Apprentissage automatique

CentraleSupélec - 3A

Projet : prédiction des retards de trains

2023-10-23

CentraleSupélec

Cindy Hua
Rémi Boutonnier
Romain Ageron

Table des matières

I Introduction

Nous nous plaçons dans le contexte où la SNCF souhaite évaluer un planning des trains (pour les 6 mois à venir par exemple). Nous considérons donc seulement les variables qui caractérisent les liaisons, et cherchons à prédire le retard à l'arrivée et ses causes.

Variables explicatives :

- date
- service
- gare_depart
- gare_arrivee
- duree_moyenne
- nb_train_prevu
- Longitude_gare_depart (ajoutée)
- Lattitude_gare_depart (ajoutée)
- Longitude_gare_arrivee (ajoutée)
- Lattitude_gare_arrivee (ajoutée)
- distance (ajoutée)

Varaibles cibles :

- retard_moyen_arrivee
- prct_cause_externe
- prct_cause_infra
- prct_cause_gestion_trafic
- prct_cause_materiel_roulant
- prct_cause_gestion_gare
- prct_cause_prise_en_charge_voyageurs

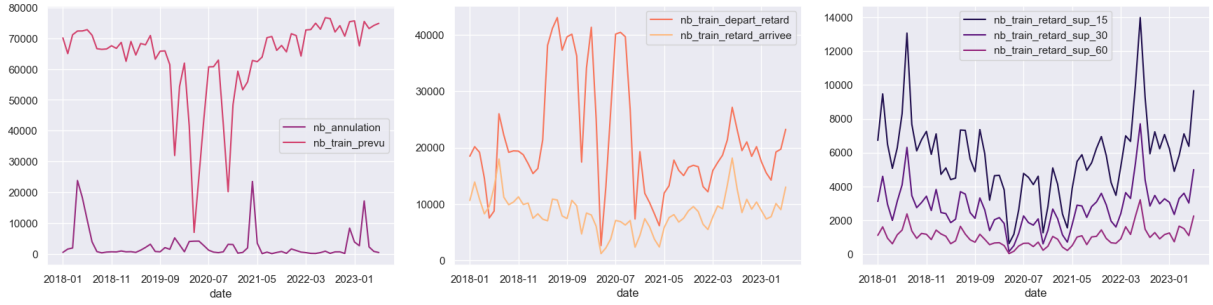
Cela ne laisse pas beaucoup de variables pour la prédiction, mais nous pourrions potentiellement utiliser d'autres données. Par exemple, la météo pourrait être liée au retard d'un train. Cependant, dans notre contexte cela n'est pas possible car il faudrait prédire la météo des 6 prochains mois.

II Analyse des données

II.1 Analyses

II.1.1 Analyse temporelle des variables

Nous avons commencé par une analyse temporelle des variables d'entrée et de sortie :



(a) Figure 1 (b) Figure 2 (c) Figure 3

FIGURE 2.1 – Analyse temporelle : de gauche à droite, évolution du nombre de trains prévus et annulés, du nombre de trains retardés, du nombre de trains retardés selon la catégorie de retard

On note notamment 3 pics du nombre de trains annulés en :

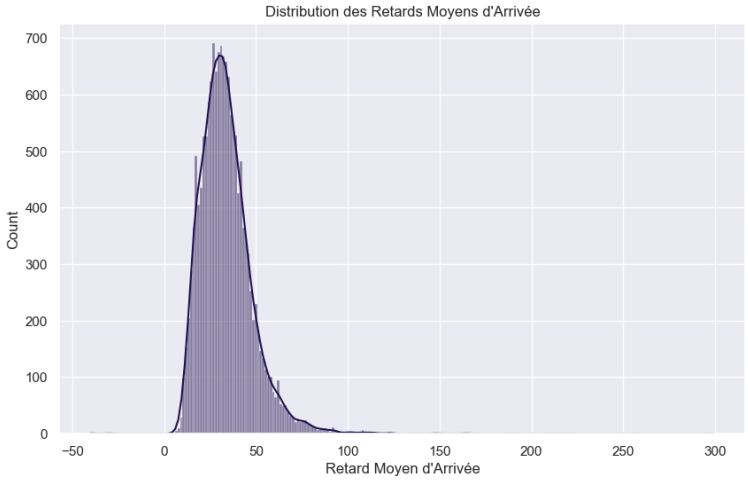
- avril et mai 2018 : grève des cheminots
- avril 2021 : période de confinement du 3 avril au 3 mai 2021
- mars 2023 : grève contre la réforme des retraites

On remarque que le premier confinement n'a pas été si problématique pour les annulations de train car la SNCF avait déjà prévu moins de trains.

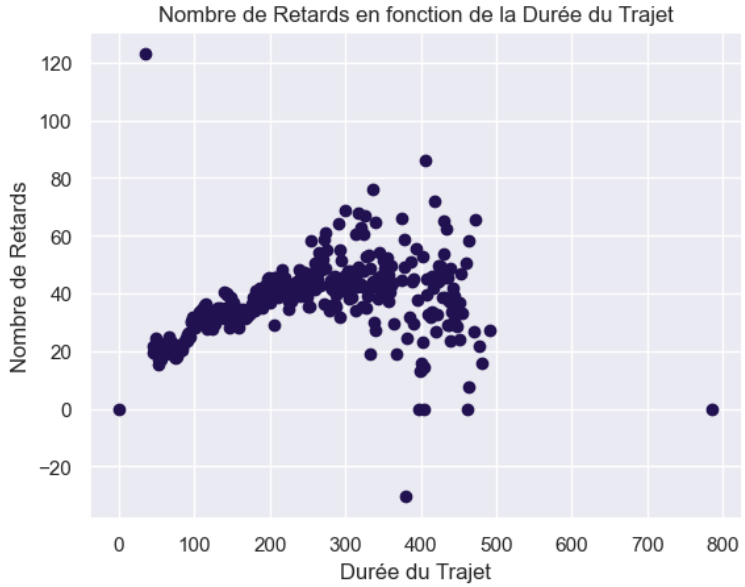
De même pour les retards de train, on note une grosse fluctuation autour du début de la période COVID, sans doute dû à un manque de personnels entrainant des contraintes à tous les niveaux.

II.1.2 Analyse de la répartition des variables

Nous avons également étudié la répartition des variables en terme de proportion. En particulier, nous trouvons une répartition des retards moyens très semblables à une loi de Poisson :



Nous pouvons aussi nous demander si les trajets longs sont plus amenés à rencontrer des retards :



Jusqu'à des trajets de 4 heures, il semblerait qu'on pourrait modéliser le nombre de retards en fonction de la durée de trajet comme une fonction linéaire. Pour des trajets plus longs, tout paraît possible : de très gros retards comme de l'avance.

II.1.3 Matrice de corrélation

Nous avons voulu voir si certaines corrélations pouvaient être intéréssantes à discuter dans la grande matrice de corrélation :

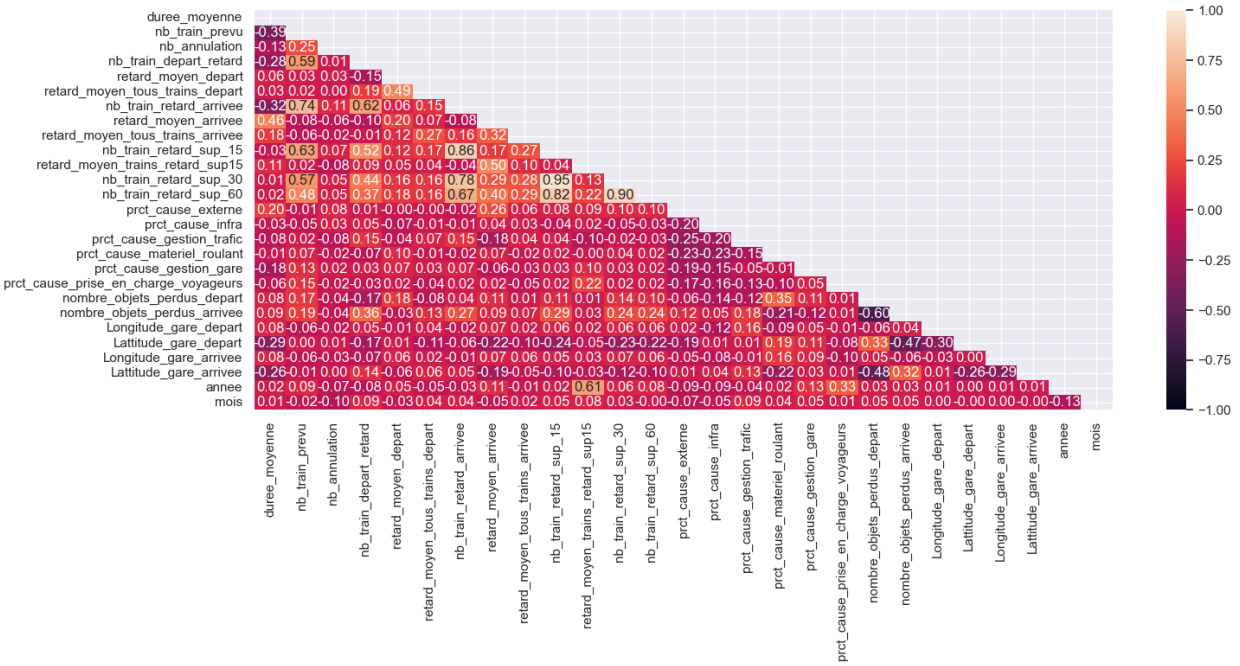


FIGURE 2.2 – Matrice de corrélation de toutes les variables du dataset

Certaines grosses valeurs de corrélation sont prévisibles, par exemple celles reliant les différentes variables de retard. L'année est corrélée positivement au retard moyen puisque nous avons vu que l'année du COVID a été une année très importante pour les retards. D'autres relations ont été mise en avant comme la corrélation du retard moyen à l'arrivée et de la durée moyenne du trajet (corrélation de 0.46).

II.2 Ajout de données

Afin de compléter la base de données, nous avons cherché d'autres datasets sur le site de la SNCF : l'une contenant les coordonnées GPS de chacune des gares , l'autre référençant l' historique des objets perdus dans les gares. La première nous permettra d'avoir la distance entre deux gares (en plus d'avoir déjà la durée moyenne entre les deux gares) tandis que la seconde permettra peut-être d'expliquer certaines causes de retard, notamment celles liées aux bagages abandonnés et au plan vigipirate.

II.3 Preprocessing

II.3.1 Features numériques

Les pourcentages ont été normalisés entre 0 et 1. Les durées moyennes, nombres de trains prévus, distances et coordonnées GPS ont été centrés-réduits. La date était normalisée de manière affine pour avoir janvier 2018 = -1 et décembre 2022 = 1.

II.3.2 Features catégorielles

Les service, gare départ, gare d'arrivée et les mois ont été encodés par un one-hot encoding dans un premier temps puis de manière différente (voir II.3.3).

II.3.3 Feature engineering

Ajout des mois : Même si l'information des mois est contenu dans la "partie décimale" de l'année, cela n'est pas vraiment accessible pour les modèles. Ajouter les mois avec un one-hot embedding donne accès facile à cette information pour les modèles. En pratique cela se traduit par une diminution de la rmse de 5% à 10% selon les modèles et les hyperparamètres. Cet embedding donne une séparation linéaire des différent mois (bien pour la régression linéaire par exemple) et est pratique pour les modèles basés sur des arbres de décisions (facile de faire une décision sur le mois).

L'un des problèmes avec cet embedding pour les KNNs est que tous les mois sont équidistants. Il semble pourtant raisonnable de supposer que le mois de Juillet est plus similaire au mois d'Août qu'au mois de Février. D'où l'embedding des mois sur un cercle : $f(mois) = \exp(i * mois * \pi/6)$. Cela donne en pratique des résultats équivalents pour le KNN la plupart des autres méthodes, et des résultats nettement dégradés pour la régression linéaire (on ne peut plus prendre une contribution indépendante pour chaque mois). Un autre avantage est celui d'utiliser 2 dimensions au lieu de 12.

Importance des gares : Cette partie ne concerne que le KNN puisque les autres méthodes utilisées ne sont pas sensibles aux ordres de grandeurs relatifs des différentes features d'entrée.

On peut supposer que pour deux liaisons avoir la même gare de départ et / ou d'arrivée peut avoir une forte influence sur les variables que l'ont souhaite prédire. Ainsi, dans le calcul de la distance dans le KNN, plutôt que de pénaliser la distance entre deux vecteurs par 0 si c'est la même liaison, 1 si c'est la même gare d'arrivée ou (exclusif) de départ, et 2 sinon, on peut utiliser un coefficient λ de manière à accorder plus ou moins d'importance aux gares composant la liaison vis-à-vis des autres features afin de définir une meilleur notion de proximité pour le KNN. Plus généralement, on n'est pas obligé de se limiter à l'utilisation du même λ pour la gare de départ ou d'arrivée. On va donc encoder les gares de départ par $\alpha \times one - hot$ et les gares d'arrivée par $\beta \times one - hot$. La figure 2.3 montre l'erreur en fonction de ces paramètres.

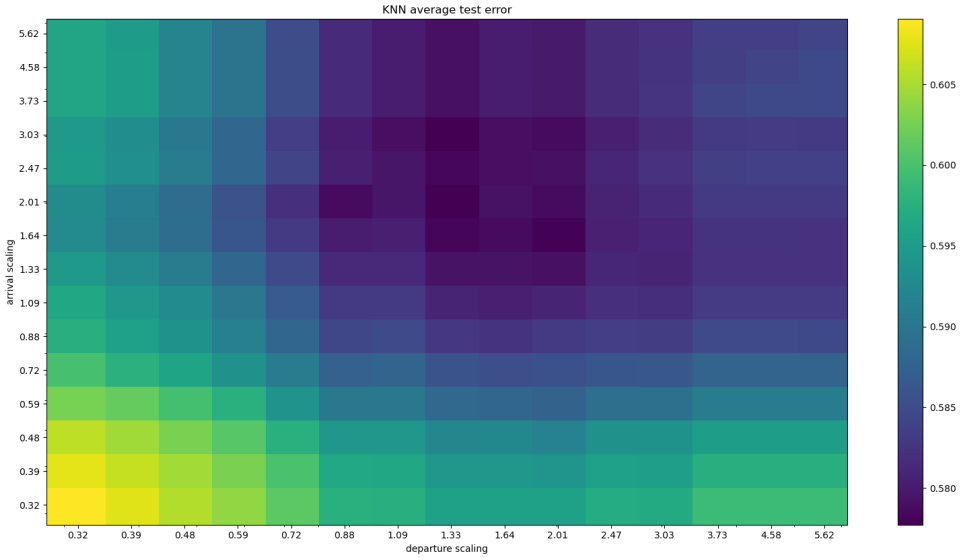


FIGURE 2.3 – Erreur moyenne de test selon α (abscisses) et β (ordonnées)

On observe qu'augmenter ces coefficients permet effectivement de diminuer l'erreur. Par ailleurs, on observe que la gare d'arrivée (coefficient plus élevé) semble avoir plus d'influence sur le résultat que la gare de départ. Les coefficients optimaux obtenus sont $\alpha = 4/3, \beta = 2$. Cela permet une (faible) amélioration, diminuant la rmse moyenne d'environ 2%.

Similarité des liaisons : L'attribution d'un coefficient différent pour les gares d'arrivée et de départ est en fait une manière de définir une similarité entre les différentes liaisons : puisque en pratique $\beta > \alpha$ on considère que partager une gare d'arrivée rend deux liaisons plus similaires que partager une gare de départ, et que n'avoir aucune gare en commun rend deux liaisons encore plus dis-similaires. On pourrait aller plus loin en ajoutant des informations extérieures pour définir la similarité ; un Paris-Valence est plus similaire à Paris-Lyon (une partie du trajet est commune) qu'à un Paris-Le Havre ; ou alors en apprenant une similarité pour chaque paire de gare. Cependant ces approches ne sont pas convenables en pratique. Un bon proxy pour ce score de similarité est la distance entre les labels deux liaisons dans les données de test : $embedding(liaison) = moyenne(labels\ de\ la\ liaison\ au\ mois\ M \mid M \in training)$. En pratique, les résultats sont équivalents et cela permet d'utiliser un embedding de dimension 7 au lieu de 118 (2 pour le one-hot sur le service, 58 pour la gare de départ, 58 pour la gare d'arrivée). Cela a pour conséquence de fortement accélérer l'entraînement des modèles.

Pour aller plus loin, apprentissage d'un produit scalaire : La réduction de la dimension de la dimension des données d'entrée (14 grâce à l'embedding des mois sur un cercle et l'embedding des liaisons en utilisant les labels) rend possible l'apprentissage de ce produit scalaire. Optimiser pour S la matrice symétrique définie positive d'un produit scalaire quelconque demanderait d'optimiser beaucoup de paramètres (91) pour une fonction de coût peu sympathique (espérance de l'erreur du KNN en test pour la distance euclidienne associée à S). En se restreignant aux matrices S diagonales le nombre de paramètres descend à 13 (il y a 14 coefficients, mais le la fonction de coût est invariante par rescaling des paramètres par un facteur $\lambda > 0$) : il devient donc envisageable de l'optimiser, par exemple avec un algorithme génétique (pour l'espérance, on peut l'estimer de manière empirique, par exemple avec de la cross-validation). Nous n'avons pas implémenté cette approche.

Pour aller plus loin, dépendance temporelle et dépendance "réseau" : Les données possède une dépendance temporelle ainsi qu'une dépendance "réseau" (deux lignes peuvent utiliser des infrastructures communes). Il pourrait être intéressant de développer des features et mêmes des modèles prenant ces aspects en compte (par exemple modéliser le réseau par un graphe et utiliser des méthodes de machine learning adaptées aux graphes).

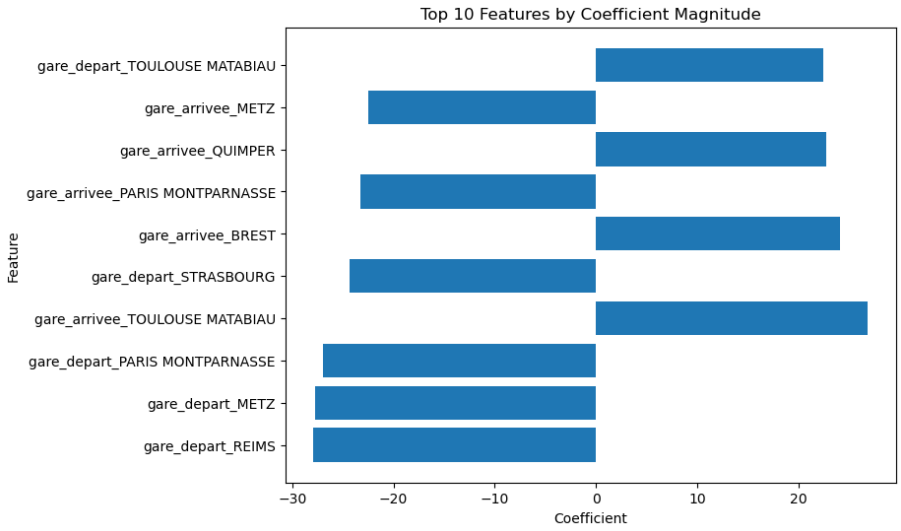
II.4 Utilisation de paramètres de modèles

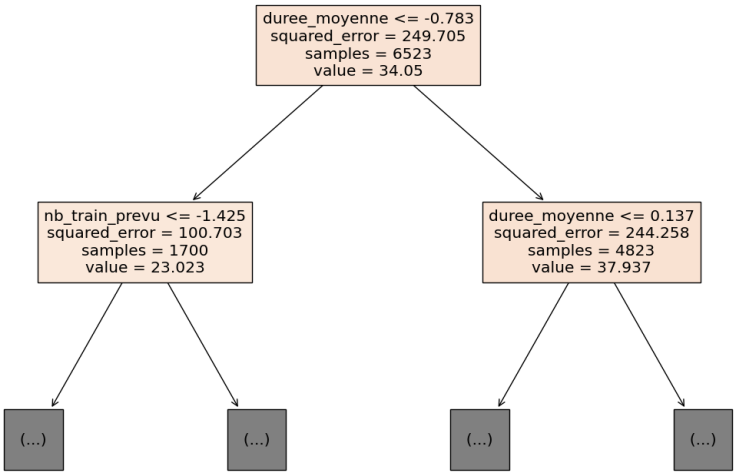
Pour se donner une idée de l'importance de certaines features, nous avons utilisé des paramètres de modèles :

- les poids d'une régression linéaire
- les noeuds d'un arbre de décision

L'idée est que les poids les plus élevés (en valeur absolu) d'une régression linéaire traduisent les features ayant le plus d'importance dans la prédiction. L'idée est similaire pour les premiers noeuds de l'arbre de décision.

On obtient ces résultats :





Pour les poids de la régression linéaire, on voit que ce sont des gares issues du one hot encoding qui sont les plus importantes. Par exemple, le départ / l'arrivée de / à Toulouse Matabiau a fortement tendance à augmenter le retard. Pour l'arbre de décision, c'est la durée moyenne et le nombre de trains qui paraissent être les features les plus importantes.

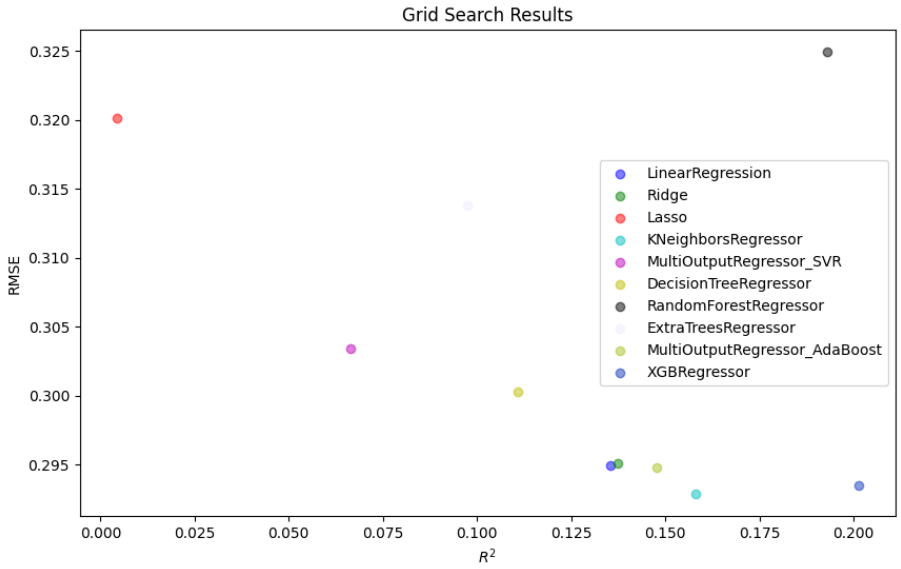
III Sélection de modèles

Nous avons testé différents modèles pour ce problème de régression :

- Régression linéaire
- Régression linéaire avec régularisation Ridge
- Régression linéaire avec régularisation Lasso
- k Nearest Neighbors (kNN)
- Support Vector Machine (SVM)
- Arbre de décision
- Random Forest
- Extremely randomized trees
- AdaBoost
- Extreme Gradient Boosting

III.1 Grid Search

Pour sélectionner les modèles en testant différentes combinaisons d'hyperparamètres, nous avons utilisé une méthode de Grid Search avec cross-validation. Nous évaluons les modèles avec leur R^2 et leur RMSE (Root Mean Squared Error) sur un ensemble de test séparé des données d'entraînement. Voici les résultats dans un tableau ainsi que sous forme de nuage de points.



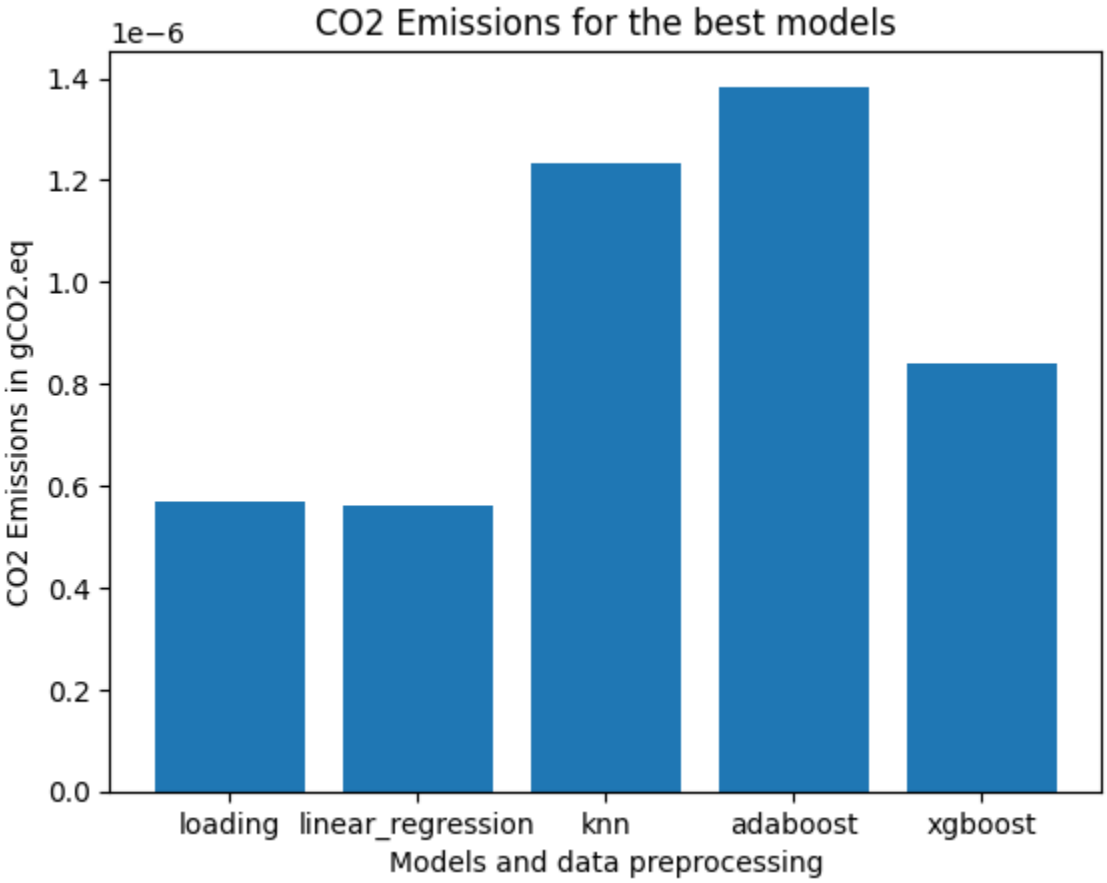
Model *Best parameters*	R^2	**RMSE**
LinearRegression *'fit_intercept' : True*	0.135	0.295
Ridge *'alpha' : 1.0*	0.138	0.295
Lasso *'alpha' : 0.1*	0.004	0.320
KNeighborsRegressor *'n_neighbors' : 76, 'weights' : 'distance'*	0.158	0.293
MultiOutputRegressor_SVR *'estimator__C' : 0.1, 'estimator__kernel' : 'linear'*	0.066	0.303
DecisionTreeRegressor *'max_depth' : 60, 'min_samples_leaf' : 40, 'min_samples_split' : 2*	0.111	0.300
RandomForestRegressor *'n_estimators' : 200, 'max_depth' : 50*	0.193	0.325
ExtraTreesRegressor *'n_estimators' : 200, 'max_depth' : 10*	0.097	0.314
MultiOutputRegressor_AdaBoost *'estimator__n_estimators' : 10, 'estimator__learning_rate' : 0.01*	0.148	0.295
XGBRegressor *'n_estimators' : 100, 'learning_rate' : 0.1, 'max_depth' : 3*	0.201	0.293

On remarque d'abord que tous les modèles ont des R^2 très faibles (maximum autour de 0.2), ce qui veut dire que nos modèles n'arrivent globalement pas à expliquer la variance de nos variables cibles. Nos variables explicatives ne sont donc sûrement pas suffisantes pour prédire correctement nos variables cibles.

Parmi les modèles testés, les meilleurs sont le XGBRegressor, le kNN, Adaboost, et la régression linéaire (avec ou sans Ridge). Le XGBRegressor atteint le meilleur R^2, mais a une RMSE équivalente au kNN et même à la régression linéaire. En tenant compte de l'argument qu'un modèle simple généralisera mieux qu'un modèle plus complexe en temps général, la régression linéaire pourrait être choisie comme meilleur modèle ici.

IV Ouverture : émissions CO2

Les émissions CO2 sont rarement un facteur pris en compte lors de la mise en place de solution de Machine Learning, alors que c'est un facteur important dans le contexte actuel. Voici donc un aperçu des émissions de CO2 enregistrées pour nos modèles les plus performants en terme d'erreur :



Cela permet d'avoir un 2ème critère de choix pour choisir le modèle. "loading" correspond aux ressources nécessaires pour charger les données et les transformer à travers toute la pipeline de preprocessing. C'est plus de ressources que pour entraîner et effectuer une prédiction sur l'ensemble de test pour la régression linéaire. Ici, on choisirait donc plutôt une régression linéaire car elle consomme moins de ressources pour être entraînée et faire la prédiction que les autres modèles, pour une performance équivalente. On s'y attendait car c'est un modèle très simple. Dans le cas où aucun modèle simple ne rivalise avec des modèles plus complexes, il peut être intéressant de voir si un compromis entre performance et ressources nécessaires est possible.