Amine Ait Laamim

Importation des bibliothèques

In [282...]
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
import joblib
```

Chargement des données

In [283...]
```
df = pd.read_csv("Car_Price_Prediction.csv")
```

Exploration des données

In [284...]
```
df.head()
```

Out[284...]

	Make	Model	Year	Engine Size	Mileage	Fuel Type	Transmission	Price
0	Honda	Model B	2015	3.9	74176	Petrol	Manual	30246.207931
1	Ford	Model C	2014	1.7	94799	Electric	Automatic	22785.747684
2	BMW	Model B	2006	4.1	98385	Electric	Manual	25760.290347
3	Honda	Model B	2015	2.6	88919	Electric	Automatic	25638.003491
4	Honda	Model C	2004	3.4	138482	Petrol	Automatic	21021.386657

In [285...]
```
df.describe()
```

Out[285...]

	Year	Engine Size	Mileage	Price
count	1000.000000	1000.000000	1000.00000	1000.000000
mean	2010.688000	2.798300	97192.48700	25136.615530
std	6.288577	1.024137	59447.31576	5181.401368
min	2000.000000	1.000000	56.00000	6704.953524
25%	2005.000000	1.900000	44768.75000	21587.878370
50%	2011.000000	2.800000	94411.50000	25189.325247
75%	2016.000000	3.700000	148977.75000	28806.368974
max	2021.000000	4.500000	199667.00000	41780.504635

In [286...]
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Make          1000 non-null   object 
 1   Model         1000 non-null   object 
 2   Year          1000 non-null   int64  
 3   Engine Size   1000 non-null   float64
 4   Mileage       1000 non-null   int64  
 5   Fuel Type     1000 non-null   object 
 6   Transmission  1000 non-null   object 
 7   Price         1000 non-null   float64
dtypes: float64(2), int64(2), object(4)
memory usage: 62.6+ KB
```

In [287...]
```
df.isna().sum()
```

Out[287...]
```
Make            0
Model           0
Year            0
Engine Size     0
Mileage         0
Fuel Type       0
Transmission    0
Price           0
dtype: int64
```

Nettoyage des données

In [288...]
```
df.columns
```

Out[288...]
```
Index(['Make', 'Model', 'Year', 'Engine Size', 'Mileage', 'Fuel Type',
       'Transmission', 'Price'],
      dtype='object')
```

In [289...]
```
df.value_counts()
```

Out[289...]
```
Make    Model    Year  Engine Size  Mileage  Fuel Type  Transmission  Price      
Audi    Model A  2000  2.3          32175    Petrol     Automatic     23103.147508   1
Honda   Model B  2010  1.1          140777   Diesel     Manual        19619.339512   1
                 2000  1.0          25037    Petrol     Manual        20172.866054   1
                       3.7          113726   Diesel     Manual        28742.652141   1
                 2001  2.9          16498    Electric   Automatic     25839.542162   1
                                                                                    ..
BMW     Model D  2011  2.8          46622    Petrol     Manual        27411.650083   1
                       3.9          99668    Diesel     Manual        26243.468247   1
                       4.5          25653    Electric   Automatic     30828.316109   1
                 2013  2.3          128391   Petrol     Manual        26386.334096   1
Toyota  Model E  2021  3.8          158497   Electric   Automatic     32061.750020   1
Name: count, Length: 1000, dtype: int64
```

In [290...]
```
num_cols = df.select_dtypes(include=["int", "float"]).columns
num_cols
```

Out[290...]
```
Index(['Year', 'Engine Size', 'Mileage', 'Price'], dtype='object')
```

In [291...]
```
for col in num_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = (df[col] < lower_bound) | (df[col] > upper_bound)
    print(outliers.sum())
    df[col] = df[col].clip(lower=lower_bound, upper=upper_bound)
```

```
0
0
0
3
```

Encodage des variables catégorielles

OneHotEncoder

In [292...]
```
cat_cols = df.select_dtypes(include='object').columns
print("cat_cols : \n", cat_cols)
```

```
cat_cols : 
 Index(['Make', 'Model', 'Fuel Type', 'Transmission'], dtype='object')
```

In [293...]
```
for col in cat_cols:
    print("**************************")
    print(df[col].value_counts())
```

```
**************************
Make
Ford      225
Audi      212
Honda     198
Toyota    187
BMW       178
Name: count, dtype: int64
**************************
Model
Model B    212
Model C    205
Model A    202
Model D    197
Model E    184
Name: count, dtype: int64
**************************
Fuel Type
Diesel    344
Petrol    331
Electric  325
Name: count, dtype: int64
**************************
Transmission
Manual       511
Automatic    489
Name: count, dtype: int64
```

In [294...]
```
cat_multi_cols = ["Make", "Model", "Fuel Type"]
```

In [295...]
```
OHE = OneHotEncoder()
col_ohe = OHE.fit_transform(df[cat_multi_cols])
```

In [296...]
```
OHE.get_feature_names_out(cat_multi_cols)
```

Out[296...]
```
array(['Make_Audi', 'Make_BMW', 'Make_Ford', 'Make_Honda', 'Make_Toyota',
       'Model_Model A', 'Model_Model B', 'Model_Model C', 'Model_Model D',
       'Model_Model E', 'Fuel Type_Diesel', 'Fuel Type_Electric',
       'Fuel Type_Petrol'], dtype=object)
```

In [297...]
```
col_ohe = pd.DataFrame(col_ohe.toarray(), columns=OHE.get_feature_names_out(cat_multi_cols), dtype='int')
col_ohe.head()
```

Out[297...]

	Make_Audi	Make_BMW	Make_Ford	Make_Honda	Make_Toyota	Model_Model A	Model_Model B	Model_Model C	Model_Model D	Model_Model E	Fuel Type_Diesel	Fuel Type_Electric	Fuel Type_Petrol
0	0	0	0	1	0	0	1	0	0	0	0	0	1
1	0	0	1	0	0	0	0	1	0	0	0	1	0
2	0	1	0	0	0	0	1	0	0	0	0	1	0
3	0	0	0	1	0	0	1	0	0	0	0	1	0
4	0	0	0	0	1	0	0	1	0	0	0	0	1

In [298...]
```
df = df.drop(cat_multi_cols, axis=1)
df = pd.concat([col_ohe ,df], axis=1)
```

LabelEncoder

In [299...]
```
le = LabelEncoder()
df['Transmission'] = le.fit_transform(df['Transmission'])
```

In [300...]
```
print(le.classes_)
```

```
['Automatic' 'Manual']
```

In [301...]
```
df.head()
```

Out[301...]

	Make_Audi	Make_BMW	Make_Ford	Make_Honda	Make_Toyota	Model_Model A	Model_Model B	Model_Model C	Model_Model D	Model_Model E	Fuel Type_Diesel	Fuel Type_Electric	Fuel Type_Petrol	Year	Engine Size	Mileage	Transmission	Price
0	0	0	0	1	0	0	1	0	0	0	0	0	1	2015	3.9	74176	1	30246.207931
1	0	0	1	0	0	0	0	1	0	0	0	1	0	2014	1.7	94799	0	22785.747684
2	0	1	0	0	0	0	1	0	0	0	0	1	0	2006	4.1	98385	1	25760.290347
3	0	0	0	1	0	0	1	0	0	0	0	1	0	2015	2.6	88919	0	25638.003491
4	0	0	0	1	0	0	0	1	0	0	0	0	1	2004	3.4	138482	0	21021.386657

Préparation des données pour l'entraînement

In [302...]
```
X = df.drop("Price", axis=1)
y = df["Price"]
```

In [303...]
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Normalisation des données

In [304...]
```
num_cols
```

Out[304...]
```
Index(['Year', 'Engine Size', 'Mileage', 'Price'], dtype='object')
```

In [305...]
```
cols_scaler = ['Year', 'Engine Size', 'Mileage']
```

In [306...]
```
scaler = StandardScaler()
X_train[cols_scaler] = scaler.fit_transform(X_train[cols_scaler])
#pour evite data leaking
X_test[cols_scaler] = scaler.transform(X_test[cols_scaler])
```

Entraînement du modèle

In [307...]
```
for n_e in [10, 20, 50, 70, 100]:
    rf = RandomForestRegressor(n_estimators=n_e, max_depth=15, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"n_estimators = {n_e}")
    print(f"       mse = {mse}, r2 = {r2}")

    print(f"       feature importances : {rf.feature_importances_}")
```

```
n_estimators = 10
       mse = 6039646.090796605, r2 = 0.7793866255215932
       feature importances : [0.00352443 0.00376965 0.00323345 0.00445768 0.00438649 0.00480284
 0.00375982 0.00407909 0.00582683 0.00354661 0.00332984 0.00332337
 0.00472898 0.38026298 0.18962855 0.37420724 0.00616496]
n_estimators = 20
       mse = 5935488.769857965, r2 = 0.7831126151982641
       feature importances : [0.00364764 0.00427681 0.00347316 0.00399094 0.00411369 0.00477871
 0.00416563 0.0050497  0.00453352 0.00362289 0.00334693 0.0033841
 0.00552326 0.37568919 0.1872475  0.37901657 0.00464257]
n_estimators = 50
       mse = 5761130.073854303, r2 = 0.7894837877073437
       feature importances : [0.00404919 0.00438064 0.00348487 0.0040445  0.0040467  0.00467618
 0.00415602 0.00463188 0.00463286 0.00383406 0.00323577 0.00329397
 0.00441147 0.37552212 0.19247537 0.3736235  0.00404895]
n_estimators = 70
       mse = 5663737.643333536, r2 = 0.7931164254939964
       feature importances : [0.00396255 0.0043793  0.00348882 0.0045903  0.00402154 0.00479146
 0.00405913 0.00473498 0.00476614 0.00393626 0.00360647 0.0032499
 0.00423387 0.37751465 0.19188129 0.3733475  0.00475662]
n_estimators = 100
       mse = 5665174.1801893302, r2 = 0.7931216170380146
       feature importances : [0.00393844 0.00432437 0.00339374 0.00404442 0.00423142 0.00487199
 0.00406662 0.00473485 0.00468099 0.00412838 0.00328218 0.00340623
 0.00414389 0.37704373 0.18983387 0.37521816 0.00476628]
```

In [308...]
```
rf = RandomForestRegressor(n_estimators=70, max_depth=5, random_state=42)
```

Bagging

In [310...]
```
for n_e in [10, 20, 50, 70, 100]:
    bagging = BaggingRegressor(n_estimators=n_e, max_samples=0.8, bootstrap=True, random_state=42)
    bagging.fit(X_train, y_train)
    y_pred = bagging.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"n_estimators = {n_e}")
    print(f"       mse = {mse}, r2 = {r2}")
```

```
n_estimators = 10
       mse = 6458237.134331172, r2 = 0.7640189082619375
n_estimators = 20
       mse = 5958634.678140086, r2 = 0.7825591719073867
n_estimators = 50
       mse = 5808167.8629984664, r2 = 0.7877650203480431
n_estimators = 70
       mse = 5685385.789725223, r2 = 0.7922549992577287
n_estimators = 100
       mse = 5595250.737153206, r2 = 0.7956455178770098
```

Enregistrer les modèles entraînés

In [311...]
```
joblib.dump(rf, "RF_Regression")
joblib.dump(scaler, "Scaler_RF_Regression")
```

Out[311...]
```
['Scaler_RF_Regression']
```

Quel modèle offre la meilleure précision entre Bagging et Random Forest ? Justifiez votre réponse.

Random Forest offre la meilleure precision, car dans bagging, tous les arbres utilisent toutes les variables, ce qui les rend tres similaires, surtout si une variable est beaucoup plus forte que les autre, les predictions des arebres sont tres correlees, donc la moyenne ne reduit pas beaucoup l'erreur. random forest corrige ca par utiliser une petite selection aleatoire de variables, alors les arbres va etre differents les uns des autre, cette redult la corelation entre les arbresic-a-d diminue la variance) et amelorie la precision finale du modele.

En quoi Random Forest améliore-t-il le simple Bagging ?

J'ai trouvé le paragraphe suivant dans le livre *'Introduction to Statistical Learning'*, qui explique pourquoi Random Forest est plus efficaces que le bagging :

" In other words, in building a random forest, at each split in the tree, the algorithm is not even allowed to consider a majority of the available predictors. This may sound crazy, but it has a clever rationale. Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors. Then in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split. Consequently, all of the bagged trees will look quite similar to each other. Hence the predictions from the bagged trees will be highly correlated. Unfortunately, averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging uncorrelated quantities. In particular, this means that bagging will not lead to a substantial reduction in variance over a single tree in this setting. Random forests overcome this problem by forcing each split to consider only a subset of the predictors. Therefore, on average (p − m)/p of the splits will not even consider the strong predictor, and so other predictors will have more of a chance. We can think of this process as decorrelating the trees, thereby making the average of the resulting trees less variable and hence more reliable. The main difference between bagging and random forests is the choice of predictor subset size m. For instance, if a random forest is built using m = p, then this amounts simply to bagging. On the Heart data, random forests using m = √p leads to a reduction in both test error and OOB error over bagging (Figure 8.8). "