

0000

HOUSE PRICES -

PRESENTATION
PARTICIPEZ À UNE
COMPÉTITION
KAGGLE

HAFIDA HAMDACHE

TABLE OF CONTENTS

- Présentation du projet
- Nettoyage du jeu de donnée, analyse exploratoire et features engineering
- Modèles Machines learning
- Choix du modèle final



PRÉDICTION DES PRIX DES MAISONS À AMES, IOWA

Cette compétition met à votre disposition un ensemble de données comprenant 79 variables explicatives décrivant presque tous les aspects des maisons résidentielles à Ames, Iowa. Votre défi consiste à prédire le prix final de chaque maison en utilisant des techniques avancées de régression et d'ingénierie de variables.

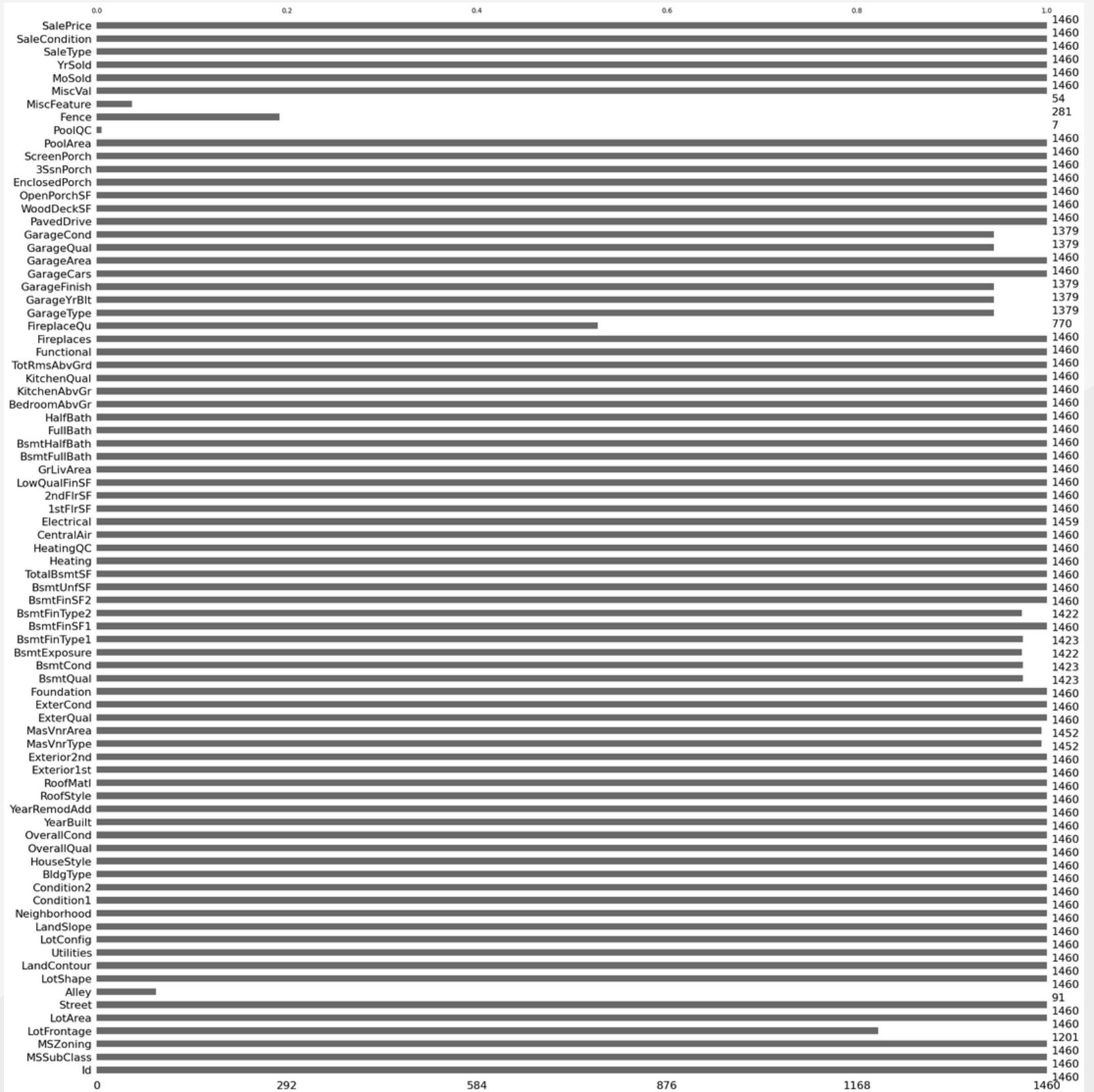
1. Maîtriser l'ingénierie de variables créative pour améliorer les performances de prédiction.

2. Utiliser des techniques avancées de régression telles que les forêts aléatoires et le gradient boosting.

3. Appliquer des compétences en science des données pour résoudre un problème réel dans le domaine de l'immobilier.

4 DATASETS

- *train.csv* y = Saleprice
- *test.csv*
- *data_description.txt*
- *sample_submission.csv*



Valeurs manquantes : 6965 NaN pour 118260 données (5.89 %)

Nombre et pourcentage de valeurs manquantes par variable

	Nombres de valeurs manquantes	% de valeurs manquantes
PoolQC	1453	99.52
MiscFeature	1406	96.30
Alley	1369	93.77
Fence	1179	80.75
FireplaceQu	690	47.26
LotFrontage	259	17.74
GarageType	81	5.55
GarageYrBlt	81	5.55
GarageFinish	81	5.55
GarageQual	81	5.55
GarageCond	81	5.55
BsmtExposure	38	2.60
BsmtFinType2	38	2.60
BsmtFinType1	37	2.53
BsmtCond	37	2.53
BsmtQual	37	2.53
MasVnrArea	8	0.55
MasVnrType	8	0.55
Electrical	1	0.07

TRAITEMENT DES VALEURS MANQUANTES

1

Suppression des colonnes avec un seuil de NaN à 0.80

2

Imputation par inexistant
`Fillna("inexistant")`
(object)

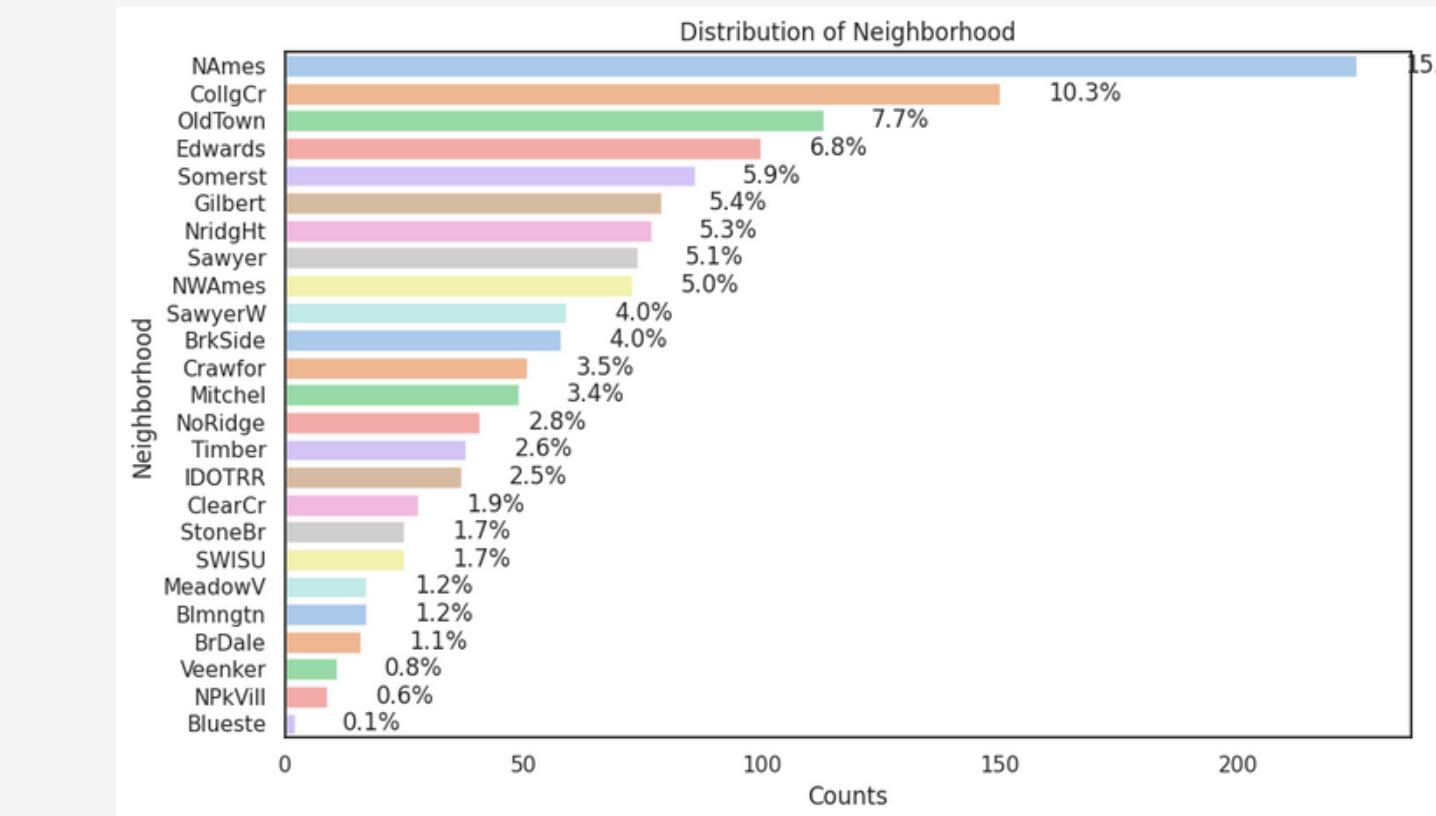
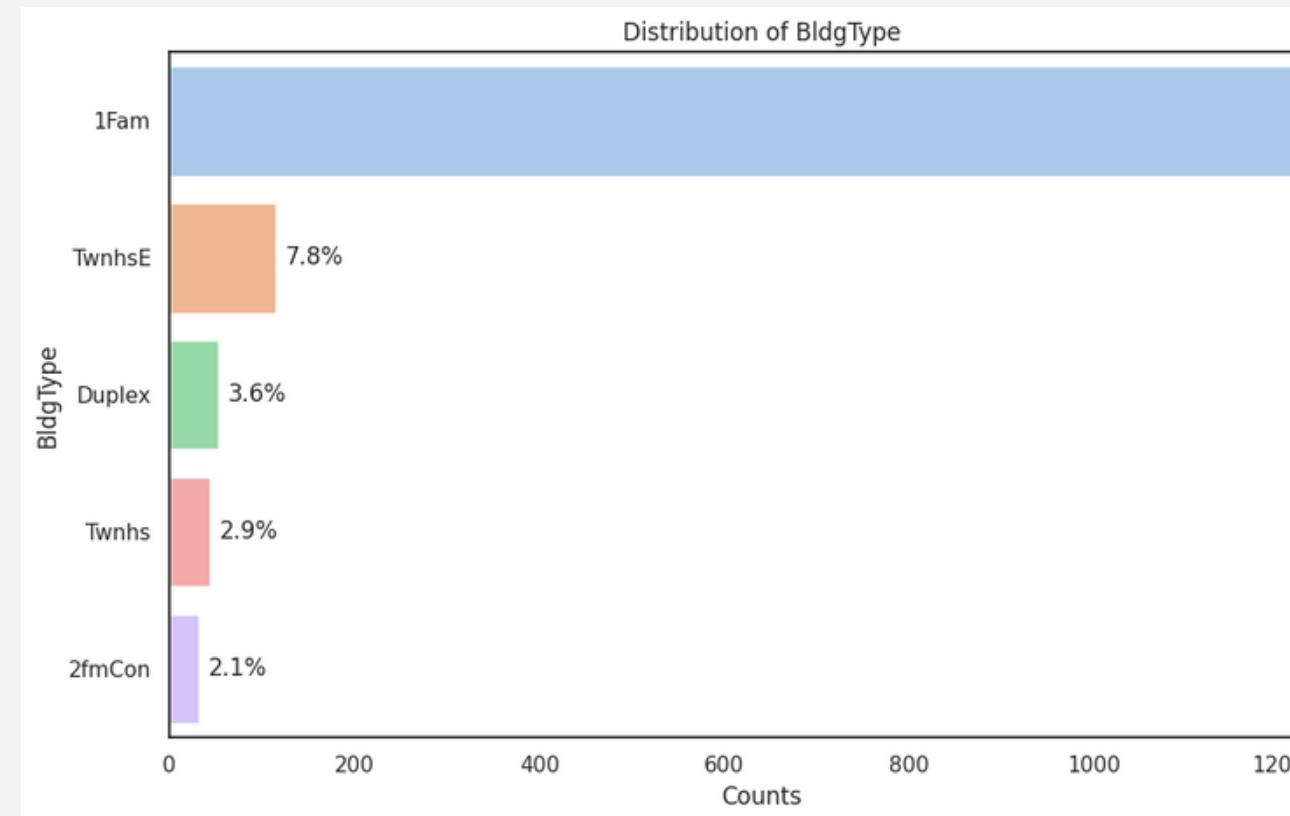
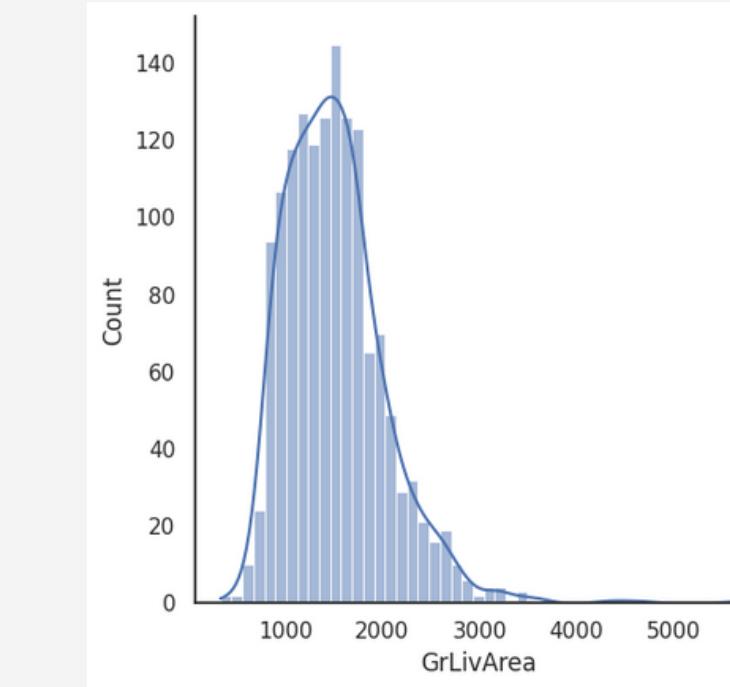
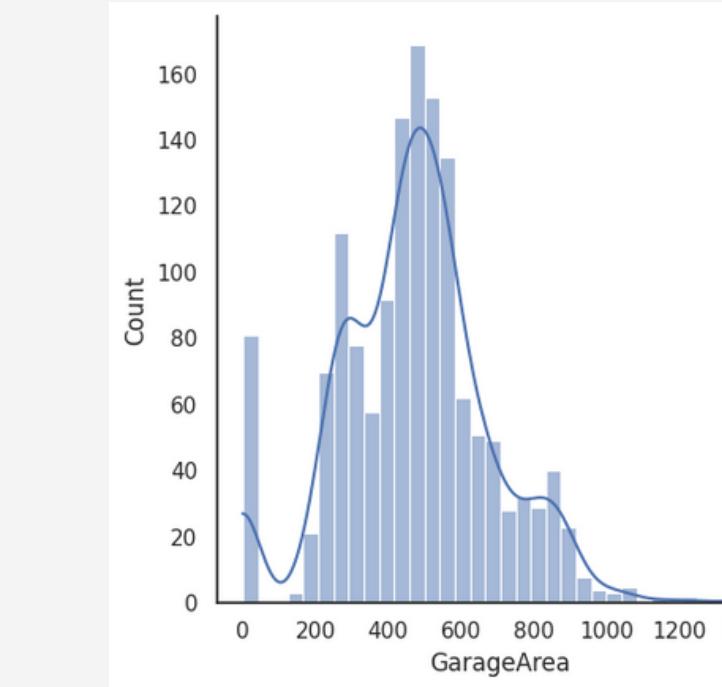
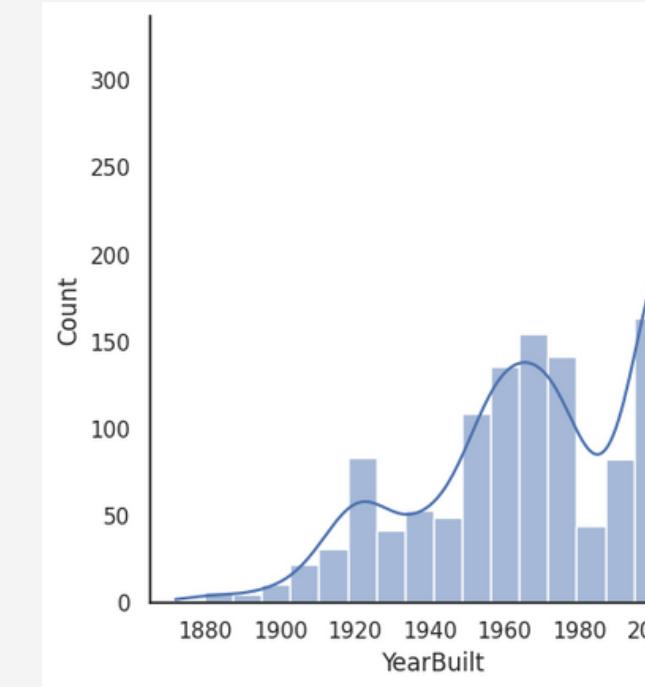
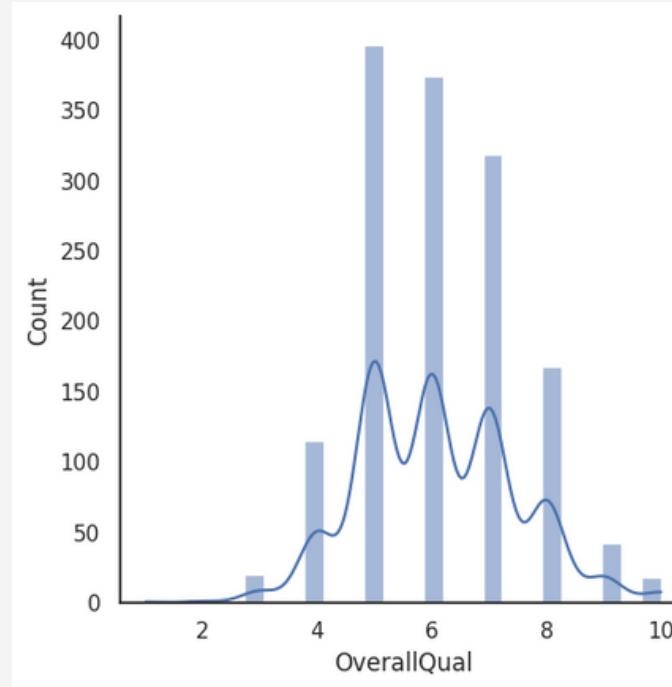
4

Imputation par la moyenne du groupe
`Fillna("mean")`

3

Imputation par 0
`Fillna(0)`
(float, int)

ANALYSE EXPLORATOIRE UNIVARIÉE



FEATURES ENGINEERING

```
[ ] # Age du bien
def calculate_age(df):
    df['Age'] = df['YrSold'] - df['YearBuilt']
    return df

# Age de la rénovation
def calculate_renovation_age(df):
    df['RenovationAge'] = df['YrSold'] - df['YearRemodAdd']
    return df

# Surface totale
def calculate_total_area(df):
    df['TotalArea'] = df['1stFlrSF'] + df['2ndFlrSF'] + df['TotalBsmtSF']
    return df

# Total des salles de bains
def calculate_total_bathrooms(df):
    df['TotalBathrooms'] = df['FullBath'] + 0.5 * df['HalfBath']
    return df

# Qualité globale
def calculate_overall_quality(df):
    df['OverallQuality'] = df['OverallQual'] * df['OverallCond']
    return df

# Total des superficies du sous-sol
def calculate_total_bsmt_area(df):
    df['TotalBsmtArea'] = df['BsmtFinSF1'] + df['BsmtFinSF2'] + df['BsmtUnfSF']
    return df

# Présence d'une piscine
def calculate_pool_presence(df):
    df['HasPool'] = df['PoolArea'].apply(lambda x: 1 if x > 0 else 0)
    return df

# Total des superficies extérieures
def calculate_total_outdoor_area(df):
    df['TotalOutdoorArea'] = df['WoodDeckSF'] + df['OpenPorchSF'] + df['EnclosedPorch'] + df['3SsnPorch'] + df['ScreenPorch']
    return df

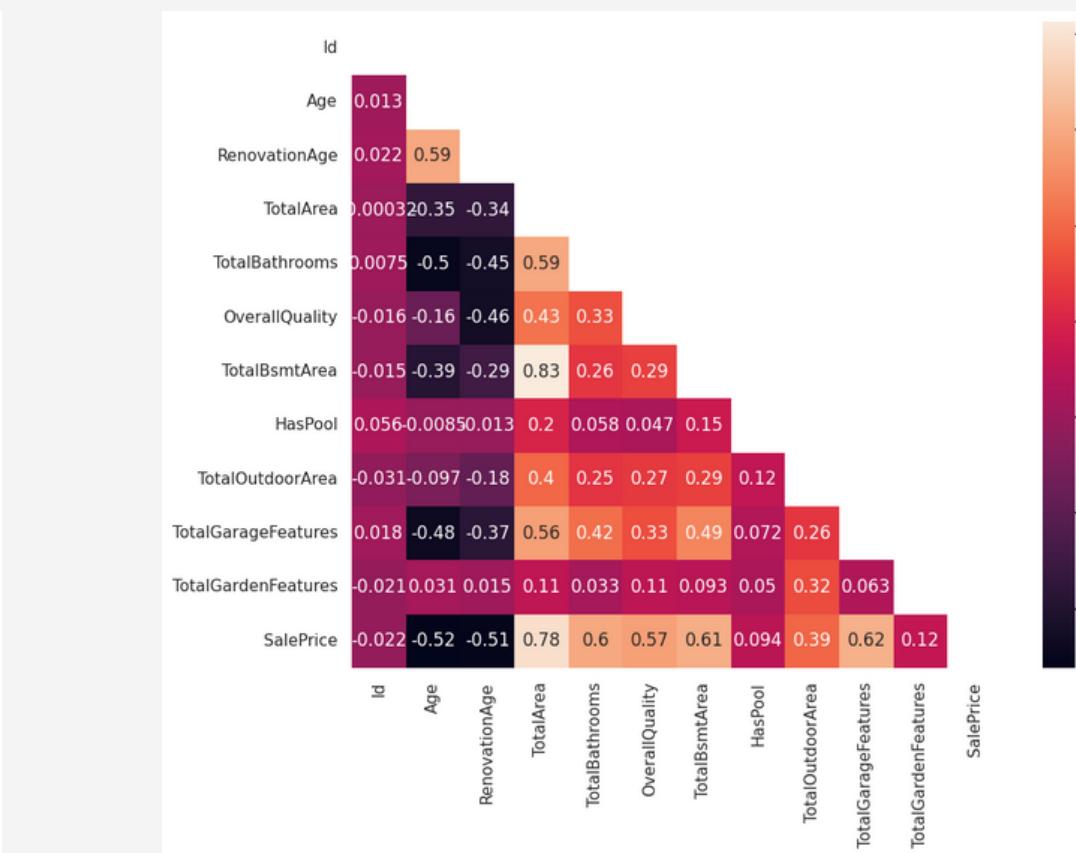
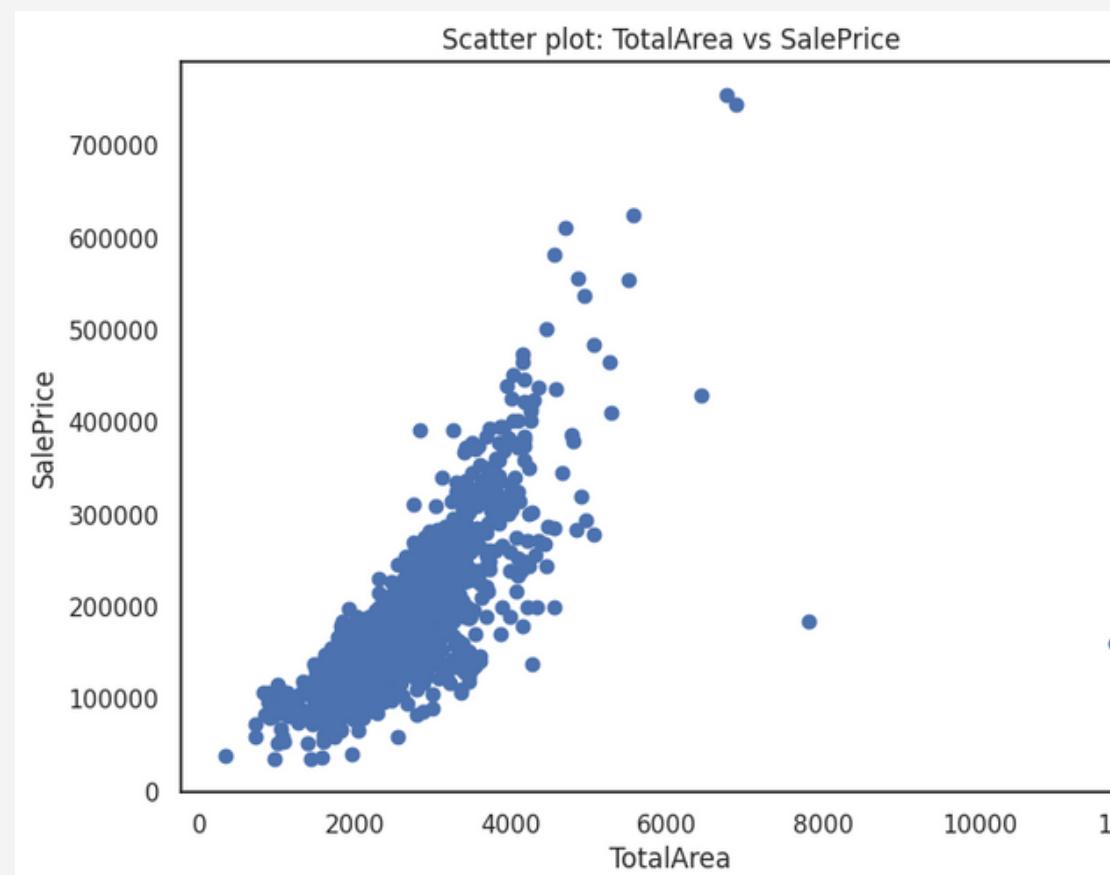
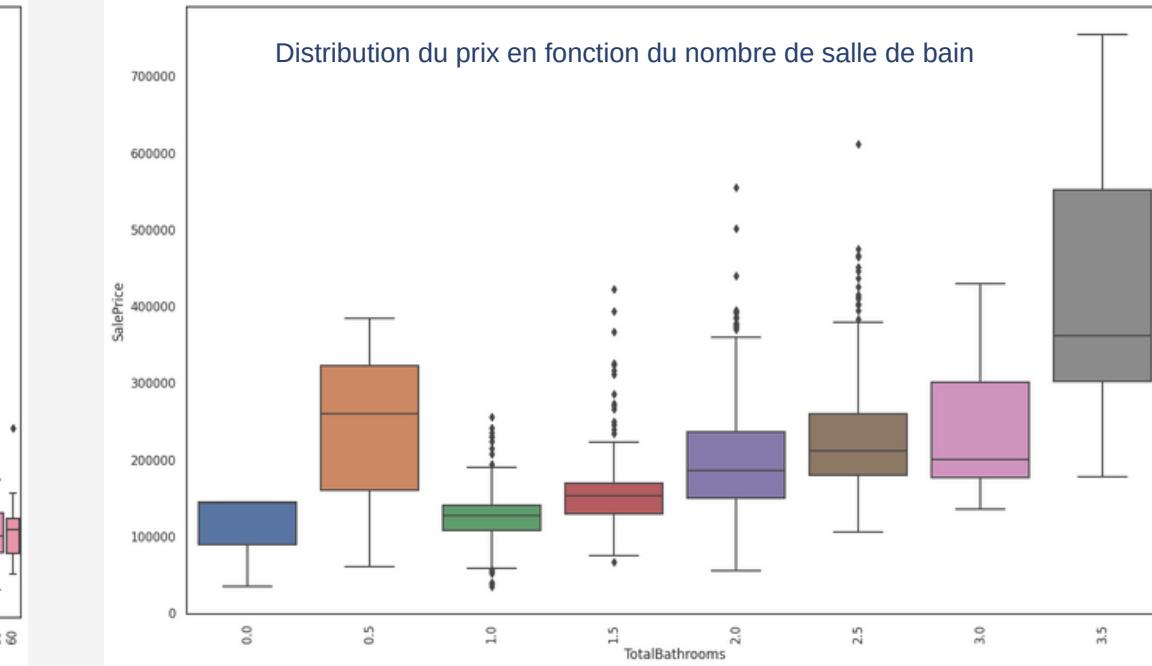
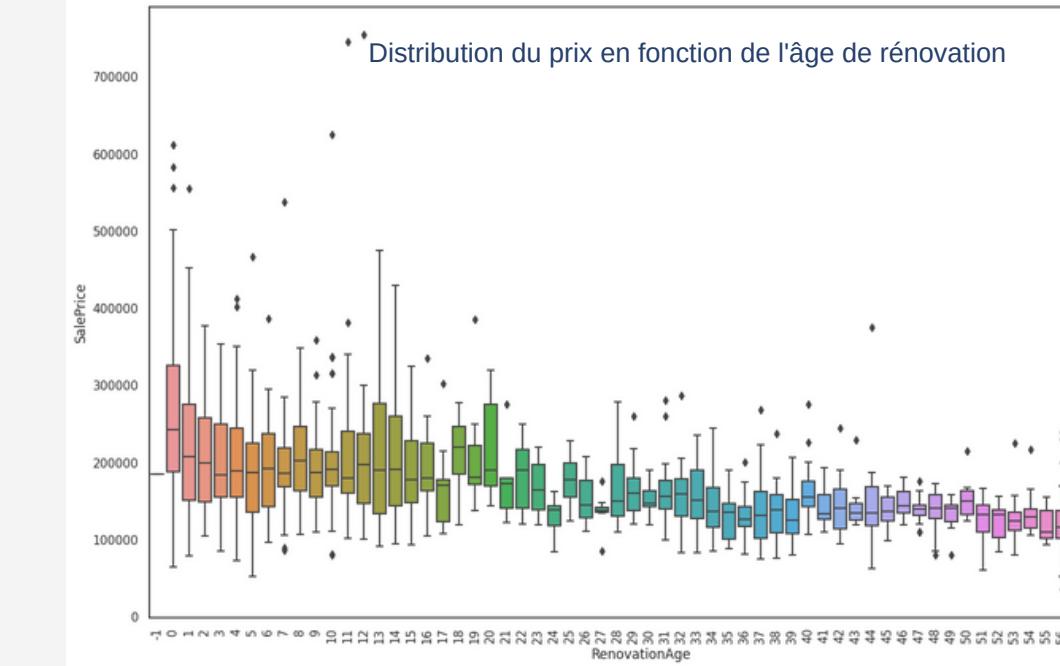
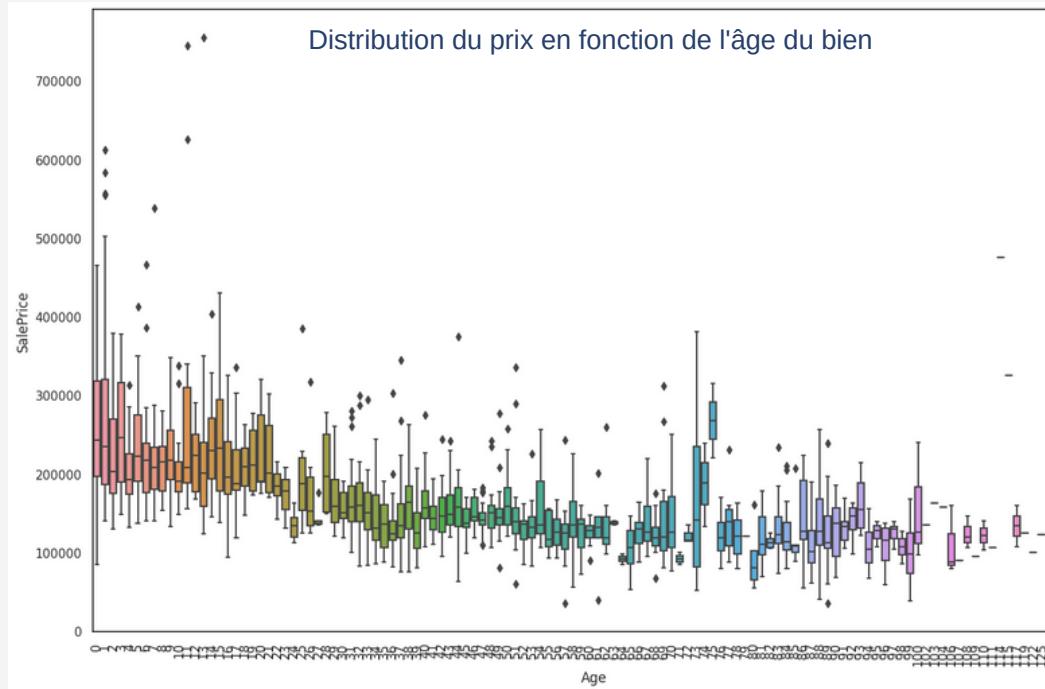
# Total des caractéristiques du garage
def calculate_total_garage_features(df):
    df['TotalGarageFeatures'] = df['GarageCars'] + df['GarageArea']
    return df

# Total des caractéristiques du jardin
def calculate_total_garden_features(df):
    df['TotalGardenFeatures'] = df['3SsnPorch'] + df['ScreenPorch']
    return df

# Appliquer toutes les fonctions de features engineering
def apply_feature_engineering(df):
    df = calculate_age(df)
    df = calculate_renovation_age(df)
    df = calculate_total_area(df)
    df = calculate_total_bathrooms(df)
    df = calculate_overall_quality(df)
    df = calculate_total_bsmt_area(df)
    df = calculate_pool_presence(df)
    df = calculate_total_outdoor_area(df)
    df = calculate_total_garage_features(df)
    df = calculate_total_garden_features(df)

    return df[['Id', 'Age', 'RenovationAge', 'TotalArea', 'TotalBathrooms', 'OverallQuality',
              'TotalBsmtArea', 'HasPool', 'TotalOutdoorArea', 'TotalGarageFeatures',
              'TotalGardenFeatures']]
```

ANALYSE EXPLORATOIRE BIVARIÉE



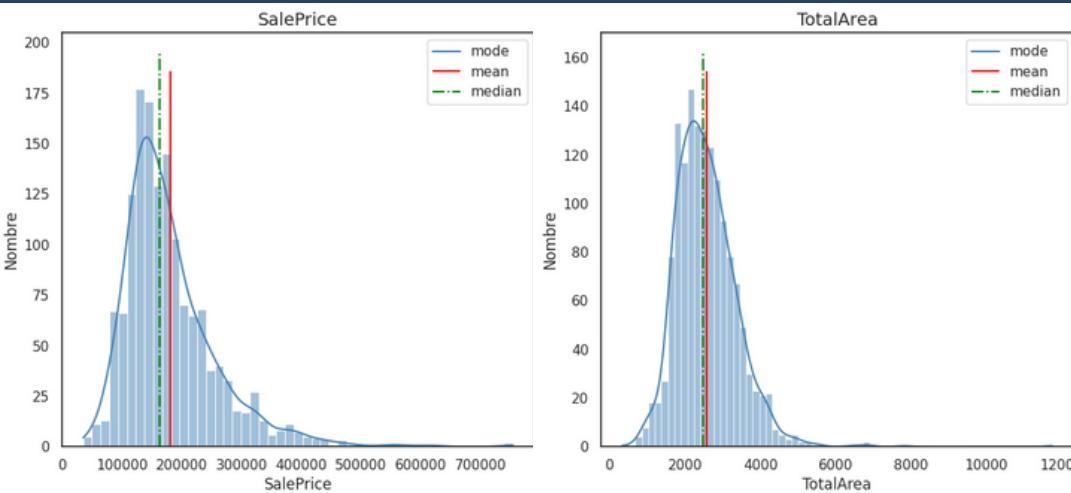
AVANT DE PASSER À LA MODÉLISATION...

TEST DE COLINÉARITÉ

```
[ ] # Observation des variables avec de fortes corrélations
corr_min = 0.7
corr_pairs = matrix.unstack().sort_values(kind="quicksort")
strong_corr = (pd.DataFrame(corr_pairs[(abs(corr_pairs) > corr_min)])
               .reset_index().rename(columns={0:'coef_correl'}))
strong_corr = strong_corr[(strong_corr.index[2] == 0) & (strong_corr['level_0'] != strong_corr['level_1'])]
strong_corr.sort_values('coef_correl', ascending=False).head(15)

      level_0  level_1  coef_correl
2  TotalBsmtArea  TotalArea     0.826742
0    TotalArea  SalePrice     0.782260
```

PASSAGE LOG



TEST D'HOMOSCEDASTICITÉ

```
[ ] from scipy.stats import levene

def test_homoscedasticity(df):
    features = df.columns.tolist()
    for feature in features:
        stat, p = levene(df[feature], df['SalePrice_log'])
        print(f"Feature: {feature}")
        print(f"Levene's test statistic: {stat}")
        print(f"p-value: {p}")
        print("-" * 50)

# Appel de la fonction pour tester l'homoscédasticité des features
test_homoscedasticity(train_features)
```

Les deux variables fortement corrélées peuvent entraîner une redondance d'informations, ce qui peut affecter négativement les performances du modèle d'apprentissage automatique. Suppression de 'TotalBsmtArea'

forte asymétrie à droite (skewed > 1). Le passage au log a pour objectif principal d'aider les algorithmes de machine learning à travailler de manière plus précise et efficace en transformant les variables explicatives et cible de manière objective. Cette transformation consiste à se rapprocher le plus possible d'une distribution normale.

Les résultats indiquaient que pour la plupart des features, la valeur p est très faible, ce qui suggère une inégalité des variances entre les groupes.

MODÈLES DE TYPE LINÉAIRE

Dummy regressor

Modèle de régression simplifié qui prédit constamment la valeur moyenne de la variable cible, sans prendre en compte les caractéristiques des données d'entrée, servant ainsi de point de référence pour évaluer la performance des modèles de régression plus complexes.

Lasso

Méthode de régression linéaire régularisée qui ajoute une pénalité L1 à la fonction de coût, ce qui favorise la sélection automatique des variables et permet d'obtenir des solutions parcimonieuses en forçant certains coefficients à zéro.

Ridge

Méthode de régression linéaire régularisée qui ajoute une pénalité L2 à la fonction de coût, permettant de réduire la variance et de prévenir le surajustement, tout en maintenant une certaine quantité de biais.

SVR

Utilise des vecteurs de support pour estimer une fonction qui s'ajuste aux données tout en minimisant les erreurs de prédiction.

Kernel Ridge

Utilise une fonction noyau pour projeter les données dans un espace de dimension supérieure, permettant ainsi de capturer des relations non linéaires entre les variables explicatives et la variable cible

MODÈLES DE TYPE NOYEAU

MODÈLES DE TYPE ENSEMBLISTE

Random Forest

algorithme d'apprentissage supervisé qui combine plusieurs arbres de décision pour créer un modèle prédictif robuste et précis.

XGboost

algorithme d'apprentissage supervisé qui utilise une approche en boosting pour créer un modèle prédictif en combinant plusieurs modèles plus faibles de manière séquentielle, en se concentrant sur les observations mal prédites lors des étapes précédentes.

RÉSULTATS MODÈLES DE TYPE LINÉAIRE

	Model	Params	Train_RMSE	Val_RMSE	Fit_time
1	Ridge	{}	0.370902	0.342137	0.118414
3	ElasticNet	{}	0.424046	0.400980	0.209106
2	Lasso	{}	0.437029	0.414787	0.273628
0	Dummy	{}	0.629403	0.615121	0.053417

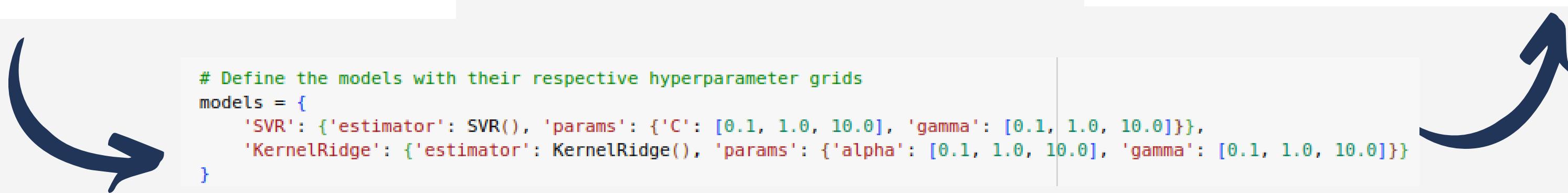
	Model	Params	Train_RMSE	Val_RMSE	Fit_time
1	Ridge	{'alpha': 0.1}	0.370895	0.342448	0.278642
3	ElasticNet	{'alpha': 10.0, 'l1_ratio': 0.8}	0.510465	0.494750	1.752027
2	Lasso	{'alpha': 10.0}	0.512257	0.496895	0.514678
0	Dummy	{}	0.629403	0.615121	0.049566

```
# Define the models with their respective hyperparameter grids
models = {
    'Dummy': {'estimator': DummyRegressor(), 'params': {}},
    'Ridge': {'estimator': Ridge(), 'params': {'alpha': [0.1, 1.0, 10.0]}},
    'Lasso': {'estimator': Lasso(), 'params': {'alpha': [0.1, 1.0, 10.0]}},
    'ElasticNet': {'estimator': ElasticNet(), 'params': {'alpha': [0.1, 1.0, 10.0], 'l1_ratio': [0.2, 0.5, 0.8]}}}
```

RÉSULTATS MODÈLES DE TYPE NOYEAU

	Model	Params	Train_RMSE	Val_RMSE	Fit_time
0	SVR	{}	0.448252	0.436519	1.034293
1	KernelRidge	{}	0.481294	0.450448	0.430688

	Model	Params	Train_RMSE	Val_RMSE	Fit_time
1	KernelRidge	{'alpha': 10.0, 'gamma': 0.1}	0.481728	0.451089	2.601053
0	SVR	{'C': 1.0, 'gamma': 0.1}	0.344585	0.615196	12.205376



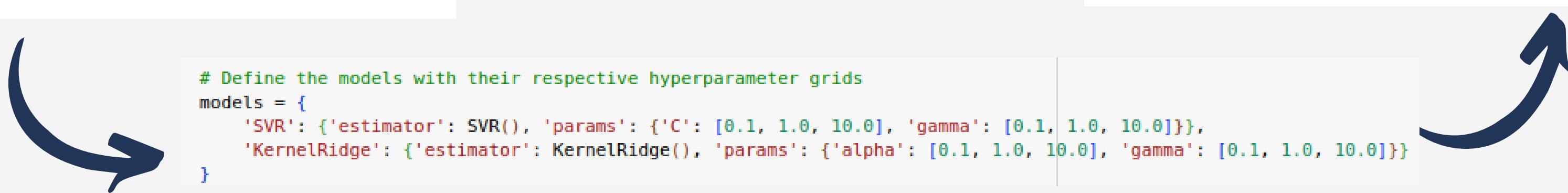
RÉSULTATS MODÈLES DE TYPE ENSEMBLISTE

	Model	Params	Train_RMSE	Val_RMSE	Fit_time
0	Random Forest	{'max_depth': 10, 'n_estimators': 500}	0.936412	0.696761	52.923970
1	XGBoost	{'max_depth': 5, 'n_estimators': 100}	0.993079	0.707670	12.340215

RÉSULTATS MODÈLES DE TYPE NOYEAU

	Model	Params	Train_RMSE	Val_RMSE	Fit_time
0	SVR	{}	0.448252	0.436519	1.034293
1	KernelRidge	{}	0.481294	0.450448	0.430688

	Model	Params	Train_RMSE	Val_RMSE	Fit_time
1	KernelRidge	{'alpha': 10.0, 'gamma': 0.1}	0.481728	0.451089	2.601053
0	SVR	{'C': 1.0, 'gamma': 0.1}	0.344585	0.615196	12.205376



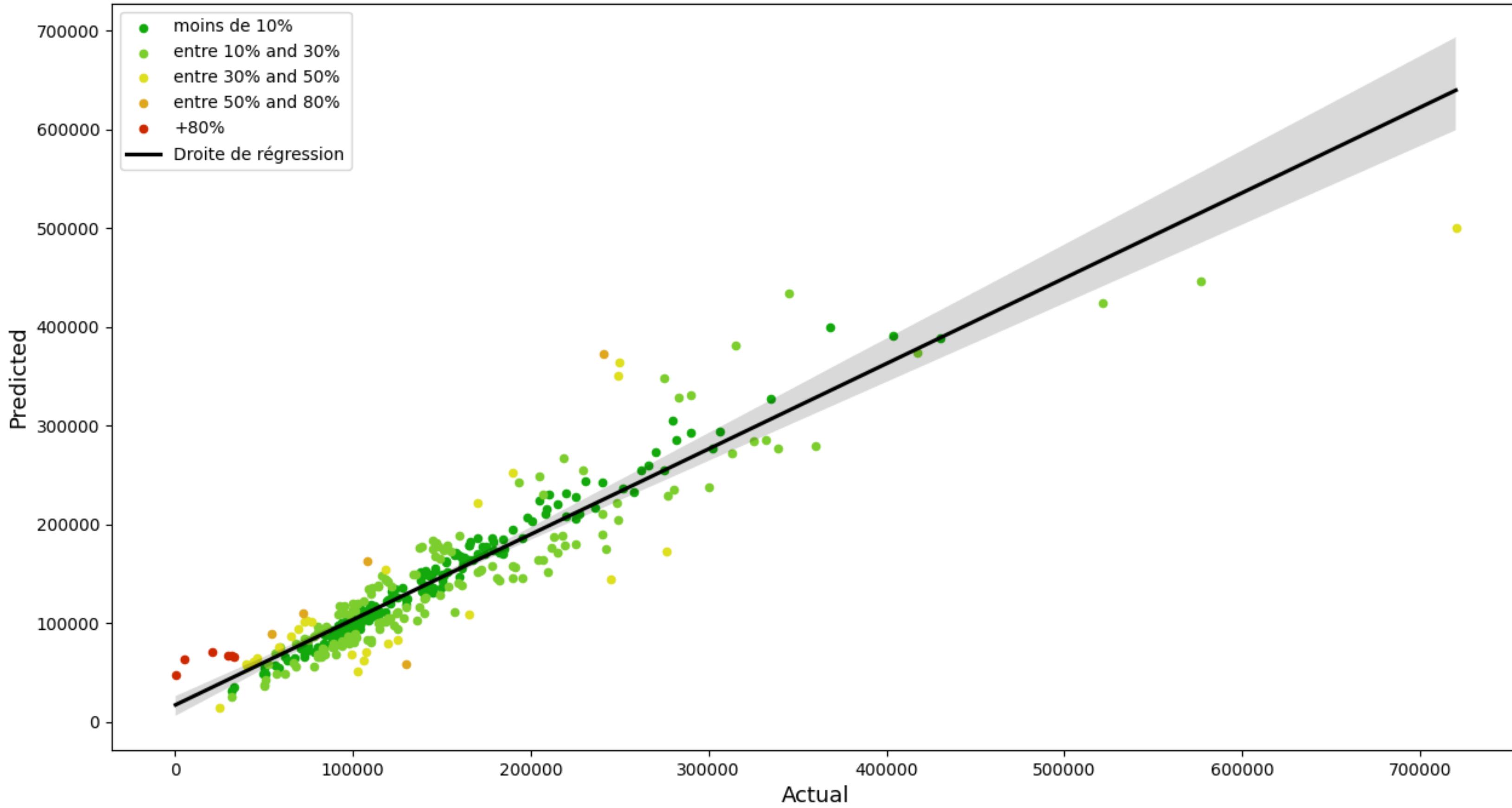
RÉSULTATS MODÈLES DE TYPE ENSEMBLISTE

	Model	Params	Train_RMSE	Val_RMSE	Fit_time
0	Random Forest	{'max_depth': 10, 'n_estimators': 500}	0.936412	0.696761	52.923970
1	XGBoost	{'max_depth': 5, 'n_estimators': 100}	0.993079	0.707670	12.340215

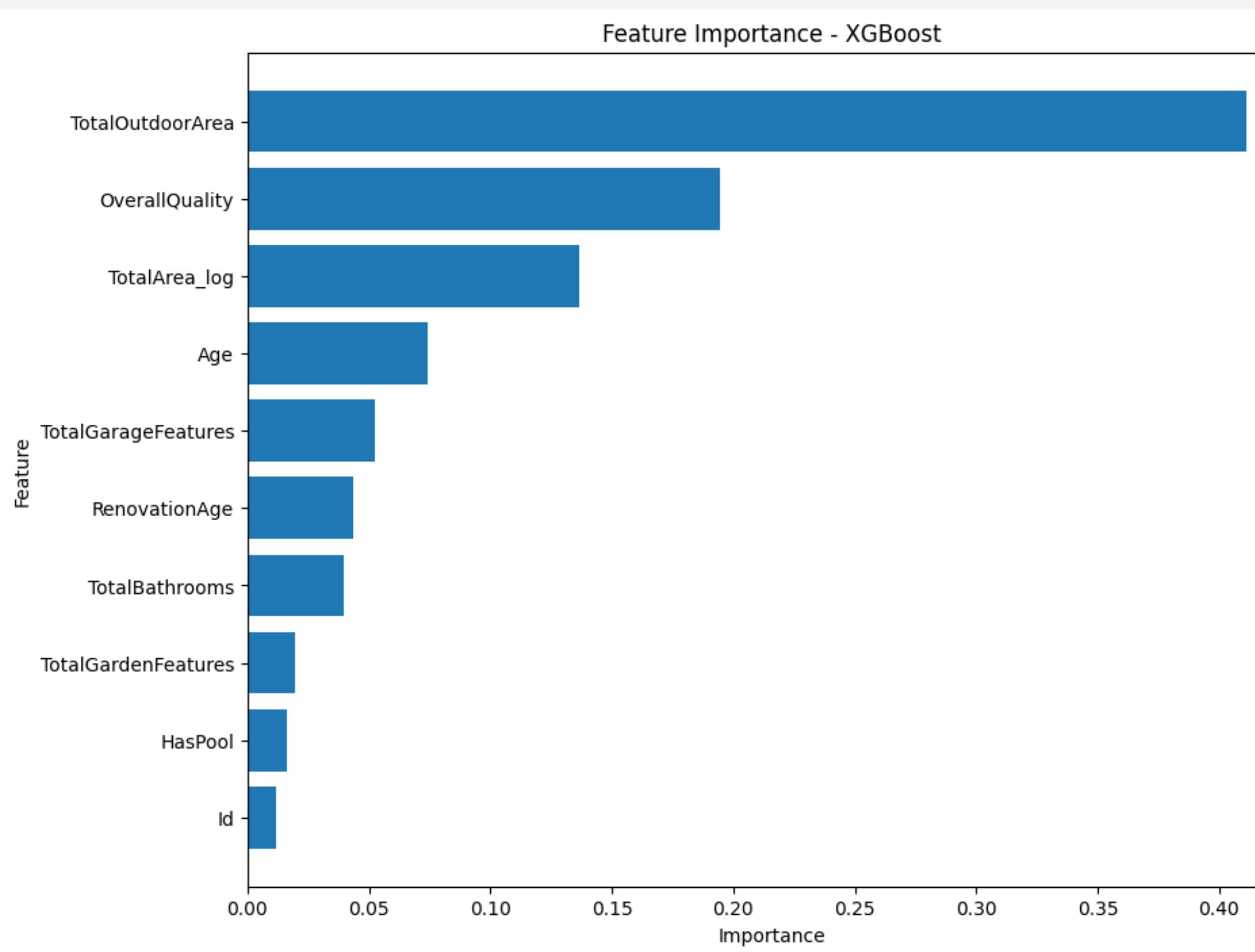
CHOIX DU MODÈLE FINAL

CHOIX DU MODÈLE FINAL

Résultats sur le test set coloré par le taux d'erreur sur la prédiction



CHOIX DU MODÈLE FINAL



```
[ ] import xgboost as xgb

# Separate the features and target variable in the training data
X_train = train.drop('SalePrice_log', axis=1)
y_train = train['SalePrice_log']

# XGBoost model with selected hyperparameters
xgb_model = xgb.XGBRegressor(max_depth=5, n_estimators=100)
xgb_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = xgb_model.predict(test)

# Create the output DataFrame with 'Id' and 'SalePrice' columns
output = pd.DataFrame({'Id': test['Id'], 'SalePrice': y_pred})

output.head()

      Id  SalePrice
0  1461  11.422921
1  1462  11.689876
2  1463  11.747937
3  1464  11.870954
4  1465  11.824861
```



**THANK
YOU**

*We look forward to working
with you*

