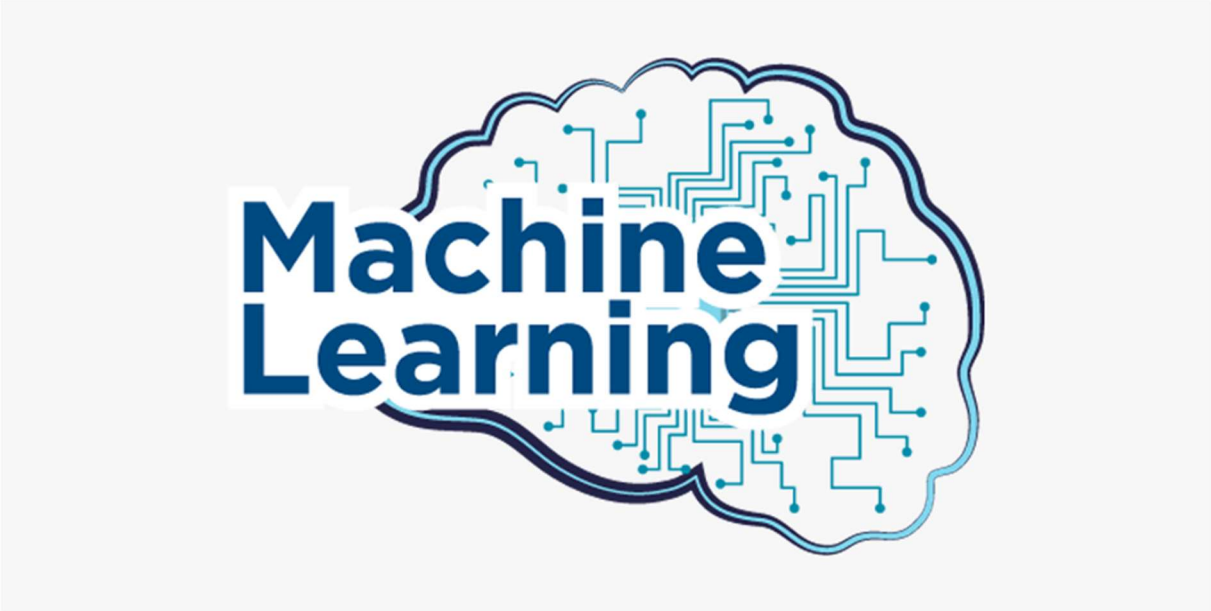
Rapport de projet

Machine Learning



Noms & Prénoms	Projet	Encadrant
-NGARI LENDOYE Alix -NANGA NDENGUE Théophile	Prédiction du prix d'un véhicule automobile	Monsieur TEKOUABOU Stéphane

Année académique : 2022-2023

Table des matières

Introduction 3
1. Analyse et traitement des données 4
1.1. Choix et structure des données 4
1.2. Visualisation des données 6
1.3. Pré-processing 9
2. Élaboration du modèle 11
2.1. Apprentissage et test 11
2.2. Optimisation du modèle 13
2.3. Sélection des variables importantes 16
2.4. Enregistrement du modèle 18
3. Déploiement du modèle 20
3.1. Présentation de Streamlit 20
3.2. Présentation de notre application 20
Conclusion 22

Introduction

Le Machine Learning, ou apprentissage automatique, est une discipline de l'intelligence artificielle qui permet aux ordinateurs d'apprendre à partir de données et de prendre des décisions ou de faire des prédictions sans être explicitement programmés. Cette approche révolutionnaire a trouvé de nombreuses applications dans divers domaines, notamment la prédiction de prix.

Dans le cadre de ce projet, notre objectif principal est de développer un modèle de Machine Learning capable de prédire le prix d'un véhicule en fonction de ses caractéristiques. Cette prédiction du prix est d'une importance capitale dans le secteur de l'automobile, tant pour les acheteurs que pour les vendeurs, car elle permet d'évaluer la juste valeur d'un véhicule sur le marché.

Cependant, la prédiction du prix des véhicules est une tâche complexe en raison de plusieurs facteurs à prendre en compte. De nombreuses caractéristiques telles que l'année du véhicule, le kilométrage, la consommation de carburant, la marque, le modèle, la puissance du moteur, etc., peuvent influencer considérablement le prix final. De plus, d'autres facteurs économiques et contextuels peuvent également jouer un rôle dans la fluctuation des prix des véhicules.

Ainsi, nous effectuerons une analyse approfondie des données traitées. Nous examinerons les différentes variables explicatives, explorerons les distributions, identifierons les valeurs manquantes ou aberrantes, et procéderons à des transformations ou des nettoyages si nécessaire. Ensuite, nous développerons notre modèle de prédiction basé sur une méthode de régression. Enfin, nous déploierons notre modèle afin qu'il puisse être utilisé dans un environnement réel. Nous développerons une application interactive à l'aide de la bibliothèque Streamlit, qui permettra aux utilisateurs de prédire le prix d'un véhicule en fonction de ses caractéristiques.

1. Analyse et traitement des données

1.1. Choix et structure des données

Le choix des données est une étape cruciale dans tout projet de Machine Learning, car il détermine en grande partie la qualité des prévisions et la pertinence des conclusions tirées. Nous devons nous assurer de détenir une compréhension claire de toutes les variables disponibles dans la dataset, ainsi que leur signification et leur utilité potentielle pour notre projet. Il est également important d'examiner la qualité des données, en cherchant des valeurs manquantes ou des données aberrantes qui pourraient affecter négativement les résultats. Une autre considération importante est la taille de la dataset. Une dataset plus grande peut souvent fournir des prévisions plus précises, mais elle peut également augmenter le temps de calcul et nécessiter des ressources informatiques plus importantes. Il est donc important de trouver un équilibre entre la taille de la dataset et les ressources disponibles pour notre projet.

Dans le cadre de notre projet sur la prévision du prix d'un véhicule automobile, nous nous sommes tournés vers une base de données contenant les informations de ventes des véhicules de la marque Ford aux Étas-Unis y compris les caractéristiques de chaque véhicule et les informations de ventes. Il s'agit là d'une dataset disponible sur la plateforme *Kaggle* mais également dans le répertoire GitHub dédié à notre projet. Les colonnes de la dataset comprennent notamment le modèle et la marque du véhicule, l'année de fabrication, le type de carburant, le type de transmission, le kilométrage, l'état du véhicule, le lieu de vente, la date de vente et le prix final de vente. Certaines des colonnes contiennent des données catégorielles, telles que le type de carburant ou le type de transmission, tandis que d'autres colonnes contiennent des données numériques, telles que le kilométrage et le prix final de vente. Toutefois, nous élaborons au travers de notre programme une analyse plus approfondie.

Une fois l'ensemble des bibliothèques indispensables importées dans notre script, nous importons également notre dataset via son lien associé au répertoire GitHub correspondant.

```
#Lire le fichier csv en utilisant pandas
url="https://raw.githubusercontent.com/NGALENAL1004/datasets/master/ford.csv"
dataset = pd.read_csv(url)
#Afficher les données
print(dataset)
#Suppression de la colonne model
df = dataset.drop(['model'], axis=1)
print(df)
```

Par la suite, nous établissons un script dédié à l'analyse en profondeur du jeu de données.

```python
####Analyse du dataset
#Fonction shape
print("Le nombre de lignes et de colonnes est de : ", df.shape)
#Fonction describe
print("Analyse statistique:")
print(df.describe(include = 'all'))
#Le type de variables descriptives
print('Le type de variables:')
print(df.dtypes)
# 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)
```

Au travers de ce script, nous découvrons le nombre de lignes et de colonnes constituant notre dataset pour notre étude. Nous avons également une analyse statistique détaillée ainsi que le type de chaque variable dans le dataset. Ainsi, notre jeu de données comprend 17966 instances pour 8 variables :

- L'année de fabrication de la voiture ;
- Le prix de la voiture en livres sterling ;
- Le type de transmission (automatique ou manuelle) ;
- Le nombre de kilomètres parcourus par la voiture ;
- Le type de carburant utilisé par la voiture (essence, diesel, hybride etc…) ;
- Le montant de la taxe sur le véhicule ;
- La consommation d'essence de la voiture en miles par gallon ;
- La teille du moteur de la voiture.

De plus, nous dénombrons 6 variables numériques contre 2 variables catégorielles.

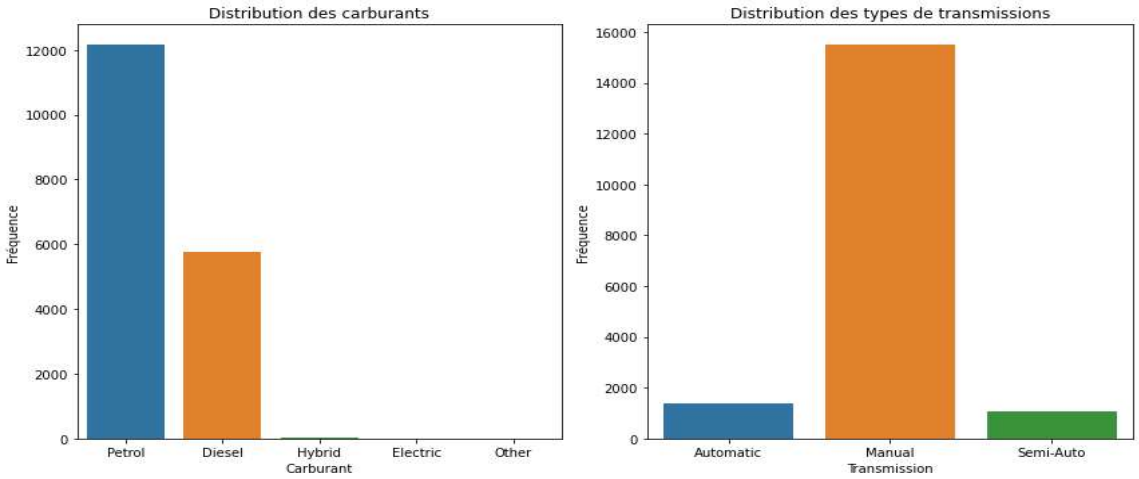
```
Le nombre de lignes et de colonnes est de :  (17966, 8)
Analyse statistique:
                year         price transmission        mileage fuelType           tax           mpg    engineSize
count   17966.000000  17966.000000        17966   17966.000000    17966  17966.000000  17966.000000  17966.000000
unique           NaN           NaN            3            NaN        5           NaN           NaN           NaN
year             int64
price            int64
transmission    object
mileage          int64
fuelType        object
tax              int64
mpg            float64
engineSize     float64
dtype: object
Variable Binaire: []
Variable Ordinale: ['transmission', 'fuelType']
Variable Nominale: []
```

1.2. Visualisation des données

Une fois la structure de notre dataset maitrisé, nous élaborons à présent des visualisations différentes pour nos variables. L'objectif ici est de représenter les données de manière graphique afin de faciliter leur compréhension et leur analyse. Nous mettons alors en évidence les relations entre variables qui seraient difficilement perceptibles par une simple lecture des chiffres.

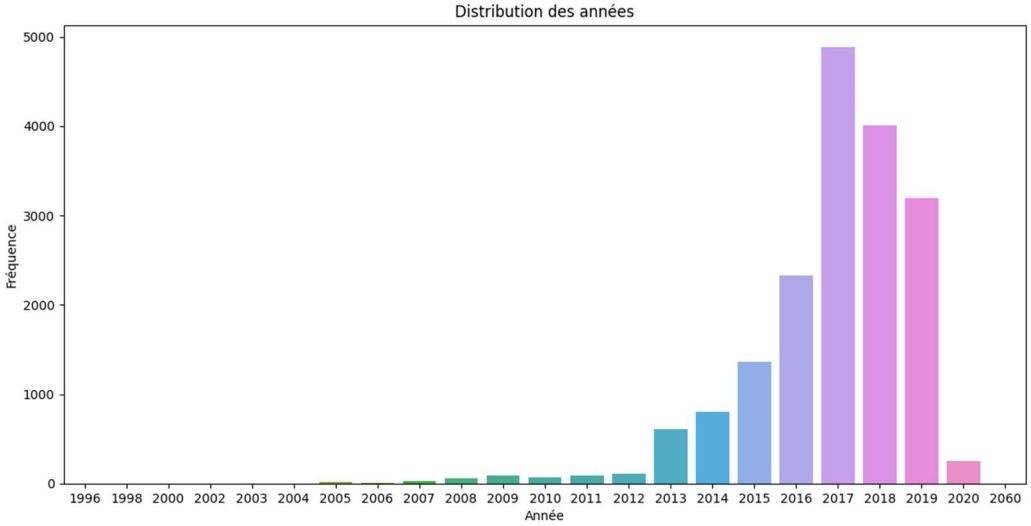
Voici donc les différentes visualisations opérées :

- Distribution des variables catégorielles



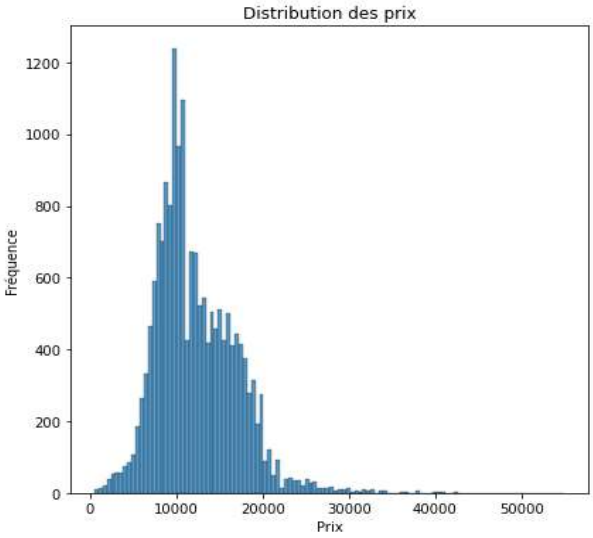
Nos deux variables catégorielles sont « FuelType » et « transmission » pour respectivement le type de carburant consommé par la voiture et le type de transmission du véhicule. En ce qui concerne le type de carburant, nous avons environ 12000 véhicules de notre jeu de données fonctionnant avec de l'essence. Quant au type de transmission, près de 16000 véhicules sont de transmission manuelle.

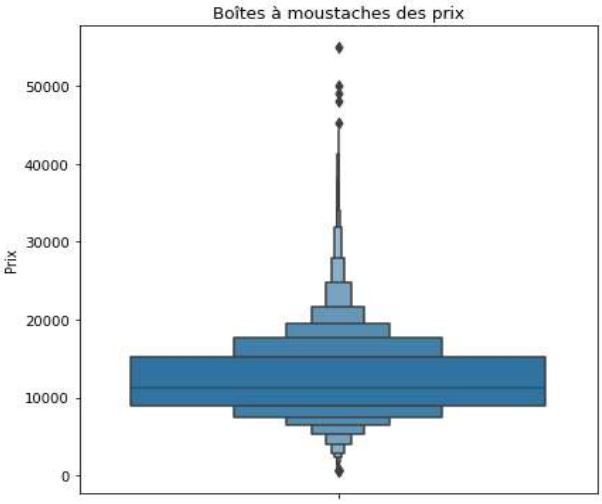
- Distribution des années



Nous observons à l'aide de ce graphique que la plupart des véhicules ont été produit entre 2016 et 2019, avec une majeure partie produite en 2017.

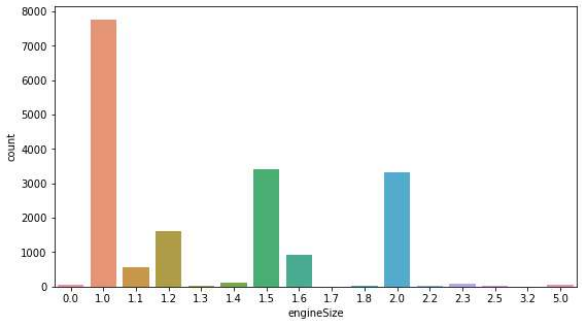
- Analyse des prix

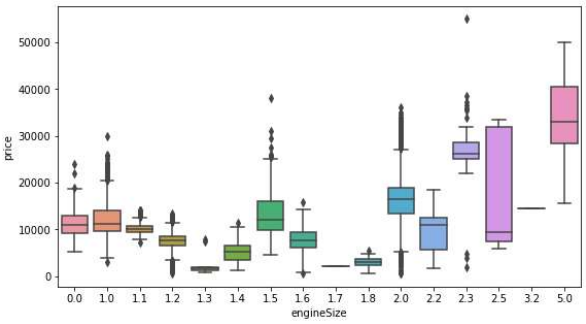




Ici, nous obtenons une idée sur la fourchette de prix prépondérante. Il se trouve que c'est à peu près entre 8000 dollars et 15000 dollars.

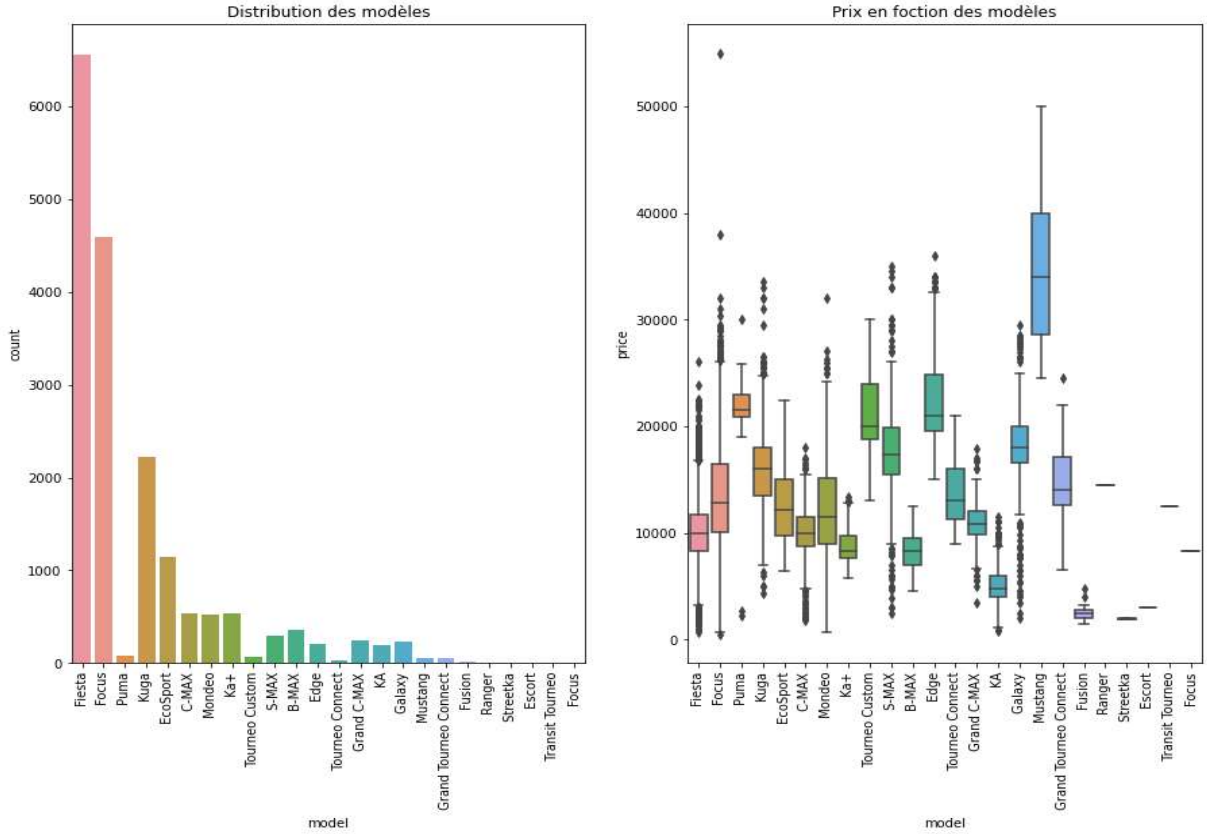
- Analyse de la capacité des moteurs





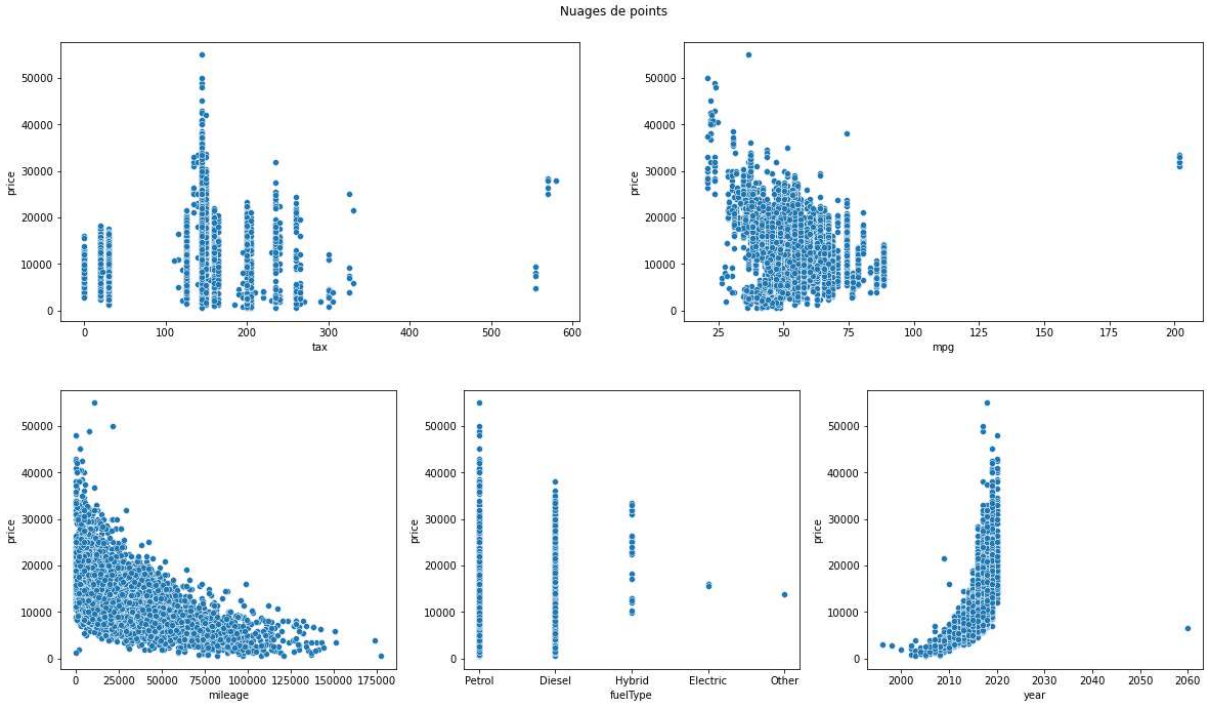
En ce qui concerne la capacité du moteur, nos visualisations choisies nous permettent d'en ressortir deux informations. Tout d'abord, les voitures dont le moteur à une capacité de 1.0 litres, de 1.5 litres et de 2.0 sont les plus présentes. Ensuite, nous remarquons que plus le moteur du véhicule détient une grande capacité, plus les véhicules détiennent un prix conséquent.

- Analyse des différents modèles de voiture



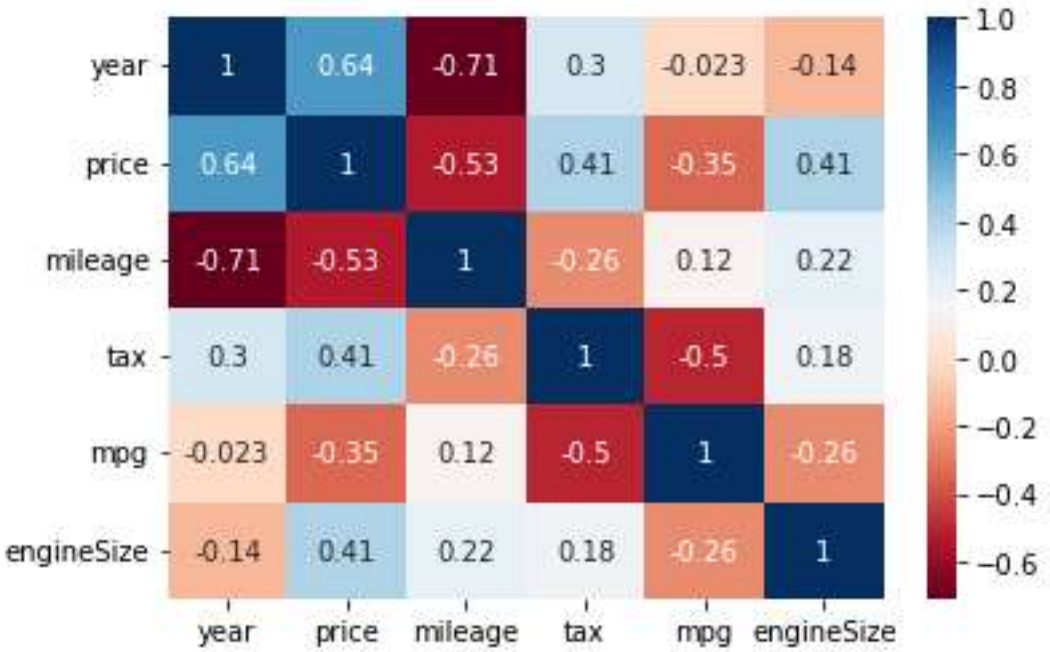
Nous remarquons que les voitures *Ford fiesta* et *Ford focus* sont les voitures les plus sur le marché mais dont les tarifs ne sont pas les plus élevés. Le modèle *mustang* est incontestablement le modèle de voiture le plus cher chez Ford.

- Les nuages de points



Ici, nous visualisons le prix de vente de chaque véhicule en fonction de la valeur de sa taxe, de sa consommation en mpg, de son kilométrage, de son type de carburant et de son année de production. Nous pouvons observer quelques individus isolés que ce soit au niveau des taxes, de la consommation en mpg, du kilométrage mais aussi pour l'année de production. Plus tard dans notre étude nous nous occuperons de ces individus.

- Matrice de corrélation/Carte de chaleur



Une *heatmap* est un graphique qui utilise des couleurs pour représenter des valeurs numériques dans une matrice. Chaque cellule de la matrice est représentée par une couleur qui varie en fonction de la valeur numérique de la cellule. La carte de chaleur est souvent utilisée pour visualiser des corrélations entre différentes variables, en particulier dans des datasets de grande taille. Il permet de visualiser rapidement les relations entre les différentes variables d'un dataset. Ainsi, nous pouvons voir que les variables les plus corrélées entre-elles sont l'année de production et le nombre de kilométrage du véhicule.

1.3. Pré-processing

Le pré-processing est une étape cruciale de notre travail. Il consiste à nettoyer, transformer et préparer les données brutes avant leur utilisation dans un modèle d'apprentissage automatique. En effet, il se trouve que les données collectées peuvent contenir des erreurs, des valeurs manquantes ou encore des doublons. Pour résoudre ces problèmes, nous faisons appel aux techniques de pré-processing.

```
####Pré-Processing
#Recherche des valeurs manquantes
print(df.isna().sum())
#Remplacer les valeurs vides par NaN
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)
   upper_bound = Q3+(1.5*IQR)
   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)
```

Tel qu'il est visible sur le code ci-dessus, nous avons traiter les valeurs manquantes, encoder les variables catégorielles mais également supprimer les valeurs aberrantes. Pour la suppression des valeur aberrantes, nous nous basons sur « *la règle des 1,5 fois l'écart interquartile*" encore appelé *"règle de Tukey"*. Cette règle est une méthode courante pour identifier et supprimer les valeurs aberrantes dans un ensemble de données. La règle de Tukey s'appuie sur l'écart interquartile (IQR) qui est une mesure de la dispersion des données. Elle définit les bornes inférieure et supérieure comme étant respectivement Q1 - 1,5IQR et Q3 + 1,5IQR. Les observations situées en dehors de ces bornes sont considérées comme des valeurs aberrantes et peuvent être supprimées. Ainsi, à l'issu de cette étape de pré-processing, nous obtenons une nouvelle structure de données composé de 11961 instances pour 8 variables. Cela permet de maximiser les performances du modèle et d'obtenir des résultats fiables.

```
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]
```

2. Élaboration du modèle

2.1. Apprentissage et test

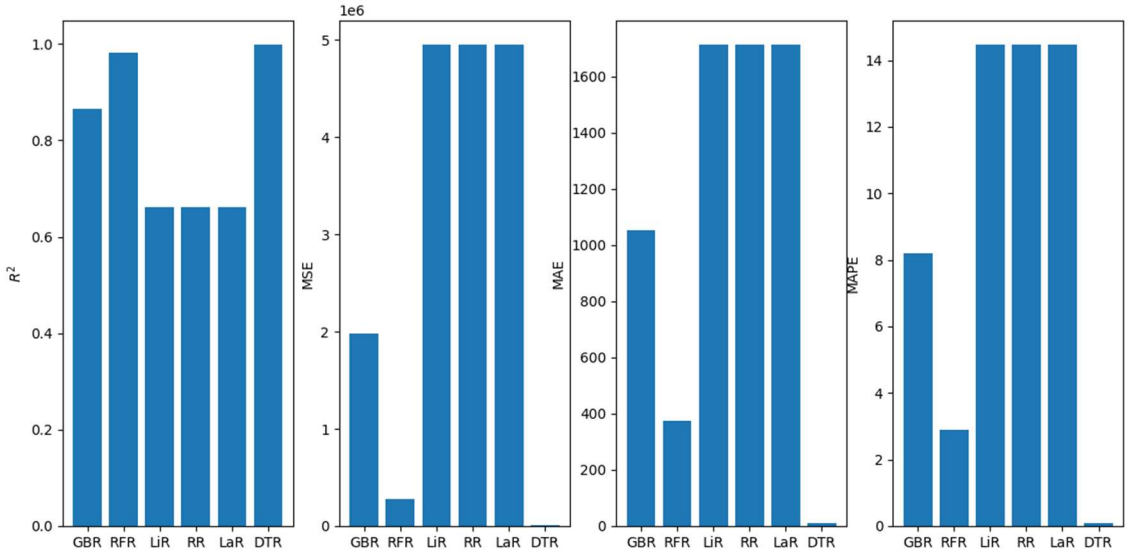
Notre projet de prédiction du prix d'un véhicule est un problème de régression. A cet effet, nous évaluerons ici les performances de 6 modèles de régression différents afin de sélectionner le plus optimal pour notre projet. Les 6 modèles sélectionnés sont alors initialisés dans un dictionnaire Python. Il s'agit du *Gradient Boost Regressor, Random Forest Regressor, Linear Regression, Ridge Regression, Lasso Regression et le Decision Tree Regressor.* Après l'initialisation des variables d'entrées et de la variable de sortie, nous séparons notre jeu de données en deux parties. Une première partie est alors destinée à l'apprentissage de nos modèles, puis une seconde partie de données est utilisé pour l'évaluation. Ainsi, 67% de notre jeu de données initiales seront des données d'apprentissage, soit 8013 instances, et 33% de notre dataset, soit 3948 instances, seront des données de test.

```
####Apprentissage et test
#Séparation des données en bases d'apprentissage et de test
x = df.drop(['price'], axis=1)
y = df['price']
print(x)
print(y)
# 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)
# 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)
# 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()
```

Ensuite, une boucle est effectuée pour chaque modèle de régression. En effet, le code entraîne chaque modèle sur les données d'apprentissage en utilisant la fonction *"fit"*, puis prédit les valeurs de la variable cible (prix des voitures) à partir des variables explicatives (x_train) avec la fonction *"predict"*. Pour chaque modèle, les métriques d'évaluation (*Mean Squared Error, Mean Absolute Error, R2 Score* et *Mean Absolute Percentage Error*) sont calculées à partir des prédictions du modèle sur les données

d'apprentissage, puis ajoutées à des listes de résultats correspondantes. Les résultats des métriques d'évaluation sont affichés pour chaque modèle sur les données d'apprentissage. Cela permet de comparer les performances des différents modèles et de sélectionner celui qui fournit les meilleurs résultats.

Enfin, nous effectuons une visualisation des résultats de chaque métrique pour chacun de nos modèles sélectionnés. A partir de là, il devient plus facile pour nous de déterminer le modèle le plus performant pour notre projet.



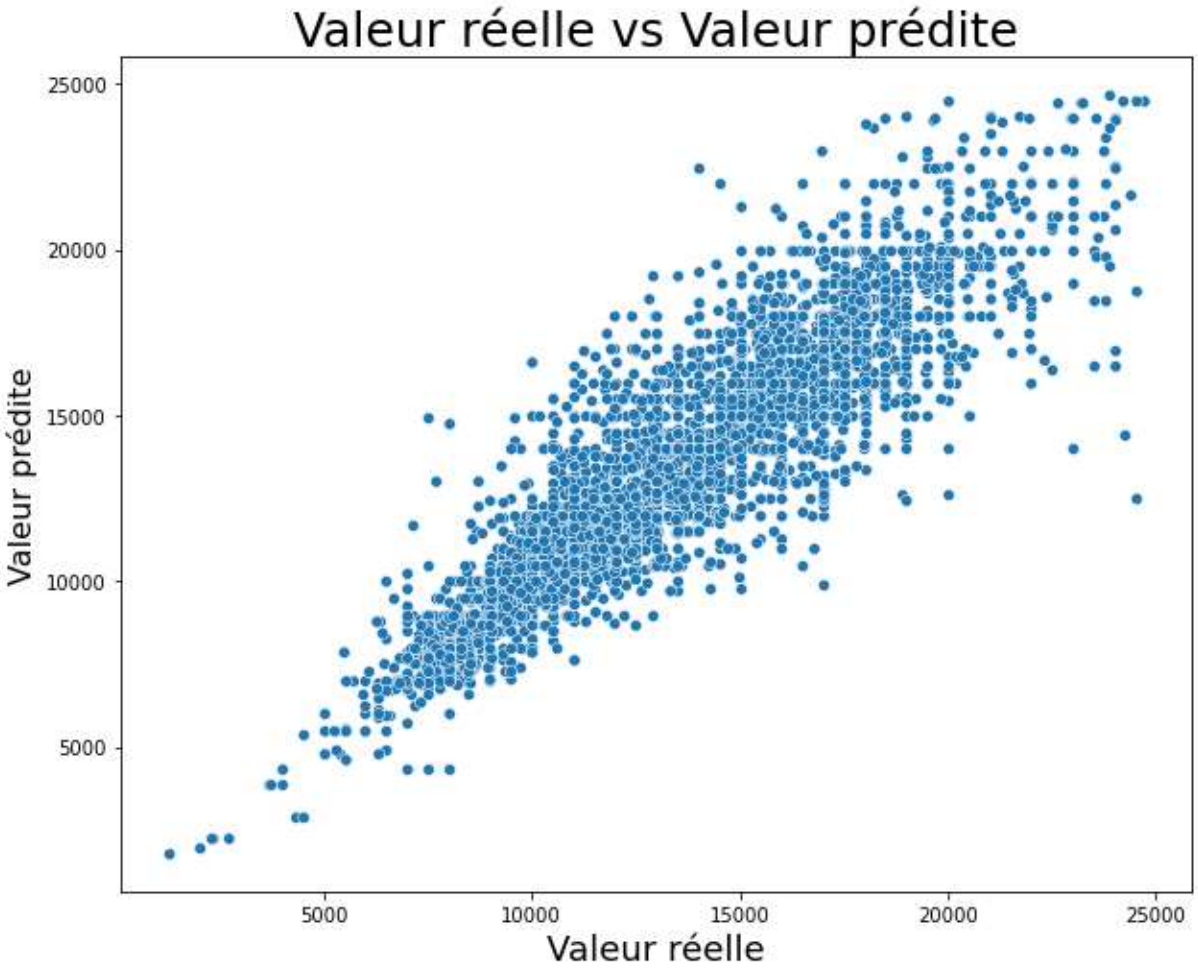
Sur la figure ci-dessus, nous décelons facilement que le *DecisionTreeRegressor* (DTR) est le modèle de régression le plus approprié dans notre cas de figure. Par la suite, nous travaillerons exclusivement avec celui-ci.

Nous effectuons à présent une évaluation de notre modèle sélectionnée sur nos données de test, à l'aide de la métrique R^2. Afin de visualiser les résultats de cette évaluation, nous adoptons une double stratégie. Tout d'abord nous utilisons une trame de données. Dans cette trame, 15 valeurs sont aléatoirement sélectionnées dans notre jeu de données. Pour chaque valeur sélectionnée, nous avons sa valeur réelle, sa valeur prédite, ainsi que la différence entre ses deux valeurs. Cela nous permet d'avoir une idée sur les écarts entre valeur réelle et valeur prédite. Nous obtenons ainsi un score d'environ 0.80 et une analyse statistique des différences pour chaque instance nous révèle une différence moyenne d'environ 8.6 entre les valeurs réelles et les valeurs prédites par le modèle.

```
le scrore sur le jeu de test est: 0.8029044362492936
       valeur_predite  valeur_reelle  difference
2491          15999.0          13200      2799.0
8808          12000.0          14000     -2000.0
733            9298.0           8998       300.0
5986          17499.0          16990       509.0
12393         15295.0          18500     -3205.0
7854          13691.0          14000      -309.0
8619          15800.0          17500     -1700.0
16549         12880.0          12395       485.0
1743          10498.0           9998       500.0
10034         17500.0          17500         0.0
16123          9610.0           9995      -385.0
2992          12500.0          16000     -3500.0
10653          9899.0          11495     -1596.0
482           19499.0          20994     -1495.0
8606           7500.0           8495      -995.0
```

count	3948.000000
mean	8.614362
std	1717.195728
min	-12000.000000
25%	-898.000000
50%	0.000000
75%	900.000000
max	8451.000000

Assi, nous visualisons les performances du modèle à l'aide d'un graphique. Ce dernier représente la valeur réelle de notre variable cible (prix des voitures) en abscisse et la valeur prédite par notre modèle en ordonné. L'arrangement des points nous informe sur la fiabilité du modèle. En effet, plus les points sont recentrés, plus notre modèle est performant.



2.2. Optimisation du modèle

Après l'obtention des premiers résultats déjà assez satisfaisant pour notre modèle, nous tentons d'améliorer davantage ses performances afin d'avoir des résultats encore plus précis et adéquats. Pour ce faire, nous adoptons diverses techniques à savoir la validation croisée ainsi que la variation de la proportion de test.

- Validation croisée

Lorsque l'on entraîne un modèle sur un ensemble de données, il est important de s'assurer que les performances de ce modèle ne soient pas influencées par la répartition spécifique des données d'apprentissage et de test. La validation croisée est utilisée pour résoudre ce problème

en fournissant une estimation plus fiable des performances du modèle. Le processus de validation croisée consiste à diviser l'ensemble de données en plusieurs sous-ensembles appelés "folds". Le modèle est ensuite entraîné et évalué plusieurs fois en utilisant différents plis (folds) comme ensemble d'apprentissage et comme ensemble de test. Cela permet d'obtenir plusieurs mesures de performance du modèle.

Dans notre situation, nous définissons en premier lieu une liste d'hyperparamètres que nous souhaitons explorer pour trouver les meilleures combinaisons. Ces hyperparamètres contrôlent le comportement de l'arbre de décision, tels que sa profondeur maximale *(max_depth)*, le nombre minimum d'échantillons requis pour diviser un nœud interne *(min_samples_split)*, le nombre minimum d'échantillons requis pour être une feuille *(min_samples_leaf)*, et le nombre de caractéristiques à considérer lors de la recherche de la meilleure division *(max_features)*.

```
###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)
# 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
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())
```

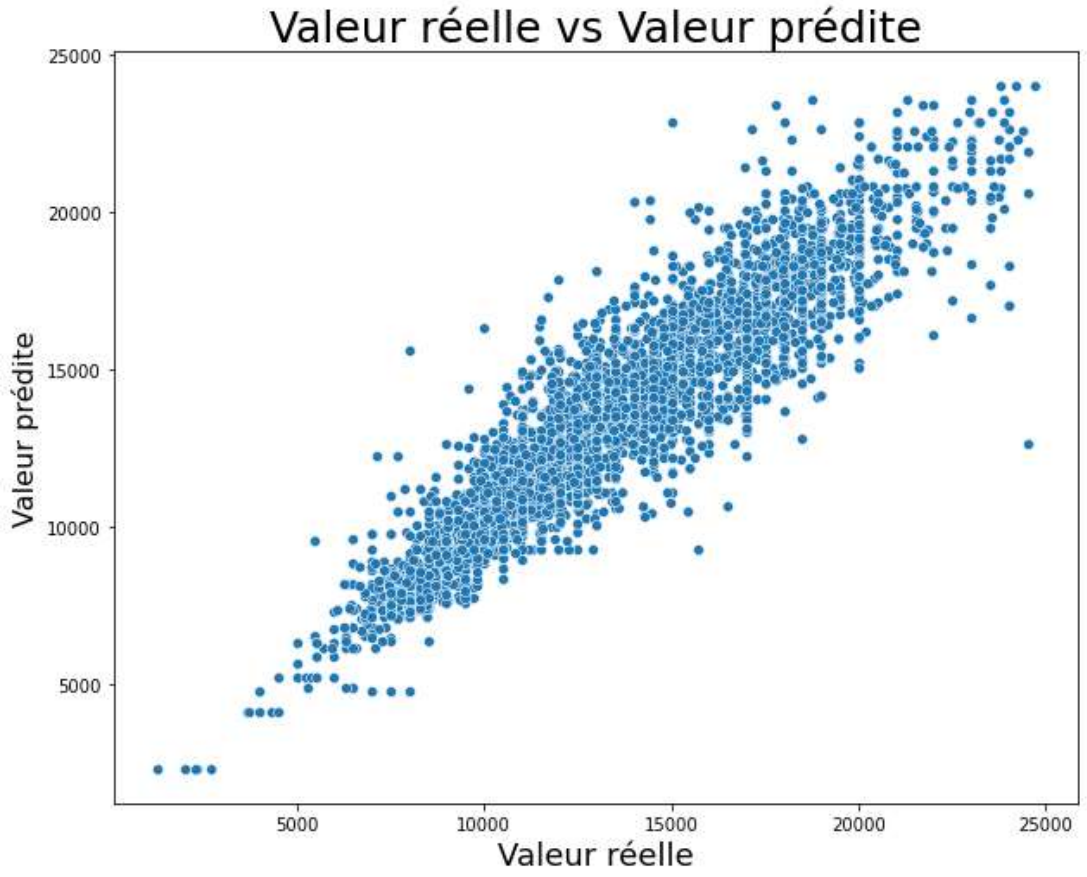
Ensuite, nous initialisons un modèle de régression basé sur l'arbre de décision. C'est le modèle que nous allons optimiser. Pour trouver les meilleurs hyperparamètres, une validation croisée est effectuée avec la fonction *GridSearchCV()* qui prend en entrée le modèle initialisé, les hyperparamètres définis et un nombre de "folds" (cv) égal à 5. La performance du modèle est alors mesurée avec le score R^2. Les meilleurs hyperparamètres ainsi que la meilleure performance obtenue en apprentissage sont affichés.

Après cela, nous entrainons à nouveau notre modèle avec les hyperparamètres optimaux sur l'ensemble d'apprentissage. La performance du modèle est évaluée par la suite sur l'ensemble de test en calculant le score R^2.

Enfin, nous visualisons les nouvelles performances de notre modèle tel que nous l'avons fait précédemment. Les prédictions sont stockées dans un DataFrame où 15 échantillons seront affichés aléatoirement. La différence entre les valeurs prédites et les valeurs réelles est également calculée et affichée sous forme de statistiques descriptives avec la méthode describe(). Les résultats obtenus sont alors les suivants :

```
Performance (R²) sur l'ensemble de test :  0.8593525932256711
       valeur_predite  valeur_reelle  difference
1970          18388.50          17865      523.50
11757         12248.50           7695     4553.50
7424          13573.25          10995     2578.25
16210         13090.25          12895      195.25
15340         18513.50          20800    -2286.50
5392          10502.89           8500     2002.89
2020          10076.50           9295      781.50
7846          14441.25          10591     3850.25
3425          11656.80           9980     1676.80
14405          9298.14           9999     -700.86
3123          10769.25          10995     -225.75
17953          8048.00           7199      849.00
5407          10741.75          14990    -4248.25
664           16826.20          17498     -671.80
9834          16635.57          16500      135.57
count      3948.000000
mean         -6.095041
std        1451.334088
min      -11839.333333
25%       -785.906250
50%        -14.277778
75%        784.616071
max       7886.875000
```

Nous avons en premier lieu une amélioration du score de performance qui est à présent de 0.85 contre 0.80 obtenu avant la validation croisée et l'ajout des hyperparamètres. Aussi, la moyenne de différence entre les valeurs réelles et les valeurs prédites est passée en absolue de 8.614 à 6.095, ce qui traduit des valeurs prédites s'éloignant moins des valeurs réelles. Cela est également perceptible au travers du graphique ci-dessous où nous pouvons observer une meilleure corrélation entre les valeurs réelles et prédites, comparé au graphique obtenu précédemment.



- Variation de la proportion de test et d'apprentissage

Toujours dans l'optique d'obtenir un modèle plus performant, nous varions la proportion de notre jeu de données destinée à l'apprentissage et celle réservée au test, afin de déterminer celle qui permet d'obtenir le score le plus optimal.

```
##Variation de la proportion de test
# 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)
```

Ainsi, ce code effectue une variation de la proportion de données de test utilisée pour évaluer la performance du modèle. Nous initialisons une boucle qui parcourt différentes valeurs de proportions, allant de 10% à 50%, avec un pas de 0,1. À chaque itération de la boucle, les données sont séparées en données d'apprentissage et de test en utilisant la fonction *train_test_split()*. La proportion spécifiée dans la boucle est utilisée comme taille de l'ensemble de test, tandis que le reste des données est utilisé comme ensemble d'apprentissage.

Ensuite, le modèle est entraîné avec les meilleurs hyperparamètres obtenus précédemment (à partir de la recherche par validation croisée) sur l'ensemble d'apprentissage. Le modèle entraîné est utilisé pour prédire les valeurs cibles *(y_pred)* à partir des données de test *(x_test)*.

Enfin, le score R² est calculé pour évaluer la performance du modèle sur l'ensemble de test. À chaque itération de la boucle, le code affiche le score R² correspondant à la proportion de données de test utilisée. Cela permet de voir comment la performance du modèle varie en fonction de la taille de l'ensemble de test.

```
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 :
0.8581248575605553
Performance (R²) sur l'ensemble de test avec la proportion 0.5 :
0.8530979542183029
Performance (R²) sur l'ensemble de test : 0.8581248575605553
```

D'après les résultats obtenus, la performance la plus optimale est obtenu avec une proportion de 10% pour les données de test. Toutefois, cette proportion étant jugée assez faible, nous opterons pour une proportion de 40%.

2.3. Sélection des variables importantes

L'analyse de l'importance des variables permet de mieux comprendre quelles caractéristiques ou variables ont le plus d'influence sur les prédictions du modèle. Cela peut fournir des informations précieuses pour interpréter les résultats et expliquer les relations entre les variables

d'entrée et la variable cible. En identifiant les variables les plus importantes, l'analyse peut aider à effectuer une sélection de variables plus efficace. En éliminant les variables moins importantes, on peut simplifier le modèle, améliorer sa performance et réduire les coûts de calcul. Ainsi, le code qui suit effectue une analyse de l'importance de chaque variable dans notre modèle de régression par arbre de décision.

```
###Analyse de l'importance de chaque variable
# Séparer les données en données d'apprentissage et de test
x_train, x_test, y_train, y_test = train_test_split(x, 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)
# 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)
# 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)
```

Tout d'abord, les données sont séparées en ensembles d'apprentissage et de test à l'aide de la fonction *train_test_split()*. L'ensemble de test est défini avec une taille de 40% des données totales.

Ensuite, le modèle est une fois de plus entraîné avec les meilleurs hyperparamètres obtenus précédemment sur l'ensemble d'apprentissage. Le modèle entraîné est utilisé pour prédire les valeurs cibles à partir des données de test, et le score R^2 est calculé pour évaluer la performance du modèle sur l'ensemble de test.

Par la suite, les scores d'importance sont calculés pour chaque variable à l'aide de la propriété *feature_importances_* du modèle. Les scores d'importance, qui représentent la contribution de chaque variable dans la prédiction du modèle, sont affichés dans une boucle, indiquant le numéro de la variable et le score d'importance correspondant. Les indices des variables ayant un score d'importance supérieur à 0.05 sont conservés. Ces variables sont considérées comme pertinentes pour la prédiction du modèle. Les colonnes pertinentes sont extraites des données d'origine *(x)* en utilisant les indices conservés, ce qui donne un nouvel ensemble de données (*x_new*) contenant uniquement les colonnes des variables pertinentes. Cela permet de sélectionner les variables les plus importantes pour le modèle et de réduire la dimensionnalité des données en ne conservant que les variables les plus influentes.

```
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 = 0.10
Variable 6: Importance Score = 0.42
Index(['year', 'mileage', 'mpg', 'engineSize'], dtype='object')
```

Ainsi, à l'issu de cette étape, les seules variables restantes sont *year* (année de production du véhicule), *mpg* (consommation du véhicule), *engineSize* (Capacité du moteur) et mileage (kilométrage de la voiture).

2.4. Enregistrement du modèle

Une fois que nous avons déterminé le modèle adéquat face à notre problème, les meilleurs hyperparamètres donnant des résultats optimaux, la proportion de test idéale ainsi que les variables essentielles afin de limiter les temps de calcul, il est à présent temps d'enregistrer notre modèle sous sa forme optimale. Pour ce faire, nous reprenons les étapes précédentes.

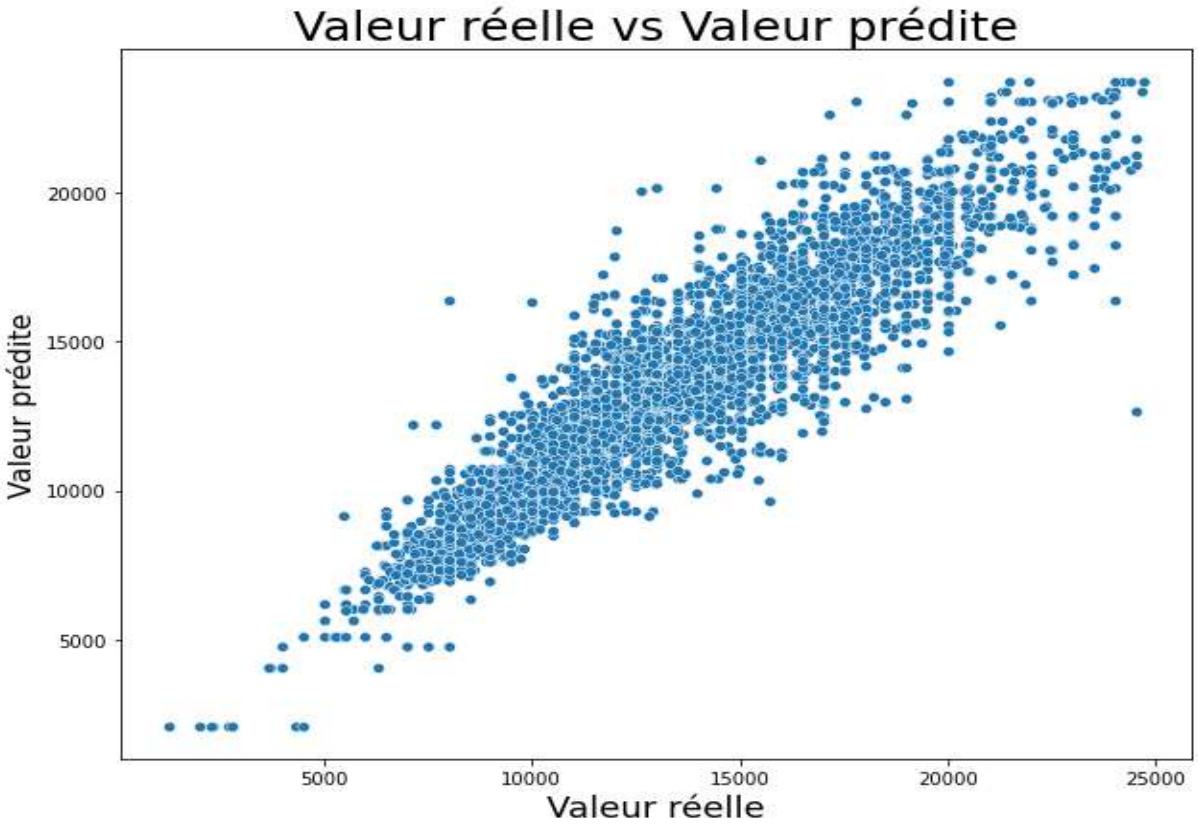
```
###Structure finale du modèle optimisé avec les variables pertinentes
# Séparer les données en données d'apprentissage et de test
x_train, x_test, y_train, y_test = train_test_split(x_new, y, test_size=0.4, random_state=42)
# Entraînement du modèle avec les meilleurs hyperparamètres sur l'ensemble 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)
# 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()
#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))
```

Dans cette partie du code, les données sont séparées en ensembles d'apprentissage et de test toujours à l'aide de la fonction *train_test_split*. L'ensemble d'apprentissage est utilisé pour entraîner le modèle et l'ensemble de test est utilisé pour évaluer sa performance. Cette

séparation des données est essentielle pour évaluer la capacité du modèle à généraliser et à prédire de nouvelles données.

Une fois les données séparées, le modèle de régression par arbre de décision est initialisé avec les meilleurs hyperparamètres obtenus précédemment par validation croisée. Le modèle est entraîné sur l'ensemble d'apprentissage en utilisant les variables pertinentes *x_new* comme données d'entrée et la variable cible *y* comme données de sortie. Une fois le modèle entrainé, des prédictions sont effectuées sur l'ensemble de test et la performance du modèle est évaluée en calculant le coefficient de détermination (R^2).

Ensuite, comme nous l'avions fait précédemment, les prédictions du modèle sont comparées aux valeurs réelles dans un tableau, où la différence entre les valeurs prédites et réelles est calculée. Les statistiques des différences sont affichées, ce qui permet de comprendre l'erreur de prédiction globale du modèle optimisé. Un nouveau graphique analysant les valeurs réelles et les valeurs prédites est alors généré.



Enfin, le modèle entraîné peut être utilisé pour effectuer des prédictions sur de nouvelles données. Dans l'exemple donné, une nouvelle voiture avec des caractéristiques spécifiques est créée et le modèle prédit son prix. De plus, le modèle est sauvegardé sous le nom *« predictPriceCar.pkl »,* dans le répertoire de notre projet pour une utilisation ultérieure, et un modèle préalablement sauvegardé peut également être chargé pour effectuer des prédictions sans avoir à entrainer de nouveau le modèle à chaque fois.

3. Déploiement du modèle

3.1. Présentation de Streamlit

Streamlit est un framework open-source qui facilite la création d'applications web interactives en utilisant du code Python. Il nous permet de transformer rapidement notre code en une application web élégante et intuitive, sans nécessiter des connaissances approfondies en développement web.

Streamlit est conçu pour être simple et intuitif, en fournissant une approche de programmation déclarative. Nous pouvons créer des applications avec une ou plusieurs pages, appelées "lay-out", qui permettent d'organiser le contenu et la navigation de notre application.

Pour exécuter l'application, les utilisateurs devront avoir Python et les bibliothèques requises installés sur leur machine. Il suffit d'utiliser l'outil de gestion de paquets de Python, tel que pip, pour installer Streamlit. L'installation se fait à l'aide de la commande suivante :

```
pip install streamlit
```

L'importation du module Streamlit dans notre application se fait à l'aide de cette commande :

```
import streamlit as st
```

En ce qui concerne l'exécution d'une application Streamlit, elle s'effectue à l'aide de cette commande :

```
streamlit run nom.py
```

La commande d'exécution doit être effectuée dans le répertoire où se situe l'application Streamlit.

3.2. Présentation de notre application

La finalité de notre travail est l'élaboration d'une application donnant la possibilité aux utilisateurs d'avoir une idée sur le prix d'un véhicule qu'ils souhaiteraient vendre ou acheter. C'est ainsi qu'à l'aide du module Streamlit, nous avons créé l'application *« PredictCarPrice »*.

Cette application Streamlit vise à aider les utilisateurs à prédire le prix d'une voiture en fonction de ses caractéristiques. Elle se compose de deux pages principales : "Vendeur" et "Acheteur".

Sur la page "Vendeur", l'utilisateur peut entrer les informations spécifiques à sa voiture, telles que l'année du véhicule, le kilométrage, la consommation de carburant en miles par gallon (mpg) et la capacité du moteur. Ensuite, il peut saisir le prix auquel il souhaite vendre sa voiture. L'application utilise ces informations pour prédire le prix de la voiture à l'aide du modèle pré-entrainé. En fonction de la prédiction, l'application indique si le prix demandé est faible, adéquat ou élevé par rapport à la valeur prédite.

Sur la page "Acheteur", l'utilisateur peut également entrer les informations spécifiques à la voiture qui l'intéresse. Ensuite, il peut saisir le prix affiché sur l'annonce de la voiture. L'application utilise ces informations pour prédire le prix de la voiture à l'aide du modèle pré-entrainé. En fonction de la prédiction, l'application indique si le prix affiché est une bonne affaire ou s'il est élevé par rapport à la valeur prédite.

Dans les deux cas, l'application vérifie si l'utilisateur a saisi un prix valide et fournit des messages d'erreur le cas échéant.

L'objectif global de l'application est de permettre aux utilisateurs d'évaluer le prix d'une voiture en fonction de ses caractéristiques et de faciliter la prise de décision d'achat ou de vente.

En ce qui concerne le lancement de l'application, nous exécutons la commande adéquate qui nous délivrera les adresses pour y accéder.

```
PS D:\Documents\EIGSI\Modules\Semestre 8\IA & Machine Learning\Projet> streamlit run app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.11.106:8501
```

La première adresse est une adresse locale propre à l'appareil sur lequel la commande a été exécutée. La seconde est une adresse à partir de laquelle l'ensemble des appareils connectés à un même réseau peuvent accéder à l'application. Ainsi, nous avons déployer notre programme et permettons à plus d'un de s'en servir.

Conclusion

Dans le cadre de ce projet de prédiction du prix d'un véhicule, nous avons exploré l'application du Machine Learning pour résoudre une problématique complexe et pertinente dans le secteur de l'automobile. Notre objectif était de développer un modèle capable de prédire le prix d'un véhicule en fonction de ses caractéristiques.

Au cours de notre étude, nous avons réalisé une analyse approfondie des données, en identifiant les variables significatives et en effectuant des traitements pour préparer les données à l'entraînement du modèle. Nous avons utilisé la méthode de l'arbre de décision pour construire notre modèle de prédiction, en optimisant ses hyperparamètres grâce à une validation croisée. Aussi, les performances de notre modèle ont été évaluées à l'aide de métriques d'évaluation telles que le coefficient de détermination (R^2). Nous avons constaté que notre modèle était capable de fournir des prédictions précises du prix des véhicules, ce qui peut être extrêmement bénéfique tant pour les acheteurs que pour les vendeurs.

De plus, nous avons mis en œuvre une application interactive en utilisant la bibliothèque Streamlit, permettant aux utilisateurs de prédire le prix d'un véhicule en entrant ses caractéristiques. Cela rend notre modèle accessible et convivial pour les personnes intéressées par l'estimation du prix d'un véhicule sur le marché.

Néanmoins, notre projet présente certaines limitations. La prédiction du prix des véhicules est influencée par de nombreux facteurs, et notre modèle ne prend en compte qu'un ensemble limité de variables. Une amélioration possible consisterait à intégrer davantage de données, telles que des informations spécifiques sur la marque, le modèle, le marché local, etc.

Ainsi, ce projet démontre l'efficacité du Machine Learning dans la prédiction du prix des véhicules. Notre modèle offre un outil pratique et fiable pour estimer le prix d'un véhicule en fonction de ses caractéristiques. Cependant, il convient de noter que les décisions finales d'achat ou de vente de véhicules doivent prendre en compte d'autres aspects, tels que l'état du véhicule, les fluctuations du marché et les négociations entre les parties.

Références

- https://github.com/NGALENAL1004/MachineLeaningProject_PredictPriceCar
- https://www.kaggle.com/datasets/adhurimquku/ford-car-price-prediction
- https://mrmint.fr/multivariate-regression
- https://docs.streamlit.io/
- https://medium.com/@WDlcls/mod%C3%A8le-pr%C3%A9dictif-de-la-valeur-automobile-7027ca342f1c