# Méthode d'exploration de données pour la prédiction du prix de maison

présenté à

Guy Wolf

STT 3795

Yi Xuan Luo
Cheng Chen
Weiyue Cai

24 juin 2021

# 1 Introduction

Notre équipe analyse un ensemble des données de biens immobiliers disponible sur Kaggle [1] et on s'intéresse aux prix des maisons et leur prédiction. Avec 79 variables explicatives décrivant (presque) tous les aspects des maisons résidentielles à Ames, Iowa, cet ensemble des données nous permet de prédire les prix des maisons. Nous voulons étudier les facteurs qui influencent le prix de maison. En nous basant sur les résultats obtenus, nous allons choisir les modèles de prévision appropriés pour prédire les prix des maison à Ames.

# 2 Compréhension des données

Les données *KaggleHousePrice* comportent 1460 observations avec 80 attributs parmi lesquelles 37 attributs sont quantitatifs et 43 sont qualitatifs.

```
# numerical attributes
num_cols = train.columns[train.dtypes != 'object']
num_cols
```

```
Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual',
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1',
       'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd',
       'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF',
       'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea',
       'MiscVal', 'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')
```

```
# categorical attributes
cat_cols = train.columns[train.dtypes == 'object']
cat_cols
```

```
Index(['MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilities',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st',
       'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation',
       'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
       'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
       'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual',
       'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature',
       'SaleType', 'SaleCondition'],
      dtype='object')
```

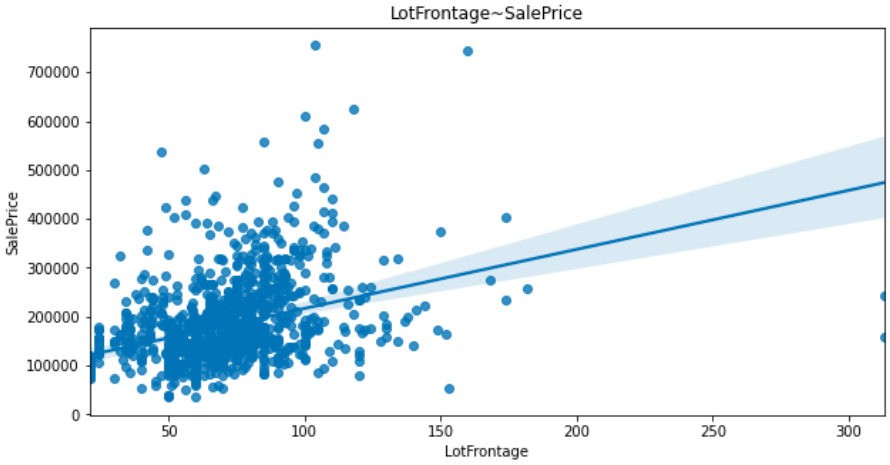
Sans tenir compte des attributs, la moyenne du prix des maisons est 180921.195890. Le prix minimum est 34900, le maximum est 755000. La médiane est 163000, Q1 et Q3 sont respectivement 129975 et 214000. La différence entre la moyenne et la médiane est grande, ce qui montre qu'il existe des valeurs aberrantes.

1. https://www.kaggle.com/c/house-prices-advanced-regression-techniques

# 3 Nettoyage des données

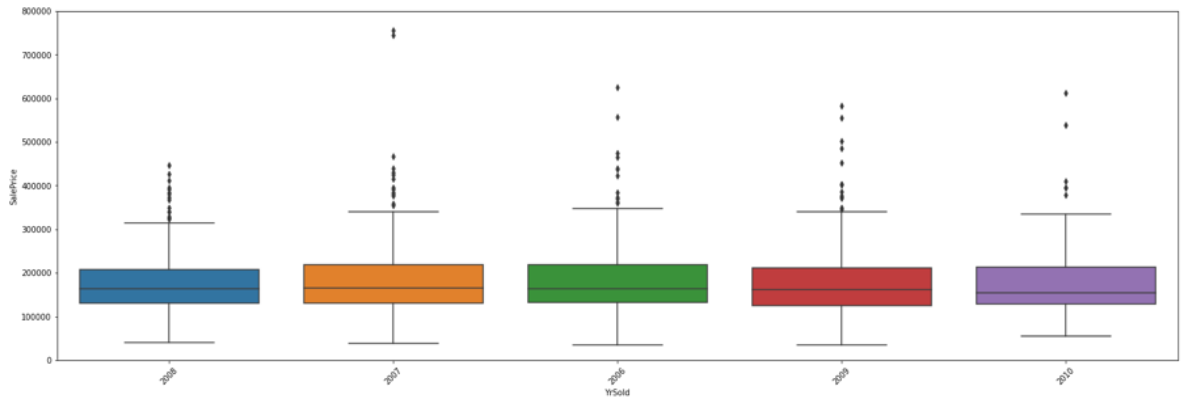
Avant de visualiser les données pour déterminer la corrélation entre les variables et identifier les caractéristiques significatives, on doit nettoyer les données en premier afin d'obtenir un meilleur résultat.

On a tout d'abord enlevé des observations aberrantes à l'aide des graphiques.

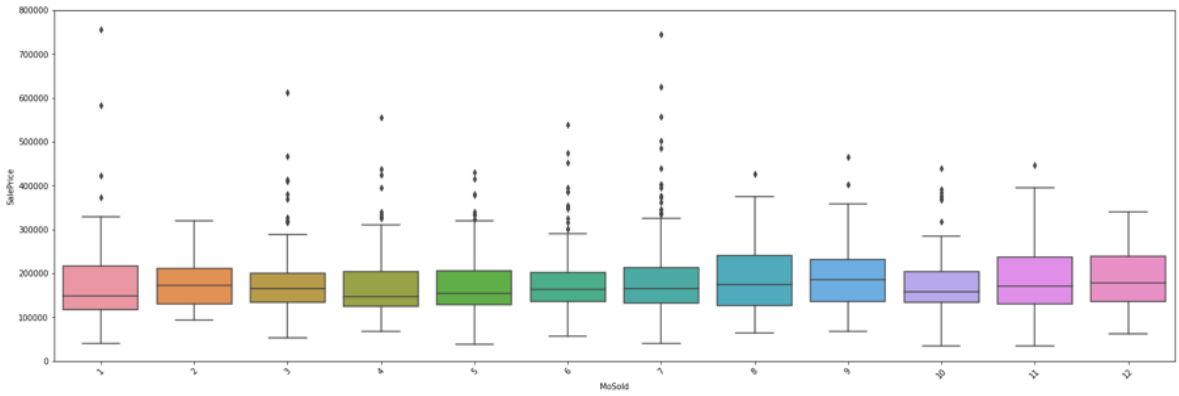



Ensuite, on a traité les valeurs manquantes en appliquant les stratégies suivantes :
(1) remplacer les données manquantes par une valeur fixe
(2) pour les attributs qualitatifs, remplacer les données manquantes par 0, la moyenne ou la médiane.
(3) pour les attributs quantitatifs, remplacer les données manquantes en fonction de la description des données ou les remplacer par la valeur qui apparaît le plus souvent. (Voir le code en annexe pour plus de détails).

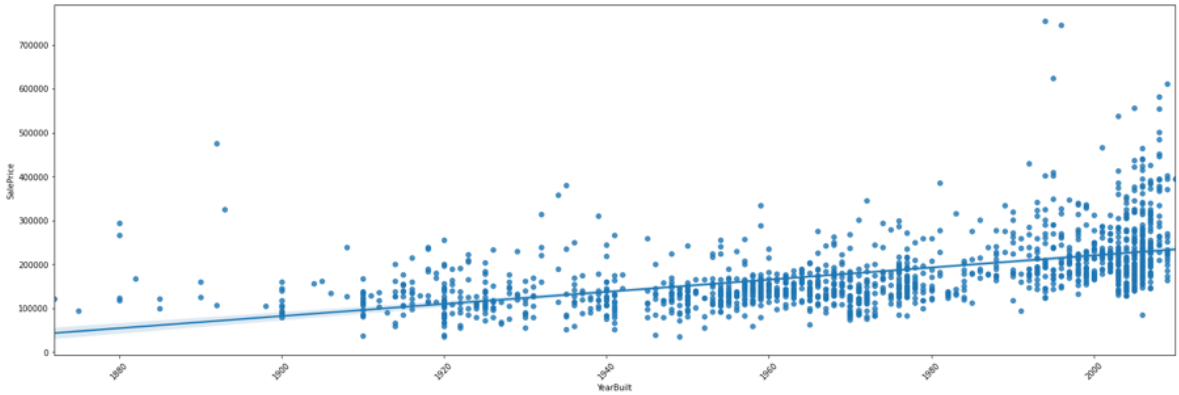
Finalement, on transforme certains attributs quantitatifs en attributs qualitatifs. Par exemple, pour la caractéristique *YrSold*, la graphique souligne les prix de maison en 2008, 2007, 2006, 2009 et 2010 sont pas mal constants.



On transforme ainsi l'année de vente en une variable catégorielle. (On fait la même chose pour le mois de vente).




Par contre, pour la graphique qui montre la corrélation entre le prix de maison et l'année de construction, on peut constater que le prix de maison a une tendance à monter avec l'année de construction. Autrement dit, plus la maison est neuve, plus possible son prix est haut. Alors, on le laisse comme ça.




# 4 Visualisation des données

## 4.1 Analyse de la Variance (ANOVA)

Comme nous disposons des variables catégorielles, on va utiliser l'Analyse de la Variance (ANOVA) comme la méthode statistique pour sélectionner les variables qui ont un impact significatif sur le prix.

L'approche de l'ANOVA consiste à effectuer un test F pour vérifier s'il existe une variance entre les groupes (i.e. les différentes variables catégorielles) en comparant la

variance entre les groupes et la variance au sein des groupes [2].

On peut calculer la somme des carrés entre les groupes (SSB) avec son degré de liberté (dfb) et la somme des carrés au sein des groupes(SSW) avec le degré de liberté associée (dfw). Puisqu'on veut comparer la variance entre les groupes et la variance au sein des groupes, on peut calculer la statistique F =

$$\frac{SSB/dfb}{SSW/dfw}$$

.
Ensuit, on va utiliser la statistique F, la loi de Fisher, pour tester l'hypothèse nulle H0 : la variance est identique pour les différents groupes. Si la valeur-p est inférieure au seuil, mettons 0,005, alors on rejette H0.

On observe qu'il y a pas mal de variables qui ont une valeur-p beaucoup plus inférieure à 0,005, p.e., la variable *neightberood* a 2.151365e-225 comme la valeur-p.

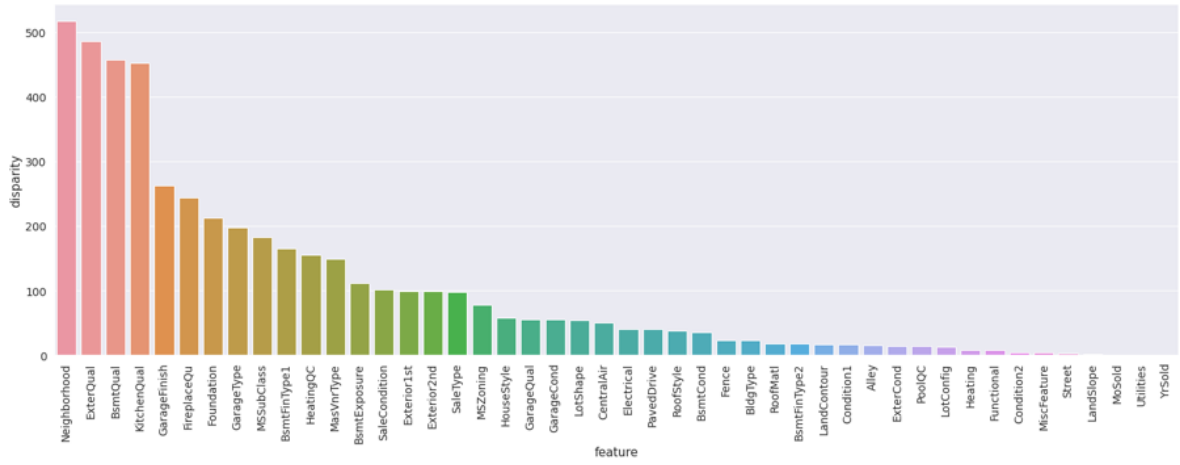
| | feature | pval | disparity |
|---|---|---|---|
| 9 | Neighborhood | 2.151365e-225 | 517.315543 |
| 19 | ExterQual | 5.868253e-212 | 486.378483 |
| 22 | BsmtQual | 3.359597e-199 | 457.002613 |
| 31 | KitchenQual | 3.691658e-197 | 452.303188 |
| 35 | GarageFinish | 9.964625e-115 | 262.498244 |
| 33 | FireplaceQu | 1.336345e-106 | 243.784082 |
| 21 | Foundation | 8.391803e-93 | 212.013158 |
| 34 | GarageType | 9.398860e-87 | 198.084315 |
| 0 | MSSubClass | 4.661331e-80 | 182.667506 |
| 25 | BsmtFinType1 | 1.704168e-72 | 165.253050 |
| 28 | HeatingQC | 2.548399e-68 | 155.640321 |

Cinq variables, *Street*, *LandSlope*, *MoSold*, *Utilities*, *YrSold*, ne sont pas significatives. Nous enlevons donc ces variables non significatives.

| | | | |
|---|---|---|---|
| 27 | Heating | 4.279670e-04 | 7.756465 |
| 32 | Functional | 5.406450e-04 | 7.522748 |
| 11 | Condition2 | 1.354969e-02 | 4.301392 |
| 41 | MiscFeature | 1.691842e-02 | 4.079352 |
| 2 | Street | 4.749178e-02 | 3.047199 |
| 8 | LandSlope | 2.840375e-01 | 1.258649 |
| 42 | MoSold | 4.694076e-01 | 0.756284 |
| 6 | Utilities | 5.866737e-01 | 0.533286 |
| 43 | YrSold | 6.197179e-01 | 0.478491 |

2. https://towardsdatascience.com/anova-for-feature-selection-in-machine-learning-d9305e228476

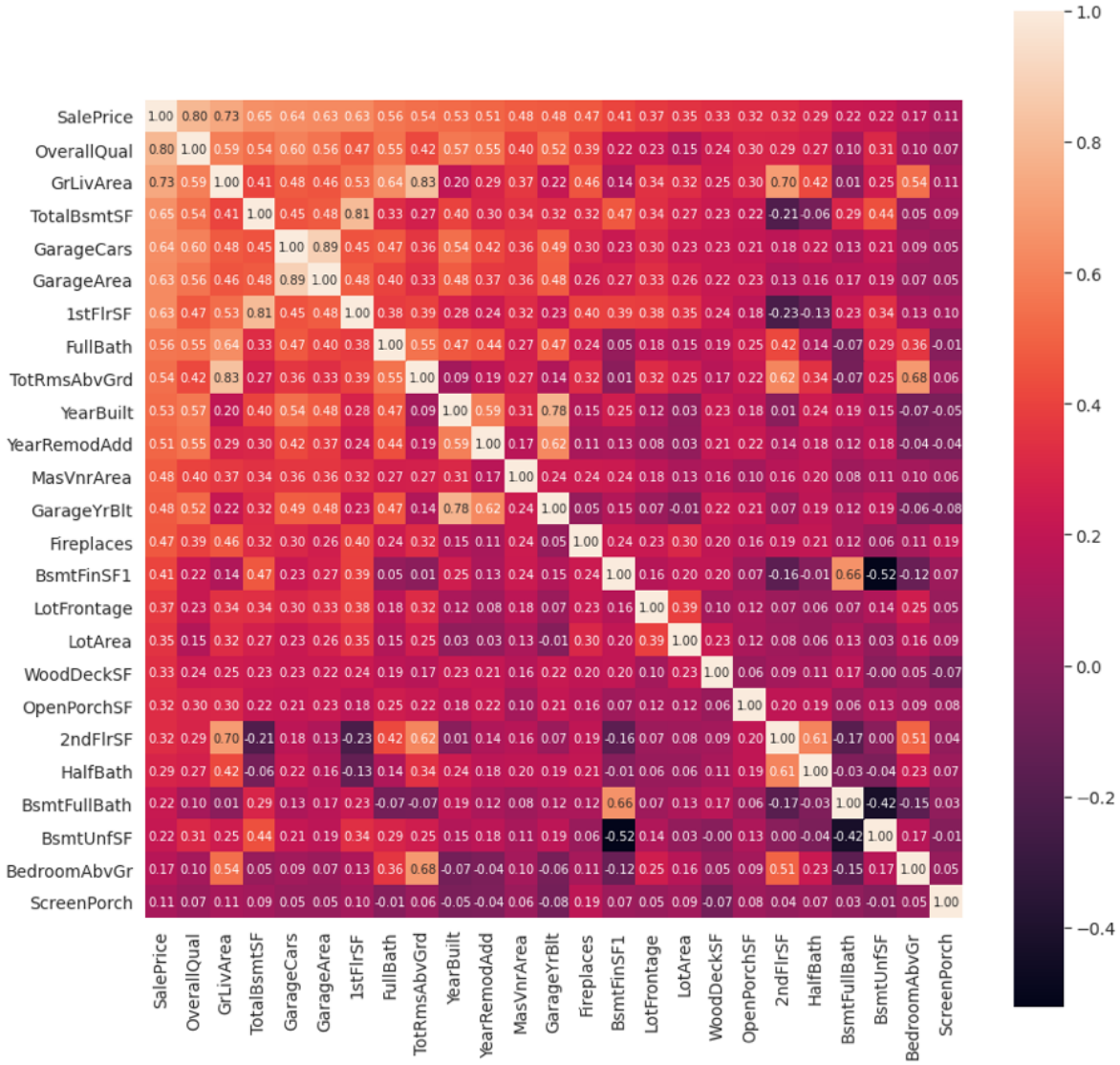
Avec l'utilisation de ANOVA, la graphique suivante montre une estimation de l'influence des attributs qualitatifs sur le prix de maison. La location de maison (*neighborhood*), la qualité de matériel de construction de maison à l'extérieur (*ExterQual*) et la qualité de sous-sol (*BsmtQual*) sont trois facteurs qualitatifs ayant plus de impact sur le prix de maison.




## 4.2 *Heatmap*

Nous avons déjà analysé l'influence des attributs qualitatifs sur le prix. Pour connaître la corrélation entre les attributs quantitatifs et le prix de la maison, on utilise *HeatMap* afin de visualiser le coefficient de la corrélation entre deux variables.

Le tableau ci-dessous affichent les 24 attributs quantitatifs ayant une corrélation plus grande avec le prix de vente :

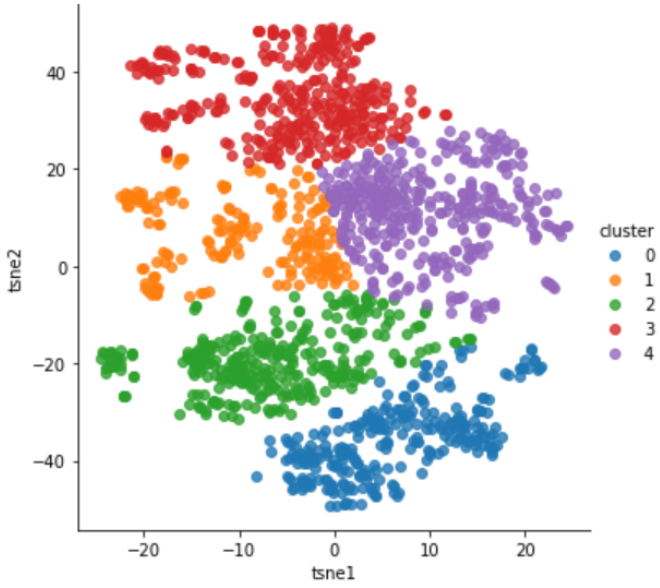



Les attributs *OverallQual*, *GrlivArea*, *TotalBsmtSF*, *GarageCar* ont plus d'influence sur le prix d'une maison par rapport aux autres attributs. Ils ont tous une corrélation positive sur le prix.

## 4.3 Réduction des dimensions

Comme ce qu'on a déjà vu en classe, la méthode de la réduction des dimensions, entres autres l'analyse en composantes principales (PCA), la notion de variété (*manifold*) comme Isomap, MDS, TSNE et l'approche de partitionnement de données (*cluster*) telle que K-means sont des outils utiles et appropriés pour visualiser des données avec plusieurs attributs.

Voici une illustration de cluster des données de notre projet :




# 5 Pré-traitement des données

Après avoir visualisé les données, on s'intéresse maintenant au pré-traitement des données. On a corrigé en premier les valeurs avec une distribution asymétrique au moyen de la normalisation des certaines variables quantitatives avec la transformation *log1p* implémentée dans python. Par la suite, on entame le processus de *feature engineering* en créant de nouvelles variables. Par exemple, pour prédire le prix de maison, on rajoute les caractéristiques suivantes pour améliorer la performace des modèles de prédiction : *isHouseOld*, *isRemodel*, *hasPool*, *has2ndfloor*, *hasGarage* et *hasBasement* (Voir le code en annexe pour plus de détails).

# 6 Modèles de prédiction

Cette partie porte sur l'analyse des modèles de prédiction utilisés pour prédire le prix de maison. On va parler principalement de quatre méthodes : la régression bayésienne (plus précisément, *bayesian ridge regression*), la machine à vecteurs de support pour la régression, l'arbre de décision pour la régression et la forêt aléatoire pour la régression.

## 6.1 Régression bayésienne (*Bayesian Ridge Regression*)

Avant de parler de la *bayesian ridge regression*, on va d'abord se rappeler la régression linéaire. La régression linéaire ajuste un modèle linéaire avec des coefficients $w = (w_1, w_2, ..., w_p)$ afin de minimiser la somme résiduelle des carrés entre les cibles observées dans l'ensemble de données (y) et les cibles prédites par l'approximation linéaire $(Xw)$ [3].

Mathématiquement, il résout un problème de la forme :

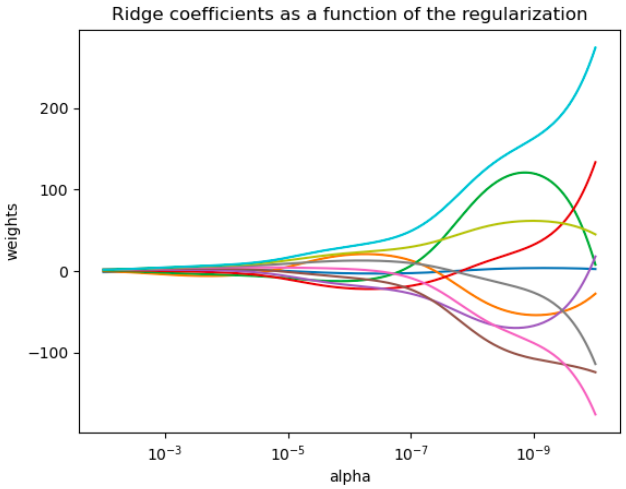
$$\min_w ||Xw - y||_2^2$$

### 6.1.1 *Ridge Regression*

La *ridge regression* résout certains des problèmes des moindres carrés ordinaires en imposant une pénalité sur les coefficients. Les *ridge coefficients* minimisent la somme résiduelle des carrés et $\alpha||w||_2^2$ :

$$\min_w ||Xw - y||_2^2 + \alpha||w||_2^2$$

Le paramètre de complexité $\alpha \geq 0$ contrôle la quantité de rétrécissement : plus la valeur de $\alpha$ est grande, plus le rétrécissement est important, ce qui implique que plus les coefficients sont robustes à la colinéarité.




(Source : scikit-learn 1.1.2.1. Regression [4])

3. https://scikit-learn.org/stable/modules/linear_model.html#ordinary-least-squares

4. https://scikit-learn.org/stable/modules/linear_model.html#ridge-regression-and-classification

### 6.1.2 *Bayesian Ridge Regression*

La régression linéaire consiste à appliquer la méthode de l'inférence bayésienne dans la régression linéaire. Lorsque le modèle de régression a des erreurs qui ont une distribution normale, et si on suppose une forme particulière de distribution de la probabilité antérieure, alors on peut essayer d'obtenir la distribution de probabilité postérieure des paramètres du modèle.

On suppose que la sortie $y$ est distribuée de manière gaussienne autour de $Xw$ [5] :

$$p(y|X, w, \alpha) = N(y|Xw, \alpha)$$

où $\alpha$ est considéré comme une variable aléatoire.

Pour la *bayesian ridge regression*, la probabilité antérieure du coefficient w est donnée par une gaussienne sphérique :

$$p(w|\lambda) = N(w|0, \lambda^{-1} I_p)$$

où $\alpha$ et $\lambda$ suivent la Loi Gamma.

Les paramètres w, $\alpha$ et $\lambda$ sont estimés conjointement pendant l'entraînement du modèle. Les paramètres de régularisation $\alpha$ et $\lambda$ sont estimés en maximisant la vraisemblance marginale logarithmique. [6]

La régression bayésienne permet d'éviter le problème de sur-ajustement (*over-fitting*) du maximum de vraisemblance, ce qui fait que cette approche s'adapte bien aux données. Voici une illustration de la distribution de la prédiction pour les modèles de régression linéaire bayésienne. [7].

5. `https://scikit-learn.org/stable/modules/linear_model.html#bayesian-regression`
6. `https://scikit-learn.org/stable/modules/linear_model.html#bayesian-regression`
7. Bishop, Christopher. Pattern Recognition and Machine Learning, p.157

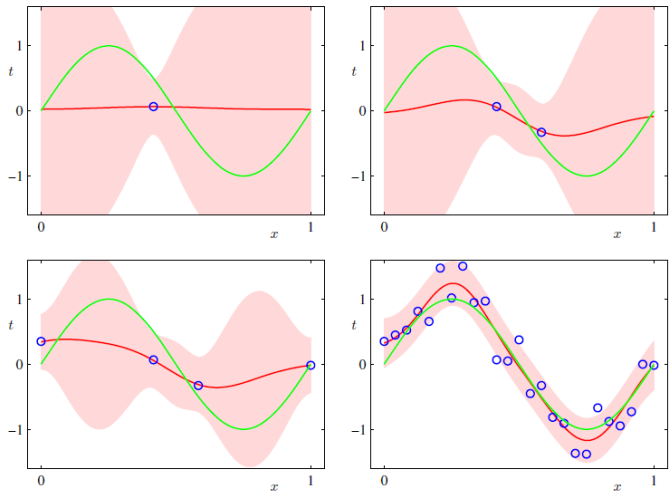


Figure 3.8 Examples of the predictive distribution (3.58) for a model consisting of 9 Gaussian basis functions of the form (3.4) using the synthetic sinusoidal data set of Section 1.1. See the text for a detailed discussion.

Les modèles de la *ridge regression* et la *bayesian ridge regression* fonctionnent très bien sur les données du projet, ce qui pourrait être expliqué par le fait qu'on a pré-traité les données asymétriques en appliquant $log1p$ avant d'entraîner les modèles de prédiction. Puisque la distribution de la probabilité antérieure et celle de la probabilité postérieure sont supposées de suivre la loi normale, les modèles de la régression Ridge et la *bayesian ridge regression* s'adaptent bien aux données standardisées.

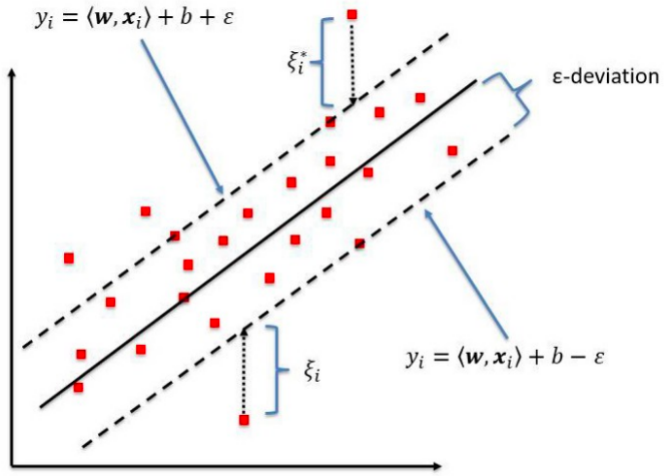
## 6.2 SVM pour la régression (*Support Vector Regression*)

Nous avons déjà vu en classe la machine à vecteurs de support pour les problèmes de classification. La machine à vecteurs de support peut aussi résoudre les problèmes de régression, *support vector regression (SVR)*.

SVR utilise presque les mêmes principes (la minimisation des erreurs et la maximisation de la marge) que SVM, mais avec quelques différences mineures. Supposons que l'équation de l'hyper plan est $wx + b = 0$,
les deux équations des frontières de décision sont :

$$wx + b = \epsilon, wx + b = -\epsilon$$

L'objectif est de maximiser le nombre des points dans les frontières.




(Source : Medium [8])

On a les fonctions objectives pour SVR [9],

$$min\frac{1}{2}||w||^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$$
$$s.t. y_i - (wx_i + b) \leq \epsilon + \xi_i$$
$$(wx_i + b) - y_i \leq \epsilon + \xi_i^*$$
$$\xi_i \geq 0$$
$$\xi_i^* \geq 0$$

La constante C est le paramètre de contrainte, une valeur positive qui contrôle la pénalité imposée aux points qui se situent en dehors de la marge $\epsilon$, et permet d'éviter le sur-ajustement afin de garantir la régularisation.

Le guide [10] d'utilisateur de *scikit-learn* nous propose quelques suggestions utiles lors d'appliquer l'algorithme de SVR :
1. Réglage d'hyperparamètres sur C (*Hyperparameter Tuning*) : C est égal à 1 par défaut. S'il y a beaucoup de bruit dans les données, on devrait diminuer la constante C, ce qui correspond à plus de régularisation.
2. Il est fortement recommandé de standardiser les données. Par exemple, nous pouvons standardiser les données X dans $[0, 1]$ ou $[-1, 1]$ ou les normaliser pour que X ait une moyenne de 0 et une variance de 1. Il faut faire le même pré-traitement sur les données de test. Ce genre de pré-traitement peut être implémenté avec l'utilisation de

8. https://medium.com/coinmonks/support-vector-regression-or-svr-8eb3acf6d0ff
9. https://www.mathworks.com/help/stats/understanding-support-vector-machine-regression.html
10. https://scikit-learn.org/stable/modules/svm.html#tips-on-practical-use

*sklearn.pipeline*.

```
>>> from sklearn.pipeline import make_pipeline
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.svm import SVC

>>> clf = make_pipeline(StandardScaler(), SVC())
```

(Source : Le guide d'utilisateur de *scikit-learn*)

Ces deux suggestions sont importantes quand on applique l'algorithme de SVR dans la pratique. Avant de faire le réglage d'hyperparamètres sur C et la standardisation des données, on a obtenu 0.27807 comme erreur sur les données d'entraînement.

Après le réglage sur le modèle et le pré-traitement (*RobustScaler*) sur les données,

```
param_grid_svr = {'C':[12, 15, 20], "gamma":[0.0003], "epsilon":[0.008]}
grid_search(SVR()).opt(X_train1, y_train_log1p, param_grid_svr, kfold)
```

```
{'C': 12, 'epsilon': 0.008, 'gamma': 0.0003} 0.18317707784189322
```

l'erreur a été beaucoup réduite (à 0.11).

```
svr_robust_scalar = make_pipeline(RobustScaler(), SVR(C = 12, epsilon = 0.008, gamma = 0.0003))

score_svr_robusts_scaler = rmse(svr_robust_scalar, X_train1, y_train_log1p, kfold)
print("SVR_robusts_scaler: {:.4f} ({:.4f})".format(score_svr_robusts_scaler.mean(), score_svr_robusts_scaler.std()))
scores_with_para['SVR_robusts_scaler'] = (score_svr_robusts_scaler.mean(), score_svr_robusts_scaler.std())
# SVR_robusts_scaler: 0.1100 (0.0143)
```

```
SVR_robusts_scaler: 0.1100 (0.0143)
```

## 6.3 Arbre de décision pour la régression (*Decision Tree Regression*)

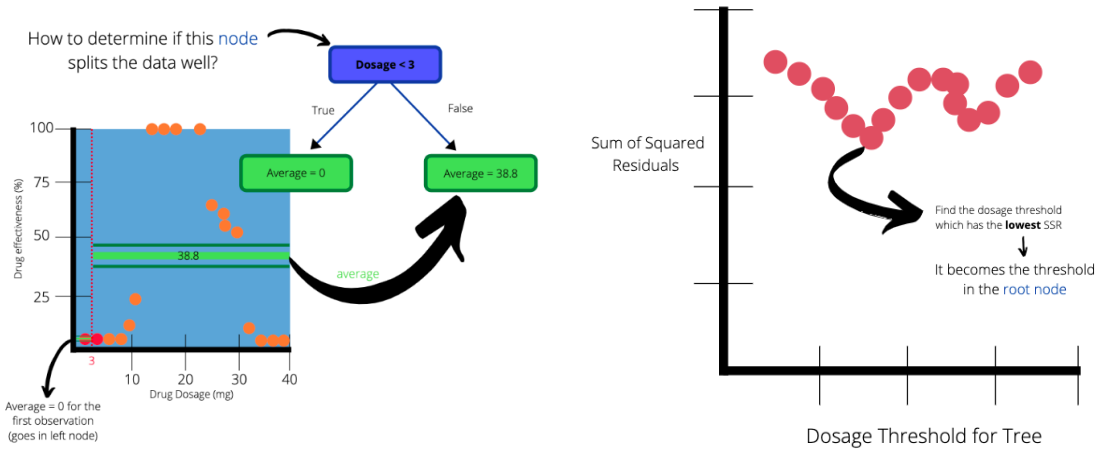
L'arbre de décision est une méthode de base de classification et de régression. Chaque arbre est composé de nœuds et d'arêtes orientées. Il existe deux types de nœuds : les nœuds internes et les nœuds externes (feuilles). La feuille représente une catégorie ou une valeur de la variable-cible. Le nœud interne correspond à des combinaisons de variables d'entrée qui mènent aux valeurs de feuilles.

En fait, l'arbre de décision consiste en une méthode de division de l'espace avec des hyperplans. Chaque fois qu'on produit des noeuds enfants, l'espace actuel (l'ensemble des données) est divisé en fonction de la valeur de sorte que chaque feuille corresponde à une partie distincte de l'espace.

### 6.3.1 Construction d'un arbre de décision par CART

L'approche par arbre de décision commence à partir du nœud racine, teste une certaine caractéristique de l'échantillon et divise l'arbre du noeud-partent vers ses enfants en sélectionnant à chaque étape une variable d'entrée qui réalise la meilleure séparation de l'ensemble des données. De cette manière, les échantillons sont testés et distribués de manière récursive jusqu'à ce que la feuille soit atteinte.

Dans le cas de l'arbre de régression, la valeur de sortie est continue et donc on utilise la minimisation de la somme des carrés de l'erreur pour obtenir la valeur de sortie optimale. Elle utilise la moyenne des feuilles comme la prédiction de la catégorie.




(Source : `https://www.datadriveninvestor.com/2020/04/13/how-do-regression-trees-work/`)

On va expliquer plus précisément la construction de l'arbre pour la régression. Soit D, un ensemble de donnée $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, supposons qu'on veut diviser X en m parties $R_1, R_2, ...R_m$, la valeur de sortie $c_m = average(y_i|x_i \in R_m)$. On choisit la j-ième variable $x^j$ et sa valeur $s$ comme la variable de coupure (*splitting variable*) et le point de coupure (*splitting point*). On définit deux régions $R_1(k, s) = \{x|x^k \leq s\}$ et $R_2(l, s) = \{x|x^l > s\}$ pour représenter le sous-arbre gauche et le sous-arbre droit du $x^j$. Afin de trouver la variable de coupure optimale, on résout la fonction objective [11] :

$$\min_{j,s}[\min_{c_1} \sum_{x_i \in R_1} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2} (y_i - c_2)^2]$$

où les $c_1, c_2$ optimaux sont $\hat{c_1} = average(y_i|x_i \in R_1)$ et $\hat{c_2} = average(y_i|x_i \in R_2)$.

L'algorithme parcourt toutes les variables et trouve la variable de coupure optimale j. Ensuite, on répète le processus de division ci-dessus jusqu'à ce que la condition d'arrêt

11. Hang LI, Statistical Learning Methods, p.68

soit remplie. De cette manière, un arbre de régression est généré.

Afin d'éviter le problème de sur-ajustement, *scikit-learn* utilise l'algorithme de *minimal cost-complexity pruning* pour faire l'élagage de l'arbre.

### 6.3.2 Avantages et Faiblesses

L'arbre de décision présente certains avantages :
(1) La vitesse d'entraînement de prédiction est rapide.
(2) Il est facile de comprendre les relations non-linéaires entre les caractéristiques des données.
(3) Les résultats sont faciles à interpréter.
(4) Il est facile de trouver les caractéristiques les plus importantes de l'ensemble de données.

Par contre, cette méthode présente aussi certaines faiblesses :
(1) La précision de la prédiction est faible.
(2) Cette approche ne convient pas parfaitement aux petits ensembles de données.
(3) L'algorithme est facile d'être influencée par le bruit des données.
(4) Lorsque de nouvelles données sont ajoutées, il n'est pas facile de mettre à jour le modèle.

Le guide d'utilisateur de $scikit-learn$ [12] nous propose également quelques suggestions utiles lors d'appliquer l'algorithme de l'arbre de décision :
(1) L'algorithme de l'arbre de décision a tendance à sur-ajuster à l'ensemble des données qui ont beaucoup de caractéristiques (*features*). Il est important de bien calculer le rapport entre le nombre d'échantillons et le nombre de caractéristiques, parce que l'arbre avec peu d'échantillons dans un espace multidimensionnel a tendance à être sur-ajusté.
(2) On peut procéder à la réduction des dimensions (PCA ou *feature selection* par exemple) à l'avance pour bien identifier les caractéristiques significatives.
(3) Il nous faut aussi régler les paramètres de l'arbre comme *max depth*, *min samples split* et *min samples leaf* pour s'assurer que des échantillons multiples informent chaque décision dans l'arbre en contrôlant les divisions qui seront prises en compte.

Si on fait le lien avec l'ensemble des données de notre projet, on a obtenu 0.2017 comme l'erreur. Même si nous avons bien pré-traité nos données et ajusté les paramètres, le résultat n'est pas satisfaisant (0.1808),

12. `https://scikit-learn.org/stable/modules/tree.html#tips-on-practical-use`

```
param_grid_decisiontree = {"max_depth": [3, 4, 5, 8, 10], "min_samples_split" : [5, 6, 7, 8], "min_samples_leaf" : [5, 6, 7, 8]}
grid_search(DecisionTreeRegressor()).opt(X_train1, y_train_log1p, param_grid_decisiontree, kfold)
```

```
{'max_depth': 10, 'min_samples_leaf': 7, 'min_samples_split': 8} 0.1807919547948937
```

ce qui s'explique probablement par les raisons suivantes :
(1) la mauvaise prédiction de l'arbre de décision
(2) le sur-ajustement causé par les observations aberrantes.
(3) le nombre des caractéristiques est grand, ce qui implique que l'algorithme de l'arbre de décision a tendance à mal indiquer les caractéristiques significatives. On pourrait appliquer PCA sur les données avant d'entraîner le modèle de prédiction.

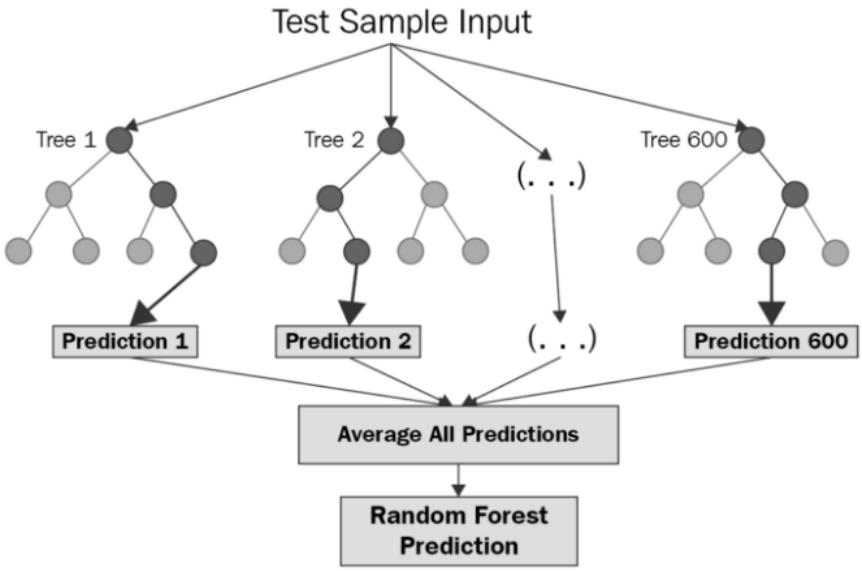
## 6.4 Forêt aléatoire pour la régression (*Random Forest Regression*)

L'algorithme de la forêt aléatoire appartient à la méthode d'apprentissage d'ensemble. Dans une forêt aléatoire, chaque arbre de l'ensemble est construit au moyen des techniques de *Bootstrap* sur la base des échantillons avec remplacement de l'ensemble des données. Les techniques de *Bootstrap* font référence aux méthodes d'inférence statistique basées sur l'échantillonnage aléatoire avec remplacement. Il nous permet de mieux comprendre le biais et la variance du jeu de données.

En outre, lors de la division de chaque nœud pendant la construction d'un arbre, la meilleure division est trouvée soit (1) à partir de toutes les caractéristiques, soit (2) à partir d'un sous-ensemble aléatoire de taille de *max features*. Ces deux sources aléatoires ont pour but de diminuer la variance de l'estimateur de la forêt et d'éviter le sur-ajustement causé pour un seul arbre. [13]

Au stade de la prédiction, certaines erreurs peuvent être éliminées en prenant la moyenne de ces estimations. Les forêts aléatoires parviennent à réduire la variance en combinant différents arbres, mais parfois le biais est légèrement croissant. En pratique, la réduction de la variance est généralement significative, ce qui se traduit par un meilleur modèle global.

13. https://scikit-learn.org/stable/modules/ensemble.html#forests-of-randomized-trees




Random Forest Structure

(Source : `https://medium.com/swlh/random-forest-and-its-implementation-71824ced454f`)

Lors d'appliquer l'algorithme de la forêt aléatoire pour la régression sur l'ensemble des données, on peut utiliser la fonction *RandomForestRegressor* pré-implémentée dans *scikit-learn*[14] :
(1) Le paramètre *n_ estimators* sert à générer 100 arbres de décision par défaut. Normalement, plus le nombre des arbres est grand, plus le résultat de prédiction est précis. Cependant, ceci prend plus de temps et le résultat va se converger si le nombre de l'arbre est trop grand.
(2) La fonction *criterion* permet de mesurer la qualité d'un fractionnement. Par exemple, si on choisit *"MSE"* comme critère, l'erreur quadratique moyenne entre le nœud parent et le nœud enfant est choisie comme la critère de la sélection des caractéristiques.
(3) L'interface *score* retourne $R^2$, le coefficient de détermination de la prédiction. On peut définir R comme suit :

$$R^2 = 1 - \frac{u}{v}$$

$$u = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

$$v = \sum_{i=1}^{N} (y_i - \overline{y})^2$$

où N est le nombre des échantillons, $\hat{y}_i$ est la valeur de prédiction, et $\overline{y}$ est la moyenne

14. `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html#sklearn.ensemble.RandomForestRegressor`

des valeurs observées. La valeur de u va être petite si $y_i$ est proche de $\hat{y}_i$. Le $\frac{u}{v}$ tend vers donc zéro, ce qui fait que la valeur de $R^2$ est proche de 1.

On peut également personnaliser la métrique pour évaluer les erreurs du modèle de prédiction. Dans notre project, on a choisi la racine de l'erreur quadratique moyenne (RMSE) entre le logarithme de la valeur prédite et le logarithme du prix de vente observé. En ce qui concerne la prédiction obtenue par le modèle de la forêt aléatoire, nous avons obtenu un résultat relativement satisfaisant même sans régler les hyperparamètres. (Veuillez voir le code en annexe pour plus d'informations).

C'est probablement grâce à des raisons suivantes : l'algorithme de la forêt aléatoire peut traiter des données avec une haute dimensionnalité parce qu'elle génère les arbres de décision à partie de la méthode statistique *Bootstrap* qui peut re-échantillonner à partir de l'ensemble de donnée. La prédiction est supportée par l'analyse de la robustesse.

# 7 Conclusion

Nous avons expliqué précisément quatre méthodes de prédiction vues en classe dans la dernière partie. À part de ces quatre algorithmes mentionnées ci-dessus, nous avons également utilisé les modèles pour la régression suivantes : *Lasso Regression*, *Extra Trees*, *Gradient boosting*, *LGBM Regressor* et *XGBoost Regressor*.

Voici la moyenne et l'écart-type de RMSE des modèles de prédiction (*without hyperparameter tuning*).

```
Ridge: 0.113670, 0.0156
Lasso: 0.264437, 0.0165
BayesianRidge: 0.111524, 0.0146
DecisionTree: 0.201714, 0.0211
RandomForest: 0.137951, 0.0112
GradientBoosting: 0.124129, 0.0131
ExtraTrees: 0.135439, 0.0112
SVR: 0.278071, 0.0189
LGBM: 0.127936, 0.0106
XGB: 0.123820, 0.0137
```

Nous avons procédé au réglage d'hyperparamètres pour certains modèles à l'aide des méthodes de *cross validation* et *grid search*. Voici la moyenne et l'écart-type de RMSE de certains modèles de prédiction (*with hyperparameter tuning*).

```
{'SVR_robusts_scaler': (0.11004557443046728, 0.014283810164719747),
 'lightgbm_para': (0.1251244358577163, 0.011736671168271097),
 'xgb_para_opt': (0.12267646814042757, 0.01347317629207774)}
```

Nous avons décidé de choisir deux stratégies de *blending* ci-dessous pour calculer la prédiction finale :

```
def mix_models_1(test_data):
  Ridge_pred = 0.25 * Ridge_model.predict(test_data)
  BayesianRidge_pred = 0.25 * BayesianRidge_model.predict(test_data)
  GradientBoosting_pred = 0.1 * GradientBoostingRegressor_model.predict(test_data)
  svr_pred = 0.2 * svr_robust_scalar_model.predict(test_data)
  lightgbm_pred = 0.1 * lightgbm_para_model.predict(test_data)
  xgb_pred = 0.1 * xgb_para_model.predict(test_data)

  return (Ridge_pred + BayesianRidge_pred + GradientBoosting_pred + svr_pred + lightgbm_pred + xgb_pred)
```

```
def mix_models_2(test_data):
  Ridge_pred = 0.25 * Ridge_model.predict(test_data)
  BayesianRidge_pred = 0.25 * BayesianRidge_model.predict(test_data)
  ExtraTrees_pred = 0.05 * ExtraTreesRegressor_model.predict(test_data)
  GradientBoosting_pred = 0.1 * GradientBoostingRegressor_model.predict(test_data)
  svr_pred = 0.15 * svr_robust_scalar_model.predict(test_data)
  lightgbm_pred = 0.05 * lightgbm_para_model.predict(test_data)
  xgb_pred = 0.15 * xgb_para_model.predict(test_data)

  return (Ridge_pred + BayesianRidge_pred + GradientBoosting_pred + svr_pred + lightgbm_pred + xgb_pred + ExtraTrees_pred)
```

Les résultats de prédiction sont évalués sur la base de l'erreur quadratique moyenne (RMSE) entre le logarithme de la valeur prédite et le logarithme du prix de vente observé. (Prendre les logarithmes signifie que les erreurs de prédiction des maisons chères et des maisons bon marché affecteront le résultat de la même manière). Nous avons obtenu 0.12255 et 0.12332 comme résultats pour la première stratégie et la deuxième stratégie respectivement, ce qui montre qu'on a obtenu un résultat relativement réussi (*top 13%* parmi tous les résultats de prédiction sur Kaggle).

Il existe plusieurs façons d'améliorer nos résultats et de rendre le modèle de prédiction plus performant.

Au niveau du pré-traitement des données, nous pouvons mettre au point les stratégies plus gagnantes pour imputer les valeurs manquantes :
(1) Remplacer les valeurs manquantes en fonction de la corrélation entre la caractéristique et le prix ou la corrélation entre les attributs au lieu de les remplacer simplement par 0, la moyenne ou la médiane.
(2) Utiliser la fonction *KNNImputer* implémentée dans python pour compléter les valeurs manquantes à l'aide de l'approche *k Nearest Neighbors*.
(3) Normaliser les données asymétriques avec la transformation de *boxcox1p* ou *boxcox normmax* et comparer les résultats avec ceux de *log1p*.
(4) Créer plus de variables pertinentes durant le processus de *feature engineering* et appliquer PCA si nécessaire.

Au niveau de l'entraînement des modèles de prédiction et de *hyperparameter tuning*,
(1) Faire un nombre beaucoup plus important d'essais pour trouver les valeurs de paramètres optimales.
(2) Essayer les autres méthodes performantes telles que *Stacked Regressor* et *Cat Boost*

*Regressor*.
(3) Utiliser la fonction *get_feature_importance* pour la sélection des caractéristiques, ce qui nous permet de mieux comprendre le fonctionnement de l'algorithme et de savoir quelles caractéristiques que l'algorithme utilise le plus pour arriver à la prédiction finale.
(4) Avec la conclusion de la sélection des caractéristiques, on peut faire encore une fois l'optimisation des hyperparamètres.

Contributions des membres de l'équipe :
Yi Xuan Luo : Introduction, Compréhension des données, Nettoyage des données et Conclusion.
Cheng Chen : ANOVA, Heatmap, Arbre de décision pour la régression et Forêt aléatoire pour la régression.
Weiyue Cai : Réduction des dimensions, Pré-traitement des données, Bayesian Ridge Regression et SVM pour la régression.

# 8 Références

# Références


[1] Kaggle, House Prices - Advanced Regression Techniques. https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview

[2] Sampath Kumar Gajawada, ANOVA for Feature Selection in Machine Learning. https://towardsdatascience.com/anova-for-feature-selection-in-machine-learning-d9305e228476

[3] Scikit-Learn User Guide, https://scikit-learn.org/stable/user_guide.html

[4] Bishop, Christopher M. Pattern Recognition and Machine Learning. New York : Springer, 2006. p.157

[5] Indresh Bhattacharyya. Support Vector Regression Or SVR. https://medium.com/coinmonks/support-vector-regression-or-svr-8eb3acf6d0ff

[6] MathWorks. Understanding Support Vector Machine Regression. https://www.mathworks.com/help/stats/understanding-support-vector-machine-regression.html

[7] Hang Li. Statistical Learning Methods, Second Edition. Tsinghua University Press, 2019