Machine Learning : Prédiction du prix d'un véhicule

Réalisé par :

NGARI LENDOYE Alix & NANGA Théophile

Présentation du problème

Imaginons que nous souhaitons vendre ou acheter une voiture. En tant qu'acheteur, nous souhaitons savoir si le prix affiché est juste, et en tant que vendeur, nous voulons nous assurer que le prix proposé soit compétitif afin d'augmenter nos chances de vendre la voiture plus facilement. L'idée de cet article est de faire ressortir l'ensemble des étapes de la conception au déploiement d'un algorithme de Machine Learning adapté à une estimation du prix d'un véhicule en fonction de ses caractéristiques.

Pour ce faire, nous avons accès aux données des prix des voitures vendues entre 2015 et 2019 au travers du dataset « ford » présent sur la plateforme Kaggle. Pour chaque vente de voiture (une observation), nous avons les informations suivantes :

- Date de Fabrication
- Le type de carburant
- La capacité du moteur
- Le kilométrage
- Le prix auquel elle a été vendu
- La consomation du véhicule en MPG (Mile per Gallon)

Entrainement du modèle

Importation des libraires

```
#####Importation des bibliothèques
import joblib
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import  StandardScaler
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.pipeline import make_pipeline
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
```

Chargement des données

Pour commencer, nous devons lire et charger les données contenues dans le fichier CSV

```
#Lire le fichier csv en utilisant pandas
url="https://raw.githubusercontent.com/NGALENAL1004/datasets/master/ford.csv" #chemin d'accès du fichier csv
dataset = pd.read_csv(url)
#Afficher les données
print(dataset)
#Suppression de la colonne model car elle n'esst pas très utile pour déterminer le prix
df = dataset.drop(['model'], axis=1)
print(df)
```

```
         model  year  price transmission  mileage fuelType  tax   mpg  \
0        Fiesta  2017  12000    Automatic    15944   Petrol  150  57.7
1         Focus  2018  14000       Manual     9083   Petrol  150  57.7
2         Focus  2017  13000       Manual    12456   Petrol  150  57.7
3        Fiesta  2019  17500       Manual    10460   Petrol  145  40.3
4        Fiesta  2019  16500    Automatic     1482   Petrol  145  48.7
...         ...   ...    ...          ...      ...      ...  ...   ...
17961    B-MAX  2017   8999       Manual    16700   Petrol  150  47.1
17962    B-MAX  2014   7499       Manual    40700   Petrol   30  57.7
17963    Focus  2015   9999       Manual     7010   Diesel   20  67.3
17964       KA  2018   8299       Manual     5007   Petrol  145  57.7
17965    Focus  2015   8299       Manual     5007   Petrol   22  57.7
```

Python propose une librairie appelée Pandas qui offre des classes et des fonctions pour lire différents formats de fichiers, y compris les fichiers Excel.

La fonction read_csv() de Pandas permet de lire un fichier CSV et renvoie un DataFrame, qui est une structure de données tabulaire à 2 dimensions. Nous devons ensuite séparer la variable cible (Y) des variables prédictives (xi).

Etude de la dataset

```
df.describe()
```

	year	price	mileage	tax	mpg	engineSize
count	17966.000000	17966.000000	17966.000000	17966.000000	17966.000000	17966.000000
mean	2016.866470	12279.534844	23362.608761	113.329456	57.906980	1.350807
std	2.050336	4741.343657	19472.054349	62.012456	10.125696	0.432367
min	1996.000000	495.000000	1.000000	0.000000	20.800000	0.000000
25%	2016.000000	8999.000000	9987.000000	30.000000	52.300000	1.000000
50%	2017.000000	11291.000000	18242.500000	145.000000	58.900000	1.200000
75%	2018.000000	15299.000000	31060.000000	145.000000	65.700000	1.500000
max	2060.000000	54995.000000	177644.000000	580.000000	201.800000	5.000000

Description de la dataset :

A l'aide de la fonction *describe()*, nous avons accès aux valeurs maximales et minimales de chaque variable numérique, ainsi qu'aux moyennes.

Le type de variables :

```
print('Le type de variables:')
print(df.dtypes)
```

```
Le type de variables:
year             int64
price            int64
transmission    object
mileage          int64
fuelType        object
tax              int64
mpg            float64
engineSize     float64
dtype: object
```

Le type de variable nous permet de mieux choisir le modèle adapté et de savoir sur quelles variables effectuer le pré-processing avant l'entraînement.

```python
# Sélectionner les variables non numériques
non_numeric_cols = df.select_dtypes(exclude=['int64', 'float64']).columns.tolist()
# Initialiser les listes pour chaque type de variable
binary_cols = []
ordinal_cols = []
nominal_cols = []
# Parcourir les variables non numériques et les classer selon leur type
for col in non_numeric_cols:
    unique_vals = df[col].unique()
    if len(unique_vals) == 2:
        binary_cols.append(col)
    elif df[col].dtype == 'object' or len(unique_vals) <= 5:
        ordinal_cols.append(col)
    else:
        nominal_cols.append(col)
# Afficher les résultats
print('Variable Binaire:', binary_cols)
print('Variable Ordinale:', ordinal_cols)
print('Variable Nominale:', nominal_cols)
```

```
Variable Binaire: []
Variable Ordinale: ['transmission', 'fuelType']
Variable Nominale: []
```

Nous sélectionnons les variables non numériques car nous appliquerons le prétraitement **Label Encoding** (encodage des labels) sur celles-ci.

Préprocessing

Le prétraitement en apprentissage automatique est une étape cruciale visant à nettoyer, transformer et normaliser les données brutes avant de les utiliser pour entraîner un modèle. Cela permet d'améliorer les performances, la précision et la robustesse du modèle, tout en réduisant les erreurs de modélisation et le temps nécessaire à l'entraînement. De plus, le prétraitement fournit des informations utiles sur la structure des données, ce qui contribue à améliorer leur qualité et à identifier des modèles intéressants pour la modélisation.

Recherche des valeurs manquantes :

```python
####Pré-Processing
#Recherche des valeurs manquantes
print(df.isna().sum())
```

```
year            0
price           0
transmission    0
mileage         0
fuelType        0
tax             0
mpg             0
engineSize      0
dtype: int64
```

Notre dataset ne présente aucune valeur manquante.

Il faut maintenant vérifier si nous avons des valeurs aberrantes.

Une valeur aberrante (ou outlier en anglais) est une observation qui diffère significativement des autres observations dans un ensemble de données, que ce soit par une valeur extrêmement élevée ou extrêmement basse. Une valeur aberrante peut résulter d'une erreur de mesure, d'un enregistrement incorrect ou tout simplement d'une variation naturelle des données.

```python
#Remplacer les valeurs vides par NaN   NaN = "Not a Number"
df.replace("", np.nan, inplace=True)
#Suppression des valeurs abérantes
len = ['price','mileage','tax','mpg']
for i in len:
   x = df[i].describe()
   Q1 = x[4]
   Q3 = x[6]
   IQR = Q3-Q1
   lower_bound = Q1-(1.5*IQR) # Valeur aberrante trop faible
   upper_bound = Q3+(1.5*IQR) # Valeur aberrante trop forte
   df = df[(df[i]>lower_bound)&(df[i]<upper_bound)]
#Encoder les variables catégorielles
df['transmission'] = df['transmission'].map({'Automatic': 1, 'Manual': 2,'Semi-Auto':3})
df['fuelType'] = df['fuelType'].map({'Petrol': 1, 'Diesel': 2, 'Electric': 3, 'Hybrid':4,'Other':5})
#Le dataset après le pré-processing
print('Le dataset après le pré-processing:')
print(df)
```

```
Le dataset après le pré-processing:
       year  price  transmission  mileage  fuelType  tax   mpg  engineSize
0      2017  12000             1    15944         1  150  57.7         1.0
1      2018  14000             2     9083         1  150  57.7         1.0
2      2017  13000             2    12456         1  150  57.7         1.0
3      2019  17500             2    10460         1  145  40.3         1.5
4      2019  16500             1     1482         1  145  48.7         1.0
...     ...    ...           ...      ...       ...  ...   ...         ...
17957  2015   7650             2    46123         1  125  53.3         1.0
17958  2019  13250             2    13359         1  145  48.7         1.0
17960  2016   7999             2    31348         1  125  54.3         1.2
17961  2017   8999             2    16700         1  150  47.1         1.4
17964  2018   8299             2     5007         1  145  57.7         1.2

[11961 rows x 8 columns]
```

Après la suppression des valeurs aberrantes suivant la *règle de Tukey,* nous constatons une réduction drastique du nombre d'enregistrement

Entrainement des données

Entrée et target

En Machine Learning, entrée = données d'entrée, target = réponse attendue. Supervisé = entraînement sur données avec targets connues, non supervisé = détection de structures et de modèles dans les données.

```python
####Apprentissage et test
#Séparation des données en bases d'apprentissage et de test
# Entrées : variables qui nous permettrons d'évaluer les prix
x = df.drop(['price'], axis=1)
print(x)
```

```
       year  transmission  mileage  fuelType  tax   mpg  engineSize
0      2017             1    15944         1  150  57.7         1.0
1      2018             2     9083         1  150  57.7         1.0
2      2017             2    12456         1  150  57.7         1.0
3      2019             2    10460         1  145  40.3         1.5
4      2019             1     1482         1  145  48.7         1.0
...     ...           ...      ...       ...  ...   ...         ...
17957  2015             2    46123         1  125  53.3         1.0
17958  2019             2    13359         1  145  48.7         1.0
17960  2016             2    31348         1  125  54.3         1.2
17961  2017             2    16700         1  150  47.1         1.4
17964  2018             2     5007         1  145  57.7         1.2

[11961 rows x 7 columns]
```

```python
# Target car nous voulons prédire les prix
y = df['price']
print(y)
```

```
0        12000
1        14000
2        13000
3        17500
4        16500
         ...
17957     7650
17958    13250
17960     7999
17961     8999
17964     8299
Name: price, Length: 11961, dtype: int64
```

Normalisation des données

Afin d'obtenir des résultats pertinents, nous avons décidé de normaliser les données pour les mettre à l'échelle appropriée. Nous utilisons la fonction **StandardScaler().**

```
# Normaliser les données numériques
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.33, random_state=42)
print("x train: ",x_train.shape)
print("x test: ",x_test.shape)
print("y train: ",y_train.shape)
print("y test: ",y_test.shape)
```

```
x train:  (8013, 7)
x test:  (3948, 7)
y train:  (8013,)
y test:  (3948,)
```

Choix du modèle

A présent, nous recherchons le modèle le plus adéquat pour notre problème. Nous sélectionnons 6 modèles de régression différents qui seront évalués à l'aide de 4 métriques.

```
# Initialiser les modèles de régression
models = {
    "Gradient Boosting Regressor": GradientBoostingRegressor(),
    "Random Forest Regressor": RandomForestRegressor(),
    "Linear Regression": LinearRegression(),
    "Ridge Regression": Ridge(),
    "Lasso Regression": Lasso(),
    "DecisionTreeRegressor":DecisionTreeRegressor()
}
models_names=["GBR","RFR","LiR","RR","LaR","DTR"]
```

```
# Initialisation des listes de résultats pour le training data
mse_training = []
mae_training = []
r2_training = []
mape_training = []
# Boucle pour le trainning data
for name, model in models.items():
    model.fit(x_train, y_train)
    y_pred = model.predict(x_train)
    # Calculer les métriques d'évaluation
    mse = mean_squared_error(y_train, y_pred)
    mae = mean_absolute_error(y_train, y_pred)
    r2 = r2_score(y_train, y_pred)
    mape = np.mean(np.abs((y_train - y_pred) / y_train)) * 100
    # Ajouter les résultats aux listes
    mse_training.append(mse)
    mae_training.append(mae)
    r2_training.append(r2)
    mape_training.append(mape)
    # Afficher les résultats
    print(name + " Metrics (Training Data):")
    print("Mean Squared Error: ", mse)
    print("Mean Absolute Error: ", mae)
    print("R2 Score: ", r2)
    print("Mean Absolute Percentage Error: ", mape)
```

Mesures obtenues

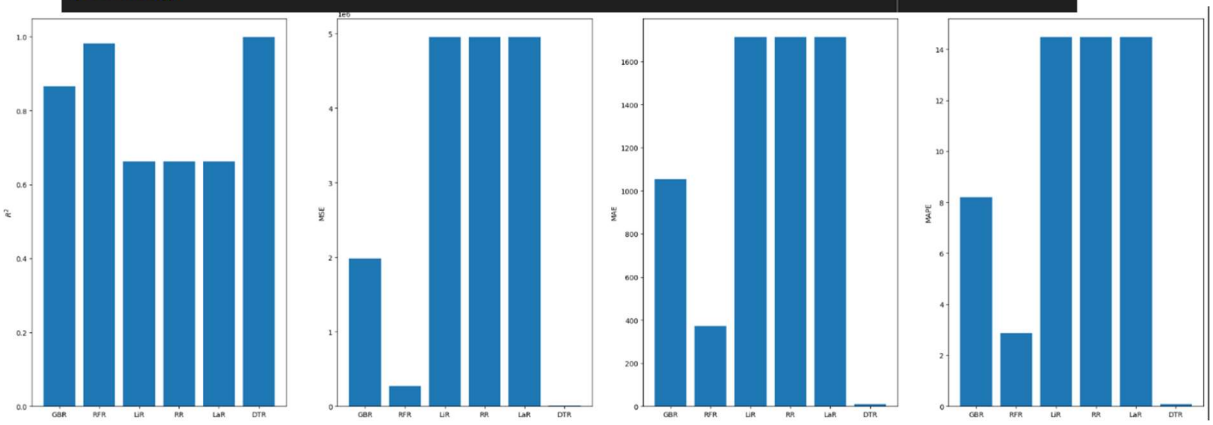
```
Gradient Boosting Regressor Metrics (Training Data):
Mean Squared Error:  1979085.2530763666
Mean Absolute Error:  1054.2550907083655
R2 Score:  0.8651560853770177
Mean Absolute Percentage Error:  8.206806092717875
Random Forest Regressor Metrics (Training Data):
Mean Squared Error:  274785.1056048165
Mean Absolute Error:  373.89858518003484
R2 Score:  0.9812776638791855
Mean Absolute Percentage Error:  2.881802923755231
Linear Regression Metrics (Training Data):
Mean Squared Error:  4954880.503269486
Mean Absolute Error:  1713.8949190611534
R2 Score:  0.662401868483748
Mean Absolute Percentage Error:  14.481654912051207
Ridge Regression Metrics (Training Data):
Mean Squared Error:  4954880.656375548
Mean Absolute Error:  1713.9135134614608
R2 Score:  0.6624018580519486
Mean Absolute Percentage Error:  14.481638150866685
Lasso Regression Metrics (Training Data):
Mean Squared Error:  4954886.035557664
Mean Absolute Error:  1714.0022543754794
R2 Score:  0.6624014915442537
Mean Absolute Percentage Error:  14.482261137074637
DecisionTreeRegressor Metrics (Training Data):
Mean Squared Error:  10325.006489454636
Mean Absolute Error:  10.793294230209243
R2 Score:  0.9992965112082052
Mean Absolute Percentage Error:  0.08492375046155612
```

Comparaison

Afin de faciliter l'interprétation et la comparaison, nous avons décidé de représenter les résultats obtenus par chaque modèle pour chacune des métriques.

Parfois, en apprentissage automatique, il est possible d'obtenir des modèles performants lors de l'entraînement, mais moins performants lors des tests. C'est pourquoi nous avons entraîné et testé tous les modèles.

```
# Tracer les histogrammes
f, (axe1,axe2,axe3,axe4)=plt.subplots(ncols=4, sharex= True, sharey=False, figsize=(30,10))
axe1.bar(models_names, r2_training)
axe1.set_ylabel('$R^2$')
axe2.bar(models_names, mse_training)
axe2.set_ylabel('MSE')
axe3.bar(models_names, mae_training)
axe3.set_ylabel('MAE')
axe4.bar(models_names, mape_training)
axe4.set_ylabel('MAPE')
plt.show()
```



Résultats du test

```
# Instancier le modèle DecisionTreeRegressor
model = DecisionTreeRegressor()
model.fit(x_train,y_train)
y_predicted = model.predict(x_test)
accuracy=r2_score(y_test, y_pred)
print('le scrore sur le jeu de test est:',accuracy)
```

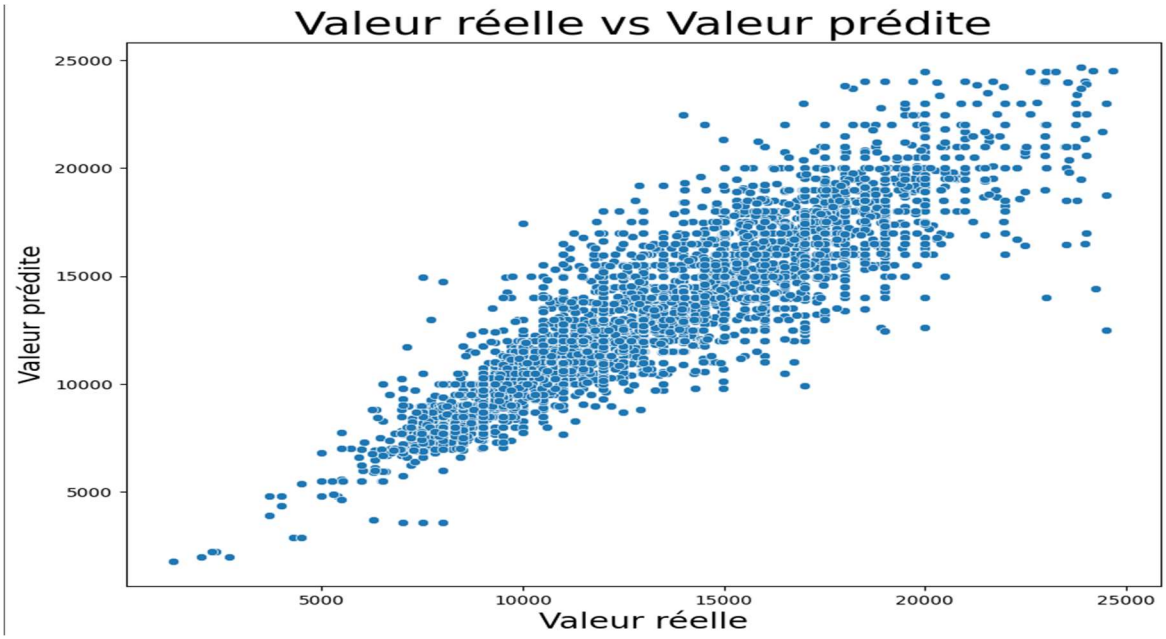
```
le scrore sur le jeu de test est: 0.8048032186804418
```

D'après les résultats obtenus précédemment, il en ressort que le modèle de régression par arbre de décision est le plus adéquat pour notre problème. Nous l'instancions donc à nouveau et il nous donne un score d'environ 0,80.

Analyse des résultats du modèle

```
#Résultat du modèle
pred=pd.DataFrame.from_dict({'valeur_predite':y_predicted,'valeur_reelle':y_test})
# Apres on doit aujouter le pourcentage d'erreur
pred['difference']=pred.valeur_predite-pred.valeur_reelle
print(pred.sample(n=15).round(2))
print(pred.difference.describe())
#Affichage du résultat du modèle
plt.figure(figsize=(10,8))
plt.title('Valeur réelle vs Valeur prédite',fontsize=25)
sns.scatterplot(x = y_test,y = y_predicted)
plt.xlabel('Valeur réelle', fontsize=18)
plt.ylabel('Valeur prédite', fontsize=16)
plt.show()
```

Nous avons testé le modèle final en comparant le prix prédit avec le prix réel indiqué dans l'ensemble de données.



Dans le graphique, nous constatons que les valeurs sont regroupées et semblent correspondre à la médiane, ce qui traduit une corrélation assez forte entre le prix prédit et le prix réel.

Optimisation du modèle

Après avoir choisi le modèle, nous passons à son optimisation en recherchant les meilleurs hyperparamètres. Pour cette recherche, nous utiliserons la fonction GridSearch.

```
###Optimisation du modèle
##Validation croisée
# Définition des hyperparamètres
parameters = {'max_depth':[None, 3, 5, 7],
              'min_samples_split':[2, 5, 10],
              'min_samples_leaf':[1, 2, 4],
              'max_features':[None, 'sqrt', 'log2']}
```

```
# Initialisation du modèle
tree = DecisionTreeRegressor()
# Recherche des meilleurs hyperparamètres par validation croisée
grid_search = GridSearchCV(tree, parameters, cv=5,scoring='r2', n_jobs=-1)
grid_search.fit(x_train, y_train)
# Affichage des meilleurs hyperparamètres et de la performance correspondante
print("Meilleurs hyperparamètres : ")
print(grid_search.best_params_)
print("Meilleure performance (R²) en apprentissage : ")
print(grid_search.best_score_)
# Entraînement du modèle avec les meilleurs hyperparamètres sur l'ensemble d'apprentissage
best_model = DecisionTreeRegressor(max_depth=grid_search.best_params_['max_depth'],
                                   min_samples_split=grid_search.best_params_['min_samples_split'],
                                   min_samples_leaf=grid_search.best_params_['min_samples_leaf'],
                                   max_features=grid_search.best_params_['max_features'])
best_model.fit(x_train, y_train)
```

```
Meilleurs hyperparamètres :
{'max_depth': None, 'max_features': None, 'min_samples_leaf': 4, 'min_samples_split': 10}
Meilleure performance (R²) en apprentissage :
0.8483811432759097
```

```
DecisionTreeRegressor
DecisionTreeRegressor(min_samples_leaf=4, min_samples_split=10)
```

Nous faisons donc varier les paramètres de l'arbre de décision pour obtenir la meilleure configuration.

Une fois la meilleure configuration obtenue, nous entrainons et testons le modèle à nouveau.

TestSPLIT et Feature Importance

Nous cherchons désormais la meilleure proportion d'entrainement et la contribution de chaque variable dans le résultat.

```
# Boucle sur les valeurs de proportions allant de 10% à 50%
for t in np.arange(0.1, 0.6, 0.1):
    # Séparer les données en données d'apprentissage et de test
    x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size=t, random_state=42)
    # Entraînement du modèle avec les meilleurs hyperparamètres sur l'ensemble d'apprentissage
    best_model = DecisionTreeRegressor(max_depth=grid_search.best_params_['max_depth'],
                                       min_samples_split=grid_search.best_params_['min_samples_split'],
                                       min_samples_leaf=grid_search.best_params_['min_samples_leaf'],
                                       max_features=grid_search.best_params_['max_features'])
    best_model.fit(x_train, y_train)
    # Évaluation du modèle sur l'ensemble de test
    y_pred = best_model.predict(x_test)
    r2 = r2_score(y_test, y_pred)
    print("Performance (R²) sur l'ensemble de test avec la proportion",t,":", r2)
```

```
Performance (R²) sur l'ensemble de test avec la proportion 0.1 : 0.8669629324553232
Performance (R²) sur l'ensemble de test avec la proportion 0.2 : 0.8542573210220223
Performance (R²) sur l'ensemble de test avec la proportion 0.30000000000000004 : 0.8540818645178612
Performance (R²) sur l'ensemble de test avec la proportion 0.4 : 0.8583142712672837
Performance (R²) sur l'ensemble de test avec la proportion 0.5 : 0.8532744837828686
```

Nous avons sélectionné les variables les plus importantes et supprimé les autres.

```
y_pred = best_model.predict(x_test)
r2 = r2_score(y_test, y_pred)
print("Performance (R²) sur l'ensemble de test :", r2)
# Score d'importance pour chaque variable
importance_scores = best_model.feature_importances_
# Afficher les scores d'importance pour chaque variable
for i, score in enumerate(importance_scores):
    print("Variable {}: Importance Score = {:.2f}".format(i, score))
# Garder les indices des variables avec un score d'importance supérieur à 0.05
indices_to_keep = np.where(importance_scores > 0.05)[0]
columns_to_keep = x_train.columns[indices_to_keep]
print(columns_to_keep)
# Sélection des colonnes pertinentes dans les données
x_new = x[columns_to_keep]
print(x_new)
```

```
Performance (R²) sur l'ensemble de test : 0.8581248575605553
Variable 0: Importance Score = 0.40
Variable 1: Importance Score = 0.01
Variable 2: Importance Score = 0.07
Variable 3: Importance Score = 0.00
Variable 4: Importance Score = 0.00
Variable 5: Importance Score = 0.09
Variable 6: Importance Score = 0.43
Index(['year', 'mileage', 'mpg', 'engineSize'], dtype='object')
       year  mileage   mpg  engineSize
0      2017    15944  57.7         1.0
1      2018     9083  57.7         1.0
2      2017    12456  57.7         1.0
3      2019    10460  40.3         1.5
4      2019     1482  48.7         1.0
...     ...      ...   ...         ...
17957  2015    46123  53.3         1.0
17958  2019    13359  48.7         1.0
17960  2016    31348  54.3         1.2
17961  2017    16700  47.1         1.4
17964  2018     5007  57.7         1.2

[11961 rows x 4 columns]
```

Nous avons éliminé les colonnes qui contribuent à moins de 0,05 (5 %) du résultat.

Maintenant, nous ne conservons que 4 variables d'entrée au lieu des 6 initiales.

La configuration finale

```
###Structure finale du modèle optimisé avec les variables pertinentes
# Séparer les données en données d'apprentissage et de test
x_train, x_test, y_train, y_test = train_test_split(x_new, y, test_size=0.4, random_state=42)
# Entraînement du modèle avec les meilleurs hyperparamètres sur l'ensemble d'apprentissage
best_model = DecisionTreeRegressor(max_depth=grid_search.best_params_['max_depth'],
                                   min_samples_split=grid_search.best_params_['min_samples_split'],
                                   min_samples_leaf=grid_search.best_params_['min_samples_leaf'],
                                   max_features=grid_search.best_params_['max_features'])
best_model.fit(x_train, y_train)
```

```
DecisionTreeRegressor
DecisionTreeRegressor(min_samples_leaf=4, min_samples_split=10)
```

Maintenant, nous testons le modèle avec les conditions optimales.

Nous sélectionnons 15 échantillons au hasard et nous effectuons à nouveau des tests en calculant la différence entre la prédiction et la valeur réelle, ce qui nous donnera une idée sur la précision de notre modèle.

```
# Évaluation du modèle sur l'ensemble de test
y_pred = best_model.predict(x_test)
r2 = r2_score(y_test, y_pred)
print("Performance (R²) sur l'ensemble de test :", r2)

#Résultat du modèle avec la validation croisée et la meilleur proportion
pred=pd.DataFrame.from_dict({'valeur_predite':y_pred,'valeur_reelle':y_test})
pred['difference']=pred.valeur_predite-pred.valeur_reelle
print(pred.sample(n=15).round(2))
print(pred.difference.describe())

#Affichage du résultat du modèle avec la validation croisée et la meilleur proportion
plt.figure(figsize=(10,8))
plt.title('Valeur réelle vs Valeur prédite',fontsize=25)
sns.scatterplot(x = y_test,y = y_pred)
plt.xlabel('Valeur réelle', fontsize=18)
plt.ylabel('Valeur prédite', fontsize=16)
plt.show()
```

```
Performance (R²) sur l'ensemble de test : 0.8544155230348706
       valeur_predite  valeur_reelle  difference
968          12657.50          12598       59.50
4210         10235.00           9790      445.00
4328         20699.75          21595     -895.25
10789        17485.20          14995     2490.20
3728         14000.67          15500    -1499.33
17069        18731.50          18000      731.50
1328         19985.50          19557      428.50
2847         17161.60          16090     1071.60
411          11692.38          11598       94.38
9886         15573.17          16400     -826.83
7152          9847.33           8995      852.33
9949         10595.50          10000      595.50
5982         13141.00          11490     1651.00
13904         8374.00           7495      879.00
17320         5111.00           5395     -284.00
```

Prédiction d'une nouvelle donnée

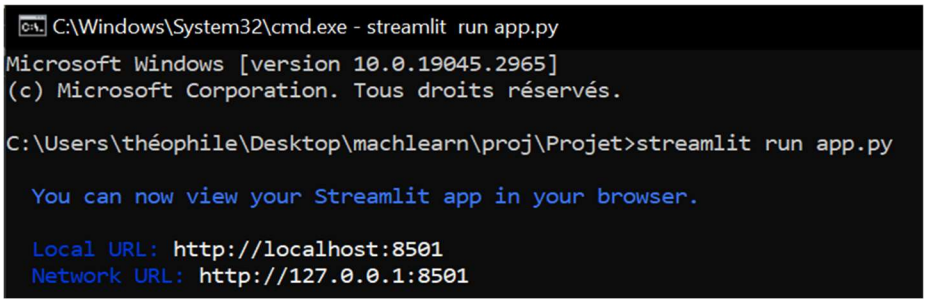
```
#Prédiction de nouvelles données
new_car = pd.DataFrame({'year': [2022],'mileage': [10000], 'mpg': [40.5], 'engineSize': [1.0]})
y_pred=best_model.predict(new_car)
print("Le modèle prédit un prix de:", int(y_pred))
#Sauvegardage du modèle
joblib.dump(value = best_model, filename = 'predictPriceCar.pkl')
# Utilisation du modèle sauvegarder pour réaliser des prédictions
model_loaded = joblib.load(filename = 'predictPriceCar.pkl')
price=model_loaded.predict(new_car)
print('Le prix est: ',int(price))
```

```
Le modèle prédit un prix de: 20893
Le prix est:  20893
```

Déploiement du modèle

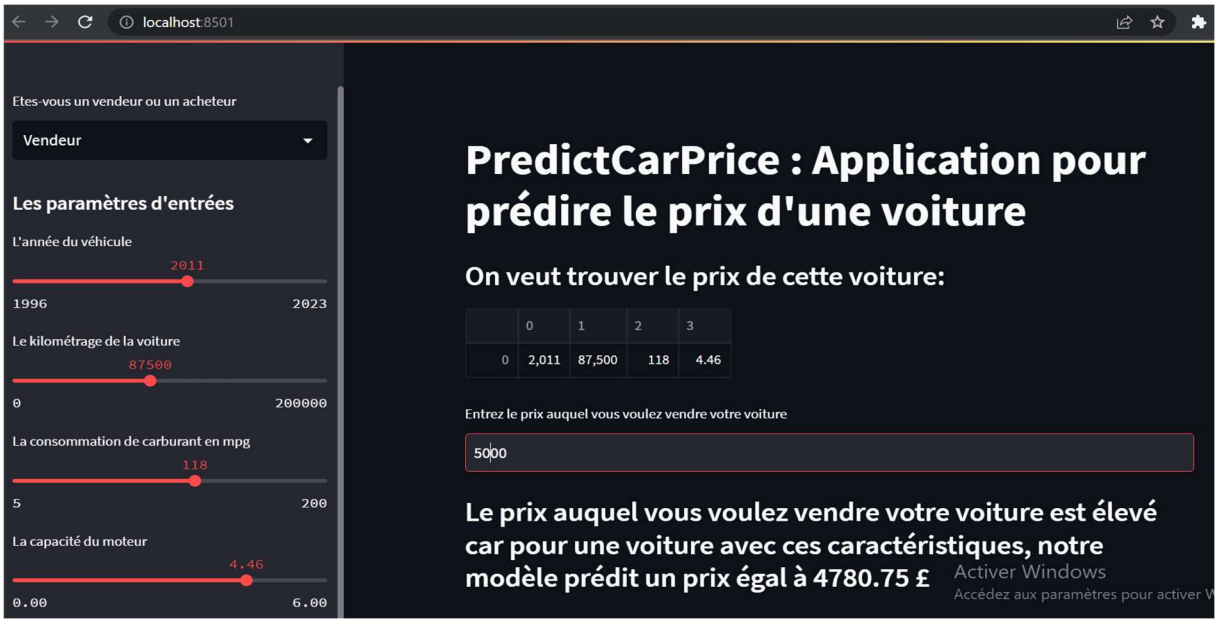
En ce qui concerne le déploiement de notre modèle, nous faisons appel au module Streamlit. Une fois les fonctionnalités et l'interface de notre application développés (code complet présent dans le répertoire GitHub), il suffit de l'exécuter. Pour ce faire, le module Streamlit doit impérativement être installé (pip install Streamlit).

Ensuite, il suffit d'utiliser la commande « streamlit run nom_du_fichier.py »



Nous obtenons une adresse locale et une adresse réseau permettant d'accéder à notre application

Le résultat final est alors le suivant :



Nous avons une application permettant aux utilisateurs d'avoir une idée sur le tarif auquel il devrait vendre ou acheter leur véhicule suivant les caractéristiques de celui-ci.

Vous retrouverez tous les fichiers utiliser ainsi que la dataset sur notre github en cliquant ici.