et de mieux comprendre les individus en fonction des données que j'ai sur eux. Cela conclut mes projets et mes perspectives actuelles.

II : Conclusion

Même si je n'ai pas eu l'occasion de réaliser une alternance durant ce premier semestre de M2, je suis en réalité très heureux d'avoir eu l'opportunité de mener à bien ce projet. Alors qu'au sein de l'Atlantic, je me limitais à la création de certains graphiques et dashboards sur Power BI, ce projet m'a permis de mettre en pratique ce qui me passionne et ce que j'ai appris durant mon master MIASHS. J'ai eu l'occasion d'approfondir mes connaissances dans des domaines qui m'intéressent réellement et de voir des résultats concrets.

Grâce au projet Kaggle et à la dynamique de ce site et de cette application, j'ai trouvé la motivation de progresser dans le classement, avançant parfois à grands pas et parfois constatant un arrêt brutal sur certains points. Ce projet a été très épanouissant pour moi, et je suis vraiment heureux d'avoir eu l'opportunité d'y participer. Maintenant que j'ai appris et progressé, je réalise toutes les erreurs commises et le temps perdu sur des problèmes qui me paraissent désormais simples. Les points de compréhension qui me semblaient auparavant compliqués me paraissent maintenant évidents. J'ai en tête de nombreux points d'amélioration et plusieurs pistes pour essayer de continuer à obtenir de meilleurs résultats et de grimper dans le classement de ce projet.

Si mon projet devait se poursuivre, je sais que j'irais plus loin dans le développement et l'amélioration de mes modèles. Tout d'abord, je me concentrerais sur le modèle AdaBoost, qui a été jusqu'à présent le plus performant pour moi. Je suis conscient qu'il y a des possibilités d'améliorer les hyperparamètres et d'autres éléments de ce modèle. De même pour l'algorithme des KNN (k-nearest neighbors), je sais que je peux explorer l'utilisation d'autres hyperparamètres que j'avais jusqu'à présent délaissés. En outre, concernant le traitement de ma base de données, je pense qu'il y a encore des éléments et des aspects que je peux améliorer.

Et surtout, même si pour ce type de projet les modèles de réseaux de neurones ne sont pas les plus performants, je me suis rendu compte qu'ils sont très vastes, complexes et complets. Je sais qu'il me reste encore de nombreuses approches à explorer et des architectures à mettre en place. Il y a beaucoup de choses à tester qui pourraient sûrement améliorer grandement les résultats de ce dernier type de modèle. La profondeur et la flexibilité des réseaux de neurones offrent un potentiel significatif pour l'innovation et l'optimisation dans des projets de deep learning.

Je sais qu'il existe des modèles que je n'ai pas encore eu l'occasion de tester et qui auraient potentiellement été plus performants. Ces modèles auraient pu m'offrir une perspective différente et contribuer à l'amélioration d'autres modèles que j'utilise actuellement.

En résumé, je suis extrêmement heureux d'avoir eu l'occasion de réaliser ce projet. J'y ai pris beaucoup de plaisir et il m'a énormément épanoui. Grâce à ce projet, j'ai beaucoup appris et j'espère avoir transmis au mieux l'enthousiasme et l'investissement que j'ai mis dans ce projet. Cela a été une expérience enrichissante qui a non seulement renforcé mes compétences techniques, mais a aussi stimulé ma curiosité et ma passion pour le domaine de la data science.

Merci pour votre lecture.

Bibliographie

Site de documentation :

- https://fr.wikipedia.org (Toutes les pages concernant les modèles cités)
- ChatGPT
- Copilot
- https://www.tensorflow.org/
- https://scikit-learn.org/

Autre type de documentation :

- Rapport semainier d'alternance
- Google image (Logo, Figure 11 et 13)
- Cours dispensés en DUT STID, License CESTAT, et Master MIASHS

Table des illustrations

Table des matières

Annexes

Lien du projet complet :

https://drive.google.com/drive/folders/1wyMw8WM3EbVOqQoHn08xU8rVkzK1DrND?usp=sharing

- AFCM.R
- GraphQuanti.jpg
- GraphQuanti.pdf
- GraphVar.jpg
- GraphVar.pdf
- 'Projet Knn.ipynb'
- 'Projet Random forest.ipynb'
- 'Projet Réseau.ipynb'
- 'Projet XG Boost.ipynb'
- 'Projet ancova.ipynb'
- Submission_Modele.csv
- Submission_Modele_Forward.csv
- TrainP.csv
- data_description.txt
- house-prices-advanced-regression-techniques.zip
- sample_submission.csv
- 'submission Adaboost Param.csv'
- 'submission Adaboost.csv'
- 'submission Knn Stand et para.csv'
- 'submission Knn Stand.csv'
- 'submission Knn.csv'
- 'submission Random Forest Param.csv'
- 'submission Random Forest.csv'
- 'submission Réseau.csv'
- 'submission XGBoost Param.csv'
- 'submission XGBoost.csv'
- submission.csv
- test.csv
- train.csv