Partie 2 : Classification

Dans cette partie, le problème d'apprentissage consiste à prédire la variable cible, c'est-à-dire le prix (Elevé, Bas), en fonction des caractéristiques du véhicule.

In [31]:

```
#  Importation des données et les trransformer du nominales vers des données numérique

import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_excel('C:/Users/utulisateur/Downloads/mini-projet/CaracteristiquesVehicules.xls')
from sklearn.preprocessing import LabelEncoder
enc=LabelEncoder()
df.Carosserie=enc.fit_transform(df["Carosserie"])
df.Nombre_cylindres=enc.fit_transform(df["Nombre_cylindres"])

#valeurs caractéristiques et valeur cible
X = df.iloc[:,:-1].values
y = df.iloc[:,-1].values

print(y.shape)
print(X.shape)
print("==>Opération terminée avec succé")
```

```
(58,)
(58, 7)
==>Opération terminée avec succé
```

In [32]:

```
from sklearn import svm
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import train_test_split
```

In [33]:

```
#transformation de données nominales vers des données numériques
from sklearn.preprocessing import LabelEncoder
enc=LabelEncoder()
df.Carosserie=enc.fit_transform(df["Carosserie"])
df.Nombre_cylindres=enc.fit_transform(df["Nombre_cylindres"])
print("-> La transformation des données nominales vers des données numérique à été bien terminée")
df
```

```
-> La transformation des données nominales vers des données numérique à été bien terminée
```

Out[33]:

	Identifiant	Carosserie	Empattement	longueur	Nombre_cylindres	Puissance	KilometrageMoyen	Prix
0	0	0	88.6	168.8	2	111	21	Bas
1	1	0	88.6	168.8	2	111	21	Bas
2	2	2	94.5	171.2	3	154	19	Bas
3	3	3	99.8	176.6	2	102	24	Bas
4	4	3	99.4	176.6	1	115	18	Bas
5	5	3	99.8	177.3	1	110	19	Bas
6	6	4	105.8	192.7	1	110	19	Bas
7	9	3	101.2	176.8	2	101	23	Bas
8	10	3	101.2	176.8	2	101	23	Bas

	Identifiant	Carosserie	Empattement	longueur	Nombre_cylindres	Puissance	KilometrageMoyen	Prix
9	11	3	101.2	176.8	3	121	21	Bas
10	13	3	103.5	189.0	3	182	16	Elevé
11	14	3	103.5	193.8	3	182	16	Elevé
12	15	3	110.0	197.0	3	182	15	Elevé
13	16	2	88.4	141.1	4	48	47	Bas
14	17	2	94.5	155.9	2	70	38	Bas
15	18	3	94.5	158.8	2	70	38	Bas
16	19	2	93.7	157.3	2	68	31	Bas
17	20	2	93.7	157.3	2	68	31	Bas
18	27	4	96.5	157.1	2	76	30	Bas
19	28	3	96.5	175.4	2	101	24	Bas
20	29	3	96.5	169.1	2	100	25	Bas
21	30	3	94.3	170.7	2	78	24	Bas
22	33	3	113.0	199.6	3	176	15	Elevé
23	34	3	113.0	199.6	3	176	15	Elevé
24	35	3	102.0	191.7	5	262	13	Elevé
25	36	2	93.1	159.1	2	68	30	Bas
26	37	2	93.1	159.1	2	68	31	Bas
27	38	2	93.1	159.1	2	68	31	Bas
28	39	2	95.3	169.0	6	101	17	Bas
29	43	3	104.9	175.0	2	72	31	Bas
30	44	3	110.0	190.9	1	123	22	Elevé
31	45	4	110.0	190.9	1	123	22	Elevé
32	46	3	120.9	208.1	0	184	14	Elevé
33	47	1	112.0	199.2	0	184	14	Elevé
34	49	2	93.7	157.3	2	68	37	Bas
35	50	2	93.7	157.3	2	68	31	Bas
36	51	3	96.3	172.4	2	88	25	Bas
37	52	3	96.3	172.4	2	88	25	Bas
38	53	3	94.5	165.3	2	55	45	Bas
39	54	3	94.5	165.3	2	69	31	Bas
40	55	3	94.5	165.3	2	69	31	Bas
41	56	4	94.5	170.2	2	69	31	Bas
42	57	3	100.4	184.6	3	152	19	Bas
43	61	1	89.5	168.9	3	207	17	Elevé
44	62	0	89.5	168.9	3	207	17	Elevé
45	66	2	95.7	158.7	2	62	35	Bas
46	67	2	95.7	158.7	2	62	31	Bas
47	68	2	95.7	158.7	2	62	31	Bas
48	69	4	95.7	169.7	2	62	31	Bas
49	70	4	95.7	169.7	2	62	27	Bas
50	71	4	95.7	169.7	2	62	27	Bas
51	79	4	104.5	187.8	3	156	19	Bas
52	80	3	97.3	171.7	2	52	37	Bas
53	81	3	97.3	171.7	2	85	27	Bas

	Identifiant	Carosserie	Empattement	longueur	Nombre_cylindres	Puissance	KilometrageMoyen	Prix
54	82	3	97.3	171.7	2	52	37	Bas
55	86	3	97.3	171.7	2	100	26	Bas
56	87	3	104.3	188.8	2	114	23	Bas
57	88	4	104.3	188.8	2	114	23	Bas

In [34]:

```
#fractionner dataset:

X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.2, random_stat
e=0)
```

Test de l'algorithmes de classification supervisée SVM

In [35]:

```
#   Calculer la SVM , On commence par importer la bilbiothèque nécessaire pour les SVM :
#   puis on applique la SVM sur le jeu de données :

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn import svm
from sklearn.svm import SVC
clf = svm.SVC()

#training
clf.fit(X_train1, y_train1)
clf.predict(X_test1)
```

Out[35]:

```
array(['Bas', 'Bas', 'Bas', 'Bas', 'Elevé', 'Bas', 'Elevé', 'Bas',
       'Elevé', 'Bas', 'Bas', 'Elevé'], dtype=object)
```

In [36]:

```
# calcule du score:
print('Le score de cet Algorithme est  :  ')

clf.score(X_train1,y_train1)
```

```
Le score de cet Algorithme est  :
```

Out[36]:

```
0.9347826086956522
```

In [37]:

```
# affichage du matrice :
clf.support_vectors_
```

Out[37]:

```
array([[ 87. ,   3. , 104.3, 188.8,   2. , 114. ,  23. ],
       [  5. ,   3. ,  99.8, 177.3,   1. , 110. ,  19. ],
       [ 79. ,   4. , 104.5, 187.8,   3. , 156. ,  19. ],
       [ 88. ,   4. , 104.3, 188.8,   2. , 114. ,  23. ],
       [  6. ,   4. , 105.8, 192.7,   1. , 110. ,  19. ],
       [ 11. ,   3. , 101.2, 176.8,   3. , 121. ,  21. ],
       [  0. ,   0. ,  88.6, 168.8,   2. , 111. ,  21. ],
       [ 46. ,   3. , 120.9, 208.1,   0. , 184. ,  14. ],
       [ 45. ,   4. , 110. , 190.9,   1. , 123. ,  22. ],
       [ 44. ,   3. , 110. , 190.9,   1. , 123. ,  22. ],
       [ 61. ,   1. ,  89.5, 168.9,   3. , 207. ,  17. ],
       [ 15. ,   3. , 110. , 197. ,   3. , 182. ,  15. ],
       [ 34. ,   3. , 113. , 199.6,   3. , 176. ,  15. ],
       [ 62. ,   0. ,  89.5, 168.9,   3. , 207. ,  17. ]])
```

In [38]:

```
clf.support_
```

Out[38]:

```
array([ 9, 15, 27, 31, 33, 38, 42,  2,  7, 16, 25, 30, 34, 45])
```

In [39]:

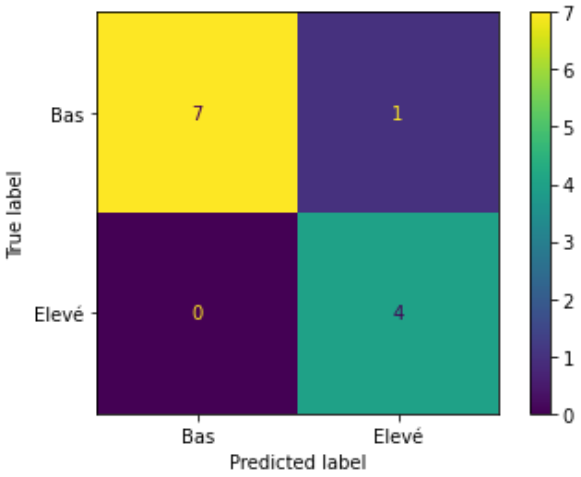
```
# Prédire la classe de données par la SVM

clf.predict(X_test1)
```

Out[39]:

```
array(['Bas', 'Bas', 'Bas', 'Bas', 'Elevé', 'Bas', 'Elevé', 'Bas',
       'Elevé', 'Bas', 'Bas', 'Elevé'], dtype=object)
```

In [40]:

```
# Matrice de Confusion pour le model SVM
plot_confusion_matrix(lr, X_test1, y_test1)
plt.show()
```



Test de l'algorithmes de classification supervisée les régression logistique

In [41]:

```
#   On commence par importer la bilbiothèque nécessaire pour les régression logistique :
#   puis on applique la régression logistique sur le jeu de données :

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

lr = LogisticRegression(max_iter=1000).fit(X_train1, y_train1)
```

In [42]:

```
print('Le score de cet Algorithme est  :  ')
lr.score(X_train1,y_train1)
```

```
Le score de cet Algorithme est  :
```

Out[42]:

```
1.0
```

In [43]:

```
lr.predict(X_test1)
```

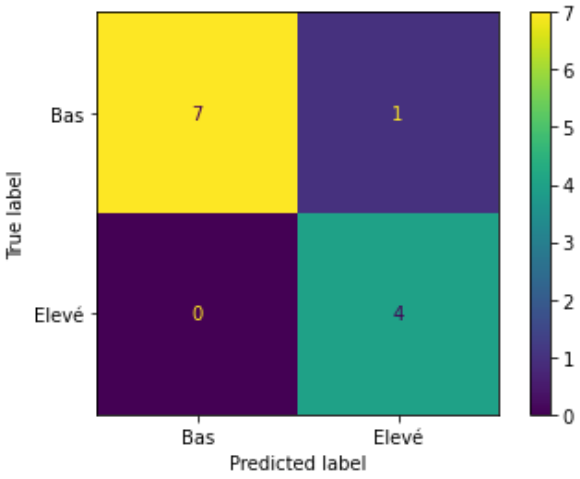
Out[43]:

Out[45]:

```
array(['Bas', 'Bas', 'Bas', 'Bas', 'Elevé', 'Elevé', 'Elevé', 'Bas',
       'Elevé', 'Bas', 'Bas', 'Elevé'], dtype=object)
```

In [44]:

```
plot_confusion_matrix(lr, X_test1, y_test1)
plt.show()
```



Test de l'algorithmes de classification supervisée le modél Naïve bayes

In [45]:

```
#   On commence par importer la bilbiothèque nécessaire pour le modél Naïve bayes:

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

gnb = GaussianNB().fit(X_train1,y_train1)
```

In [46]:

```
print('Le score de cet Algorithme est  :  ')
gnb.score(X_train1,y_train1)
```

```
Le score de cet Algorithme est  :
```

Out[46]:

```
0.9347826086956522
```

In [47]:

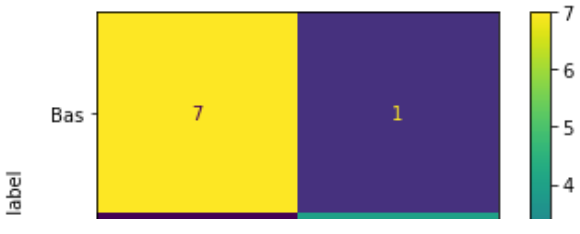
```
gnb.predict(X_test1)
```

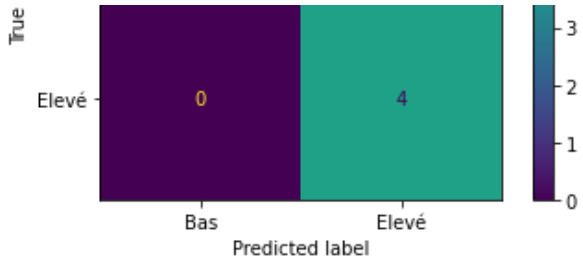
Out[47]:

```
array(['Bas', 'Bas', 'Elevé', 'Bas', 'Elevé', 'Bas', 'Elevé', 'Bas',
       'Elevé', 'Bas', 'Bas', 'Elevé'], dtype='<U5')
```

In [48]:

```
plot_confusion_matrix(gnb, X_test1, y_test1)
plt.show()
```





Test de l'algorithmes de classification supervisée le modél KNN KNeighborsClassifier

In [49]:

```
# On commence par importer la bilbiothèque nécessaire pour le modéle KNN  KNeighborsClass
ifier:

from sklearn.neighbors import KNeighborsClassifier

neigh = KNeighborsClassifier(n_neighbors = 1).fit(X_train1,y_train1)
```

In [50]:

```
print('Le score de cet Algorithme est  :  ')
neigh.score(X_train1,y_train1)
```

```
Le score de cet Algorithme est  :
```

Out[50]:

```
1.0
```

----> On remarque : dès que le nombre de voisin augmente , le score diminue

In [51]:

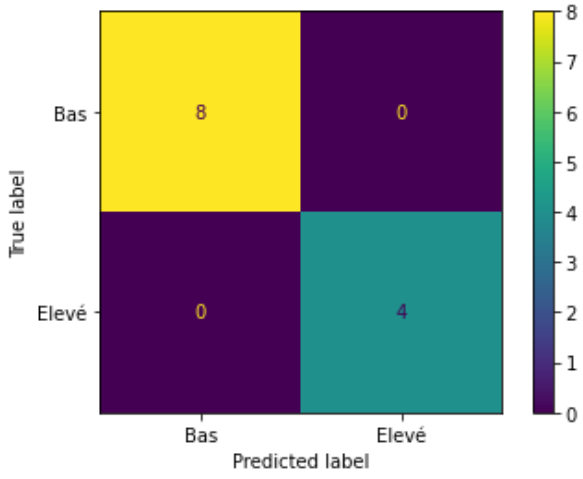
```
neigh.predict(X_test1)
```

Out[51]:

```
array(['Bas', 'Bas', 'Bas', 'Bas', 'Elevé', 'Bas', 'Elevé', 'Bas',
       'Elevé', 'Bas', 'Bas', 'Elevé'], dtype=object)
```

In [52]:

```
# Matrice de confusion:
plot_confusion_matrix(neigh, X_test1, y_test1)
plt.show()
```



Test de l'algorithmes de classification supervisée le modél

arbres de decision

In [53]:

```
#   On commence par importer la bilbiothèque nécessaire pour le modéle arbres de décision
,:

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score

dtc = DecisionTreeClassifier(random_state=0).fit(X_train1,y_train1)
```

In [54]:

```
print('Le score de cet Algorithme est  :  ')
dtc.score(X_train1,y_train1)
```

```
Le score de cet Algorithme est  :
```

Out[54]:

```
1.0
```

In [55]:

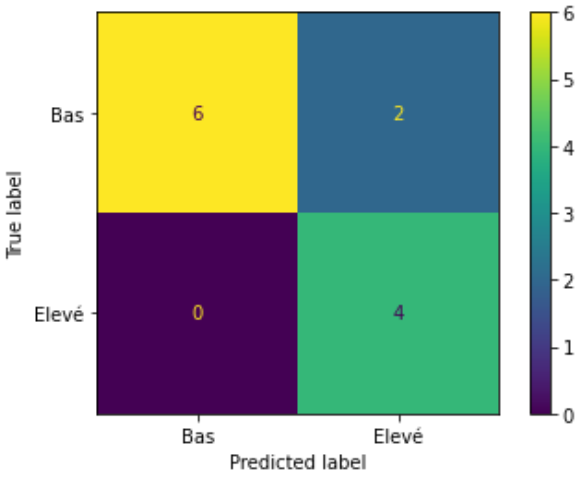
```
dtc.predict(X_test1)
```

Out[55]:

```
array(['Bas', 'Bas', 'Elevé', 'Bas', 'Elevé', 'Elevé', 'Elevé', 'Bas',
       'Elevé', 'Bas', 'Bas', 'Elevé'], dtype=object)
```

In [56]:

```
plot_confusion_matrix(dtc, X_test1, y_test1)
plt.show()
```



Test de l'algorithmes de classification supervisée le modél Multilayer Perceptron

In [57]:

```
#   On commence par importer la bilbiothèque nécessaire pour le modéle  Multilayer Percep
tron:

from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=
1).fit(X_train1,y_train1)
```

In [58]:

```
print('Le score de cet Algorithme est  :  ')
```

```
mlp.score(X_train1, y_train1)
```

```
Le score de cet Algorithme est  :
```

Out[58]:

```
0.8260869565217391
```

In [59]:

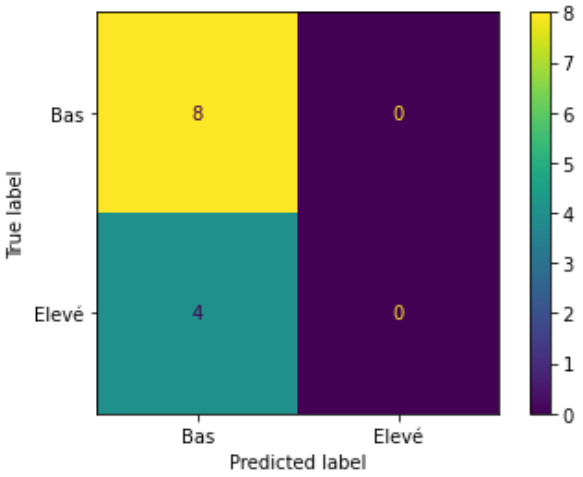
```
mlp.predict(X_test1)
```

Out[59]:

```
array(['Bas', 'Bas', 'Bas', 'Bas', 'Bas', 'Bas', 'Bas', 'Bas', 'Bas',
       'Bas', 'Bas', 'Bas'], dtype='<U5')
```

In [60]:

```
plot_confusion_matrix(mlp, X_test1, y_test1)
plt.show()
```



Test de l'algorithmes de classification supervisée le modél Adaboost

In [61]:

```
#   On commence par importer la bilbiothèque nécessaire pour le modéle  Adaboost:

from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import make_classification

ab = AdaBoostClassifier(n_estimators=100, random_state=0).fit(X_train1, y_train1)
```

In [62]:

```
print('Le score de cet Algorithme est  :  ')
ab.score(X_train1, y_train1)
```

```
Le score de cet Algorithme est  :
```

Out[62]:

```
1.0
```

In [63]:

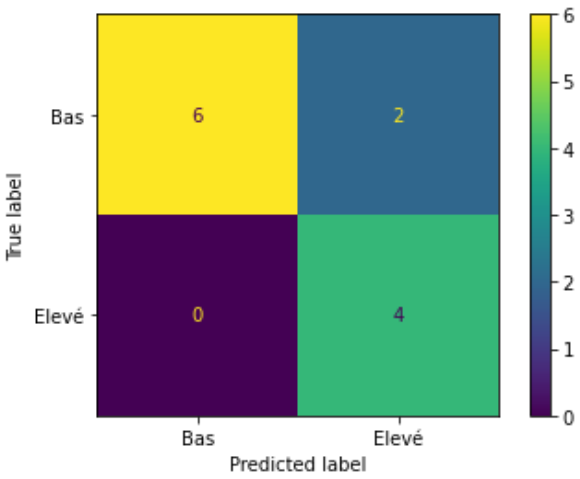
```
ab.predict(X_test1)
```

Out[63]:

```
array(['Bas', 'Bas', 'Elevé', 'Bas', 'Elevé', 'Elevé', 'Elevé', 'Bas',
       'Elevé', 'Bas', 'Bas', 'Elevé'], dtype=object)
```

In [64]:

```
plot_confusion_matrix(ab, X_test1, y_test1)
plt.show()
```



Test de l'algorithmes de classification supervisée le modél Random Forest

In [65]:

```
#   On commence par importer la bilbiothèque nécessaire pour le modéle  Random Forest:

from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification



rfc = RandomForestClassifier(max_depth=2, random_state=0).fit(X_train1,y_train1)
```

In [66]:

```
print('Le score de cet Algorithme est  :  ')
rfc.score(X_train1,y_train1)
```

```
Le score de cet Algorithme est  :
```

Out[66]:

```
1.0
```

In [67]:

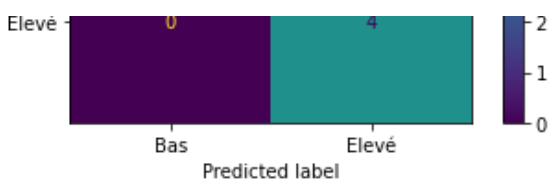
```
rfc.predict(X_test1)
```

Out[67]:

```
array(['Bas', 'Bas', 'Bas', 'Bas', 'Elevé', 'Bas', 'Elevé', 'Bas',
       'Elevé', 'Bas', 'Bas', 'Elevé'], dtype=object)
```

In [68]:

```
plot_confusion_matrix(rfc, X_test1, y_test1)
plt.show()
```



Résumé des scores : SVM : 0.8620689655172413 LogisticRegression : 0.7586206896551724 Naïve Bayes : 0.8620689655172413 KNN : 0.8620689655172413 (Nombre de voisin = 1 sinon le score diminue) DecisionTree : 0.8620689655172413 Multilayer Perceptron : 0.10344827586206896 AdaBoost : 0.6206896551724138 RandomForest : 0.7931034482758621 Les enseignements que nous avons tiré : Les algorithmes baysiens naïfs sont en effet les plus rapides (et pas si mauvais) ; SVM (ici LinearSVC) est effectivement plus efficace, et les performances se tiennent avec les autres algorithmes ; Nous avons également tenté d'utiliser de l'apprentissage profond, mais ici, ça n'a pas fonctionné mieux que SVM. Nous n'avons pas tout construit nous-mêmes, mais utlisé les fonctionnalités de scikit-learn : les lettres MLP dans le MLPClassifier signifient "Multi Layer Perceptron", c'est-à-dire un réseau de neurones relativement simple dans son fonctionnement : ====> On recommande d'utiliser l'algorithme SVM

In [ ]: