Charger les bibliothèques

In [1]:
```python
import pandas as pd
import numpy as np
from sklearn import preprocessing
from pandas import read_csv
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
```

1- Lecture des données

In [4]:
```python
# Charger la feuille "cars" du fichier Excel dans un DataFrame nommé df
df = pd.read_excel('carsPreprocessing.xlsx', sheet_name='cars')

# Charger la feuille "carsMod" du même fichier Excel dans un autre DataFrame nommé df1
df1 = pd.read_excel('carsPreprocessing.xlsx', sheet_name='carsMod')
```

2- Détermination du nombre d'individus et de variables

Dimensions de la table de données : nombre de lignes, nombre de colonnes la ligne d'en-tête n'est pas comptabilisée dans le nombre de lignes

In [7]:
```python
# Obtenir les dimensions (nombre de lignes et de colonnes) du DataFrame df1
dimension1 = df1.shape

# Afficher les dimensions de la base de données "carsMod" issue du fichier Excel
print("Dimension de la base de données carsPreprocessing.xlsx:",dimension1)
```

```
Dimension de la base de données carsPreprocessing.xlsx: (31, 8)
```

In [9]:
```python
# Afficher le contenu du DataFrame df1
display(df1)
```

	type	prix	cylindree	puissance	poids	conso	origine	abb
0	Daihatsu Cuore	11600.0	846	32	650	5.7	Japon	JP
1	Suzuki Swift 1.0 GLS	12490.0	993	39	790	5.8	Japon	JP
2	Fiat Panda Mambo L	10450.0	899	29	730	6.1	Italie	IT
3	VW Polo 1.4 60	17140.0	1390	44	955	6.5	Allemagne	DE
4	Opel Corsa 1.2i Eco	14825.0	1195	33	895	6.8	Allemagne	NaN
5	Toyota Corolla	19490.0	1331	55	1010	7.1	Japon	JP
6	Mercedes S 600	183900.0	5987	300	2250	18.7	Allemagne	DE
7	Maserati Ghibli GT	92500.0	2789	209	1485	14.5	Italie	IT
8	Opel Astra 1.6i 16V	25000.0	1597	74	1080	7.4	Allemagne	DE
9	Peugeot 306 XS 108	22350.0	1761	74	1100	9.0	France	NaN
10	Renault Safrane 2.2. V	36600.0	2165	101	1500	11.7	France	FR
11	Seat Ibiza 2.0 GTI	22500.0	1983	85	1075	9.5	Espagne	ES
12	VW Golt 2.0 GTI	31580.0	1984	85	1155	9.5	Allemagne	DE
13	Citroen ZX Volcane	28750.0	1998	89	1140	8.8	France	FR
14	Fort Escort 1.4i PT	20300.0	1390	54	1110	8.6	États-Unis	USA
15	Honda Civic Joker 1.4	19900.0	1396	66	1140	7.7	Japon	JP
16	Volvo 850 2.5	39800.0	2435	106	1370	10.8	Suède	SE
17	Ford Fiesta 1.2 Zetec	19740.0	1242	55	940	6.6	États-Unis	USA
18	Hyundai Sonata 3000	38990.0	2972	107	1400	11.7	Corée	KR
19	Lancia K 3.0 LS	50800.0	2958	150	1550	11.9	Italie	IT
20	Mazda Hachtback V	36200.0	2497	122	1330	10.8	Japon	JP
21	Mitsubishi Galant	31990.0	1998	66	1300	7.6	Japon	JP
22	Opel Omega 2.5i V6	47700.0	2496	125	1670	11.3	Allemagne	DE
23	Peugeot 806 2.0	36950.0	1998	89	1560	10.8	France	FR
24	Seat Alhambra 2.0	36400.0	1984	85	1635	11.6	Espagne	ES
25	Toyota Previa salon	50900.0	2438	97	1800	12.8	Japon	JP
26	Subaru Vivio 4WD	NaN	658	32	740	6.8	Japon	JP
27	Ferrari 456 GT	285000.0	5474	325	1690	21.3	Italie	IT
28	Fiat Tempra 1.6 Liberty	22600.0	1580	65	1080	9.3	Italie	IT
29	Nissan Primera 2.0	26950.0	1997	92	1240	9.2	Japon	JP
30	Volvo 960 Kombi aut	49300.0	2473	125	1570	12.7	Suède	SE

3- Types de variables

In [12]:
```python
# Afficher les types de données de chaque colonne du DataFrame df1
print(df1.dtypes)
```

```
type          object
prix         float64
cylindree      int64
puissance      int64
poids          int64
conso        float64
origine       object
abb           object
dtype: object
```

Nous disposons de 8 variables. type, origine et abb sont des varaibles qualitatives. prix,cylindree, puissance poids et conso sont quantitatives.

Ex2 : Traitement des données manquantes

1 - Détection des données manquantes : Comptage des données manquantes

On peut utiliser '.isany()' ou bien '.sum()' pour trouver le nombre des données manquantes

In [17]:
```python
# Compter et afficher le nombre de valeurs manquantes dans chaque colonne du DataFrame df1
df1.isnull().sum()
print("Nombre de données manquantes de la variable prix:",df1.isnull().sum())
```

```
Nombre de données manquantes de la variable prix: type         0
prix         1
cylindree    0
puissance    0
poids        0
conso        0
origine      0
abb          2
dtype: int64
```

In [87]:
```python
# Afficher spécifiquement le nombre de données manquantes dans la colonne 'prix' (si elle existe)
print("Nombre de données manquantes de la variable prix:", df1['prix'].isnull().sum())
```

```
Nombre de données manquantes de la variable prix: 1
```

In [89]:
```python
# Afficher spécifiquement le nombre de données manquantes dans la colonne 'prix' (si elle existe)
print("Nombre de données manquantes de la variable prix:", df1['abb'].isnull().sum())
```

```
Nombre de données manquantes de la variable prix: 2
```

In [91]:
```python
print(df1.shape)
```

```
(31, 8)
```

2-Détection des données manquantes : Affichage des données manqauntes

In [22]:
```python
Highlighted=df1.style.highlight_null('red')
Highlighted
```

Out[22]:

	type	prix	cylindree	puissance	poids	conso	origine	abb
0	Daihatsu Cuore	11600.000000	846	32	650	5.700000	Japon	JP
1	Suzuki Swift 1.0 GLS	12490.000000	993	39	790	5.800000	Japon	JP
2	Fiat Panda Mambo L	10450.000000	899	29	730	6.100000	Italie	IT
3	VW Polo 1.4 60	17140.000000	1390	44	955	6.500000	Allemagne	DE
4	Opel Corsa 1.2i Eco	14825.000000	1195	33	895	6.800000	Allemagne	nan
5	Toyota Corolla	19490.000000	1331	55	1010	7.100000	Japon	JP
6	Mercedes S 600	183900.000000	5987	300	2250	18.700000	Allemagne	DE
7	Maserati Ghibli GT	92500.000000	2789	209	1485	14.500000	Italie	IT
8	Opel Astra 1.6i 16V	25000.000000	1597	74	1080	7.400000	Allemagne	DE
9	Peugeot 306 XS 108	22350.000000	1761	74	1100	9.000000	France	nan
10	Renault Safrane 2.2. V	36600.000000	2165	101	1500	11.700000	France	FR
11	Seat Ibiza 2.0 GTI	22500.000000	1983	85	1075	9.500000	Espagne	ES
12	VW Golt 2.0 GTI	31580.000000	1984	85	1155	9.500000	Allemagne	DE
13	Citroen ZX Volcane	28750.000000	1998	89	1140	8.800000	France	FR
14	Fort Escort 1.4i PT	20300.000000	1390	54	1110	8.600000	États-Unis	USA
15	Honda Civic Joker 1.4	19900.000000	1396	66	1140	7.700000	Japon	JP
16	Volvo 850 2.5	39800.000000	2435	106	1370	10.800000	Suède	SE
17	Ford Fiesta 1.2 Zetec	19740.000000	1242	55	940	6.600000	États-Unis	USA
18	Hyundai Sonata 3000	38990.000000	2972	107	1400	11.700000	Corée	KR
19	Lancia K 3.0 LS	50800.000000	2958	150	1550	11.900000	Italie	IT
20	Mazda Hachtback V	36200.000000	2497	122	1330	10.800000	Japon	JP
21	Mitsubishi Galant	31990.000000	1998	66	1300	7.600000	Japon	JP
22	Opel Omega 2.5i V6	47700.000000	2496	125	1670	11.300000	Allemagne	DE
23	Peugeot 806 2.0	36950.000000	1998	89	1560	10.800000	France	FR
24	Seat Alhambra 2.0	36400.000000	1984	85	1635	11.600000	Espagne	ES
25	Toyota Previa salon	50900.000000	2438	97	1800	12.800000	Japon	JP
26	Subaru Vivio 4WD	nan	658	32	740	6.800000	Japon	JP
27	Ferrari 456 GT	285000.000000	5474	325	1690	21.300000	Italie	IT
28	Fiat Tempra 1.6 Liberty	22600.000000	1580	65	1080	9.300000	Italie	IT
29	Nissan Primera 2.0	26950.000000	1997	92	1240	9.200000	Japon	JP
30	Volvo 960 Kombi aut	49300.000000	2473	125	1570	12.700000	Suède	SE

In [93]:
```python
display(df1[df1.isna().any(axis=1)].style.highlight_null('red'))
NaN_Rows_Ids=df1[df1.isna().any(axis=1)].index
```

	type	prix	cylindree	puissance	poids	conso	origine	abb
4	Opel Corsa 1.2i Eco	14825.000000	1195	33	895	6.800000	Allemagne	nan
9	Peugeot 306 XS 108	22350.000000	1761	74	1100	9.000000	France	nan
26	Subaru Vivio 4WD	nan	658	32	740	6.800000	Japon	JP

3 et 4- Suppression des données manquantes

In [96]:
```python
df1Del=df1.copy()
df1Del.dropna(inplace=True)
```

5- Dimension de df1Del

In [99]:
```python
print("Dimension de la base de df1:",df1.shape)
print("Dimension de la base de df1Del:",df1Del.shape)
```

```
Dimension de la base de df1: (31, 8)
Dimension de la base de df1Del: (28, 8)
```

6- Réparation des données manqantes : Imputation par moyenne

In [102…
```python
df1ImputeMean=df1.copy()
a = df1.select_dtypes('number')
df1ImputeMean[a.columns] = a.fillna(a.mean())
display(df1ImputeMean.iloc[NaN_Rows_Ids,:])
```

	type	prix	cylindree	puissance	poids	conso	origine	abb
4	Opel Corsa 1.2i Eco	14825.0	1195	33	895	6.8	Allemagne	NaN
9	Peugeot 306 XS 108	22350.0	1761	74	1100	9.0	France	NaN
26	Subaru Vivio 4WD	44756.5	658	32	740	6.8	Japon	JP

7- Réparation des données manqantes : Imputation par médiane

In [105…
```python
# Créer une copie de df1 pour effectuer l'imputation des valeurs manquantes
df1ImputeMedian=df1.copy()

# Sélectionner uniquement les colonnes de type numérique dans df1
a = df1.select_dtypes('number')

# Imputer les valeurs manquantes avec la médiane de chaque colonne numérique
df1ImputeMedian[a.columns] = a.fillna(a.median())

# Afficher les lignes contenant des valeurs manquantes après l'imputation (NaN_Rows_Ids doit être défini auparavant)
display(df1ImputeMedian.iloc[NaN_Rows_Ids,:])
```

	type	prix	cylindree	puissance	poids	conso	origine	abb
4	Opel Corsa 1.2i Eco	14825.0	1195	33	895	6.8	Allemagne	NaN
9	Peugeot 306 XS 108	22350.0	1761	74	1100	9.0	France	NaN
26	Subaru Vivio 4WD	30165.0	658	32	740	6.8	Japon	JP

8-Comparaison des résultats

In [39]:
```python
# Afficher la valeur réelle dans la colonne 'prix' à l'indice 26 dans le DataFrame df
print("Valeur réelle", df['prix'].loc[26])

# Afficher la valeur imputée par la moyenne dans la colonne 'prix' à l'indice 26 dans le DataFrame df1ImputeMean
print("Imputation par la moyenne", df1ImputeMean['prix'].loc[26])

# Afficher la valeur imputée par la médiane dans la colonne 'prix' à l'indice 26 dans le DataFrame df1ImputeMedian
print("Imputation par la médiane", df1ImputeMedian['prix'].loc[26])
```

```
Valeur réelle 13730
Imputation par la moyenne 44756.5
Imputation par la médiane 30165.0
```

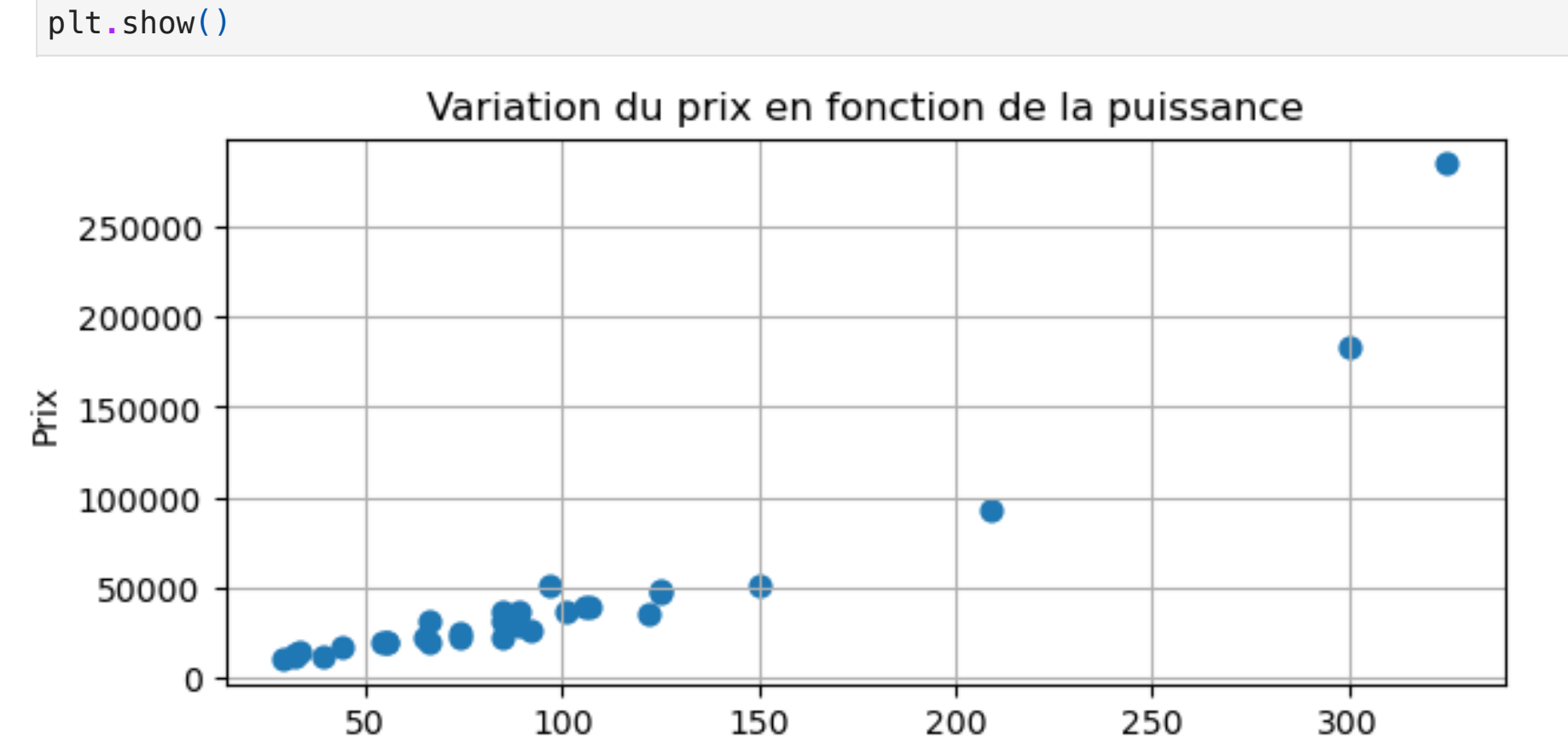
9- Comparaison des résultats

On remarque que les résultats obtenus sont sensiblement différents. Le résultat obtenu par la médiane est plus proche de la valeure réelle.

Ex3 - Traitement des données aberrantes

1- Détection des valeurs Aberrantes : Analyse graphique

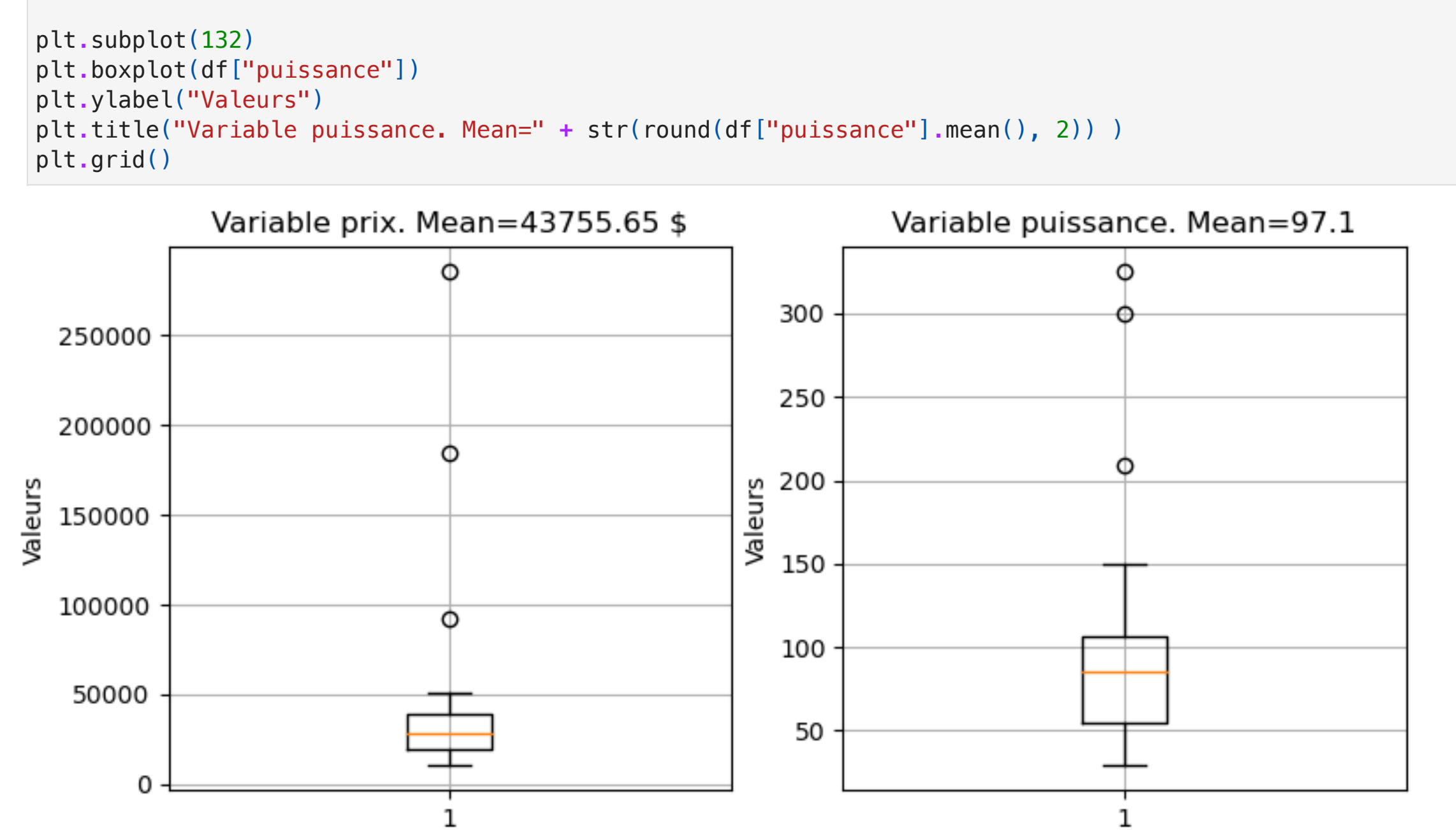
In [107…
```python
plt.rcParams.update({'font.size': 10})
fig = plt.figure(figsize=(7, 3))
plt.scatter(df["puissance"], df["prix"],s=40)
plt.xlabel("Puissance")
plt.ylabel("Prix")
plt.title("Variation du prix en fonction de la puissance")
plt.grid(True)
plt.show()
```



2- Diagramme en boite

In [109…
```python
plt.rcParams.update({'font.size': 10})
fig = plt.figure(figsize=(14, 4))
plt.subplot(131)
plt.boxplot(df["prix"])
plt.ylabel("Valeurs")
plt.title("Variable prix. Mean=" + str(round(df["prix"].mean(), 2))+" $")
plt.grid()

plt.subplot(132)
plt.boxplot(df["puissance"])
plt.ylabel("Valeurs")
plt.title("Variable puissance. Mean=" + str(round(df["puissance"].mean(), 2)) )
plt.grid()
```



4- Détection des valeurs Aberrantes : Score Z

In [115…
```python
# Définir les seuils pour identifier les outliers
threshold1 = 2 # Seuil 1 pour détecter les outliers (écart-type * 2)
threshold2 = 3 # Seuil 2 pour détecter les outliers (écart-type * 3)

# Créer des copies du DataFrame original pour travailler sur les outliers sans modifier df directement
dfOutliers1 = df.copy()
dfOutliers2 = df.copy()

# Afficher la dimension initiale du DataFrame df
print("Dimension de la base de df:",df.shape)

# Calculer les bornes supérieures et inférieures pour le seuil 1
upper1 = dfOutliers1.prix.mean() + threshold1*dfOutliers1.prix.std() # Limite supérieure : moyenne + seuil * écart-type
lower1 = dfOutliers1.prix.mean() - threshold1*dfOutliers1.prix.std() # Limite inférieure : moyenne - seuil * écart-type

# Filtrer les lignes du DataFrame dfOutliers1 pour supprimer les valeurs en dehors des bornes (outliers)
dfOutliers1= dfOutliers1[(dfOutliers1.prix<upper1) & (dfOutliers1.prix>lower1)]

# Afficher la dimension après suppression des outliers pour dfOutliers1
print("Dimension de la base de dfOutliers1:",dfOutliers1.shape)

# Calculer les bornes supérieures et inférieures pour le seuil 2
upper2 = dfOutliers2.prix.mean() + threshold2 * dfOutliers2.prix.std()  # Limite supérieure : moyenne + seuil * écart-type
lower2 = dfOutliers2.prix.mean() - threshold2 * dfOutliers2.prix.std()  # Limite inférieure : moyenne - seuil * écart-type

# Filtrer les lignes du DataFrame dfOutliers2 pour supprimer les valeurs en dehors des bornes (outliers)
dfOutliers2 = dfOutliers2[(dfOutliers2.prix < upper2) & (dfOutliers2.prix > lower2)]

# Afficher la dimension après suppression des outliers pour dfOutliers2
print("Dimension de la base de dfOutliers2:", dfOutliers2.shape)
```

```
Dimension de la base de df: (31, 7)
Dimension de la base de dfOutliers1: (29, 7)
Dimension de la base de dfOutliers2: (30, 7)
```

6- Compraisons des résultats

Nous remarquons la présence de 2 valeurs abérrantes avec $\eta = 3$ et de une seule avec $\eta = 2$. En effet lorsque la valeur de η augmente le nombre de valeur abérrante peut augmenter également.

Ex 4 - Normalisation et standardisation

1 & 2 Normalisation : MinMaxScaler

In [57]:
```python
x = df['conso'].values.reshape(-1,1)
min_max_scaler = preprocessing.MinMaxScaler()
x_minmax = min_max_scaler.fit_transform(x)
print(np.mean(x))
print(np.mean(x_minmax))
print(np.min(x_minmax))
print(np.std(x_minmax))
```

```
9.954838709677418
0.27274607113316784
0.0
0.2250629314789705
```

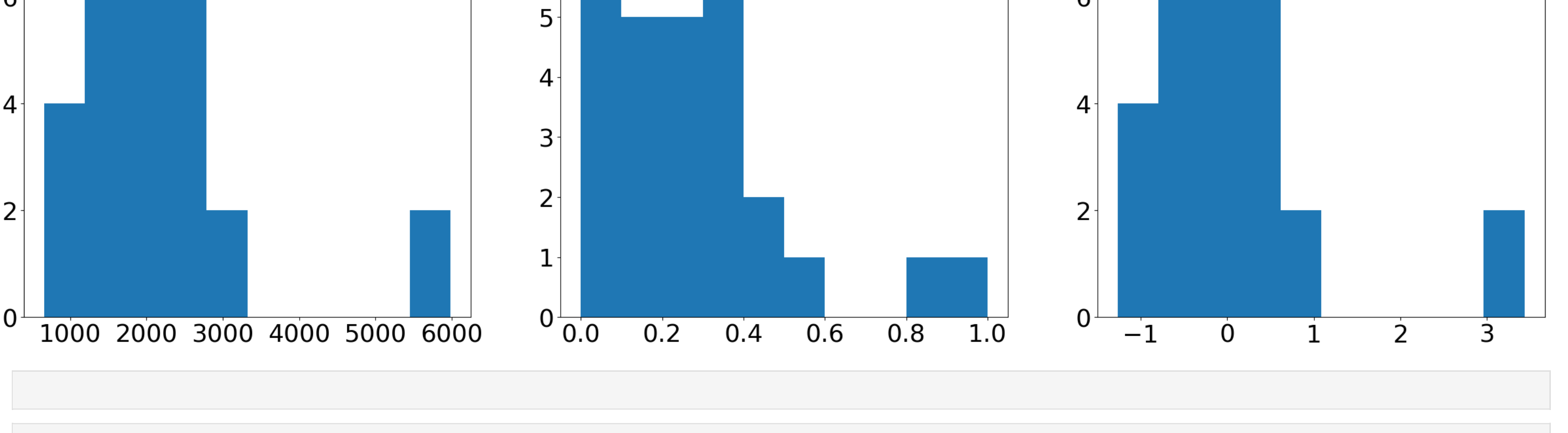
3- Standardisation

In [60]:
```python
x = df['cylindree'].values.reshape(-1,1)
scaler = preprocessing.StandardScaler()
x_standard= scaler.fit_transform(x)
print(np.mean(x))
print(np.mean(x_standard))
print(np.std(x_standard))
```

```
2093.6774193548385
1.3967321922703582e-16
1.0
```

4- Histogrammes

In [63]:
```python
plt.rcParams.update({'font.size': 27})
fig = plt.figure(figsize=(30, 10))
plt.subplot(131)
plt.hist(x)
plt.title("Variable prix")
plt.subplot(132)
plt.hist(x_minmax)
plt.title("Variable prix normalisée")
plt.subplot(133)
plt.hist(x_standard)
plt.title("Variable prix standardisée")
None
```



In [ ]:

In [ ]: