ml_python_delvich

September 18, 2023

1 Machine Learning Avancé

Ce document présente les techniques d'optimisation de différents modèles d'apprentissage supervisé. Les manipulations de la donnée ne sont pas présentés dans le présent document. Il permet également de s'initier aux modèles de stacking et de voting. Qu'est ce que le stacking ? Le stacking, ou l'empilement, est une technique d'ensemble learning utilisée dans la régression et d'autres tâches d'apprentissage automatique. L'objectif du stacking est de combiner les prédictions de plusieurs modèles de régression de base (appelés également "learners" ou "modèles de base") pour améliorer les performances de la prédiction par rapport à l'utilisation d'un seul modèle de régression. Le stacking permet souvent d'obtenir de meilleures performances prédictives que l'utilisation de chaque modèle de base individuellement et de réduire la dépendance des modèles aux graines fixées.

[1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score,␣
 ↪mean_absolute_error, f1_score
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Dropout
from sklearn.model_selection import GridSearchCV
```

1.1 Importation de la base Sur google Collab

Comment importer les données sur google collab pour utiliser la puissance de calcul de google collab ?

[2]:
```
# Première étape
from google.colab import files
import io
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving base.csv to base.csv
```

[3]:
```
# Deuxième étape
data = pd.read_csv(io.BytesIO(uploaded["base.csv"]), index_col = 0)
data.head(3)
```

[3]:
```
            id             date     price  bedrooms  bathrooms  sqft_living  \
1  7129300520  20141013T000000  221900.0         3       1.00         1180
2  6414100192  20141209T000000  538000.0         3       2.25         2570
3  5631500400  20150225T000000  180000.0         2       1.00          770

   sqft_lot  floors  waterfront  view  …  zipcode      lat     long  \
1      5650     1.0           0     0  …    98178  47.5112 -122.257
2      7242     2.0           0     0  …    98125  47.7210 -122.319
3     10000     1.0           0     0  …    98028  47.7379 -122.233

   sqft_living15  sqft_lot15  Ech  part_maison_sur_terrain  \
1           1340        5650    1                 0.208850
2           1690        7639    1                 0.354874
3           2720        8062    1                 0.077000

   Part_logement_dessus_sol_sur_terrai  part_sous_sol_sur_logement  \
1                             1.000000                    0.000000
2                             0.844358                    0.155642
3                             1.000000                    0.000000

   yr_renovated_rec
1                 0
2                 4
3                 0

[3 rows x 26 columns]
```

1.2 Importation de la base Sur kaggle

La base de donnée utilisée pour la présente utilisation est également présente sur kaggle. Pour l'importer, utilisez :

[ ]:
```
data = pd.read_csv("/kaggle/input/data-ml/base.csv", index_col = 0)
```

1.3 Importation en local

[2]:
```
data = pd.read_csv("base.csv", index_col= 0)
```

2 Analyse

[3]:
```
data.columns
```

[3]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15', 'Ech',
       'part_maison_sur_terrain', 'Part_logement_dessus_sol_sur_terrai',
```

```
       'part_sous_sol_sur_logement', 'yr_renovated_rec'],
      dtype='object')
```

Diviser en data_set et la data sans les "price". La data_set contient les données à diviser en échantillon test et d'apprentissage. La base de données *price* contient les données dont on ne connait pas le prix (à prévoir)

[4]:
```
data_set = data.loc[data["Ech"]== 1]
price = data.loc[data["Ech"]== 0]
```

On construit les deux modèles à partir desquels je veux essayer

[5]:
```
data_test_1 = data_set.drop(columns= ["id", "date", "Ech", "sqft_above",␣
 ↪"sqft_basement", "yr_renovated_rec"])
price = price.drop(columns=["sqft_above", "sqft_basement", "yr_renovated_rec",␣
 ↪"date", "Ech"])
```

Constitution des échantillons

[6]:
```
X1 = data_test_1.drop("price", axis= 1)
Y1 = data_test_1["price"]
X_train, X_test, y_train, y_test = train_test_split(X1, Y1, test_size=0.25,␣
 ↪random_state=42)
```

3 Random Forest

Un modèle de Random Forest, ou "forêt aléatoire" en français, est un type d'algorithme d'apprentissage automatique utilisé pour la classification, la régression et d'autres tâches de modélisation prédictive. Il appartient à la famille des modèles d'ensemble, ce qui signifie qu'il combine les prédictions de plusieurs arbres de décision pour améliorer la précision globale du modèle. Voici les principales caractéristiques d'un modèle de Random Forest :

Voici les principales caractéristiques d'un modèle de Random Forest :

1 - Arbres de décision: Un modèle de Random Forest est composé de multiples arbres de décision. Chaque arbre de décision est construit à partir d'un échantillon aléatoire des données d'entraînement et un sous-ensemble aléatoire des caractéristiques (variables indépendantes). Cela rend chaque arbre légèrement différent, ce qui réduit le risque de surapprentissage.

2- Bagging (Bootstrap Aggregating): Le terme "forêt" fait référence à l'idée de construire de multiples arbres de décision de manière indépendante. Pour cela, on utilise une technique appelée "bagging", qui consiste à créer plusieurs ensembles de données d'entraînement en échantillonnant les données avec remplacement. Chaque arbre est ensuite formé sur l'un de ces ensembles de données.

3- Agrégation des prédictions: Une fois que tous les arbres de décision ont été formés, le modèle de Random Forest agrège leurs prédictions pour obtenir la prédiction finale. En classification, cela peut être fait par vote majoritaire (le label le plus fréquent parmi tous les arbres est retenu), tandis qu'en régression, cela peut être fait en prenant la moyenne des prédictions.

4- Importance des caractéristiques: Un avantage notable des Random Forests est leur capacité à estimer l'importance des caractéristiques. Chaque arbre de décision conserve une trace de la façon dont il utilise les caractéristiques pour prendre des décisions, ce qui permet de calculer l'importance relative de chaque caractéristique dans la prédiction globale du modèle.

5- Robustesse et performances élevées: Les Random Forests sont robustes aux valeurs aberrantes et aux données manquantes, et elles sont souvent moins sensibles au surapprentissage par rapport à un arbre de décision unique. Elles sont également connues pour offrir de bonnes performances sur une variété de tâches de classification et de régression.

6- Paramètres de réglage: Les Random Forests ont quelques paramètres importants à régler, tels que le nombre d'arbres dans la forêt, la profondeur maximale des arbres, le nombre minimum d'échantillons par feuille, etc. Ces paramètres peuvent être ajustés via une validation croisée pour optimiser les performances du modèle.

En résumé, un modèle de Random Forest est un puissant modèle d'ensemble basé sur des arbres de décision qui combine la prédiction de multiples arbres pour obtenir des résultats plus robustes et précis. Il est largement utilisé dans divers domaines de l'apprentissage automatique en raison de sa performance et de sa polyvalence

[7]:
```
from sklearn.ensemble import RandomForestRegressor
```

[8]:
```
param_grid = {'n_estimators': [100, 200, 500], 'max_depth': [10, 20, 30]}
rfc = RandomForestRegressor(random_state=42)
grid_search_rf = GridSearchCV(rfc, param_grid=param_grid, cv=5)
grid_search_rf.fit(X_train, y_train)
```

[8]:
```
GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
             param_grid={'max_depth': [10, 20, 30],
                         'n_estimators': [100, 200, 500]})
```

[9]:
```
grid_search_rf.best_params_
```

[9]:
```
{'max_depth': 30, 'n_estimators': 500}
```

[9]:
```
rf_1 = RandomForestRegressor(n_estimators= 500, max_depth= 30)
rf_1.fit(X_train, y_train)
y_pred = rf_1.predict(X_test)
```

[10]:
```
print("R2 Score : ", r2_score(y_test, y_pred))
print("RMSE : ", np.sqrt(mean_squared_error(y_test, y_pred)))
print("MSE : ", mean_squared_error(y_test, y_pred))
print("MAE :", mean_absolute_error(y_test, y_pred))
```

```
R2 Score :  0.8705591816644898
RMSE :  135369.49053842155
MSE :  18324898968.631805
MAE : 71951.299435234
```

4 Adaboost

Un modèle AdaBoost, ou “Adaptive Boosting” en anglais, est un algorithme d’apprentissage automatique utilisé pour améliorer la précision des modèles de classification faibles en les combinant de manière adaptative. L’objectif principal de l’AdaBoost est de donner plus de poids aux échantillons mal classés ou difficiles à classer, ce qui permet de construire un modèle fort à partir de plusieurs modèles faibles.

[ ]:
```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
```

[ ]:
```
base_estimator = DecisionTreeRegressor(max_depth=1)
adaboost_model = AdaBoostRegressor(base_estimator=base_estimator,␣
 ↪random_state=42)
```

[ ]:
```
params = {
    'n_estimators': [50, 100, 200, 300, 400],
    'learning_rate': [0.01, 0.05, 0.1, 0.5, 1]
}

grid_search_ada = GridSearchCV(estimator=adaboost_model, param_grid=params,␣
 ↪cv=5, scoring='neg_mean_squared_error')
grid_search_ada.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5,
estimator=AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=1),
                                           random_state=42),
             param_grid={'learning_rate': [0.01, 0.05, 0.1, 0.5, 1],
                         'n_estimators': [50, 100, 200, 300, 400]},
             scoring='neg_mean_squared_error')
```

[ ]:
```
print("Best parameters found: ", grid_search_ada.best_params_)
```

```
Best parameters found:  {'learning_rate': 0.01, 'n_estimators': 50}
```

[ ]:
```
adaboost_model = AdaBoostRegressor(
    base_estimator=DecisionTreeRegressor(max_depth=1),
    n_estimators=50,
    learning_rate=0.01,
    random_state=42)

adaboost_model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/ensemble/_base.py:166:
FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and
will be removed in 1.4.
  warnings.warn(
```

[ ]:
```
AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=1),
                  learning_rate=0.01, random_state=42)
```

[ ]:
```
y_pred_ad = adaboost_model.predict(X_test)
```

[ ]:
```
print("R2 score : ", r2_score(y_test, y_pred_ad))
print("RMSE : ", np.sqrt(mean_squared_error(y_test, y_pred_ad)))
print("MSE : ", mean_squared_error(y_test, y_pred_ad))
print("MAE :", mean_absolute_error(y_test, y_pred_ad))
```

```
R2 score :  0.3795363759794702
RMSE :  296376.2045470328
MSE :  87838854621.7046
MAE : 192276.90646711245
```

5 XGBOOST

XGBoost, ou "eXtreme Gradient Boosting," est un algorithme d'apprentissage automatique très populaire et puissant qui appartient à la famille des modèles d'ensemble. Il est particulièrement efficace pour les tâches de classification et de régression. XGBoost est apprécié pour sa capacité à produire des modèles de haute performance tout en étant robuste et rapide.

[11]:
```
import xgboost as xgb
```

[12]:
```
# Création d'un modèle XGBoost
model = xgb.XGBRegressor()
```

[ ]:
```
# Définition de la grille de recherche pour les hyperparamètres
param_grid = {
    'max_depth': [3, 4, 5],
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': [100, 500, 1000]
}
```

[ ]:
```
grid_search = GridSearchCV(model, param_grid, cv=5,␣
 ↪scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5,
             estimator=XGBRegressor(base_score=None, booster=None,
                                    callbacks=None, colsample_bylevel=None,
                                    colsample_bynode=None,
                                    colsample_bytree=None,
                                    early_stopping_rounds=None,
                                    enable_categorical=False, eval_metric=None,
                                    gamma=None, gpu_id=None, grow_policy=None,
                                    importance_type=None,
                                    interaction_constraints=None,
```

```
                                    learning_rate=None, max_bin=None,
                                    max_cat…tep=None,
                                    max_depth=None, max_leaves=None,
                                    min_child_weight=None, missing=nan,
                                    monotone_constraints=None, n_estimators=100,
                                    n_jobs=None, num_parallel_tree=None,
                                    predictor=None, random_state=None,
                                    reg_alpha=None, reg_lambda=None, …),
             param_grid={'learning_rate': [0.1, 0.01, 0.001],
                         'max_depth': [3, 4, 5],
                         'n_estimators': [100, 500, 1000]},
             scoring='neg_mean_squared_error')
```

[ ]:
```
# Affichage des meilleurs hyperparamètres trouvés
print('Best parameters:', grid_search.best_params_)
```

```
Best parameters: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 1000}
```

[13]:
```
xgb_model = xgb.XGBRegressor(
    objective='reg:squarederror',
    learning_rate=0.1,
    max_depth=4,
    min_child_weight=1,
    subsample=0.8,
    colsample_bytree=0.8,
    n_estimators=1000,
    seed=42)

xgb_model.fit(X_train, y_train)
```

[13]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.8, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=4, max_leaves=None,
             min_child_weight=1, missing=nan, monotone_constraints=None,
             n_estimators=1000, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, …)
```

[14]:
```
y_pred_xgb = xgb_model.predict(X_test)
```

[15]:
```
print("R2 score : ", r2_score(y_test, y_pred_xgb))
print("RMSE : ", np.sqrt(mean_squared_error(y_test, y_pred_xgb)))
print("MSE : ", mean_squared_error(y_test, y_pred_xgb))
print("MAE :", mean_absolute_error(y_test, y_pred_xgb))
```

```
R2 score :  0.895461576897249
RMSE :  121653.08925455222
MSE :  14799474125.176048
MAE : 66752.2755748057
```

6 Bagging

Un modèle de bagging, ou “Bootstrap Aggregating”, est une technique d’ensemble learning en apprentissage automatique. L’objectif du bagging est d’améliorer la stabilité et la précision d’un modèle en construisant plusieurs modèles similaires à partir d’échantillons de données différents et en combinant leurs prédictions. Le modèle de bagging le plus connu est le “Random Forest,” bien que d’autres algorithmes de bagging puissent également être utilisés.

[16]:
```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
```

[ ]:
```
parameters_bag = {
    "n_estimators": [10, 50, 100],
    "max_samples": [0.5, 1.0],
    "max_features": [0.5, 1.0],
    "bootstrap": [True, False],
    "bootstrap_features": [True, False],
}
```

[ ]:
```
model = BaggingRegressor()
grid_search = GridSearchCV(model, parameters_bag, cv=5,␣
 ↪scoring="neg_root_mean_squared_error")
grid_search.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=BaggingRegressor(),
             param_grid={'bootstrap': [True, False],
                         'bootstrap_features': [True, False],
                         'max_features': [0.5, 1.0], 'max_samples': [0.5, 1.0],
                         'n_estimators': [10, 50, 100]},
             scoring='neg_root_mean_squared_error')
```

[ ]:
```
print("Meilleurs hyperparamètres:", grid_search.best_params_)
```

```
Meilleurs hyperparamètres: {'bootstrap': True, 'bootstrap_features': False,
'max_features': 1.0, 'max_samples': 1.0, 'n_estimators': 50}
```

[17]:
```
bagging = BaggingRegressor(base_estimator=DecisionTreeRegressor(max_features=1.
 ↪0), n_estimators=50,
                           max_samples=1.0, max_features=1.0, bootstrap=True,␣
 ↪bootstrap_features=False,
                           random_state=42)
```

[18]:
```
# Entraîner le modèle Bagging
bagging.fit(X_train, y_train)

# Prédire les valeurs pour l'ensemble de test
y_pred_bag = bagging.predict(X_test)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/ensemble/_base.py:166:
FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and
will be removed in 1.4.
  warnings.warn(
```

[19]:
```
print("R2 score : ", r2_score(y_test, y_pred_bag))
print("RMSE : ", np.sqrt(mean_squared_error(y_test, y_pred_bag)))
print("MSE : ", mean_squared_error(y_test, y_pred_bag))
print("MAE :", mean_absolute_error(y_test, y_pred_bag))
```

```
R2 score :  0.8678351207542938
RMSE :  136786.4886553237
MSE :  18710543478.652992
MAE : 72715.26703224279
```

7 sklearn.neural_network.MLPRegressor

Un modèle MLPRegressor est un modèle de régression basé sur un réseau de neurones artificiels appelé un perceptron multicouche (Multilayer Perceptron, MLP). Il s'agit d'un type de modèle d'apprentissage automatique utilisé pour résoudre des tâches de régression, où l'objectif est de prédire une valeur numérique continue plutôt que de classer des données en catégories discrètes.

[ ]:
```
from sklearn.neural_network import MLPRegressor
```

[ ]:
```
# Initialiser le modèle MLPRegressor
mlp = MLPRegressor(max_iter=100, solver = "adam")
```

[ ]:
```
parameters = {'hidden_layer_sizes': [(10,), (50,), (100,), (50, 50), (100, 50)],
              'activation': ['identity', 'logistic', 'tanh', 'relu'],
              'alpha': [0.0001, 0.001, 0.01],
              'learning_rate': ['constant', 'invscaling', 'adaptive']}
```

[ ]:
```
# Effectuer une recherche d'hyperparamètres en utilisant la méthode GridSearchCV
clf = GridSearchCV(mlp, parameters, n_jobs=-1)
clf.fit(X_train, y_train)
```

```
/usr/local/lib/python3.8/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

```
[ ]: GridSearchCV(estimator=MLPRegressor(max_iter=100), n_jobs=-1,
                  param_grid={'activation': ['identity', 'logistic', 'tanh', 'relu'],
                              'alpha': [0.0001, 0.001, 0.01],
                              'hidden_layer_sizes': [(10,), (50,), (100,), (50, 50),
                                                     (100, 50)],
                              'learning_rate': ['constant', 'invscaling',
                                                'adaptive']})
```

```
[ ]: print("Meilleurs hyperparamètres : ", clf.best_params_)
```

```
Meilleurs hyperparamètres :  {'activation': 'relu', 'alpha': 0.0001,
'hidden_layer_sizes': (100, 50), 'learning_rate': 'constant'}
```

```
[ ]: mlp_model = MLPRegressor(
    activation = "relu",
    alpha = 0.0001,
    solver = "adam",
    hidden_layer_sizes= (100, 50),
    learning_rate="constant"
)
```

```
[ ]: mlp_model.fit(X_train, y_train)
```

```
[ ]: MLPRegressor(hidden_layer_sizes=(100, 50))
```

```
[ ]: y_pred_mlp = mlp_model.predict(X_test)
```

```
[ ]: print("R2 score : ", r2_score(y_test, y_pred_mlp))
print("RMSE : ", np.sqrt(mean_squared_error(y_test, y_pred_mlp)))
print("MSE : ", mean_squared_error(y_test, y_pred_mlp))
print("MAE :", mean_absolute_error(y_test, y_pred_mlp))
```

```
R2 score :  0.5654009643277536
RMSE :  248044.46971982645
MSE :  61526058958.5899
MAE : 168319.69837780044
```

8 Neuronal Network

```
[ ]: from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
```

```
[ ]: model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dropout(0.2))
model.add(Dense(1))
```

```
[ ]: model.compile(loss='mean_squared_error', optimizer=Adam(lr=0.01))
```

```
/usr/local/lib/python3.9/dist-
packages/keras/optimizers/optimizer_v2/adam.py:117: UserWarning: The `lr`
argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

```
[ ]: history = model.fit(X_train, y_train, epochs=100, batch_size=32,␣
     ↪validation_split=0.2)
```

```
Epoch 1/100
362/362 [==============================] - 5s 10ms/step - loss:
130349711360.0000 - val_loss: 105619521536.0000
Epoch 2/100
362/362 [==============================] - 2s 6ms/step - loss: 82859630592.0000
- val_loss: 66503950336.0000
Epoch 3/100
362/362 [==============================] - 1s 3ms/step - loss: 66516852736.0000
- val_loss: 60952924160.0000
Epoch 4/100
362/362 [==============================] - 1s 2ms/step - loss: 63251554304.0000
- val_loss: 60739403776.0000
Epoch 5/100
362/362 [==============================] - 1s 2ms/step - loss: 62028410880.0000
- val_loss: 61536038912.0000
Epoch 6/100
362/362 [==============================] - 1s 3ms/step - loss: 62543196160.0000
- val_loss: 60685365248.0000
Epoch 7/100
362/362 [==============================] - 1s 3ms/step - loss: 61665177600.0000
- val_loss: 57776193536.0000
Epoch 8/100
362/362 [==============================] - 1s 3ms/step - loss: 60698832896.0000
- val_loss: 58074357760.0000
Epoch 9/100
362/362 [==============================] - 1s 2ms/step - loss: 59817869312.0000
- val_loss: 58627235840.0000
Epoch 10/100
362/362 [==============================] - 1s 3ms/step - loss: 60728315904.0000
- val_loss: 57120231424.0000
Epoch 11/100
362/362 [==============================] - 1s 3ms/step - loss: 59932893184.0000
- val_loss: 58544623616.0000
Epoch 12/100
362/362 [==============================] - 1s 4ms/step - loss: 59927293952.0000
- val_loss: 59103395840.0000
Epoch 13/100
362/362 [==============================] - 1s 3ms/step - loss: 59739549696.0000
```

```
- val_loss: 57338920960.0000
Epoch 14/100
362/362 [==============================] - 1s 3ms/step - loss: 60859248640.0000
- val_loss: 61248421888.0000
Epoch 15/100
362/362 [==============================] - 1s 2ms/step - loss: 59761950720.0000
- val_loss: 58627788800.0000
Epoch 16/100
362/362 [==============================] - 1s 3ms/step - loss: 59514060800.0000
- val_loss: 58308804608.0000
Epoch 17/100
362/362 [==============================] - 1s 3ms/step - loss: 58571276288.0000
- val_loss: 58470989824.0000
Epoch 18/100
362/362 [==============================] - 1s 3ms/step - loss: 60157550592.0000
- val_loss: 58782810112.0000
Epoch 19/100
362/362 [==============================] - 1s 3ms/step - loss: 59108769792.0000
- val_loss: 62143307776.0000
Epoch 20/100
362/362 [==============================] - 1s 3ms/step - loss: 59754692608.0000
- val_loss: 61145956352.0000
Epoch 21/100
362/362 [==============================] - 1s 3ms/step - loss: 59617869824.0000
- val_loss: 60594237440.0000
Epoch 22/100
362/362 [==============================] - 1s 3ms/step - loss: 60630638592.0000
- val_loss: 57011466240.0000
Epoch 23/100
362/362 [==============================] - 1s 3ms/step - loss: 59031281664.0000
- val_loss: 57858846720.0000
Epoch 24/100
362/362 [==============================] - 2s 4ms/step - loss: 59001147392.0000
- val_loss: 60564770816.0000
Epoch 25/100
362/362 [==============================] - 2s 6ms/step - loss: 58854883328.0000
- val_loss: 58903801856.0000
Epoch 26/100
362/362 [==============================] - 2s 6ms/step - loss: 58455449600.0000
- val_loss: 65201844224.0000
Epoch 27/100
362/362 [==============================] - 2s 6ms/step - loss: 58712895488.0000
- val_loss: 58249502720.0000
Epoch 28/100
362/362 [==============================] - 1s 3ms/step - loss: 58386411520.0000
- val_loss: 55696330752.0000
Epoch 29/100
362/362 [==============================] - 1s 3ms/step - loss: 56432533504.0000
```

```
- val_loss: 55646879744.0000
Epoch 30/100
362/362 [==============================] - 1s 3ms/step - loss: 58249920512.0000
- val_loss: 64049324032.0000
Epoch 31/100
362/362 [==============================] - 1s 3ms/step - loss: 58310811648.0000
- val_loss: 56733794304.0000
Epoch 32/100
362/362 [==============================] - 2s 5ms/step - loss: 57196171264.0000
- val_loss: 56631455744.0000
Epoch 33/100
362/362 [==============================] - 1s 4ms/step - loss: 57035321344.0000
- val_loss: 61548707840.0000
Epoch 34/100
362/362 [==============================] - 1s 3ms/step - loss: 56540925952.0000
- val_loss: 55811788800.0000
Epoch 35/100
362/362 [==============================] - 1s 3ms/step - loss: 57599336448.0000
- val_loss: 57823641600.0000
Epoch 36/100
362/362 [==============================] - 1s 3ms/step - loss: 57381658624.0000
- val_loss: 56121757696.0000
Epoch 37/100
362/362 [==============================] - 1s 3ms/step - loss: 56751947776.0000
- val_loss: 55912886272.0000
Epoch 38/100
362/362 [==============================] - 1s 3ms/step - loss: 58060750848.0000
- val_loss: 60387930112.0000
Epoch 39/100
362/362 [==============================] - 1s 3ms/step - loss: 57800413184.0000
- val_loss: 56291713024.0000
Epoch 40/100
362/362 [==============================] - 1s 3ms/step - loss: 57699823616.0000
- val_loss: 56133238784.0000
Epoch 41/100
362/362 [==============================] - 1s 3ms/step - loss: 57830674432.0000
- val_loss: 55490703360.0000
Epoch 42/100
362/362 [==============================] - 1s 3ms/step - loss: 57878855680.0000
- val_loss: 59497426944.0000
Epoch 43/100
362/362 [==============================] - 2s 4ms/step - loss: 57495040000.0000
- val_loss: 55946428416.0000
Epoch 44/100
362/362 [==============================] - 2s 4ms/step - loss: 57371918336.0000
- val_loss: 55208771584.0000
Epoch 45/100
362/362 [==============================] - 1s 3ms/step - loss: 57727127552.0000
```

```
- val_loss: 58785492992.0000
Epoch 46/100
362/362 [==============================] - 1s 3ms/step - loss: 57355669504.0000
- val_loss: 56102633472.0000
Epoch 47/100
362/362 [==============================] - 1s 3ms/step - loss: 56874651648.0000
- val_loss: 55721660416.0000
Epoch 48/100
362/362 [==============================] - 1s 3ms/step - loss: 57852006400.0000
- val_loss: 57025171456.0000
Epoch 49/100
362/362 [==============================] - 1s 3ms/step - loss: 56819834880.0000
- val_loss: 55915663360.0000
Epoch 50/100
362/362 [==============================] - 1s 3ms/step - loss: 55749197824.0000
- val_loss: 58365517824.0000
Epoch 51/100
362/362 [==============================] - 1s 3ms/step - loss: 57870360576.0000
- val_loss: 55485554688.0000
Epoch 52/100
362/362 [==============================] - 1s 3ms/step - loss: 56799760384.0000
- val_loss: 55605215232.0000
Epoch 53/100
362/362 [==============================] - 1s 3ms/step - loss: 58351046656.0000
- val_loss: 58050686976.0000
Epoch 54/100
362/362 [==============================] - 1s 3ms/step - loss: 58240045056.0000
- val_loss: 56304787456.0000
Epoch 55/100
362/362 [==============================] - 1s 4ms/step - loss: 56973635584.0000
- val_loss: 55857664000.0000
Epoch 56/100
362/362 [==============================] - 2s 5ms/step - loss: 57536851968.0000
- val_loss: 54703652864.0000
Epoch 57/100
362/362 [==============================] - 1s 3ms/step - loss: 56966258688.0000
- val_loss: 55565430784.0000
Epoch 58/100
362/362 [==============================] - 1s 3ms/step - loss: 56727384064.0000
- val_loss: 56662274048.0000
Epoch 59/100
362/362 [==============================] - 1s 3ms/step - loss: 57057841152.0000
- val_loss: 56131297280.0000
Epoch 60/100
362/362 [==============================] - 1s 3ms/step - loss: 57106432000.0000
- val_loss: 57162002432.0000
Epoch 61/100
362/362 [==============================] - 1s 3ms/step - loss: 56996057088.0000
```

```
- val_loss: 55509757952.0000
Epoch 62/100
362/362 [==============================] - 1s 3ms/step - loss: 56949288960.0000
- val_loss: 56910897152.0000
Epoch 63/100
362/362 [==============================] - 1s 3ms/step - loss: 57478504448.0000
- val_loss: 56916652032.0000
Epoch 64/100
362/362 [==============================] - 1s 4ms/step - loss: 57280950272.0000
- val_loss: 57675218944.0000
Epoch 65/100
362/362 [==============================] - 1s 3ms/step - loss: 56205631488.0000
- val_loss: 55291617280.0000
Epoch 66/100
362/362 [==============================] - 2s 5ms/step - loss: 57029038080.0000
- val_loss: 56058564608.0000
Epoch 67/100
362/362 [==============================] - 1s 4ms/step - loss: 57152798720.0000
- val_loss: 56628772864.0000
Epoch 68/100
362/362 [==============================] - 1s 3ms/step - loss: 56707006464.0000
- val_loss: 55682416640.0000
Epoch 69/100
362/362 [==============================] - 1s 3ms/step - loss: 57880010752.0000
- val_loss: 56076984320.0000
Epoch 70/100
362/362 [==============================] - 1s 3ms/step - loss: 56574627840.0000
- val_loss: 55310065664.0000
Epoch 71/100
362/362 [==============================] - 1s 2ms/step - loss: 57581318144.0000
- val_loss: 57118375936.0000
Epoch 72/100
362/362 [==============================] - 1s 3ms/step - loss: 56771571712.0000
- val_loss: 56669999104.0000
Epoch 73/100
362/362 [==============================] - 1s 3ms/step - loss: 56921722880.0000
- val_loss: 55360172032.0000
Epoch 74/100
362/362 [==============================] - 1s 3ms/step - loss: 57648848896.0000
- val_loss: 57195765760.0000
Epoch 75/100
362/362 [==============================] - 1s 3ms/step - loss: 56947154944.0000
- val_loss: 60914499584.0000
Epoch 76/100
362/362 [==============================] - 1s 3ms/step - loss: 56933867520.0000
- val_loss: 55683928064.0000
Epoch 77/100
362/362 [==============================] - 2s 4ms/step - loss: 57165938688.0000
```

```
- val_loss: 55621263360.0000
Epoch 78/100
362/362 [==============================] - 2s 5ms/step - loss: 56995581952.0000
- val_loss: 56577425408.0000
Epoch 79/100
362/362 [==============================] - 1s 3ms/step - loss: 56160059392.0000
- val_loss: 58967621632.0000
Epoch 80/100
362/362 [==============================] - 1s 3ms/step - loss: 56674836480.0000
- val_loss: 55958745088.0000
Epoch 81/100
362/362 [==============================] - 1s 3ms/step - loss: 56419098624.0000
- val_loss: 55170904064.0000
Epoch 82/100
362/362 [==============================] - 1s 3ms/step - loss: 56387543040.0000
- val_loss: 56538132480.0000
Epoch 83/100
362/362 [==============================] - 1s 3ms/step - loss: 57166389248.0000
- val_loss: 55411765248.0000
Epoch 84/100
362/362 [==============================] - 1s 3ms/step - loss: 56693239808.0000
- val_loss: 57887612928.0000
Epoch 85/100
362/362 [==============================] - 2s 4ms/step - loss: 56454176768.0000
- val_loss: 54893154304.0000
Epoch 86/100
362/362 [==============================] - 1s 3ms/step - loss: 55829000192.0000
- val_loss: 54488121344.0000
Epoch 87/100
362/362 [==============================] - 1s 3ms/step - loss: 56232845312.0000
- val_loss: 56132669440.0000
Epoch 88/100
362/362 [==============================] - 2s 5ms/step - loss: 56657674240.0000
- val_loss: 59243200512.0000
Epoch 89/100
362/362 [==============================] - 1s 4ms/step - loss: 57006252032.0000
- val_loss: 55493210112.0000
Epoch 90/100
362/362 [==============================] - 1s 3ms/step - loss: 57289580544.0000
- val_loss: 56840744960.0000
Epoch 91/100
362/362 [==============================] - 1s 4ms/step - loss: 56579969024.0000
- val_loss: 55479316480.0000
Epoch 92/100
362/362 [==============================] - 1s 3ms/step - loss: 56307843072.0000
- val_loss: 54750064640.0000
Epoch 93/100
362/362 [==============================] - 1s 3ms/step - loss: 57106923520.0000
```

```
- val_loss: 55930478592.0000
Epoch 94/100
362/362 [==============================] - 1s 3ms/step - loss: 56176627712.0000
- val_loss: 57079234560.0000
Epoch 95/100
362/362 [==============================] - 1s 3ms/step - loss: 56924119040.0000
- val_loss: 55493402624.0000
Epoch 96/100
362/362 [==============================] - 1s 3ms/step - loss: 56580837376.0000
- val_loss: 61588004864.0000
Epoch 97/100
362/362 [==============================] - 1s 3ms/step - loss: 55827578880.0000
- val_loss: 55033217024.0000
Epoch 98/100
362/362 [==============================] - 2s 5ms/step - loss: 56983724032.0000
- val_loss: 55230414848.0000
Epoch 99/100
362/362 [==============================] - 2s 5ms/step - loss: 55896170496.0000
- val_loss: 54352429056.0000
Epoch 100/100
362/362 [==============================] - 1s 3ms/step - loss: 57560989696.0000
- val_loss: 57595682816.0000
```

[ ]:
```
y_pred_neurone = model.predict(X_test)
```

```
151/151 [==============================] - 0s 2ms/step
```

[ ]:
```
print("R2 score : ", r2_score(y_test, y_pred_neurone))
print("RMSE : ", np.sqrt(mean_squared_error(y_test, y_pred_neurone)))
print("MSE : ", mean_squared_error(y_test, y_pred_neurone))
print("MAE :", mean_absolute_error(y_test, y_pred_neurone))
```

```
R2 score :  0.5603324218892574
RMSE :  249486.69510825898
MSE :  62243611036.041374
MAE : 157189.59336949483
```

9 KNN(k-nearest neighbors)

Un modèle KNN (k-nearest neighbors) est un algorithme d'apprentissage automatique utilisé pour résoudre des problèmes de classification et de régression. Il appartient à la catégorie des méthodes d'apprentissage supervisé, ce qui signifie qu'il utilise un ensemble de données d'entraînement avec des étiquettes (dans le cas de la classification) ou des valeurs cibles (dans le cas de la régression) pour effectuer des prédictions sur de nouvelles données. Le principe fondamental du modèle KNN est basé sur la similarité des données : il suppose que des points de données similaires ont tendance à avoir la même étiquette (en classification) ou une valeur cible similaire (en régression). L'idée centrale est de trouver les k voisins les plus proches (d'où le nom "k-nearest neighbors") d'un point de données donné dans l'ensemble de données d'entraînement, puis d'utiliser les étiquettes ou les

valeurs cibles de ces voisins pour faire une prédiction.

[ ]:
```
from sklearn.neighbors import KNeighborsRegressor
```

[ ]:
```
hyperparameters = {'n_neighbors': range(1, 10), 'weights': ['uniform',␣
↪'distance']}
```

[ ]:
```
# initialiser le modèle KNN
knn = KNeighborsRegressor()

# initialiser la recherche de grille avec validation croisée
grid_search = GridSearchCV(knn, hyperparameters, cv=5)
```

[ ]:
```
# entraîner la recherche de grille sur les données d'apprentissage
grid_search.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=KNeighborsRegressor(),
             param_grid={'n_neighbors': range(1, 10),
                         'weights': ['uniform', 'distance']})
```

[ ]:
```
print("Hyperparameters optimaux:", grid_search.best_params_)
```

```
Hyperparameters optimaux: {'n_neighbors': 9, 'weights': 'distance'}
```

[ ]:
```
knn = KNeighborsRegressor(n_neighbors=9, weights='distance')
knn.fit(X_train, y_train)
```

[ ]:
```
KNeighborsRegressor(n_neighbors=9, weights='distance')
```

[ ]:
```
y_pred_knn = knn.predict(X_test)
```

[ ]:
```
print("R2 score : ", r2_score(y_test, y_pred_knn))
print("RMSE : ", np.sqrt(mean_squared_error(y_test, y_pred_knn)))
print("MSE : ", mean_squared_error(y_test, y_pred_knn))
print("MAE :", mean_absolute_error(y_test, y_pred_knn))
```

```
R2 score :  0.48496628281329857
RMSE :  270024.39409107255
MSE :  72913173404.25085
MAE : 160286.6678020077
```

10 GradientBoosting

Un modèle de gradient boosting, également appelé “Gradient Boosting Machine” ou GBM, est une technique d’apprentissage automatique qui appartient à la famille des méthodes d’ensemble. Les méthodes de gradient boosting sont utilisées pour résoudre des problèmes de classification et de régression, et elles sont connues pour leur performance élevée dans une variété de domaines.

Le gradient boosting fonctionne en construisant itérativement un modèle prédictif fort en combinant plusieurs modèles faibles, généralement des arbres de décision peu profonds. Le processus de construction du modèle se fait de manière séquentielle, en corrigeant les erreurs des modèles précédents.

[20]:
```
from sklearn.ensemble import GradientBoostingRegressor
```

[ ]:
```
params = {
    'n_estimators': [100, 500, 1000],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 1],
    'subsample': [0.5, 0.8, 1],
    'min_samples_leaf': [1, 3, 5],
    'max_features': ['auto', 'sqrt', 'log2']
}

hyperparameters = {'learning_rate': [0.1, 0.01, 0.001], 'n_estimators': [100,␣
 ↪500, 1000], 'max_depth': [3, 5, 7]}

# Créer un modèle Gradient Boosting
model = GradientBoostingRegressor()
```

[ ]:
```
grid_search = GridSearchCV(model, hyperparameters, cv=5, n_jobs=-1)
```

[ ]:
```
grid_search.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=GradientBoostingRegressor(), n_jobs=-1,
             param_grid={'learning_rate': [0.1, 0.01, 0.001],
                         'max_depth': [3, 5, 7],
                         'n_estimators': [100, 500, 1000]})
```

[ ]:
```
print("Hyperparameters optimaux:", grid_search.best_params_)
```

```
Hyperparameters optimaux: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators':
1000}
```

[21]:
```
model_grad = GradientBoostingRegressor(learning_rate=0.1, max_depth=5,␣
 ↪n_estimators= 1000)
```

[22]:
```
model_grad.fit(X_train, y_train)
```

[22]:
```
GradientBoostingRegressor(max_depth=5, n_estimators=1000)
```

[23]:
```
y_pred_grad_boos = model_grad.predict(X_test)
```

[24]:
```
print("R2 score : ", r2_score(y_test, y_pred_grad_boos))
print("RMSE : ", np.sqrt(mean_squared_error(y_test, y_pred_grad_boos)))
print("MSE : ", mean_squared_error(y_test, y_pred_grad_boos))
```

```
print("MAE :", mean_absolute_error(y_test, y_pred_grad_boos))
```

```
R2 score :  0.8887994098596639
RMSE :  125469.6583767593
MSE :  15742635173.180683
MAE : 68512.8185909716
```

11 Hist Gradient boosting

Le modèle d'histogramme de gradient boosting est une variante du gradient boosting, spécifiquement conçue pour améliorer les performances et l'efficacité de cette technique, principalement dans le contexte de l'apprentissage automatique sur de grands ensembles de données. Cette variante utilise une représentation en histogramme des données pour accélérer le processus d'apprentissage et réduire la complexité des calculs.

[25]:
```
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingRegressor
```

```
/usr/local/lib/python3.9/dist-
packages/sklearn/experimental/enable_hist_gradient_boosting.py:16: UserWarning:
Since version 1.0, it is not needed to import enable_hist_gradient_boosting
anymore. HistGradientBoostingClassifier and HistGradientBoostingRegressor are
now stable and can be normally imported from sklearn.ensemble.
  warnings.warn(
```

[ ]:
```
# Define hyperparameters to optimize
param_grid = {
    'learning_rate': [0.1, 0.05, 0.01],
    'max_depth': [3, 5, 7],
    'min_samples_leaf': [1, 2, 4]
}
```

[ ]:
```
# Create instance of HistGradientBoostingRegressor estimator
estimator = HistGradientBoostingRegressor()

# Create instance of GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(estimator, param_grid=param_grid, cv=5, n_jobs=-1)
```

[ ]:
```
# Fit the model using GridSearchCV
grid_search.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=HistGradientBoostingRegressor(), n_jobs=-1,
             param_grid={'learning_rate': [0.1, 0.05, 0.01],
                         'max_depth': [3, 5, 7],
                         'min_samples_leaf': [1, 2, 4]})
```

[ ]:
```
print("Hyperparameters optimaux:", grid_search.best_params_)
```

```
Hyperparameters optimaux: {'learning_rate': 0.1, 'max_depth': 7,
'min_samples_leaf': 1}
```

[26]:
```
model_hist_grad = HistGradientBoostingRegressor(learning_rate=0.1, max_depth=7,␣
↪min_samples_leaf=1)
```

[27]:
```
model_hist_grad.fit(X_train, y_train)
```

[27]:
```
HistGradientBoostingRegressor(max_depth=7, min_samples_leaf=1)
```

[28]:
```
y_pred_hist_grad = model_hist_grad.predict(X_test)
```

[29]:
```
print("R2 score : ", r2_score(y_test, y_pred_hist_grad))
print("RMSE : ", np.sqrt(mean_squared_error(y_test, y_pred_hist_grad)))
print("MSE : ", mean_squared_error(y_test, y_pred_hist_grad))
print("MAE :", mean_absolute_error(y_test, y_pred_hist_grad))
```

```
R2 score :  0.8803928818450442
RMSE :  130125.88096486217
MSE :  16932744896.881477
MAE : 69868.25791006602
```

12 Staking

12.1 StackingRegressor

[32]:
```
from sklearn.ensemble import StackingRegressor
from sklearn.linear_model import LinearRegression
```

[33]:
```
estimator = [
    ("Foret", rf_1),
    ("xgboost", xgb_model),
    ("GradientBoosting", model_grad)
]
```

[34]:
```
stack_regression = StackingRegressor(
    estimators = estimator,
    final_estimator = LinearRegression()
)
```

[35]:
```
stack_regression.fit(X_train, y_train)
```

[35]:
```
StackingRegressor(estimators=[('Foret',
                               RandomForestRegressor(max_depth=30,
                                                     n_estimators=500)),
                              ('xgboost',
                               XGBRegressor(base_score=None, booster=None,
                                            callbacks=None,
```

```
                                                colsample_bylevel=None,
                                                colsample_bynode=None,
                                                colsample_bytree=0.8,
                                                early_stopping_rounds=None,
                                                enable_categorical=False,
                                                eval_metric=None,
                                                feature_types=None, gamma=None,
                                                gpu_id=None, grow_policy=…
                                                max_cat_threshold=None,
                                                max_cat_to_onehot=None,
                                                max_delta_step=None, max_depth=4,
                                                max_leaves=None, min_child_weight=1,
                                                missing=nan,
                                                monotone_constraints=None,
                                                n_estimators=1000, n_jobs=None,
                                                num_parallel_tree=None,
                                                predictor=None, random_state=None,
    …)),
                                ('GradientBoosting',
                                 GradientBoostingRegressor(max_depth=5,
                                                           n_estimators=1000))],
                    final_estimator=LinearRegression())
```

[36]:
```
y_pred_stack = stack_regression.predict(X_test)
```

[37]:
```
print("R2 score : ", r2_score(y_test, y_pred_stack))
print("RMSE : ", np.sqrt(mean_squared_error(y_test, y_pred_stack)))
print("MSE : ", mean_squared_error(y_test, y_pred_stack))
print("MAE :", mean_absolute_error(y_test, y_pred_stack))
```

```
R2 score :  0.8967456900870034
RMSE :  120903.60869195209
MSE :  14617682594.736671
MAE : 65839.59068459326
```

12.2 Avec Voting Regressor

[30]:
```
from sklearn.ensemble import VotingRegressor
ereg = VotingRegressor(estimators=[('xgboost', xgb_model), ('GradientBoosting',␣
 ↪model_grad), ("HistGradBoos", model_hist_grad)])
ereg = ereg.fit(X_train, y_train)
y_pred_ereg = ereg.predict(X_test)
```

[38]:
```
print("R2 score : ", r2_score(y_test, y_pred_ereg))
print("RMSE : ", np.sqrt(mean_squared_error(y_test, y_pred_ereg)))
print("MSE : ", mean_squared_error(y_test, y_pred_ereg))
print("MAE :", mean_absolute_error(y_test, y_pred_ereg))
```

```
R2 score :  0.8928970138262748
RMSE :  123136.26018574712
MSE :  15162538572.532011
MAE : 66433.92296156764
```

13 Prévision des prix

[39]:
```
price.head(5)
```

[39]:
```
            id  price  bedrooms  bathrooms  sqft_living  sqft_lot  floors  \
4   2487200875    NaN         4       3.00         1960      5000     1.0
20  7983200060    NaN         3       1.00         1250      9774     1.0
27  1794500383    NaN         3       1.75         2450      2691     2.0
46  8035350320    NaN         3       2.50         3160     13603     2.0
51  5245600105    NaN         3       1.00         1190      9199     1.0

    waterfront  view  condition  …  yr_built  yr_renovated  zipcode  \
4            0     0          5  …      1965             0    98136
20           0     0          4  …      1969             0    98003
27           0     0          3  …      1915             0    98119
46           0     0          3  …      2003             0    98019
51           0     0          3  …      1955             0    98148

        lat     long  sqft_living15  sqft_lot15  part_maison_sur_terrain  \
4   47.5208 -122.393           1360        5000                 0.392000
20  47.3343 -122.306           1280        8850                 0.127890
27  47.6386 -122.360           1760        3573                 0.910442
46  47.7443 -121.977           3050        9232                 0.232302
51  47.4258 -122.322           1190        9364                 0.129362

    Part_logement_dessus_sol_sur_terrai  part_sous_sol_sur_logement
4                              0.535714                    0.464286
20                             1.000000                    0.000000
27                             0.714286                    0.285714
46                             1.000000                    0.000000
51                             1.000000                    0.000000

[5 rows x 21 columns]
```

[40]:
```
price.columns
```

[40]:
```
Index(['id', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'view', 'condition', 'grade', 'yr_built',
       'yr_renovated', 'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15',
       'part_maison_sur_terrain', 'Part_logement_dessus_sol_sur_terrai',
       'part_sous_sol_sur_logement'],
      dtype='object')
```

[41]:
```
X_test.columns
```

[41]:
```
Index(['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'yr_built', 'yr_renovated',
       'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15',
       'part_maison_sur_terrain', 'Part_logement_dessus_sol_sur_terrai',
       'part_sous_sol_sur_logement'],
      dtype='object')
```

[42]:
```
price = price.drop(columns=["id", "price"])
```

[43]:
```
prediction_random_forest = rf_1.predict(price)
prediction_xgboost = xgb_model.predict(price)
prediction_bagging = bagging.predict(price)
prediction_gradient_boost = model_grad.predict(price)
prediction_hist_gradien_boosting = model_hist_grad.predict(price)
prediction_model_stacking = stack_regression.predict(price)
prediction_model_voting = ereg.predict(price)
```

[44]:
```
mes_previsions = pd.DataFrame({'RandomForest': prediction_random_forest,
                               'XGBoost': prediction_xgboost,
                               'Bagging':prediction_bagging,
                               'GradientBoosting':prediction_gradient_boost,
                               'HistogramGradientBoosting':
 ↪prediction_hist_gradien_boosting,
                               'StackingRegressor':prediction_model_stacking,
                               'VotingRegressor':prediction_model_voting
                               })
```

[48]:
```
mes_previsions.head(5)
```

[48]:

	RandomForest	XGBoost	Bagging	GradientBoosting	HistogramGradientBoosting	StackingRegressor	VotingRegressor
0	496503.950000	502422.437500	490329.0	496792.708890	441352.664043	499129.180810	471652.204169
1	215777.880214	205533.656250	215142.0	213678.354178	201742.369009	204945.590553	209462.457743
2	819543.106000	877646.250000	838574.0	929563.016422	838501.482519	879541.849822	862798.199316
3	520824.630000	543513.125000	530470.0	516504.494205	548593.539994	535310.025014	528792.658785
4	238273.930765	213961.640625	235527.0	200053.341267	213458.174030	211201.378028	209140.145218

[49]:
```
moyennes = mes_previsions.mean(axis=1)
mes_previsions = mes_previsions.assign(Ma_prevision =moyennes)
```

[52]:
```
mes_previsions.head(5)
```

[52]:

	RandomForest	XGBoost	Bagging	GradientBoosting	\
0	496503.950000	502422.437500	490329.0	496792.708890	
1	215777.880214	205533.656250	215142.0	213678.354178	
2	819543.106000	877646.250000	838574.0	929563.016422	
3	520824.630000	543513.125000	530470.0	516504.494205	
4	238273.930765	213961.640625	235527.0	200053.341267	

	HistogramGradientBoosting	StackingRegressor	VotingRegressor	\
0	441352.664043	499129.180810	471652.204169	
1	201742.369009	204945.590553	209462.457743	
2	838501.482519	879541.849822	862798.199316	
3	548593.539994	535310.025014	528792.658785	
4	213458.174030	211201.378028	209140.145218	

	Ma_prevision
0	485454.592202
1	209468.901135
2	863738.272011
3	532001.210428
4	217373.658562

[53]:
```
mes_previsions.to_csv('resultats_predictions.csv', index=False)
```

[54]:
```
from google.colab import files
files.download('resultats_predictions.csv')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```