

Jour 43 : MTCARS

La base de données **mtcars** est un jeu de données classique en statistique et en science des données. Elle provient de la revue Motor Trend US de 1974 et contient des informations sur 32 modèles de voitures. Chaque observation correspond à un modèle de voiture ; le jeu de données comprend 11 variables décrivant des caractéristiques techniques et des performances de modèles de voiture. L'objectif de cette analyse est de mieux comprendre les relations entre les **caractéristiques des véhicules** et leur **consommation en carburant (mpg)**.

\$\bigoplus\$ Signification des variables

- mpg : Miles per gallon (consommation en miles par gallon, indicateur d'efficacité énergétique).
- cyl : Nombre de cylindres du moteur (4, 6 ou 8).
- disp : Cylindrée du moteur en pouces cubes.
- hp : Puissance du moteur (horsepower, chevaux-vapeur).
- drat : Ratio du pont arrière (rear axle ratio).
- wt : Poids du véhicule (en 1000 livres).
- qsec : Temps au quart de mile (temps en secondes pour parcourir 1/4 de mile).
- vs : Type de moteur (0 = moteur en V, 1 = moteur en ligne).
- am : Type de transmission (0 = automatique, 1 = manuelle).
- gear : Nombre de vitesses de la boîte de transmission.
- carb : Nombre de carburateurs.

0. Chargement des librairies

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import shapiro
import statsmodels.api as sm

from sklearn.linear_model import LinearRegression
```

1. Chargement de la base données

```
In [2]: data = pd.read_csv("../data/mtcars.csv")

# Les types de données
print(data.dtypes)
```

```
model      object
mpg       float64
cyl        int64
disp      float64
hp         int64
drat      float64
wt        float64
qsec      float64
vs         int64
am         int64
gear      int64
carb      int64
dtype: object
```

```
In [3]: data.head()
```

```
Out[3]:
```

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

```
In [4]: data.describe().transpose()
```

```
Out[4]:
```

	count	mean	std	min	25%	50%	75%	max
mpg	32.0	20.090625	6.026948	10.400	15.42500	19.200	22.80	33.900
cyl	32.0	6.187500	1.785922	4.000	4.00000	6.000	8.00	8.000
disp	32.0	230.721875	123.938694	71.100	120.82500	196.300	326.00	472.000
hp	32.0	146.687500	68.562868	52.000	96.50000	123.000	180.00	335.000
drat	32.0	3.596563	0.534679	2.760	3.08000	3.695	3.92	4.930
wt	32.0	3.217250	0.978457	1.513	2.58125	3.325	3.61	5.424
qsec	32.0	17.848750	1.786943	14.500	16.89250	17.710	18.90	22.900
vs	32.0	0.437500	0.504016	0.000	0.00000	0.000	1.00	1.000
am	32.0	0.406250	0.498991	0.000	0.00000	0.000	1.00	1.000
gear	32.0	3.687500	0.737804	3.000	3.00000	4.000	4.00	5.000
carb	32.0	2.812500	1.615200	1.000	2.00000	2.000	4.00	8.000

```
In [5]: # Nous allons exclure la colonne des noms de voitures pour les analyses
df = data.select_dtypes(include=[np.number])
```

```
In [6]: df.head()
```

Out[6]:

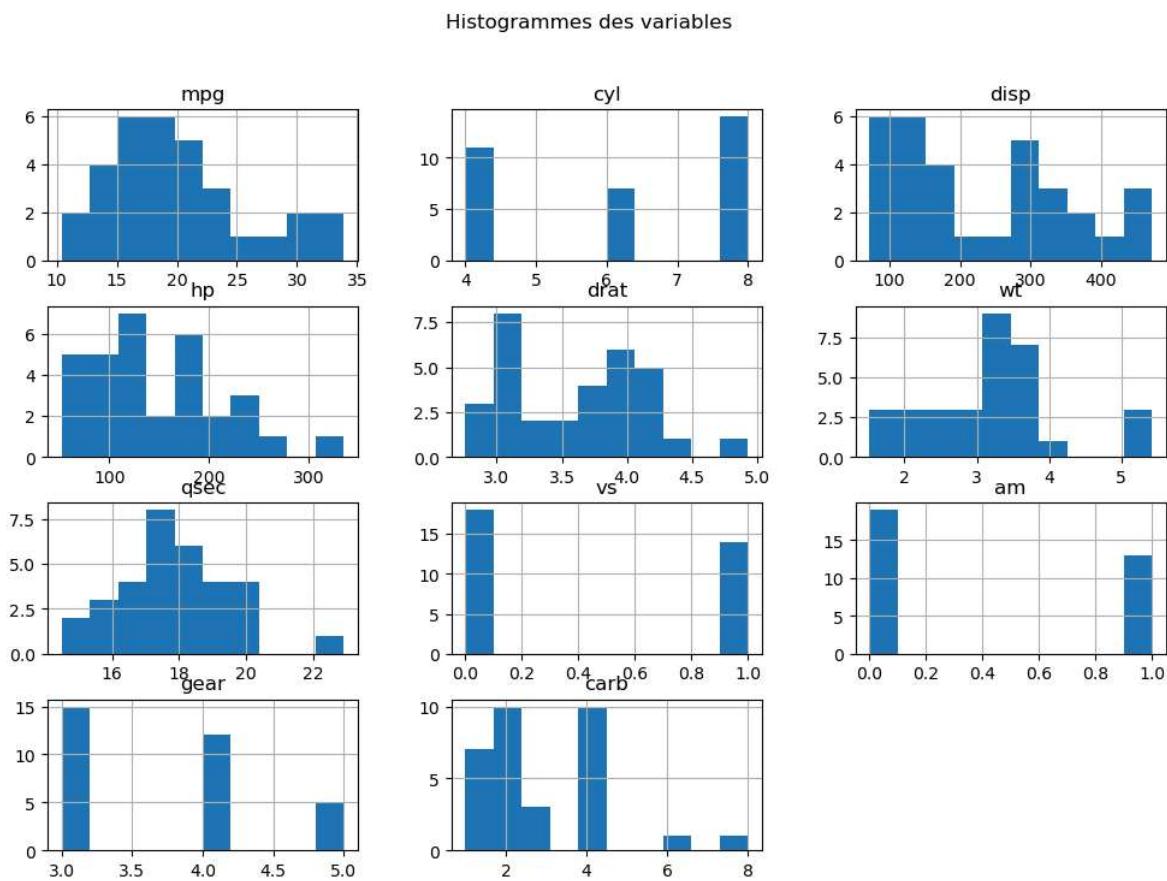
	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

2. Analyse exploratoire des données

2.1. Visualisation des distributions des variables

In [7]:

```
# Histogrammes des variables
df.hist(figsize=(12, 8), bins=10)
plt.suptitle("Histogrammes des variables")
plt.show()
```



La variable **mpg** suit une distribution légèrement asymétrique à droite avec une concentration entre 15 et 25.

Les variables **hp** et **disp** semblent avoir une distribution asymétrique à droite, indiquant qu'il y a quelques voitures très puissantes avec une cylindrée élevée.

Les variables **cyl**, **vs**, **am** et **gear** ont des distributions discrètes, confirmant qu'elles sont à considérer comme des variables catégorielles plutôt que continues.

Les variables **wt** et **qsec** suivent une distribution normale à peu près normale.

In [8]:

```
results = []

# Test de normalité pour chaque variable (colonne)
for col in df.columns:
    stat, p_value = shapiro(df[col])
    results.append({'Variable': col, 'p-valeur': round(p_value, 4)})

# Résultat sous forme de DataFrame
result_df = pd.DataFrame(results)
print(result_df)
```

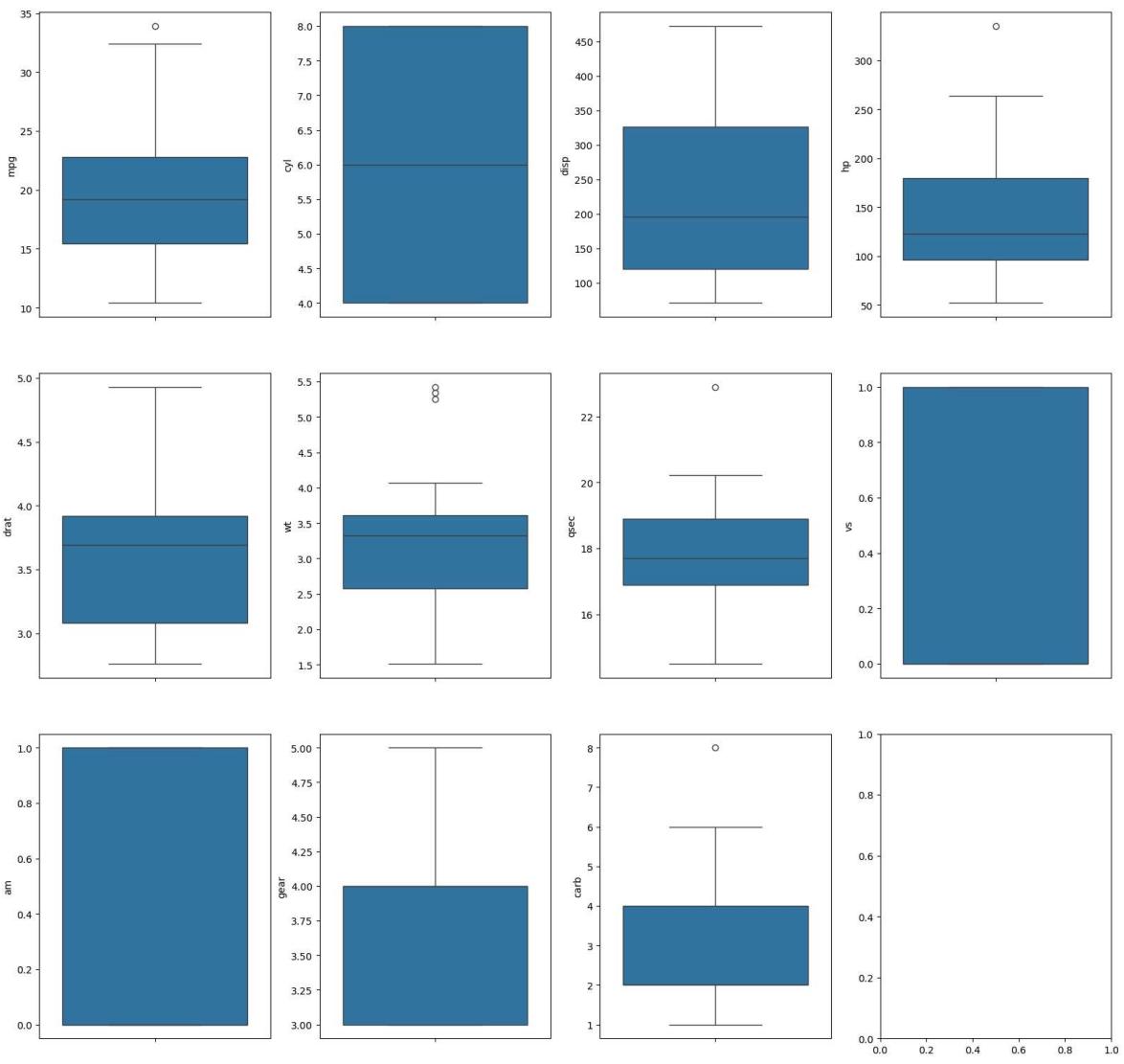
	Variable	p-valeur
0	mpg	0.1229
1	cyl	0.0000
2	disp	0.0208
3	hp	0.0488
4	drat	0.1101
5	wt	0.0927
6	qsec	0.5935
7	vs	0.0000
8	am	0.0000
9	gear	0.0000
10	carb	0.0004

Le test de normalité de Shapiro effectué révèle que seules les variables **cyl**, **disp**, **hp**, **vs**, **am**, **gear** et **carb** suivent une distribution normale.

In [9]:

```
# Création de boxplots
fig, ax = plt.subplots(ncols=4, nrows=3, figsize=(16, 15))
index = 0
ax = ax.flatten()

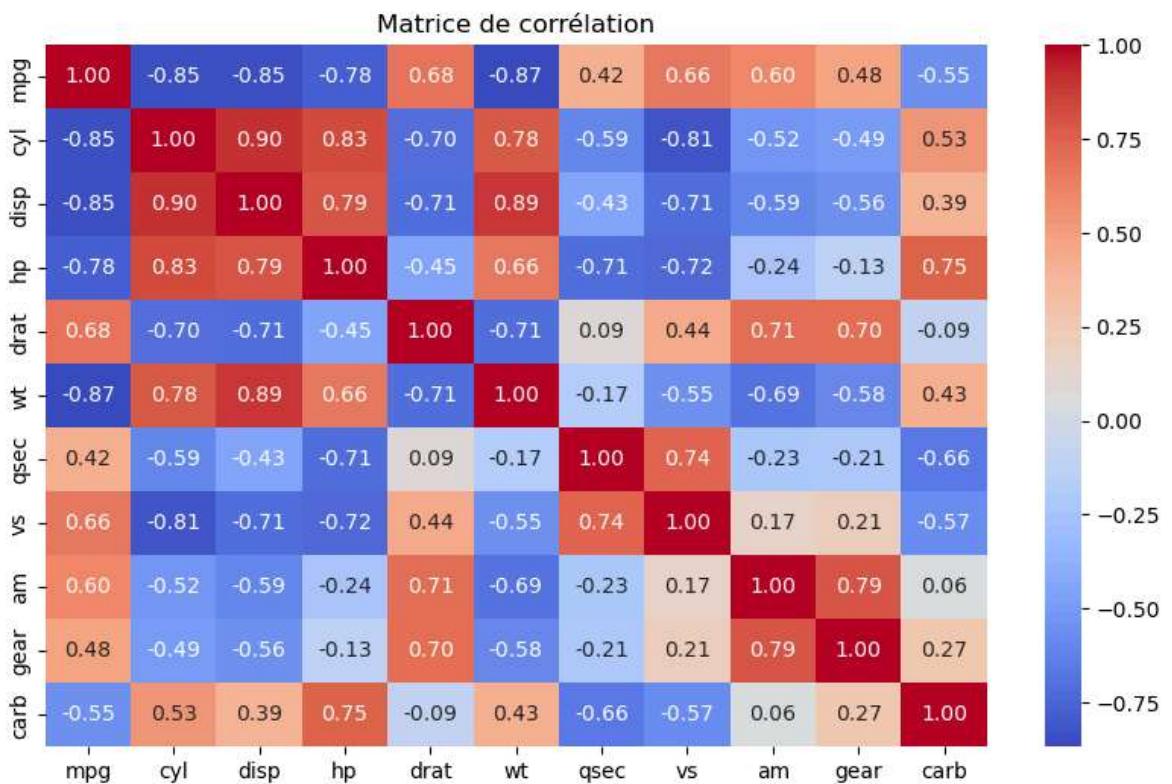
for col, value in df.items():
    sns.boxplot(y=col, data=df, ax=ax[index])
    index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



Les boxplots montrent une grande variabilité entre les données. Certaines variables, comme **hp**, **disp** et **wt**, présentent des valeurs extrêmes, tandis que d'autres, comme **mpg** ou **qsec**, sont plus symétriques. Les variables catégorielles ont peu de modalités. Cette visualisation permet d'identifier rapidement les outliers et la dispersion des données.

2.2. Analyse des corrélations

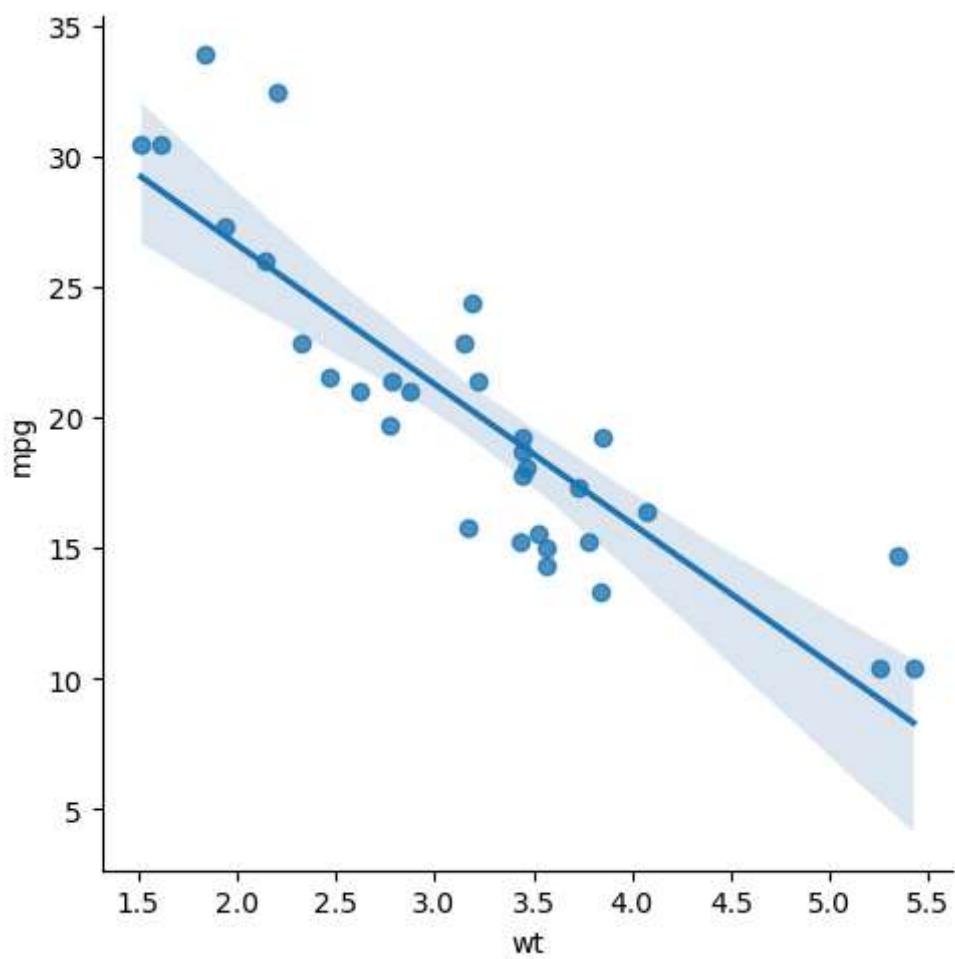
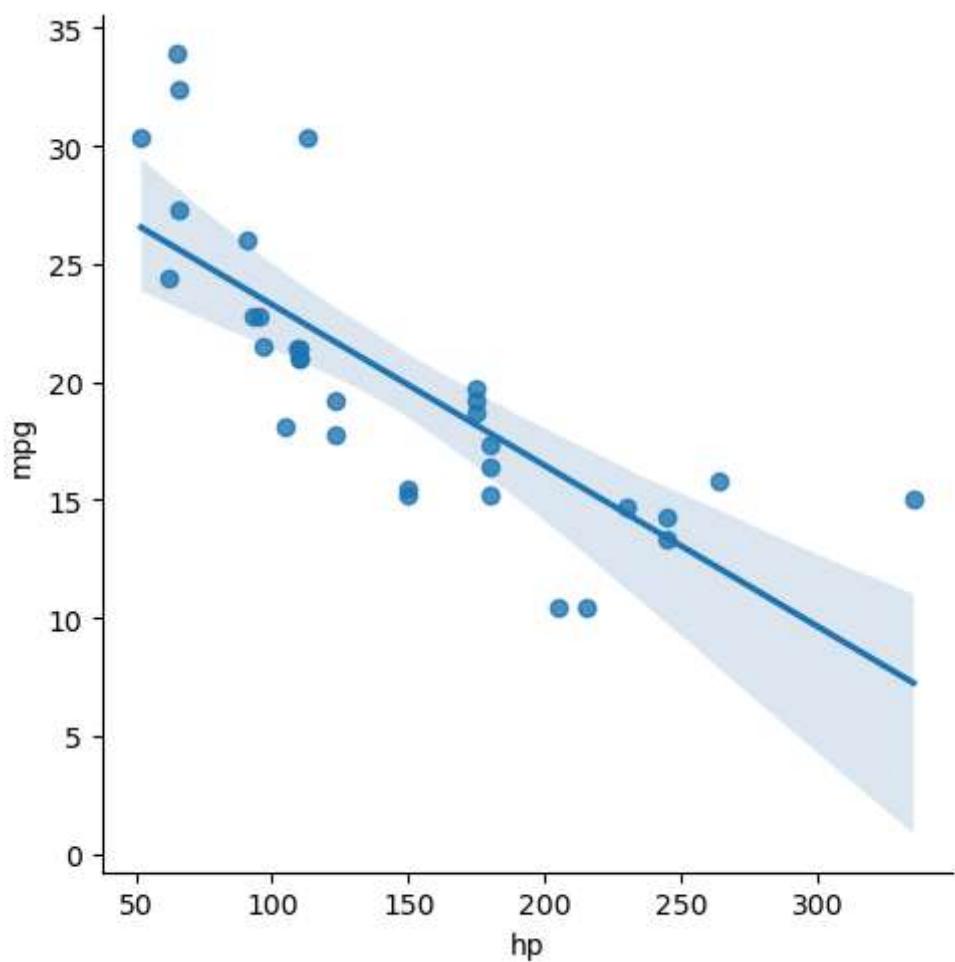
```
In [10]: # Matrice de corrélation
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Matrice de corrélation")
plt.show()
```



2.3. Analyse de quelques relations entre variables

```
In [11]: # Relation entre mpg et d'autres variables
sns.lmplot(x="hp", y="mpg", data=df)
sns.lmplot(x="wt", y="mpg", data=df)
```

```
Out[11]: <seaborn.axisgrid.FacetGrid at 0x29181ee32f0>
```



Plus le poids du véhicule (**wt**) et la puissance du moteur (**hp**) sont importants plus la consommation en miles par gallon (**mpg**) est faible. Il y a donc une corrélation négative entre (**wt**, **hp**) et **mpg**.

3. Régression linéaire multiple

```
In [12]: # Les variables explicatives (X) et la variable cible (y)
X = df.drop(columns=["mpg"])
y = df["mpg"]

# La constante pour l'intercept
X = sm.add_constant(X)

# Ajustement du modèle
model = sm.OLS(y, X).fit()

# Résumé du modèle
print(model.summary())
```

OLS Regression Results						
Dep. Variable:	mpg	R-squared:	0.869			
Model:	OLS	Adj. R-squared:	0.807			
Method:	Least Squares	F-statistic:	13.93			
Date:	Thu, 24 Apr 2025	Prob (F-statistic):	3.79e-07			
Time:	05:35:45	Log-Likelihood:	-69.855			
No. Observations:	32	AIC:	161.7			
Df Residuals:	21	BIC:	177.8			
Df Model:	10					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	12.3034	18.718	0.657	0.518	-26.623	51.229
cyl	-0.1114	1.045	-0.107	0.916	-2.285	2.062
disp	0.0133	0.018	0.747	0.463	-0.024	0.050
hp	-0.0215	0.022	-0.987	0.335	-0.067	0.024
drat	0.7871	1.635	0.481	0.635	-2.614	4.188
wt	-3.7153	1.894	-1.961	0.063	-7.655	0.224
qsec	0.8210	0.731	1.123	0.274	-0.699	2.341
vs	0.3178	2.105	0.151	0.881	-4.059	4.694
am	2.5202	2.057	1.225	0.234	-1.757	6.797
gear	0.6554	1.493	0.439	0.665	-2.450	3.761
carb	-0.1994	0.829	-0.241	0.812	-1.923	1.524
Omnibus:	1.907	Durbin-Watson:	1.861			
Prob(Omnibus):	0.385	Jarque-Bera (JB):	1.747			
Skew:	0.521	Prob(JB):	0.418			
Kurtosis:	2.526	Cond. No.	1.22e+04			

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 1.22e+04. This might indicate that there are strong multicollinearity or other numerical problems.

```
In [13]: # Sélection des variables pertinentes par la Régression pas à pas (Backward Elimination)
import statsmodels.api as sm

def backward_elimination(data, target, significance_level=0.05):
    X = data.copy()
    X = sm.add_constant(X)
    y = target
    model = sm.OLS(y, X).fit()

    while True:
        p_values = model.pvalues.drop("const")
        max_p_value = p_values.max()
        if max_p_value > significance_level:
            feature_to_remove = p_values.idxmax()
            X.drop(columns=[feature_to_remove], inplace=True)
            model = sm.OLS(y, X).fit()
        else:
            break

    return model

# Application de l'élimination pas à pas
final_model = backward_elimination(df.drop(columns=["mpg"]), df["mpg"])
print(final_model.summary())
```

OLS Regression Results

Dep. Variable:	mpg	R-squared:	0.850			
Model:	OLS	Adj. R-squared:	0.834			
Method:	Least Squares	F-statistic:	52.75			
Date:	Thu, 24 Apr 2025	Prob (F-statistic):	1.21e-11			
Time:	05:35:45	Log-Likelihood:	-72.060			
No. Observations:	32	AIC:	152.1			
Df Residuals:	28	BIC:	158.0			
Df Model:	3					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	9.6178	6.960	1.382	0.178	-4.638	23.874
wt	-3.9165	0.711	-5.507	0.000	-5.373	-2.460
qsec	1.2259	0.289	4.247	0.000	0.635	1.817
am	2.9358	1.411	2.081	0.047	0.046	5.826
Omnibus:	2.574	Durbin-Watson:			1.714	
Prob(Omnibus):	0.276	Jarque-Bera (JB):			2.213	
Skew:	0.540	Prob(JB):			0.331	
Kurtosis:	2.297	Cond. No.			296.	

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

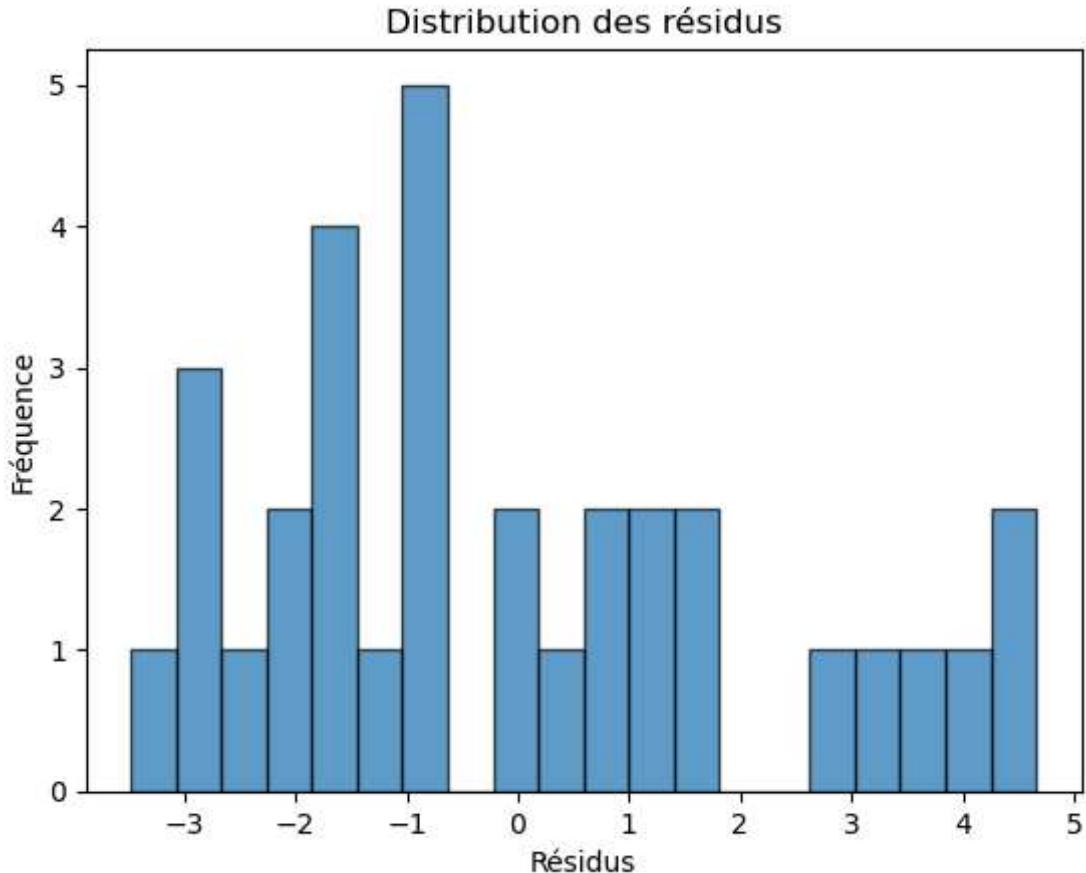
4. Vérification des hypothèses

4.1. Test de Normalité des erreurs

```
In [14]: residuals = final_model.resid
from scipy.stats import shapiro

# Histogramme des résidus
plt.hist(residuals, bins=20, edgecolor="black", alpha=0.7)
plt.title("Distribution des résidus")
plt.xlabel("Résidus")
plt.ylabel("Fréquence")
plt.show()

# Test de normalité de Shapiro-Wilk
print("Test de Shapiro-Wilk:", shapiro(residuals))
```



Test de Shapiro-Wilk: ShapiroResult(statistic=0.9410960255551162, pvalue=0.08042769363003151)

La p-valeur du test de **Shapiro-Wilk** (0,0804) est > 0.05, alors les résidus suivent une loi normale.

4.2. Homoscédasticité (test de Breusch-Pagan)

```
In [15]: from statsmodels.stats.diagnostic import het_breusvhagan

bp_test = het_breusvhagan(residuals, final_model.model.exog)
print(f"Test de Breusch-Pagan:, p-value={bp_test[1]:.3f}")
```

Test de Breusch-Pagan:, p-value=0.103

Les résidus sont homoscédastiques (variance constante) car la p-valeur est > 0,05.

5. Prédiction et évaluation du modèle

5.1. Séparation des données et prédiction

```
In [16]: from sklearn.model_selection import train_test_split

# Définition des variables
X = df.drop(columns=["mpg"])
y = df["mpg"]

# Une constante pour l'intercept
X = sm.add_constant(X)

# Séparation des données en 80% entraînement et 20% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Ajustement du modèle sur l'ensemble d'entraînement
model_train = sm.OLS(y_train, X_train).fit()

# Prédictions sur l'ensemble de test
y_pred = model_train.predict(X_test)

print(model_train.summary())
```

OLS Regression Results

Dep. Variable:	mpg	R-squared:	0.875
Model:	OLS	Adj. R-squared:	0.786
Method:	Least Squares	F-statistic:	9.804
Date:	Thu, 24 Apr 2025	Prob (F-statistic):	9.89e-05
Time:	05:35:45	Log-Likelihood:	-53.488
No. Observations:	25	AIC:	129.0
Df Residuals:	14	BIC:	142.4
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
const	-5.5937	30.491	-0.183	0.857	-70.991	59.804
cyl	-0.4762	1.183	-0.403	0.693	-3.014	2.061
disp	0.0183	0.021	0.878	0.395	-0.026	0.063
hp	-0.0095	0.031	-0.304	0.766	-0.077	0.058
drat	1.0021	2.209	0.454	0.657	-3.736	5.741
wt	-4.8620	2.600	-1.870	0.083	-10.439	0.715
qsec	1.8410	1.507	1.222	0.242	-1.391	5.073
vs	-1.9372	3.077	-0.630	0.539	-8.537	4.662
am	1.0766	2.483	0.434	0.671	-4.249	6.402
gear	1.3271	1.745	0.760	0.460	-2.416	5.070
carb	0.1471	1.146	0.128	0.900	-2.310	2.605

Omnibus:	2.280	Durbin-Watson:	2.374
Prob(Omnibus):	0.320	Jarque-Bera (JB):	1.965
Skew:	0.641	Prob(JB):	0.374
Kurtosis:	2.508	Cond. No.	1.66e+04

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 1.66e+04. This might indicate that there are strong multicollinearity or other numerical problems.

5.2. Evaluation du modèle

```
In [17]: from sklearn.metrics import mean_squared_error

# Erreur quadratique moyenne (MSE - Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)

# Calcul du Racine de l'erreur quadratique moyenne (RMSE - Root Mean Squared Err
rmse = np.sqrt(mse)
print("RMSE : {:.2f}")
```

RMSE : 3.18

Le **RMSE** de **3.18** obtenu signifie que les prédictions du modèle peuvent en moyenne s'écartez de **±3 mpg** par rapport aux vraies valeurs.