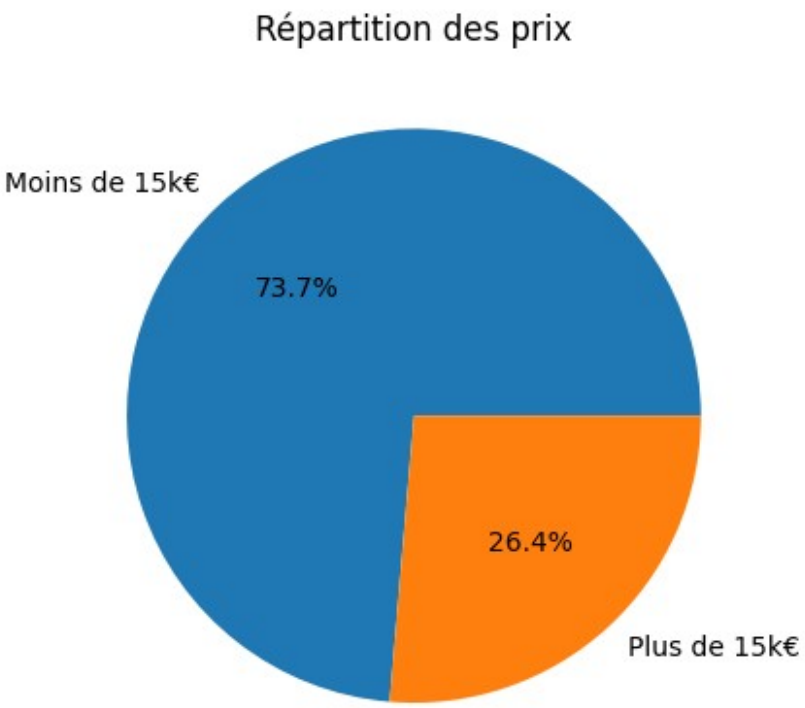
```python
# Tout d'abord je vais générer des données aléatoire ici au lieu de prendre un csv existant comme nous avions pu le faire en cours

import pandas as pd
import numpy as np

# Définir un seed pour la reproductibilité
np.random.seed(42)

# Nombre d'exemples
n = 2000

# Création du dataset de voitures d'occasion
brands = ['Toyota', 'Peugeot', 'Renault', 'BMW', 'Mercedes']
fuel_types = ['Essence', 'Diesel', 'Hybride', 'Electrique']
gearbox = ['Manuelle', 'Automatique']

data = {
    'brand': np.random.choice(brands, n),
    'year': np.random.randint(2005, 2023, n),
    'mileage_km': np.random.randint(20000, 250000, n),
    'fuel_type': np.random.choice(fuel_types, n),
    'gearbox': np.random.choice(gearbox, n),
    'power_hp': np.random.randint(60, 400, n),
    'price': np.random.randint(3000, 50000, n)
}

df = pd.DataFrame(data)

# Variable cible : 1 si le prix dépasse 15 000€, sinon 0
df['price_category'] = (df['price'] > 15000).astype(int)

# Ajouter des valeurs manquantes aléatoirement
for col in ['gearbox', 'fuel_type', 'mileage_km']:
    indices = df.sample(frac=0.05).index
    df.loc[indices, col] = np.nan

# Sauvegarder le fichier CSV localement (plutôt que de l'importer comme on faisait dans les TPs il est en local ici)
df.to_csv("voitures_occasion.csv", index=False)

# 1. Importation et exploration des données
import matplotlib.pyplot as plt
import seaborn as sns

# Charger le dataset à l'aide de pandas (même si en soit il est déjà en local mais juste à titre d'exemple pour l'importation)
df = pd.read_csv("voitures_occasion.csv")

# Vérifier les dimensions du dataset
print(f"Les dimensions du dataset sont : {df.shape}.")
```

```
# Afficher un aperçu des 5 premières lignes du dataset (et non tout le
DataFrame)
print("\nExemple des 5 premières lignes du dataset :")
print(df.head())

# Types de chaque variable
print("\nTypes des colonnes :")
print(df.dtypes)

# Nombre de valeurs nulles par colonne
print("\nNombre de valeurs nulles par colonne :")
print(df.isnull().sum())

# Statistiques descriptives des colonnes numériques
print("\nDescription statistique des colonnes numériques :")
print(df.describe())

# Analyse de la variable cible (répartition des prix)
df['price_category'].value_counts().plot.pie(autopct='%1.1f%%',
labels=['Moins de 15k€', 'Plus de 15k€'])
plt.title("Répartition des prix")
plt.ylabel("")
plt.show()

# Affiche des histogrammes pour chaque variable numérique
df.hist(bins=30, figsize=(15, 10), edgecolor='black')
plt.suptitle('Histogrammes des variables numériques', fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()

# Histogramme stratifié de 'year' selon la cible
plt.figure(figsize=(10,5))
sns.histplot(data=df, x='year', hue='price_category', bins=20,
kde=False, multiple='stack')
plt.title("Année des véhicules stratifiée par catégorie de prix")
plt.show()

# Matrice de corrélation (uniquement pour les variables numériques)
plt.figure(figsize=(10,6))
df_num = df.select_dtypes(include=['int64', 'float64'])
sns.heatmap(df_num.corr(), annot=True, cmap="coolwarm")
plt.title("Matrice de corrélation")
plt.show()

# Boxplot du prix selon la marque
plt.figure(figsize=(10,5))
sns.boxplot(data=df, x='brand', y='price')
plt.title("Distribution des prix selon la marque")
plt.show()
```

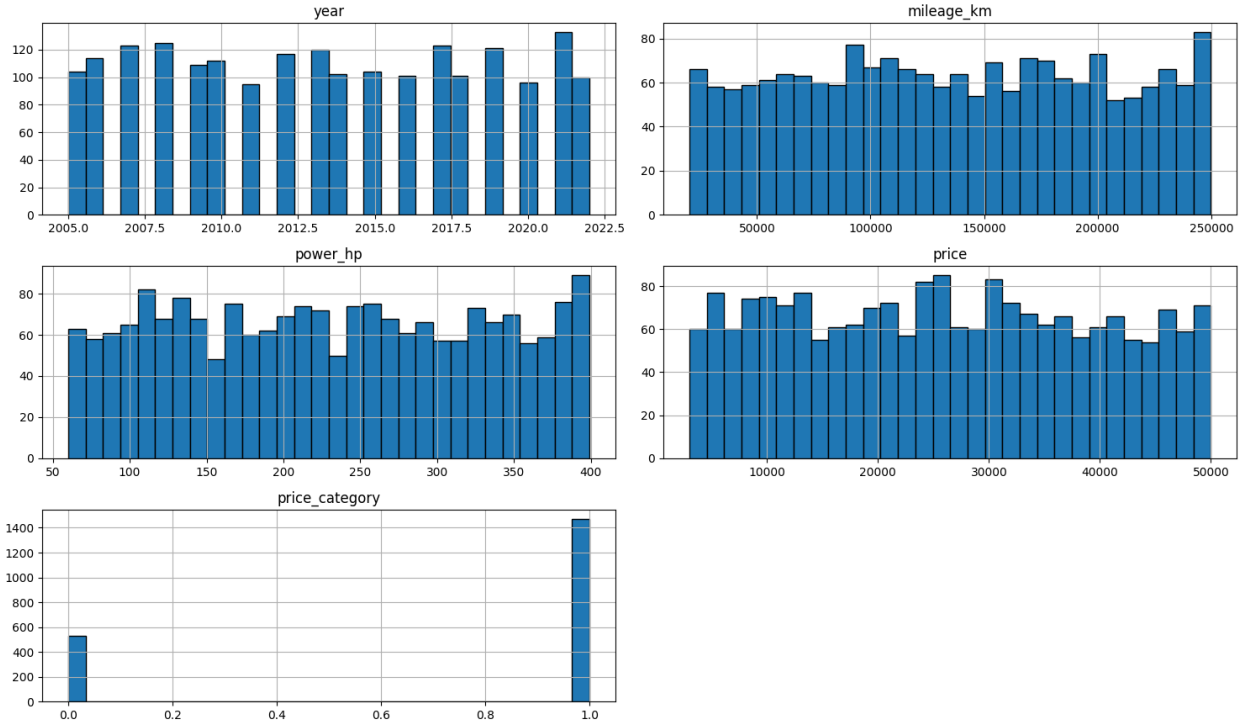
```python
# Analyse des valeurs manquantes plus poussée
missing_df = df.isnull().mean().sort_values(ascending=False)
print("Proportion de valeurs manquantes par colonne :")
print(missing_df)
```

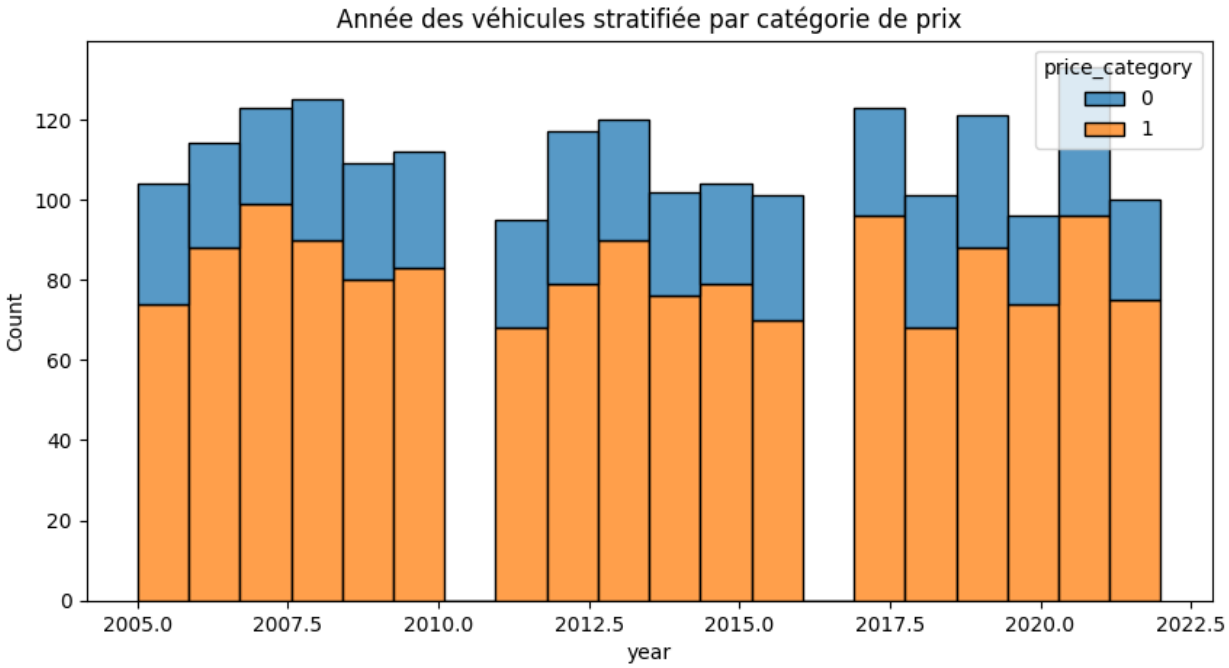
```
Les dimensions du dataset sont : (2000, 8).

Exemple des 5 premières lignes du dataset :
      brand  year  mileage_km fuel_type      gearbox  power_hp  price
\
0       BMW  2011    184529.0   Essence     Manuelle       130  36690

1  Mercedes  2015    105679.0   Essence  Automatique       162  48697

2   Renault  2014    145829.0    Diesel  Automatique       385  32176

3  Mercedes  2013    148216.0   Essence  Automatique       216  17925

4  Mercedes  2018    153175.0    Diesel     Manuelle       253  24251


   price_category
0               1
1               1
2               1
3               1
4               1

Types des colonnes :
brand              object
year                int64
mileage_km        float64
fuel_type          object
gearbox            object
power_hp            int64
price               int64
price_category      int64
dtype: object

Nombre de valeurs nulles par colonne :
brand                0
year                 0
mileage_km         100
fuel_type          100
gearbox            100
power_hp             0
price                0
price_category       0
dtype: int64
```

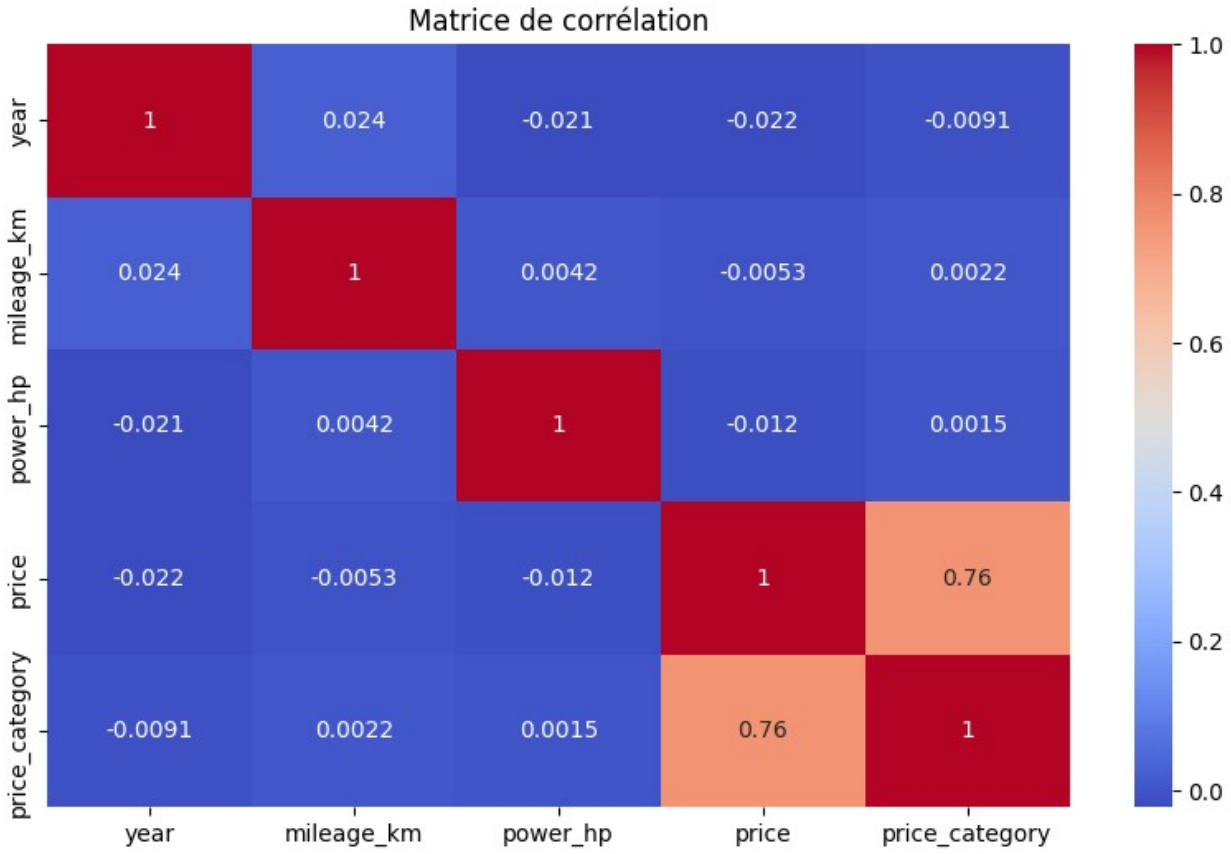
```
Description statistique des colonnes numériques :
              year     mileage_km     power_hp         price
price_category
count  2000.000000    1900.000000  2000.000000   2000.000000
2000.000000
mean   2013.450000  135826.874211   231.097000  26066.915500
0.736500
std       5.208557   66155.362098    98.565877  13444.597935
0.440641
min    2005.000000   20287.000000    60.000000   3015.000000
0.000000
25%    2009.000000   79817.500000   145.000000  14158.750000
0.000000
50%    2013.000000  135044.000000   229.000000  25777.500000
1.000000
75%    2018.000000  191849.250000   318.000000  37172.250000
1.000000
max    2022.000000  249797.000000   399.000000  49996.000000
1.000000
```

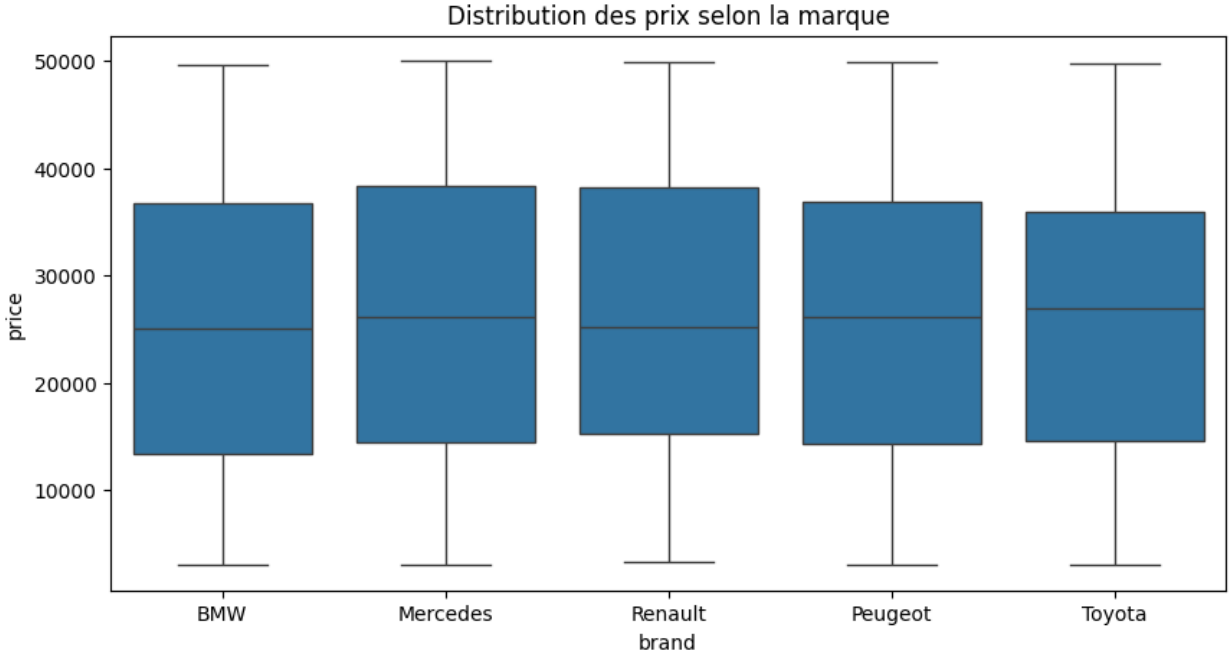


Histogrammes des variables numériques










```
Proportion de valeurs manquantes par colonne :
fuel_type         0.05
mileage_km        0.05
```

```
gearbox            0.05
year               0.00
brand              0.00
power_hp           0.00
price              0.00
price_category     0.00
dtype: float64
```

```
# 2. Prétraitement des données

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder

# Imputation des valeurs manquantes

# Pour les variables numériques
imputer_num = SimpleImputer(strategy='median')
df['mileage_km'] = imputer_num.fit_transform(df[['mileage_km']])

# Pour les variables catégorielles
imputer_cat = SimpleImputer(strategy='most_frequent')
for col in ['gearbox', 'fuel_type']:
    df[col] = imputer_cat.fit_transform(df[[col]]).ravel()

print("\nValeurs manquantes après imputation :")
print(df[['mileage_km', 'gearbox', 'fuel_type']].isnull().sum())

# Encodage des colonnes catégorielles
label_encoders = {}
colonnes_a_encoder = ['brand', 'fuel_type', 'gearbox']

for col in colonnes_a_encoder:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Vérification après imputation et encodage
print("\nAperçu après imputation et encodage des colonnes
catégorielles :")
print(df.head())
```

```
Valeurs manquantes après imputation :
mileage_km    0
gearbox       0
fuel_type     0
dtype: int64

Aperçu après imputation et encodage des colonnes catégorielles :
   brand  year  mileage_km  fuel_type  gearbox  power_hp  price  \
0      0  2011    184529.0          2        1       130  36690
```

```
1      1  2015    105679.0           2        0       162  48697
2      3  2014    145829.0           0        0       385  32176
3      1  2013    148216.0           2        0       216  17925
4      1  2018    153175.0           0        1       253  24251
```

```
   price_category
0               1
1               1
2               1
3               1
4               1
```

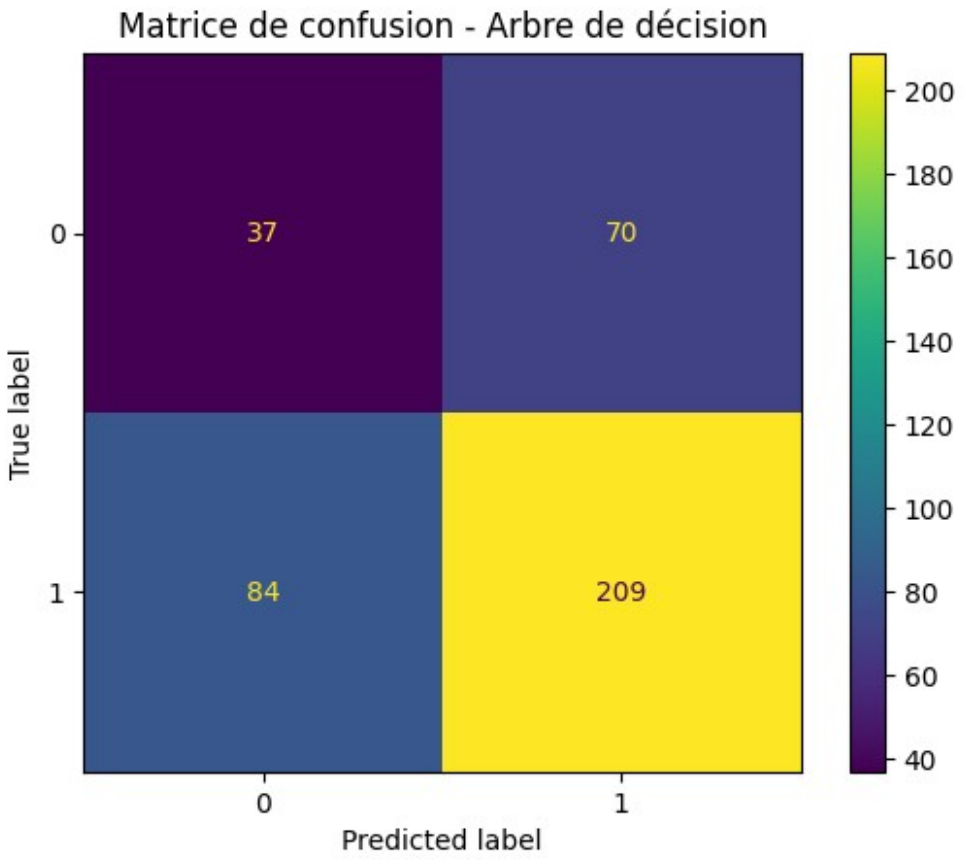
```python
# 3. Séparation des données

from sklearn.model_selection import train_test_split

# Suppression des colonnes inutiles à la prédiction
X = df.drop(columns=['price', 'price_category'])  # Variables
explicatives
y = df['price_category']  # Variable cible

# Séparation des données en ensemble d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Vérification des dimensions
print(f"Taille de X_train : {X_train.shape}")
print(f"Taille de X_test  : {X_test.shape}")
```

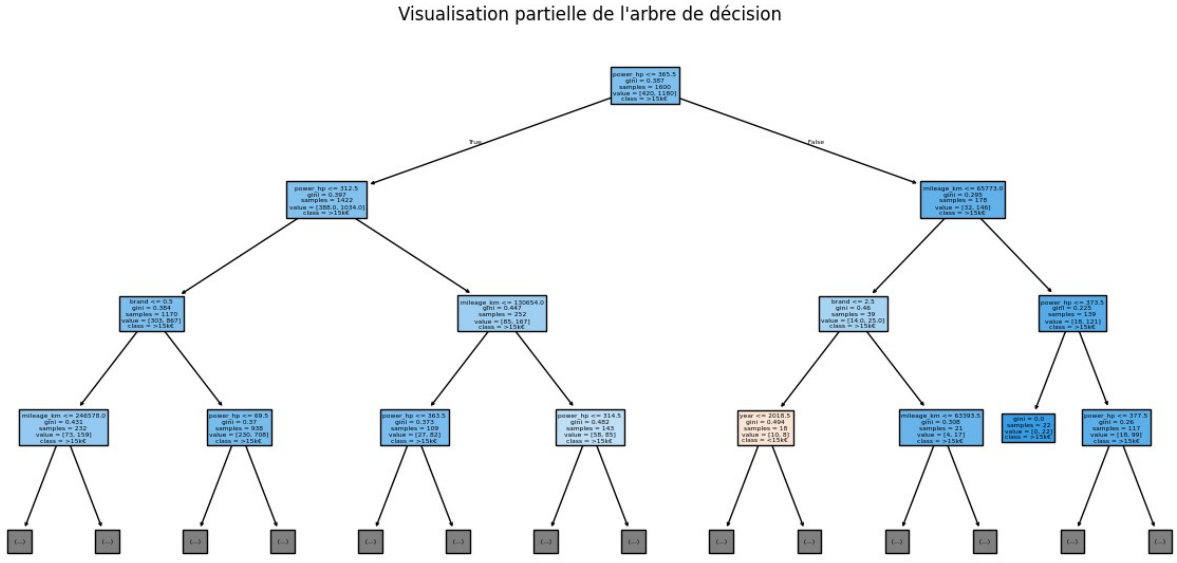
```
Taille de X_train : (1600, 6)
Taille de X_test  : (400, 6)
```

```python
# 4. Entraînement d’un modèle

from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import GridSearchCV

# Arbre de décision
tree = DecisionTreeClassifier(random_state=42)
tree.fit(X_train, y_train)
y_pred_tree = tree.predict(X_test)

print("Arbre de décision :")
print(f"Accuracy  : {accuracy_score(y_test, y_pred_tree):.3f}")
print(f"Precision : {precision_score(y_test, y_pred_tree):.3f}")
print(f"Recall    : {recall_score(y_test, y_pred_tree):.3f}")
print(f"F1-score  : {f1_score(y_test, y_pred_tree):.3f}")
```

```
ConfusionMatrixDisplay.from_predictions(y_test, y_pred_tree)
plt.title("Matrice de confusion - Arbre de décision")
plt.show()

plt.figure(figsize=(15,7))
plot_tree(tree, feature_names=X.columns, class_names=['<15k€',
'>15k€'], filled=True, max_depth=3)
plt.title("Visualisation partielle de l'arbre de décision")
plt.show()

# Random Forest avec GridSearch pour hyperparamètres
param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5]
}

forest = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(forest, param_grid, cv=3, n_jobs=-1,
scoring='accuracy')
grid_search.fit(X_train, y_train)

best_forest = grid_search.best_estimator_
y_pred_forest = best_forest.predict(X_test)

print(f"Meilleurs paramètres forêt aléatoire :
{grid_search.best_params_}")
print("Forêt aléatoire après GridSearch :")
print(f"Accuracy  : {accuracy_score(y_test, y_pred_forest):.3f}")
print(f"Precision : {precision_score(y_test, y_pred_forest):.3f}")
print(f"Recall    : {recall_score(y_test, y_pred_forest):.3f}")
print(f"F1-score  : {f1_score(y_test, y_pred_forest):.3f}")

ConfusionMatrixDisplay.from_predictions(y_test, y_pred_forest)
plt.title("Matrice de confusion - Forêt aléatoire")
plt.show()
```

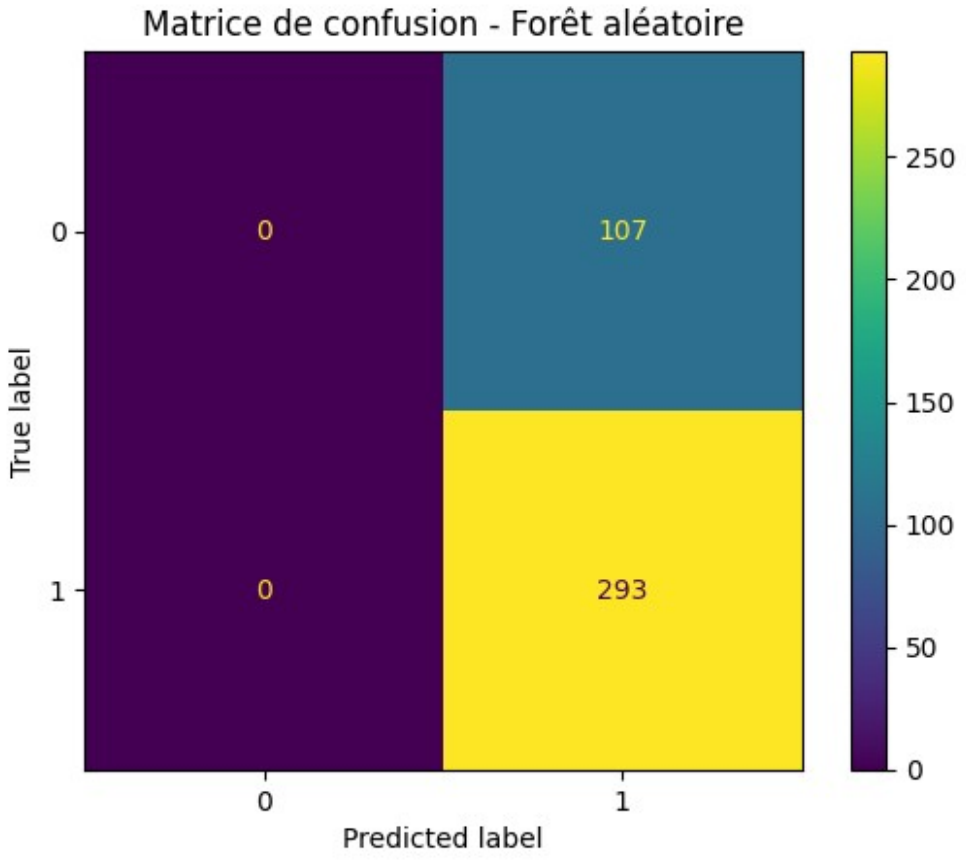
```
Arbre de décision :
Accuracy  : 0.615
Precision : 0.749
Recall    : 0.713
F1-score  : 0.731
```






```
Meilleurs paramètres forêt aléatoire : {'max_depth': 5,
'min_samples_split': 2, 'n_estimators': 50}
Forêt aléatoire après GridSearch :
Accuracy  : 0.733
```

```
Precision : 0.733
Recall    : 1.000
F1-score  : 0.846
```




```python
from sklearn.metrics import classification_report

# Regarder quelques erreurs de classification (faux positifs et faux négatifs)
errors = X_test[(y_test != y_pred_forest)]

print(f"Nombre d'erreurs : {errors.shape[0]}")
print("Exemples d'erreurs :")
print(errors.head())

# Ajout des vraies valeurs et prédictions pour analyse
errors['true_label'] = y_test.loc[errors.index]
y_pred_forest_series = pd.Series(y_pred_forest, index=y_test.index)
errors['predicted'] = y_pred_forest_series.loc[errors.index]

print(errors[['brand', 'fuel_type', 'gearbox', 'year', 'mileage_km',
'power_hp', 'true_label', 'predicted']].head())
print("\nClassification Report :")
```

```
print(classification_report(y_test, y_pred_forest,
target_names=['<15k€', '>15k€']))

Nombre d'erreurs : 107
Exemples d'erreurs :
      brand  year  mileage_km  fuel_type  gearbox  power_hp
353       0  2015    135044.0          3        0       124
1273      0  2012    172225.0          1        0       353
1323      1  2019    126244.0          3        0       368
56        0  2006    184759.0          1        1       179
1118      1  2008     79222.0          2        1       194
      brand  fuel_type  gearbox  year  mileage_km  power_hp
true_label  \
353       0          3        0  2015    135044.0       124
0
1273      0          1        0  2012    172225.0       353
0
1323      1          3        0  2019    126244.0       368
0
56        0          1        1  2006    184759.0       179
0
1118      1          2        1  2008     79222.0       194
0

      predicted
353           1
1273          1
1323          1
56            1
1118          1

Classification Report :
              precision    recall  f1-score   support

       <15k€       0.00      0.00      0.00       107
       >15k€       0.73      1.00      0.85       293

    accuracy                           0.73       400
   macro avg       0.37      0.50      0.42       400
weighted avg       0.54      0.73      0.62       400
```