Visualisation des Données

In [1]:
```
import pandas as pd
pd.plotting.register_matplotlib_converters()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
print("Setup Complete")
```

```
Setup Complete
```

Ce code prépare l'environnement pour visualiser des données à l'aide des bibliothèques pandas, matplotlib, et seaborn. Voici une explication détaillée de ce que fait chaque ligne :

```
1 import pandas as pd importe pandas pour manipuler et analyser des données via des DataFrames.
2 pd.plotting.register_matplotlib_converters() enregistre les convertisseurs pour que matplotlib pu
isse afficher correctement les données temporelles.
3 import matplotlib.pyplot as plt importe pyplot pour créer des graphiques.
4 %matplotlib inline est une commande Jupyter qui permet d'afficher les graphiques dans le noteboo
k.
5 import seaborn as sns importe seaborn, qui facilite la création de visualisations statistiques.
```

Nous travaillerons avec un fichier CSV contenant les classements historiques en football pour des pays.

Chargement des données

In [2]:
```
# Path of the file to read
foot_filepath = "./foot.csv"

# Read the file into a variable fifa_data
foot_data = pd.read_csv(foot_filepath, index_col="Date", parse_dates=True)
```

In [3]:
```
# Print the first 5 rows of the data
foot_data.head()
```

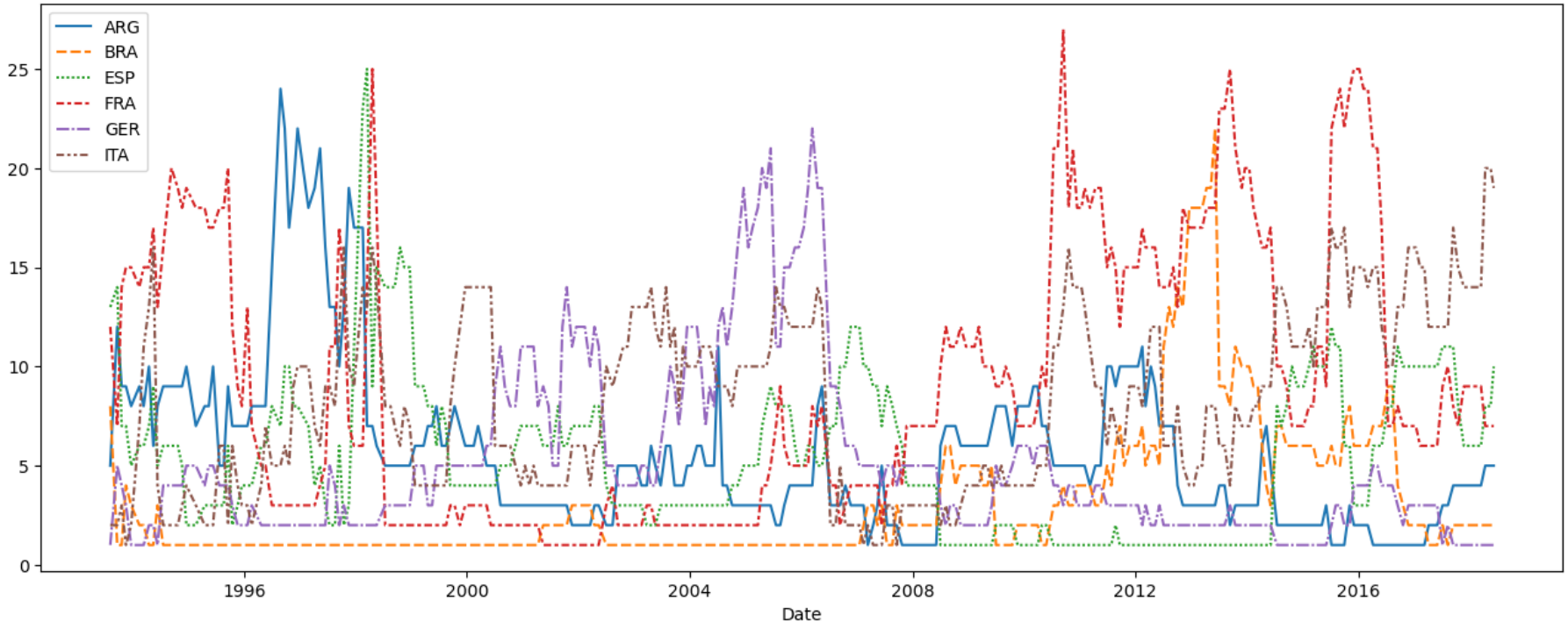
Out[3]:

Date	ARG	BRA	ESP	FRA	GER	ITA
1993-08-08	5.0	8.0	13.0	12.0	1.0	2.0
1993-09-23	12.0	1.0	14.0	7.0	5.0	2.0
1993-10-22	9.0	1.0	7.0	14.0	4.0	3.0
1993-11-19	9.0	4.0	7.0	15.0	3.0	1.0
1993-12-23	8.0	3.0	5.0	15.0	1.0	2.0

Visualisation

In [4]:
```
# Set the width and height of the figure
plt.figure(figsize=(16,6))

# Line chart showing how FIFA rankings evolved over time
sns.lineplot(data=foot_data)
```

Out[4]: <Axes: xlabel='Date'>



Le code configure et crée un graphique linéaire avec matplotlib et seaborn :

1 plt.figure(figsize=(16,6)) définit la taille du graphique à 16 pouces de large et 6 pouces de hau t, idéal pour les séries temporelles.
2 sns.lineplot(data=fifa_data) crée un graphique linéaire à partir des données contenues dans le Da taFrame foot_data. Seaborn trace les lignes pour chaque colonne de données, avec l'axe x représenta nt l'index (probablement les dates), et l'axe y les valeurs des colonnes.

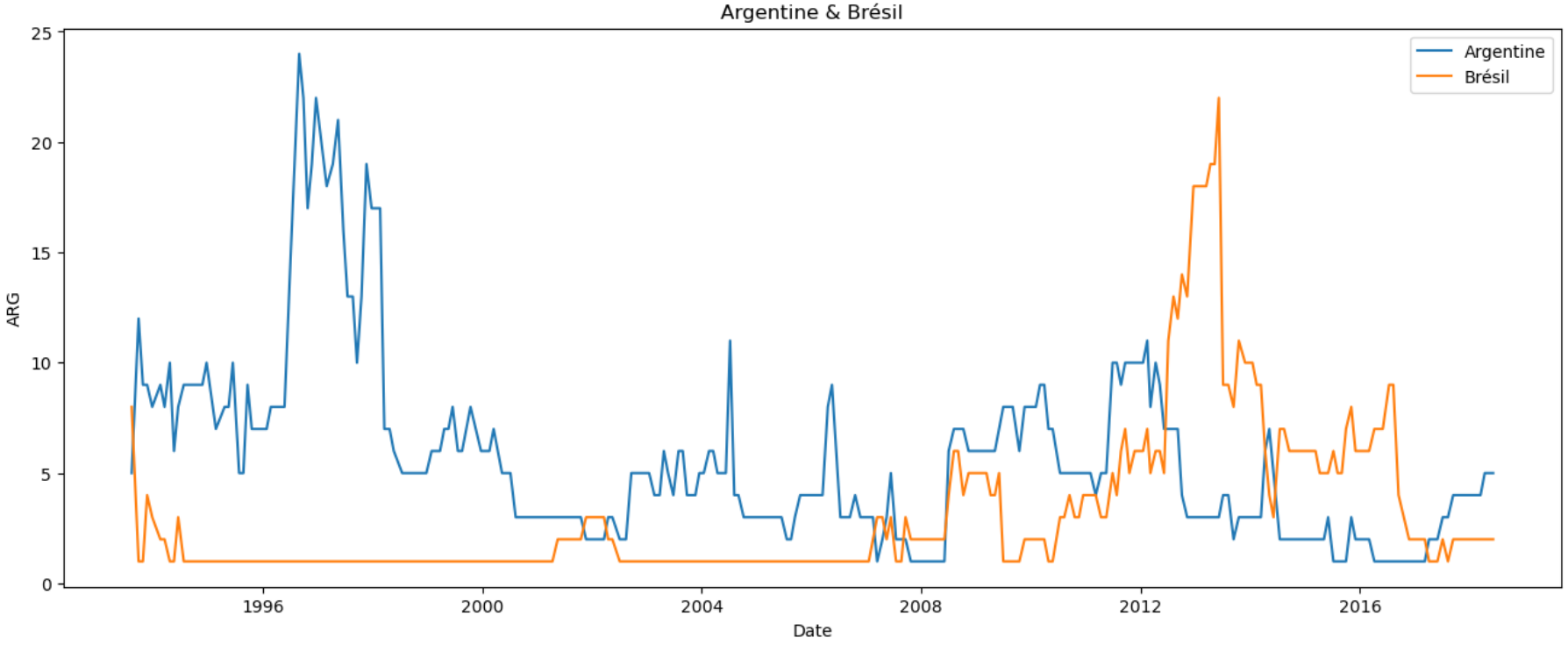
Représenter un sous-ensemble des données.

In [5]:
```
list(foot_data.columns)
```

Out[5]:
```
['ARG', 'BRA', 'ESP', 'FRA', 'GER', 'ITA']
```

In [6]:
```
# Set the width and height of the figure
plt.figure(figsize=(16,6))

# Add title
plt.title("Argentine & Brésil")

# Line chart showing daily global streams of 'Argentine'
sns.lineplot(data=foot_data['ARG'], label="Argentine")

# Line chart showing daily global streams of 'Brésil'
sns.lineplot(data=foot_data['BRA'], label="Brésil")

# Add label for horizontal axis
plt.xlabel("Date")
```

Out[6]:
```
Text(0.5, 0, 'Date')
```



Ce code crée un graphique linéaire comparant deux colonnes de données pour l'Argentine et le Brésil à l'aide de Matplotlib et Seaborn.

```
   1 plt.figure(figsize=(16,6)) définit les dimensions du graphique.
   2 plt.title("Argentine & Brésil") ajoute un titre indiquant que le graphique concerne ces deux p
ays.
   3 sns.lineplot(data=foot_data['ARG'], label="Argentine") trace une courbe pour les données de l'
Argentine à partir de la colonne 'ARG'.
   4 sns.lineplot(data=foot_data['BRA'], label="Brésil") trace une courbe pour les données du Brési
l à partir de la colonne 'BRA'.
   5 plt.xlabel("Date") étiquette l'axe des x avec "Date", suggérant une série temporelle.
```

Le graphique final affiche l'évolution des données pour les deux pays avec des étiquettes et une légende.

Revoir aussi les notions de Bar chart, Heat map, Scatter plots, Histograms and Density plot

Ingénierie des caractéristiques (feature engineering)

L'ingénierie des caractéristiques vise à rendre les données mieux adaptées au problème à résoudre. L'ingénierie des caractéristiques peut améliorer la performance prédictive, réduire les besoins en calcul ou en données, et rendre les résultats plus interprétables.

Pour être utile, une caractéristique doit avoir un lien que le modèle peut apprendre.

Cet exemple montre comment l'ajout de caractéristiques synthétiques peut améliorer les performances d'un modèle de forêt aléatoire. L'ensemble de données comprend des formulations de béton et leur résistance à la compression, une mesure de la charge que le béton peut supporter. La tâche consiste à prédire cette résistance à partir de la formulation du béton.

In [7]:
```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score

#df = pd.read_csv(".concdata.csv")
df = pd.read_excel('concdata.xls', engine='xlrd')

df.head()
```

Out[7]:

	Cement (component 1) (kg in a m^3 mixture)	Blast Furnace Slag (component 2) (kg in a m^3 mixture)	Fly Ash (component 3) (kg in a m^3 mixture)	Water (component 4) (kg in a m^3 mixture)	Superplasticizer (component 5)(kg in a m^3 mixture)	Coarse Aggregate (component 6) (kg in a m^3 mixture)	Fine Aggregate (component 7) (kg in a m^3 mixture)	Age (day)	Concrete compressive strength(MPa, megapascals)
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.986111
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.887366
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.269535
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.052780
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.296075

In [8]:
```
df.columns
```

Out[8]:
```
Index(['Cement (component 1)(kg in a m^3 mixture)',
       'Blast Furnace Slag (component 2)(kg in a m^3 mixture)',
       'Fly Ash (component 3)(kg in a m^3 mixture)',
       'Water  (component 4)(kg in a m^3 mixture)',
       'Superplasticizer (component 5)(kg in a m^3 mixture)',
       'Coarse Aggregate  (component 6)(kg in a m^3 mixture)',
       'Fine Aggregate (component 7)(kg in a m^3 mixture)', 'Age (day)',
       'Concrete compressive strength(MPa, megapascals) '],
      dtype='object')
```

On examine comment l'ajout de caractéristiques synthétiques aux ingrédients du béton peut améliorer les performances du modèle. Avant cela, une référence est établie en entraînant le modèle sur les données non modifiées. Cette pratique permet de vérifier si les nouvelles caractéristiques ajoutent de la valeur, ou s'il faut les écarter et tester d'autres options.

In [9]:
```
ccs = 'Concrete compressive strength(MPa, megapascals) '


X = df.copy()
y = X.pop(ccs)

# Train and score baseline model
baseline = RandomForestRegressor(criterion="absolute_error", random_state=0)
baseline_score = cross_val_score(
    baseline, X, y, cv=5, scoring="neg_mean_absolute_error"
)
baseline_score = -1 * baseline_score.mean()

print(f"MAE Baseline Score: {baseline_score:.4}")
```

```
MAE Baseline Score: 8.397
```

Trois nouvelles caractéristiques de ratio sont ajoutées à l'ensemble de données.

In [10]:
```
X = df.copy()
y = X.pop(ccs)

fa = "Fine Aggregate (component 7)(kg in a m^3 mixture)"
ca = "Coarse Aggregate  (component 6)(kg in a m^3 mixture)"
ce = 'Cement (component 1)(kg in a m^3 mixture)'
wa = 'Water  (component 4)(kg in a m^3 mixture)'

# Create synthetic features
X["FCRatio"] = X[fa] / X[ca]
X["AggCmtRatio"] = (X[ca] + X[fa]) / X[ce]
X["WtrCmtRatio"] = X[wa] / X[ce]

# Train and score model on dataset with additional ratio features
model = RandomForestRegressor(criterion="absolute_error", random_state=0)
score = cross_val_score(
    model, X, y, cv=5, scoring="neg_mean_absolute_error"
)
score = -1 * score.mean()

print(f"MAE Score with Ratio Features: {score:.4}")
```

```
MAE Score with Ratio Features: 8.01
```

MI Scores

Rencontrer un nouvel ensemble de données peut être intimidant, avec de nombreuses caractéristiques sans description. Un bon début est de créer un classement en utilisant une métrique d'utilité comme l'information mutuelle (MI), qui mesure la relation entre une caractéristique et la cible. Contrairement à la corrélation, MI détecte tout type de relation, pas seulement linéaire. Elle est facile à utiliser, résistante au surapprentissage et utile en début de développement des caractéristiques. MI mesure comment la connaissance d'une caractéristique réduit l'incertitude sur la cible, mais ne capture pas les interactions entre caractéristiques. Sa véritable utilité dépend du modèle utilisé.

L'objectif de l'exemple ci-après est de prédire le prix d'une voiture à partir de caractéristiques comme la marque ou la puissance. L'exemple classe les caractéristiques à l'aide de l'information mutuelle et analyse les résultats par visualisation.

In [11]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

# Utiliser directement le style de Seaborn
sns.set(style="whitegrid")

df = pd.read_csv("./autodata.csv")
df.head()
```

Out[11]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.4	10.0
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.4	8.0

5 rows × 26 columns

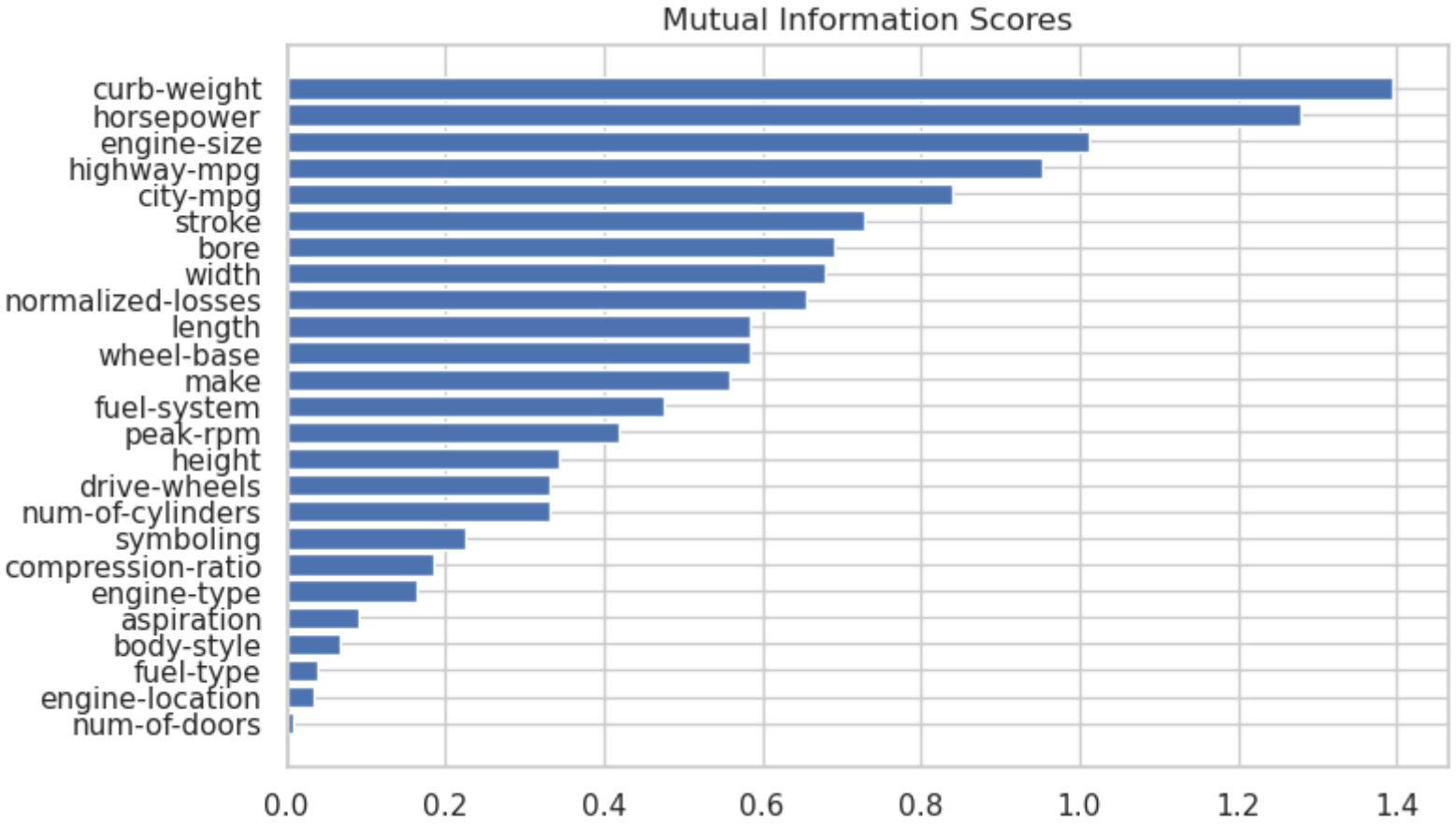
L'algorithme scikit-learn pour l'information mutuelle traite différemment les caractéristiques discrètes et continues. Vous devez indiquer lesquelles sont discrètes, comme les catégoriques, qui nécessitent un encodage de labels. Scikit-learn propose deux métriques : mutual_info_regression pour les cibles réelles, comme le prix ici, et mutual_info_classif pour les cibles catégorielles. La cellule suivante calcule et présente les scores MI dans un dataframe.

In [12]:

```python
from sklearn.feature_selection import mutual_info_regression

# Copy dataframe and extract target
X = df.copy()
y = X.pop("price")

# Handle missing or non-numeric values in y
# Replace '?' with NaN and drop those rows
y = y.replace('?', np.nan)
y = pd.to_numeric(y, errors='coerce')
y = y.dropna()

# Ensure X is consistent with y after dropping NaNs
X = X.loc[y.index]

# Label encoding for categorical features in X
for colname in X.select_dtypes("object"):
    X[colname], _ = X[colname].factorize()

# Ensure there are no NaNs in X
X = X.fillna(0)  # You can also use other imputation strategies depending on your needs

# All discrete features should now have integer dtypes (double-check this before using MI!)
discrete_features = X.dtypes == int

# Function to calculate mutual information scores
def make_mi_scores(X, y, discrete_features):
    mi_scores = mutual_info_regression(X, y, discrete_features=discrete_features)
    mi_scores = pd.Series(mi_scores, name="MI Scores", index=X.columns)
    mi_scores = mi_scores.sort_values(ascending=False)
    return mi_scores

# Calculate MI scores
mi_scores = make_mi_scores(X, y, discrete_features)

print(mi_scores)
```

```
curb-weight          1.394527
horsepower           1.278299
engine-size          1.011880
highway-mpg          0.953165
city-mpg             0.839496
stroke               0.727421
bore                 0.691395
width                0.679997
normalized-losses    0.655361
length               0.583614
wheel-base           0.583312
make                 0.558149
fuel-system          0.474674
peak-rpm             0.418110
height               0.343562
drive-wheels         0.332446
num-of-cylinders     0.331078
symboling            0.225273
compression-ratio    0.184804
engine-type          0.163244
aspiration           0.091703
body-style           0.067794
fuel-type            0.038389
engine-location      0.034534
num-of-doors         0.006943
Name: MI Scores, dtype: float64
```

In [13]:
```
def plot_mi_scores(scores):
    scores = scores.sort_values(ascending=True)
    width = np.arange(len(scores))
    ticks = list(scores.index)
    plt.barh(width, scores)
    plt.yticks(width, ticks)
    plt.title("Mutual Information Scores")


plt.figure(dpi=100, figsize=(8, 5))
plot_mi_scores(mi_scores)
```



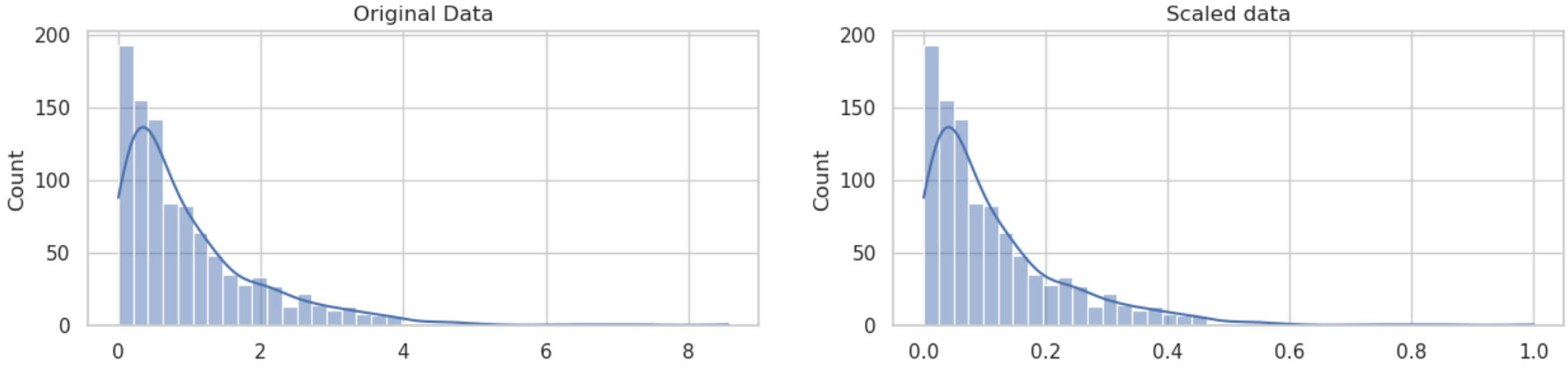
La caractéristique curb_weight, avec un score élevé, montre une forte relation avec price, la cible.

Mise à l'échelle (scaling) et la normalisation

Les 2 transforment les valeurs numériques, mais avec des objectifs différents : la mise à l'échelle modifie l'échelle des données (ex. 0-1), tandis que la normalisation modifie la forme de leur distribution. La mise à l'échelle est utile pour des méthodes comme SVM ou KNN, où les écarts entre les points sont importants. Par exemple, comparer des prix en Yen et en Dollars sans mise à l'échelle donnerait une importance égale à des différences non comparables.

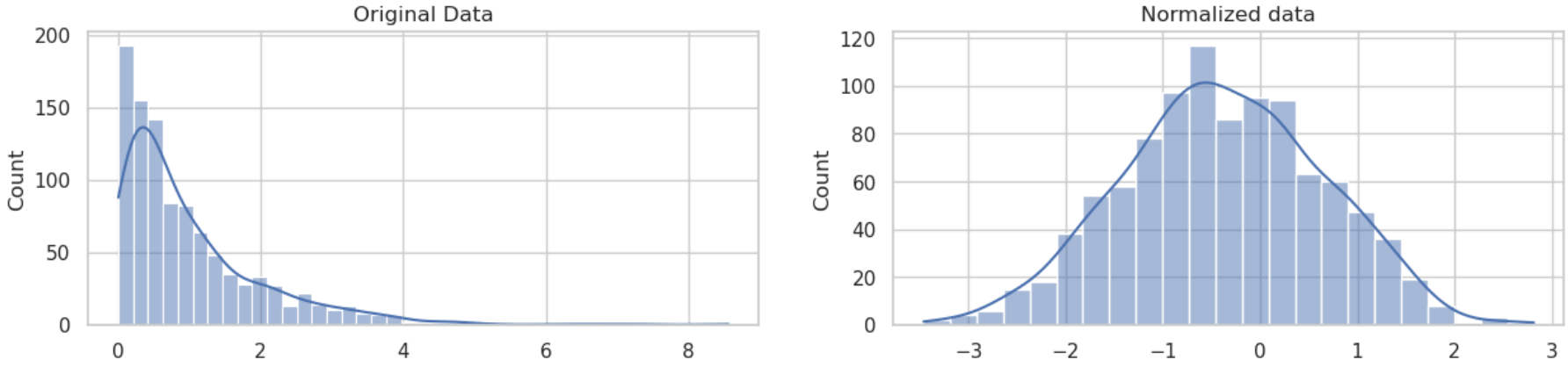
In [14]:
```
# modules we'll use
import pandas as pd
import numpy as np

# for Box-Cox Transformation
from scipy import stats

# for min_max scaling
from mlxtend.preprocessing import minmax_scaling

# plotting modules
import seaborn as sns
import matplotlib.pyplot as plt

# set seed for reproducibility
np.random.seed(0)
```

In [15]:
```
# generate 1000 data points randomly drawn from an exponential distribution
original_data = np.random.exponential(size=1000)

# mix-max scale the data between 0 and 1
scaled_data = minmax_scaling(original_data, columns=[0])

# plot both together to compare
fig, ax = plt.subplots(1, 2, figsize=(15, 3))
sns.histplot(original_data, ax=ax[0], kde=True, legend=False)
ax[0].set_title("Original Data")
sns.histplot(scaled_data, ax=ax[1], kde=True, legend=False)
ax[1].set_title("Scaled data")
plt.show()
```



La normalisation transforme les observations pour les faire correspondre à une distribution normale, c'est une distribution statistique spécifique où un nombre à peu près égal d'observations se situe au-dessus et en dessous de la moyenne. La distribution normale est également connue sous le nom de distribution gaussienne.Elle est utilisée pour des techniques qui supposent des données normalement distribuées, comme l'analyse discriminante linéaire (LDA) et le naïf bayésien gaussien. La méthode utilisée ici est la transformation de Box-Cox.

In [16]:
```
# normalize the exponential data with boxcox
normalized_data = stats.boxcox(original_data)

# plot both together to compare
fig, ax=plt.subplots(1, 2, figsize=(15, 3))
sns.histplot(original_data, ax=ax[0], kde=True, legend=False)
ax[0].set_title("Original Data")
sns.histplot(normalized_data[0], ax=ax[1], kde=True, legend=False)
ax[1].set_title("Normalized data")
plt.show()
```



Clustering

Ceci utilise des algorithmes d'apprentissage non supervisé. Ces algorithmes n'utilisent pas de cible mais cherchent à découvrir des propriétés des données, souvent en tant que technique de "découverte de caractéristiques".

Le clustering consiste à regrouper des points de données en fonction de leur similarité. En ingénierie des caractéristiques, on pourrait utiliser cette technique pour identifier des segments de marché ou des zones géographiques avec des modèles météorologiques similaires. L'ajout de labels de clusters peut aider les modèles à gérer des relations complexes d'espace ou de proximité.

Lorsqu'il est appliqué à une seule caractéristique, le clustering agit comme une transformation classique de "binning" (ou de "discrétisation"). Pour plusieurs caractéristiques, c'est similaire à un "binning multi-dimensionnel". L'ajout des labels de clusters dans un DataFrame peut ressembler à une table listant les coordonnées géographiques avec leurs clusters assignés.

Clustering k-means

c'est un algorithme de regroupement mesurant la similarité à l'aide de la distance euclidienne. Il crée des clusters en plaçant des points appelés centroids dans l'espace des caractéristiques, chaque point de données étant assigné au cluster du centroid le plus proche. Le "k" de k-means correspond au nombre de centroids que vous définissez.

Le processus fonctionne en deux étapes : assigner les points aux centroids les plus proches et déplacer les centroids pour minimiser la distance à leurs points. Le processus itère jusqu'à ce que les centroids se stabilisent ou qu'un nombre maximum d'itérations (max_iter) soit atteint. Comme l'initialisation aléatoire des centroids peut conduire à de mauvais regroupements, l'algorithme répète le processus plusieurs fois (n_init) pour trouver le meilleur clustering.

L'algorithme k-means produit une tessellation de Voronoï, montrant comment les données futures seront assignées aux clusters. Pour déterminer le meilleur nombre de clusters (n_clusters), il est recommandé d'effectuer une validation croisée ou de traiter ce choix comme un hyperparamètre à ajuster.

Exemple - Logement

> En tant que caractéristiques spatiales, les 'latitude' et 'longitude' d'un logement sont des candidats naturels pour le clustering k-means. Dans cet exemple, nous allons regrouper ces variables avec 'median_income' (revenu médian) afin de créer des segments économiques dans différentes régions.

In [17]:
```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.cluster import KMeans

# Utiliser directement le style de Seaborn
sns.set(style="whitegrid")

plt.rc("figure", autolayout=True)
plt.rc(
    "axes",
    labelweight="bold",
    labelsize="large",
    titleweight="bold",
    titlesize=14,
    titlepad=10,
)

df = pd.read_csv("./housing.csv")
X = df.loc[:, ["median_income", "latitude", "longitude"]]
X.head()
```

Out[17]:

	median_income	latitude	longitude
0	8.3252	37.88	-122.23
1	8.3014	37.86	-122.22
2	7.2574	37.85	-122.24
3	5.6431	37.85	-122.25
4	3.8462	37.85	-122.25

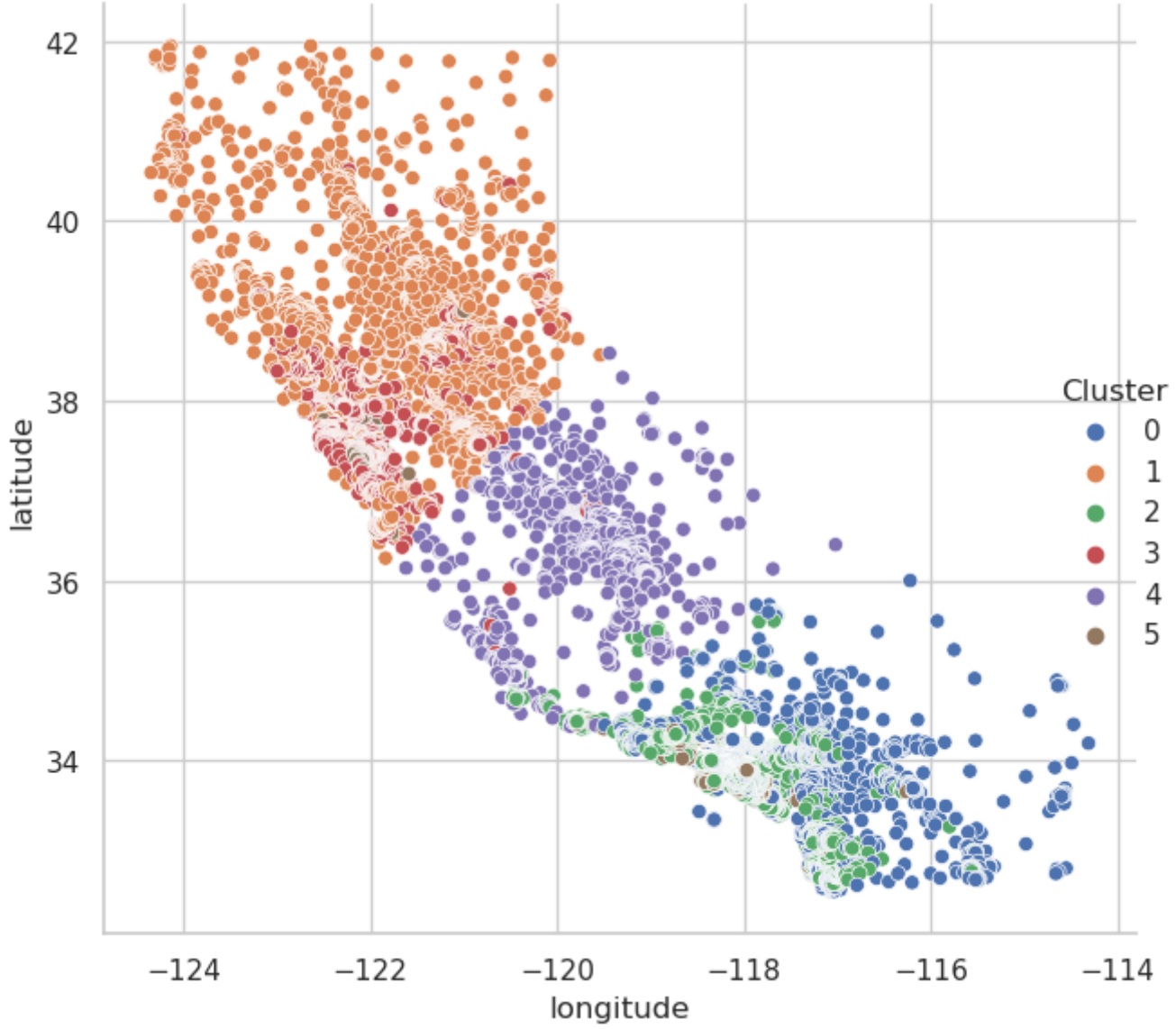
Ici les caratéristiques sont quasiment sur la même échelle, il n'est pas utile de normaliser ou de mettre à l'échelle

In [18]:
```
# Create cluster feature
kmeans = KMeans(n_clusters=6, n_init=10)  # Définir explicitement n_init à 10
X["Cluster"] = kmeans.fit_predict(X)
X["Cluster"] = X["Cluster"].astype("category")

X.head()
```

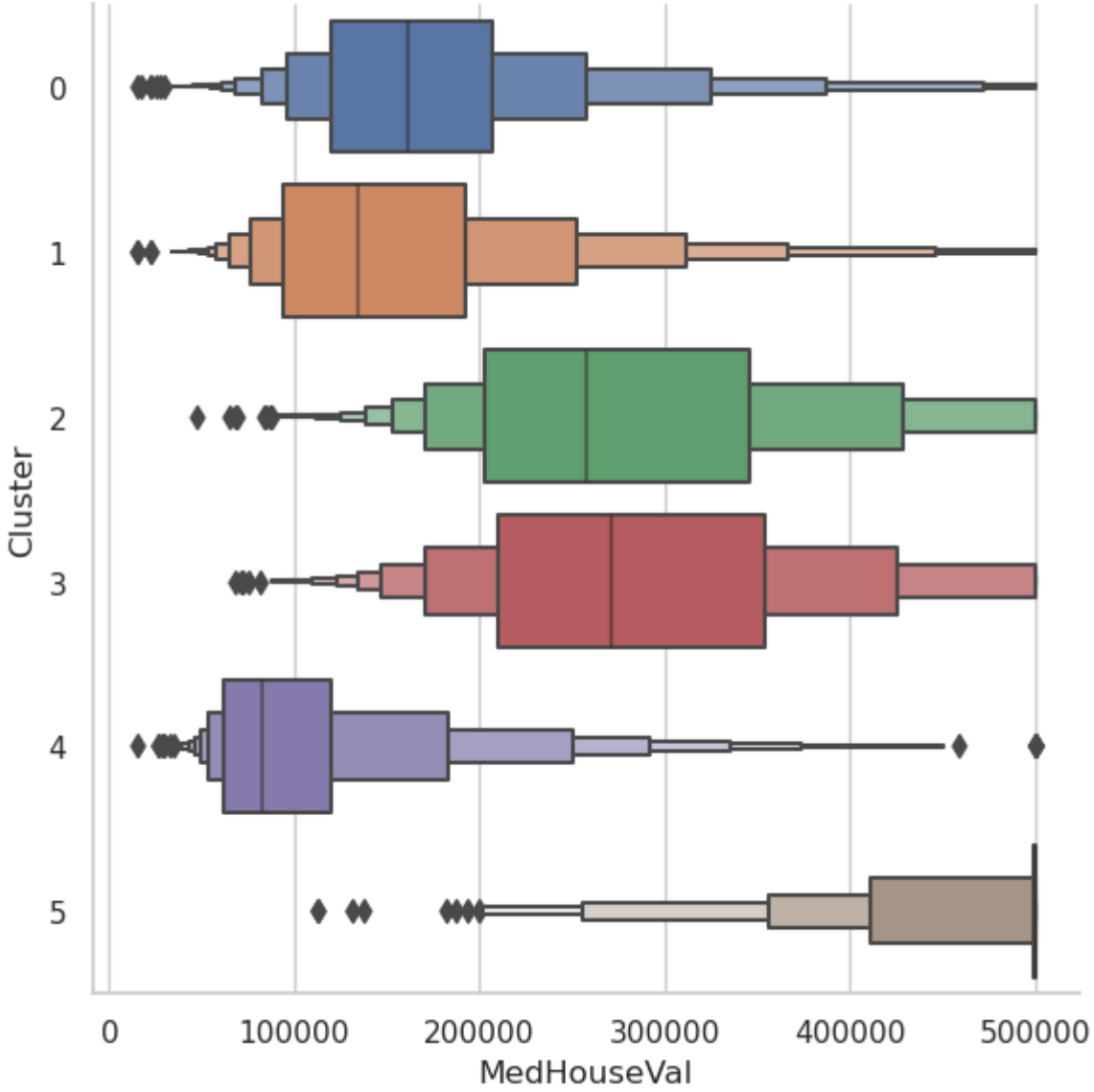
Out[18]:

	median_income	latitude	longitude	Cluster
0	8.3252	37.88	-122.23	3
1	8.3014	37.86	-122.22	3
2	7.2574	37.85	-122.24	3
3	5.6431	37.85	-122.25	3
4	3.8462	37.85	-122.25	1

In [19]:
```
sns.relplot(
    x="longitude", y="latitude", hue="Cluster", data=X, height=6,
);
```



In [20]:
```
X["MedHouseVal"] = df["median_house_value"]
sns.catplot(x="MedHouseVal", y="Cluster", data=X, kind="boxen", height=6);
```



Mixtures de Gaussiennes (Gaussian Mixture)

Le modèle de mélange gaussien (GMM) modélise des données complexes en supposant qu'elles proviennent de plusieurs sous-populations, chacune suivant une distribution gaussienne. Contrairement au k-means, le GMM attribue une probabilité d'appartenance à plusieurs clusters, permettant de gérer des groupes de données qui se chevauchent ou ont des formes complexes. Fonctionnement :

```
Chaque composante est définie par une moyenne (μ) et une matrice de covariance (Σ), représentant le
centre et la forme du cluster.
La pondération (π) indique la proportion de données dans chaque gaussienne.
L'algorithme Expectation-Maximization (EM) ajuste les paramètres du modèle en deux étapes : calcul
des probabilités (E-step) et ajustement des paramètres (M-step), jusqu'à convergence.
```

Applications :

```
Clustering : Pour des groupes qui se chevauchent ou ont des formes non circulaires.
Modélisation : Pour des distributions complexes, comme la génération de données synthétiques.
Détection d'anomalies : En identifiant les points ayant une faible probabilité d'appartenance aux c
lusters.
```

Avantages :

```
Plus flexible que k-means, gère des clusters de différentes formes.
Attribue des probabilités d'appartenance, ce qui permet une classification souple.
Peut traiter des clusters qui se chevauchent.
```

Inconvénients :

```
Complexité : Nécessite de spécifier des paramètres comme le nombre de clusters.
Sensible aux anomalies et aux minima locaux, nécessitant parfois plusieurs initialisations.
```

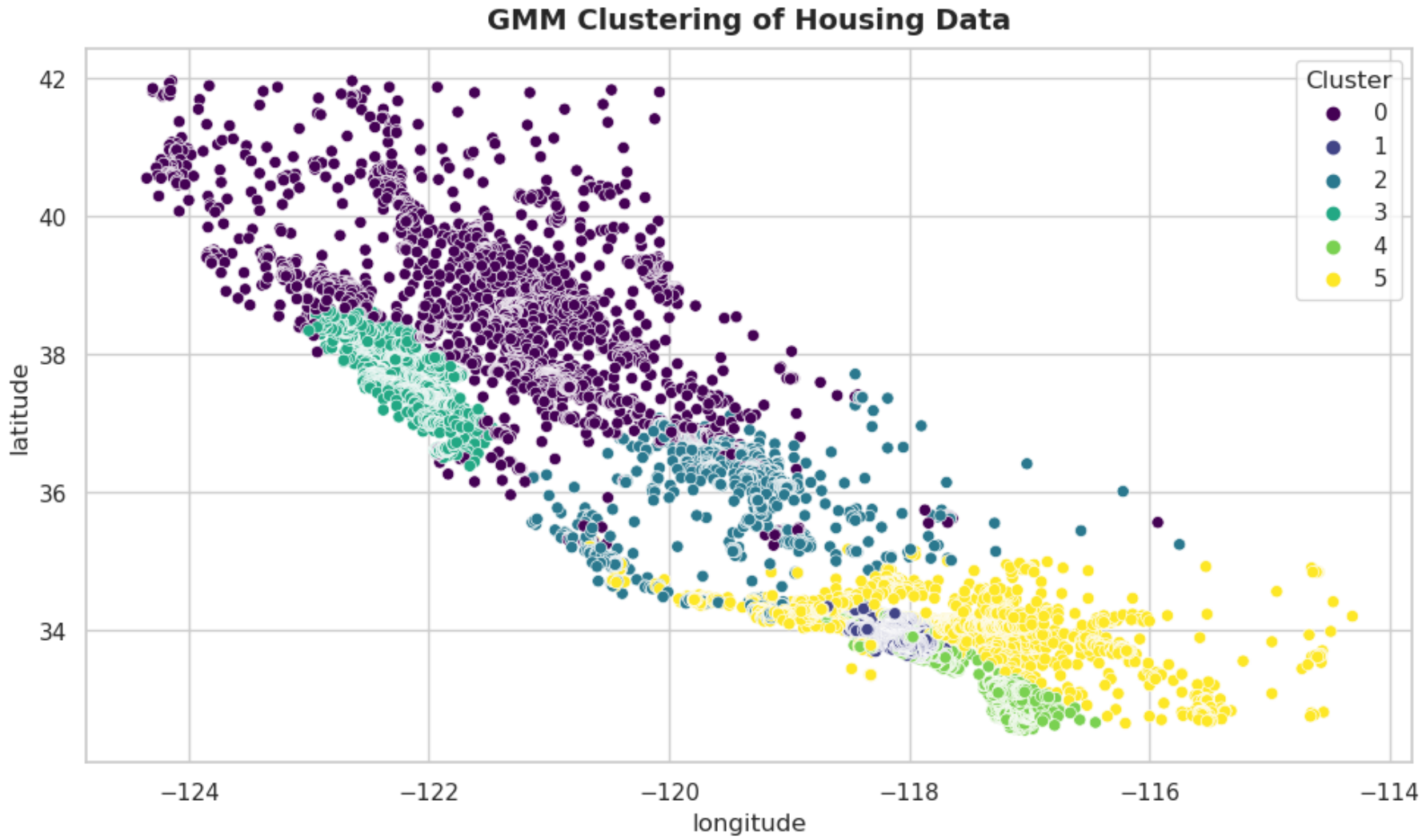
Le GMM est donc un outil puissant pour le clustering, mais il demande une bonne maîtrise pour bien paramétrer et éviter certains pièges.

Notre exemple avec Gaussian Mixture au lieu de K-means

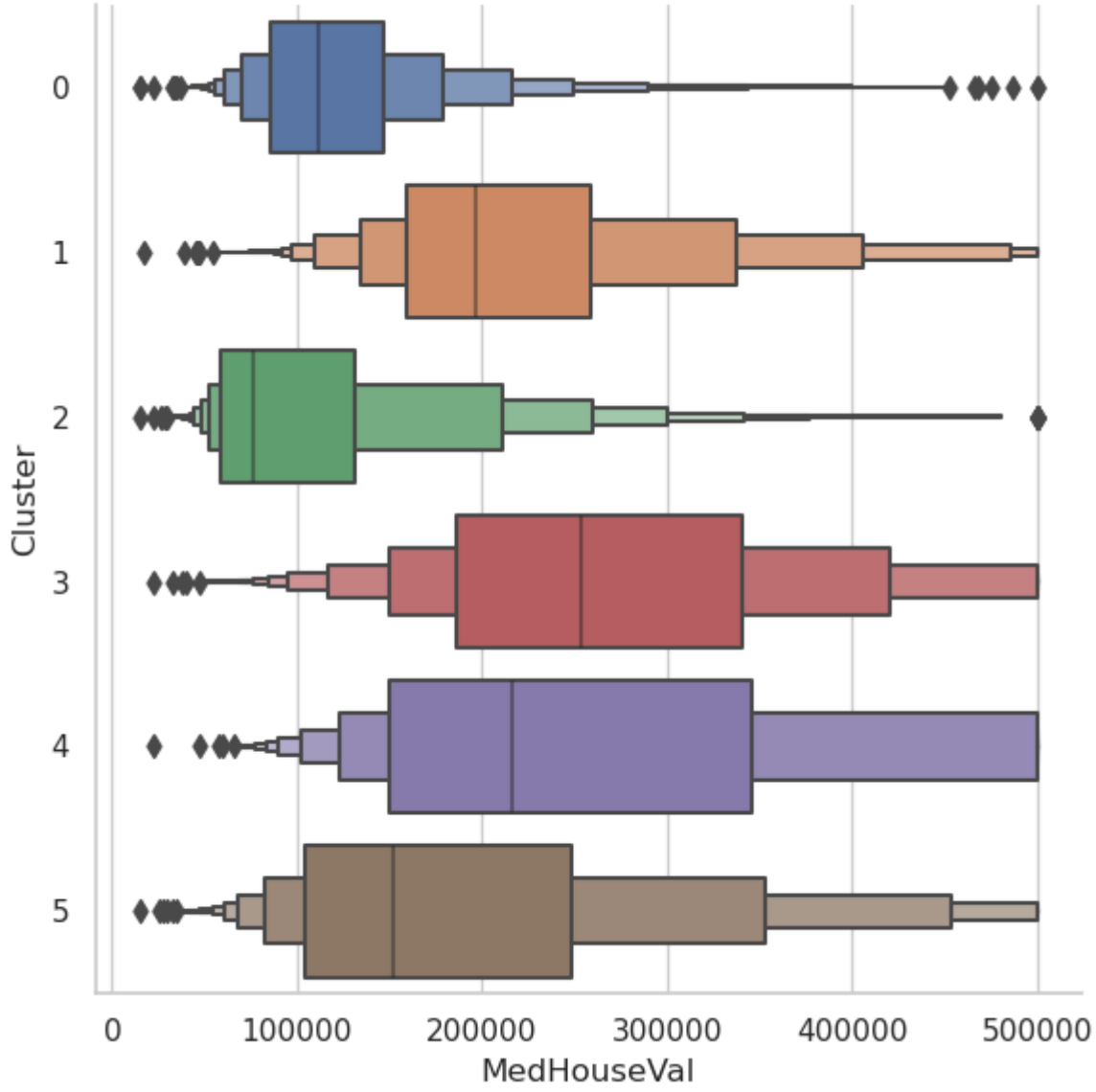
In [21]:

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.mixture import GaussianMixture  # Import GaussianMixture instead of KMeans

# Utiliser directement le style de Seaborn
sns.set(style="whitegrid")

plt.rc("figure", autolayout=True)
plt.rc(
    "axes",
    labelweight="bold",
    labelsize="large",
    titleweight="bold",
    titlesize=14,
    titlepad=10,
)

# Load data
df = pd.read_csv("./housing.csv")
X = df.loc[:, ["median_income", "latitude", "longitude"]]
X.head()

# Use Gaussian Mixture Model for clustering
gmm = GaussianMixture(n_components=6, random_state=42)  # Use 6 components (clusters)
gmm.fit(X)

# Predict the clusters
labels = gmm.predict(X)

# Add cluster labels to the dataframe
df["Cluster"] = labels
X["Cluster"] = df["Cluster"]
# Ensure 'Cluster' is treated as a categorical variable
X["Cluster"] = X["Cluster"].astype("category")

# Plot the clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x="longitude", y="latitude", hue="Cluster", palette="viridis", legend="full")
plt.title("GMM Clustering of Housing Data")
```

```
plt.show()
```


GMM Clustering of Housing Data
Cluster
0
1
2
3
4
5
latitude
42
40
38
36
34
longitude
−124
−122
−120
−118
−116
−114

In [22]:
```
X["MedHouseVal"] = df["median_house_value"]
# Plot using Seaborn's catplot
sns.catplot(x="MedHouseVal", y="Cluster", data=X, kind="boxen", height=6)
plt.show()
```



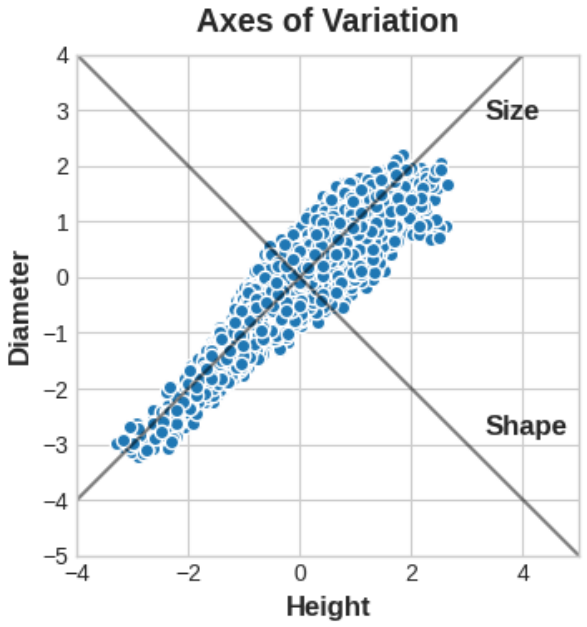
PCA

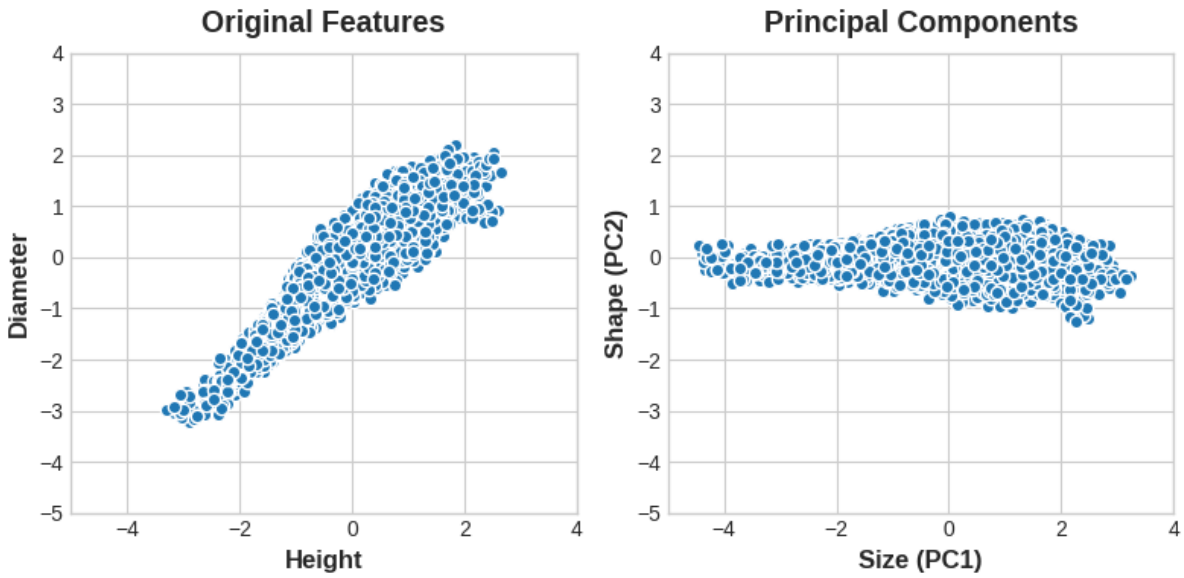
Ici, nous abordons l'analyse en composantes principales (PCA), qui partitionne la variation des données. PCA permet de découvrir des relations importantes et de créer des caractéristiques plus informatives. Les nouvelles caractéristiques, appelées composantes principales, sont des combinaisons linéaires des caractéristiques d'origine.

PCA révèle les axes de variation dans les données, tels que les composantes Taille et Forme. Les poids, appelés "loadings", indiquent la contribution de chaque caractéristique à ces axes. PCA indique aussi la variance expliquée par chaque composante, mais une composante avec plus de variance n'est pas toujours un meilleur prédicteur.

Il existe deux utilisations principales pour PCA en ingénierie des caractéristiques : descriptive (analyser la variation pour créer des caractéristiques) et utiliser directement les composantes comme caractéristiques, utile pour la réduction de dimension, la détection d'anomalies, la réduction du bruit, et la dé-corrélation.

Meilleures pratiques pour PCA : PCA s'applique uniquement aux caractéristiques numériques, est sensible à l'échelle (d'où l'importance de la standardisation), et il est recommandé de gérer les outliers avant son utilisation.





Les nouvelles caractéristiques que la PCA construit sont en fait des combinaisons linéaires (somme pondérée) des caractéristiques d'origine :

```
df["Taille"] = 0.707 * X["Hauteur"] + 0.707 * X["Diamètre"]

df["Forme"] = 0.707 * X["Hauteur"] - 0.707 * X["Diamètre"]
```

Ces nouvelles caractéristiques sont appelées les composantes principales des données. Les poids eux-mêmes sont appelés des loadings. Il y aura autant de composantes principales qu'il y a de caractéristiques dans l'ensemble de données d'origine. Les loadings d'une composante indiquent quelles variations elle exprime à travers les signes et les magnitudes :

	Taille (CP1)	Forme (CP2)
Hauteur	0.707	0.707
Diamètre	0.707	-0.707

Dans cet exemple ci-après, nous appliquons la PCA à notre ensemble de données auto comme technique descriptive pour identifier de nouvelles caractéristiques.

In [23]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from IPython.display import display
from sklearn.feature_selection import mutual_info_regression

# Utiliser directement le style de Seaborn
sns.set(style="whitegrid")

plt.rc("figure", autolayout=True)
plt.rc(
    "axes",
    labelweight="bold",
    labelsize="large",
    titleweight="bold",
    titlesize=14,
    titlepad=10,
)

def plot_variance(pca, width=8, dpi=100):
    # Create figure
    fig, axs = plt.subplots(1, 2)
    n = pca.n_components_
    grid = np.arange(1, n + 1)
    # Explained variance
    evr = pca.explained_variance_ratio_
    axs[0].bar(grid, evr)
    axs[0].set(
        xlabel="Component", title="% Explained Variance", ylim=(0.0, 1.0)
    )
    # Cumulative Variance
    cv = np.cumsum(evr)
    axs[1].plot(np.r_[0, grid], np.r_[0, cv], "o-")
    axs[1].set(
        xlabel="Component", title="% Cumulative Variance", ylim=(0.0, 1.0)
    )
    # Set up figure
    fig.set(figwidth=8, dpi=100)
    return axs
```

```
def make_mi_scores(X, y, discrete_features):
    mi_scores = mutual_info_regression(X, y, discrete_features=discrete_features)
    mi_scores = pd.Series(mi_scores, name="MI Scores", index=X.columns)
    mi_scores = mi_scores.sort_values(ascending=False)
    return mi_scores

df = pd.read_csv("./autodata.csv")
```

In [24]:
```
df.columns
```

Out[24]:
```
Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-location',
       'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-type',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price'],
      dtype='object')
```

Quatre caractéristiques ont été sélectionnées pour couvrir diverses propriétés, chacune ayant un score MI élevé avec la cible (prix). Les données seront standardisées car ces caractéristiques ne sont pas sur la même échelle.

In [25]:
```
features = ["highway-mpg", "engine-size", "horsepower", "curb-weight"]

X = df.copy()
y = X.pop('price')
X = X.loc[:, features]

# Standardize
X_scaled = (X - X.mean(axis=0, numeric_only=True)) / X.std(axis=0, numeric_only=True)
```

In [26]:
```
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer

# Imputer les valeurs manquantes (remplacer par la moyenne)
imputer = SimpleImputer(strategy='mean')  # Vous pouvez aussi utiliser 'median' ou 'most_frequent'
X_imputed = imputer.fit_transform(X_scaled)

# Create principal components
pca = PCA()
X_pca = pca.fit_transform(X_imputed)

# Convert to dataframe
component_names = [f"PC{i+1}" for i in range(X_pca.shape[1])]
X_pca = pd.DataFrame(X_pca, columns=component_names)

X_pca.head()
```

Out[26]:

	PC1	PC2	PC3
0	0.337894	-0.372875	-0.221911
1	0.337894	-0.372875	-0.221911
2	1.038570	-0.091846	-0.125912
3	-0.437361	-0.402075	-0.141693
4	1.143997	-0.779971	-0.106238

Seules 3 caractéristiques sur les 3 ont finalement de l'importance

In [27]:
```
#loadings = pd.DataFrame(
#    pca.components_.T,  # transpose the matrix of loadings
#    columns=component_names,  # so the columns are the principal components
#    index=X.columns,  # and the rows are the original features
#)
#loadings

# Vérifier les dimensions des composantes principales
print(pca.components_.shape)  # (3, 3)

# Créer les loadings avec le bon index et les bonnes colonnes
loadings = pd.DataFrame(
    pca.components_.T,  # transpose the matrix of loadings
    columns=component_names[:3],  # Take only the 3 first component names
    index=X.columns[:3]  # Adjust the index to match the 3 components
)

# Afficher les loadings
loadings
```

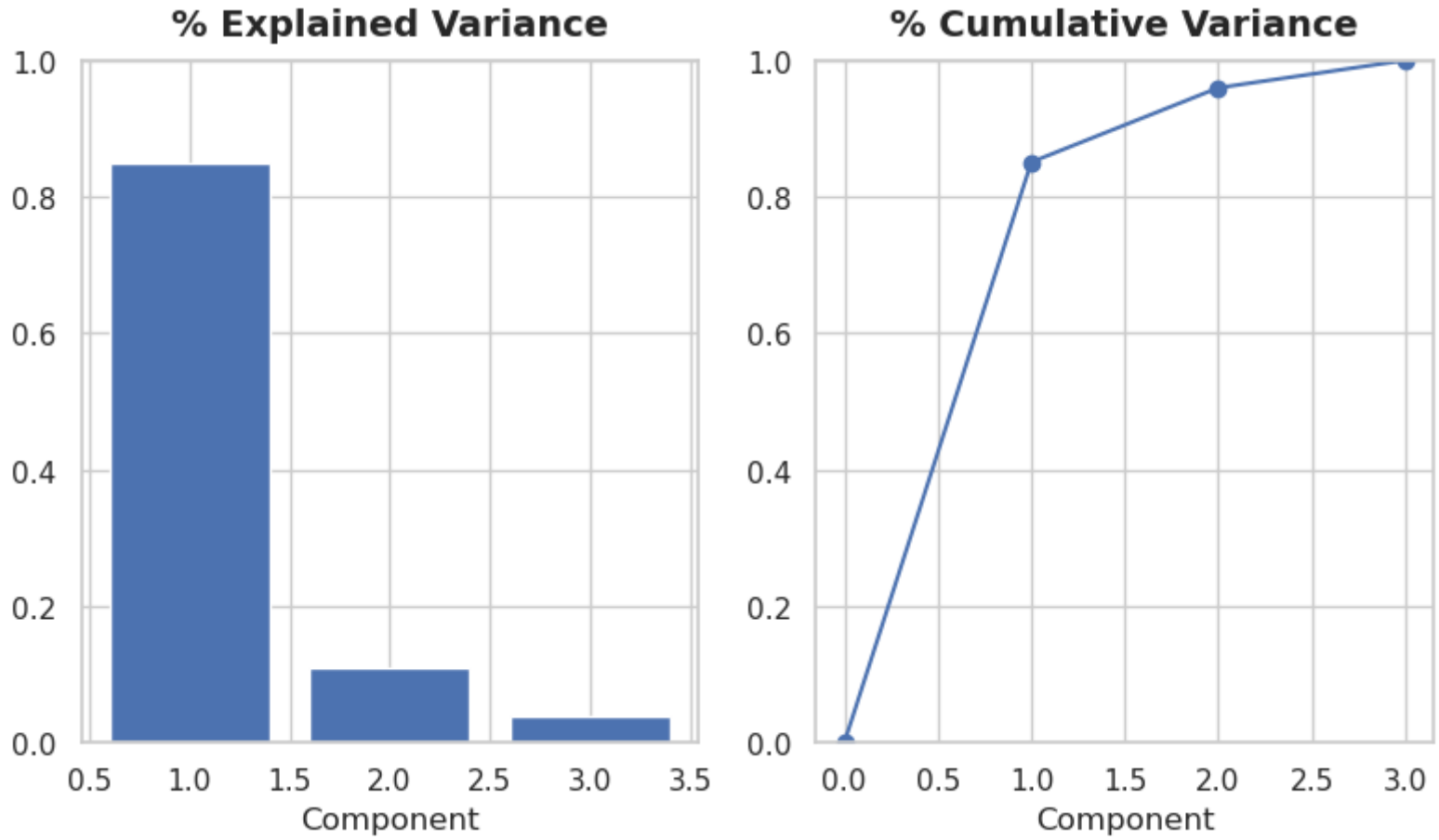
```
(3, 3)
```

Out[27]:

	PC1	PC2	PC3
highway-mpg	0.600436	0.111840	0.791813
engine-size	0.572561	0.631119	-0.523319
horsepower	-0.558256	0.767581	0.314912

Les loadings d'une composante montrent le type de variation capturé. La première composante (PC1) illustre le contraste entre les véhicules luxueux et puissants à faible consommation et les véhicules économiques à bonne consommation. Nos 3 caractéristiques principales varient principalement selon cet axe.

In [28]:
```
# Look at explained variance
plot_variance(pca);
```



Les scores MI montrent que la première composante (PC1) est très informative, mais les autres composantes, malgré leur faible variance, ont également une relation significative avec le prix.

In [29]:

```
import numpy as np
import pandas as pd
from sklearn.feature_selection import mutual_info_regression

# Step 1: Cleaning `y` (Target Variable)
# Replace '?' with NaN and convert y to numeric, coercing errors to NaN
y = y.replace('?', np.nan)
y = pd.to_numeric(y, errors='coerce')

# Drop rows with NaN in y
y_clean = y.dropna()

# Step 2: Cleaning `X` (Feature Variables)
# Select rows from X that match the clean indices of y
X_clean = X_pca.loc[y_clean.index]

# If there are any NaNs left in X, drop those rows
X_clean = X_clean.dropna()

# Ensure that `X_clean` and `y_clean` are aligned
y_clean = y_clean.loc[X_clean.index]

# Step 3: Verify Data Integrity
# Check the length of X and y
print("Length of X_clean:", len(X_clean))
print("Length of y_clean:", len(y_clean))

# Check for any remaining NaN values
assert not X_clean.isnull().values.any(), "X_clean contains NaN values."
assert not y_clean.isnull().values.any(), "y_clean contains NaN values."

# Step 4: Calculate MI scores using the provided function
# Ensure discrete_features is properly defined (all features should be numeric at this point)
discrete_features = False  # Set to True or False depending on your data, here we assume False

# Calculate mutual information scores
try:
    mi_scores = make_mi_scores(X_clean, y_clean, discrete_features)
    print(mi_scores)
except ValueError as e:
```

```
    print(f"Error: {e}")
```

```
Length of X_clean: 201
Length of y_clean: 201
PC1    1.042772
PC2    0.356371
PC3    0.205258
Name: MI Scores, dtype: float64
```

Naïve Bayes

Le Naïve Bayes est un algorithme de classification probabiliste basé sur le théorème de Bayes, utile pour des tâches comme la détection de spam ou l'analyse de sentiment. Théorème de Bayes : Il calcule la probabilité qu'une donnée appartienne à une classe en fonction des caractéristiques observées. Hypothèse naïve : L'algorithme suppose que les caractéristiques sont indépendantes, ce qui simplifie les calculs mais n'est pas réaliste en pratique. Classification : Il choisit la classe avec la plus grande probabilité en multipliant les probabilités individuelles des caractéristiques. Supervisé : Le modèle est entraîné sur des données étiquetées pour pouvoir ensuite classifier de nouvelles instances. Applications : Utilisé pour la classification de texte (spam, sentiment), la catégorisation de documents, etc. Avantages : Simple, rapide à entraîner, efficace pour les grandes données. Inconvénients : Hypothèse d'indépendance irréaliste, ce qui peut limiter les performances lorsque les caractéristiques sont corrélées.

Résumé : Naïve Bayes est efficace pour des tâches simples et des données textuelles, bien qu'il ait des limitations dans des situations plus complexes.

Exemple de classification de spams en utilisant Naïve Bayes et un jeu de spams et de non spams.

En tout une centaine d'exemples. Ce jeu de données est un excellent point de départ pour entraîner un modèle de classification sur un nombre suffisant d'exemples.

In [30]:
```
# Exemples de spam
spam_emails = [
    "Gagnez 1000 euros maintenant en cliquant ici",
    "Offre spéciale, ne manquez pas cette chance exclusive !",
    "Vous avez été sélectionné pour un prix incroyable",
    "Faites vite ! Cette offre expire bientôt, cliquez ici",
    "Obtenez votre prêt personnel sans vérification de crédit",
    "Profitez d'une réduction de 50% sur tous nos produits",
    "Cliquez ici pour recevoir une carte-cadeau gratuite",
    "Votre compte a été suspendu, veuillez vérifier vos informations",
    "Obtenez un iPhone gratuit maintenant en répondant à ce sondage",
    "Vous êtes le gagnant d'un voyage tout compris, contactez-nous",
    "Ne ratez pas cette opportunité de devenir riche rapidement",
    "Remportez un chèque de 5000 euros en vous inscrivant ici",
    "Vos gains sont prêts à être transférés, entrez vos coordonnées",
    "Dernière chance pour bénéficier de cette offre incroyable",
    "Votre solde a été mis à jour, consultez-le maintenant",
    "Réclamez votre remboursement immédiat en cliquant sur ce lien",
    "Votre compte PayPal a été compromis, réinitialisez votre mot de passe",
    "Vous êtes notre gagnant, confirmez votre identité pour recevoir votre prix",
    "Ne manquez pas cette offre exclusive pour les membres VIP",
    "Inscrivez-vous pour obtenir des échantillons gratuits de nos produits",
    "Gagnez un week-end tout inclus en répondant à ce sondage rapide",
    "Vous avez droit à un remboursement exceptionnel, cliquez ici",
    "Recevez une carte-cadeau Amazon de 100 euros maintenant",
    "Offre exclusive pour un prêt à taux zéro, approuvé instantanément",
    "Obtenez une réduction immédiate en utilisant ce code promo",
    "Votre colis est en attente, confirmez votre adresse maintenant",
    "Votre abonnement a été renouvelé, vérifiez les détails ici",
    "Vous êtes sélectionné pour notre nouvelle promotion, cliquez pour plus de détails",
    "Profitez de 70% de réduction aujourd'hui seulement, n'attendez pas !",
    "Félicitations, vous avez gagné une carte-cadeau de 500 euros",
    "Participez à notre concours pour remporter une voiture neuve",
    "Cette offre expire ce soir, cliquez maintenant pour en bénéficier",
    "Votre remboursement est en attente, confirmez vos informations",
    "Cliquez ici pour recevoir un bonus exclusif sur votre prochain achat",
    "Votre compte bancaire a été suspendu, veuillez réinitialiser votre mot de passe",
    "Votre carte de crédit est sur le point d'expirer, mettez-la à jour ici",
    "Obtenez un prêt immédiat sans justificatif en quelques clics",
    "Vous êtes qualifié pour recevoir un prêt personnel à taux réduit",
```

```
    "Confirmez votre adresse pour recevoir un iPhone 12 gratuit",
    "Votre compte est sur le point d'être fermé, action requise",
    "Réclamez vos gains de loterie non réclamés, agissez vite",
    "Votre abonnement Netflix a été suspendu, mettez à jour vos informations",
    "Félicitations ! Vous êtes le gagnant d'un voyage de luxe",
    "Recevez un iPad gratuit en répondant à notre enquête",
    "Votre compte a été sélectionné pour une récompense spéciale",
    "Participez maintenant pour gagner une voiture de sport",
    "Obtenez une remise instantanée sur tous vos achats avec ce lien",
    "Vos informations de carte bancaire sont obsolètes, mettez-les à jour ici",
    "Obtenez une réduction de 70% en utilisant ce code promo exclusif",
    "Votre accès au site premium a été suspendu, réactivez-le maintenant",
    "Vous êtes sélectionné pour gagner une somme d'argent importante"
]
```

In [31]:
```
# Exemples de non-spam
non_spam_emails = [
    "Réunion prévue demain à 9h, merci de confirmer votre présence",
    "N'oubliez pas de soumettre votre rapport avant vendredi",
    "Votre commande a été expédiée, vous la recevrez sous 3 jours",
    "Confirmation de votre réservation pour le dîner de ce soir",
    "Le projet avance bien, nous aurons une réunion la semaine prochaine",
    "Merci pour votre inscription à notre newsletter",
    "La facture de votre abonnement est disponible dans votre espace client",
    "Votre compte a été activé avec succès, bienvenue parmi nous",
    "Nous avons bien reçu votre demande, elle est en cours de traitement",
    "L'événement commence à 18h, veuillez arriver 15 minutes à l'avance",
    "Votre rendez-vous médical est confirmé pour le 10 octobre",
    "Merci pour votre participation à notre sondage",
    "Votre dossier a été approuvé, vous recevrez une confirmation sous peu",
    "Le contrat est prêt à être signé, merci de passer au bureau",
    "Votre demande de remboursement a été acceptée",
    "Nous avons mis à jour notre politique de confidentialité",
    "N'oubliez pas de renouveler votre abonnement avant la fin du mois",
    "Merci pour votre achat, votre facture est disponible en pièce jointe",
    "Votre compte est sécurisé, aucun changement détecté",
    "Nous vous informons d'une interruption de service prévue ce weekend",
    "Votre colis est en chemin, suivez son parcours ici",
    "Merci pour votre soutien lors de la dernière réunion",
    "Le projet a été approuvé par la direction, nous avançons sur la suite",
    "Le planning de la semaine prochaine est disponible en ligne",
    "Merci de nous avoir contactés, nous vous répondrons sous 48h",
    "Votre rendez-vous chez le médecin a été reporté à une date ultérieure",
    "L'ordre du jour de la réunion vous sera envoyé demain",
    "Votre participation à l'événement a bien été enregistrée",
    "Les tickets pour le concert sont maintenant disponibles",
    "Votre facture est en attente de paiement, merci de régulariser",
    "Le rapport mensuel sera disponible la semaine prochaine",
    "N'oubliez pas de vérifier les détails de votre contrat avant signature",
    "Votre compte a été mis à jour avec succès",
    "Veuillez confirmer votre adresse pour l'envoi de votre commande",
    "La mise à jour de sécurité est désormais disponible",
    "Merci de mettre à jour vos informations de compte",
    "Nous avons bien reçu votre retour de produit, il est en traitement",
    "Votre rendez-vous chez le dentiste est confirmé",
```

```
    "Merci pour votre retour d'expérience, nous en tiendrons compte",
    "La réunion d'équipe est reportée à la semaine prochaine",
    "Votre espace client a été mis à jour, consultez-le ici",
    "Merci de compléter le formulaire avant la fin du mois",
    "Le nouveau catalogue est disponible, consultez-le en ligne",
    "Votre inscription à la formation a bien été validée",
    "Nous avons bien reçu votre demande de remboursement",
    "Votre commande a été annulée à votre demande",
    "Merci pour votre participation à l'enquête, les résultats sont disponibles",
    "Le webinaire commencera à 14h, merci de vous connecter 5 minutes avant",
    "Votre changement de mot de passe a été effectué avec succès",
    "L'événement est prévu pour le 15 octobre, inscrivez-vous rapidement",
    "Votre colis est arrivé au point de retrait, vous pouvez le récupérer"
]
```

In [32]:

```python
# Importer les bibliothèques nécessaires
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

# Combiner les emails de spam et non-spam
emails = spam_emails + non_spam_emails
labels = [1] * len(spam_emails) + [0] * len(non_spam_emails)  # 1 pour spam, 0 pour non-spam

# Vérification des tailles
print(f"Nombre d'emails: {len(emails)}")
print(f"Nombre d'étiquettes: {len(labels)}")

# Étape 1 : Convertir le texte en vecteurs numériques (Bag of Words)
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(emails)

# Vérification des dimensions après transformation
print(f"Dimension de la matrice X : {X.shape[0]} emails et {X.shape[1]} caractéristiques")

# Étape 2 : Diviser le jeu de données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.3, random_state=42)

# Étape 3 : Créer et entraîner le modèle Naïve Bayes
model = MultinomialNB()
model.fit(X_train, y_train)

# Étape 4 : Prédire sur les données de test
y_pred = model.predict(X_test)

# Étape 5 : Évaluer le modèle
accuracy = accuracy_score(y_test, y_pred)
print(f"Précision du modèle : {accuracy:.2f}")

# Afficher un rapport de classification
print("\nRapport de classification :\n", classification_report(y_test, y_pred))

# Tester une prédiction manuelle
new_email = ["Félicitations, vous avez gagné un iPhone gratuit, cliquez ici pour réclamer"]
```

```python
new_email_vectorized = vectorizer.transform(new_email)
prediction = model.predict(new_email_vectorized)
print("\nL'email est classifié comme :", "spam" if prediction[0] == 1 else "non-spam")

# Explication :
# - CountVectorizer : Convertit le texte en vecteurs de mots (Bag of Words).
# - train_test_split : Divise les données en entraînement et test (70%-30%).
# - MultinomialNB : Modèle Naïve Bayes adapté à la classification de texte.
# - accuracy_score : Évalue la précision du modèle en comparant les prédictions avec les étiquettes réelles
# - classification_report : Fournit des métriques supplémentaires comme précision, rappel et score F1.
# - vectorizer.transform : Prépare de nouveaux emails pour être prédits par le modèle.
```

```
Nombre d'emails: 102
Nombre d'étiquettes: 102
Dimension de la matrice X : 102 emails et 335 caractéristiques
Précision du modèle : 0.90

Rapport de classification :
              precision    recall  f1-score   support

           0       0.86      0.92      0.89        13
           1       0.94      0.89      0.91        18

    accuracy                           0.90        31
   macro avg       0.90      0.91      0.90        31
weighted avg       0.91      0.90      0.90        31


L'email est classifié comme : spam
```

Classification de spams - Différentes solutions

```
1 Naïve Bayes :
    Avantages : Simple, rapide, adapté aux textes.
    Inconvénients : Hypothèse d'indépendance des caractéristiques,
    moins performant sur des données complexes.

2 SVM :
    Avantages : Performant pour données de haute dimension,
    bonne séparation des classes.
    Inconvénients : Plus lent, coûteux en calcul.

3 Arbres de décision et Random Forest :
    Avantages : Interprétable, évite le surapprentissage,
    gère des relations complexes.
    Inconvénients : Moins efficace sur de grandes données textuelles.

4 Réseaux de neurones (Deep Learning) :
    Avantages : Très performants pour le texte (Transformers, BERT).
    Inconvénients : Exige beaucoup de données et des ressources de calcul.

5 Régression logistique :
    Avantages : Simple, interprétation probabiliste.
    Inconvénients : Limité pour des données textuelles complexes.

6 k-NN :
    Avantages : Facile à comprendre, classe selon les voisins proches.
    Inconvénients : Peu adapté aux grands ensembles de données textuels.

7 Word Embeddings (Word2Vec, BERT) :
    Avantages : Représentation sémantique riche des mots.
    Inconvénients : Nécessite des modèles pré-entraînés ou un entraînement long.
```

Recommandations :

```
Pour des solutions rapides : Naïve Bayes ou Régression Logistique.
Pour capturer des relations complexes : SVM ou Random Forest.
Pour grands volumes de données : Réseaux de neurones ou embeddings de mots.
Techniques d'ensemble comme XGBoost ou LightGBM peuvent être envisagées pour des performances optim
ales.
```

Ci-après même code avec Naïve Bayes mais avec TF-IDF au lieu de Bag of Words

In [33]:

```python
# Importer les bibliothèques nécessaires
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

# Combiner les emails de spam et de non-spam
emails = spam_emails + non_spam_emails
labels = [1] * len(spam_emails) + [0] * len(non_spam_emails)  # 1 pour spam, 0 pour non-spam

# Vérification des tailles
print(f"Nombre d'emails: {len(emails)}")
print(f"Nombre d'étiquettes: {len(labels)}")

# Étape 1 : Convertir le texte en vecteurs numériques (TF-IDF)
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(emails)

# Vérification des dimensions après transformation
print(f"Dimension de la matrice X : {X.shape[0]} emails et {X.shape[1]} caractéristiques")

# Étape 2 : Diviser le jeu de données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.3, random_state=42)

# Étape 3 : Créer et entraîner le modèle Naïve Bayes
model = MultinomialNB()
model.fit(X_train, y_train)

# Étape 4 : Prédire sur les données de test
y_pred = model.predict(X_test)

# Étape 5 : Évaluer le modèle
accuracy = accuracy_score(y_test, y_pred)
print(f"Précision du modèle : {accuracy:.2f}")

# Afficher un rapport de classification
print("\nRapport de classification :\n", classification_report(y_test, y_pred))

# Tester une prédiction manuelle
new_email = ["Félicitations, vous avez gagné un iPhone gratuit, cliquez ici pour réclamer"]
```

```
new_email_vectorized = vectorizer.transform(new_email)
prediction = model.predict(new_email_vectorized)
print("\nL'email est classifié comme :", "spam" if prediction[0] == 1 else "non-spam")
```

```
Nombre d'emails: 102
Nombre d'étiquettes: 102
Dimension de la matrice X : 102 emails et 335 caractéristiques
Précision du modèle : 0.94

Rapport de classification :
              precision    recall  f1-score   support

           0       0.87      1.00      0.93        13
           1       1.00      0.89      0.94        18

    accuracy                           0.94        31
   macro avg       0.93      0.94      0.93        31
weighted avg       0.94      0.94      0.94        31


L'email est classifié comme : spam
```

Explication des modifications :

Changement de la méthode de vectorisation :
    Au lieu d'utiliser CountVectorizer (Bag of Words), nous utilisons TfidfVectorizer, qui génère des vecteurs pondérés basés sur la fréquence des termes et l'inverse de leur fréquence dans le corpus (TF-IDF).
Pas d'autres modifications nécessaires :
    Le reste du code reste identique car Multinomial Naïve Bayes fonctionne bien avec des données pondérées comme celles issues de TF-IDF.

Ci-après Code en utilisant SVM au lieu de Naîve Bayes

In [34]:
```python
# Importer les bibliothèques nécessaires
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC  # Remplacer Naïve Bayes par SVM
from sklearn.metrics import accuracy_score, classification_report

# Combiner les emails de spam et de non-spam
emails = spam_emails + non_spam_emails
labels = [1] * len(spam_emails) + [0] * len(non_spam_emails)  # 1 pour spam, 0 pour non-spam

# Vérification des tailles
print(f"Nombre d'emails: {len(emails)}")
print(f"Nombre d'étiquettes: {len(labels)}")

# Étape 1 : Convertir le texte en vecteurs numériques (TF-IDF)
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(emails)

# Vérification des dimensions après transformation
print(f"Dimension de la matrice X : {X.shape[0]} emails et {X.shape[1]} caractéristiques")

# Étape 2 : Diviser le jeu de données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.3, random_state=42)

# Étape 3 : Créer et entraîner le modèle SVM
model = SVC(kernel='linear')  # Utiliser un noyau linéaire pour SVM
#model = SVC(kernel='rbf')  # Utiliser un noyau non linéaire pour SVM
model.fit(X_train, y_train)

# Étape 4 : Prédire sur les données de test
y_pred = model.predict(X_test)

# Étape 5 : Évaluer le modèle
accuracy = accuracy_score(y_test, y_pred)
print(f"Précision du modèle SVM : {accuracy:.2f}")

# Afficher un rapport de classification
print("\nRapport de classification :\n", classification_report(y_test, y_pred))

# Tester une prédiction manuelle
```

```python
new_email = ["Félicitations, vous avez gagné un iPhone gratuit, cliquez ici pour réclamer"]
new_email_vectorized = vectorizer.transform(new_email)
prediction = model.predict(new_email_vectorized)
print("\nL'email est classifié comme :", "spam" if prediction[0] == 1 else "non-spam")
```

```
Nombre d'emails: 102
Nombre d'étiquettes: 102
Dimension de la matrice X : 102 emails et 335 caractéristiques
Précision du modèle SVM : 0.90

Rapport de classification :
              precision    recall  f1-score   support

           0       0.86      0.92      0.89        13
           1       0.94      0.89      0.91        18

    accuracy                           0.90        31
   macro avg       0.90      0.91      0.90        31
weighted avg       0.91      0.90      0.90        31


L'email est classifié comme : spam
```

Ci-après code utilisant des embeddings comme caractéristiques supplémentaires

En machine learning, les embeddings sont des représentations vectorielles d'objets, généralement de données textuelles ou catégorielles, dans un espace de dimensions inférieures. Ils permettent de transformer des informations complexes (comme des mots ou des catégories) en vecteurs de nombres réels de manière à capturer les similarités entre ces objets. Par exemple, dans un modèle de traitement du langage naturel, des mots ayant des significations proches auront des embeddings similaires.

Les embeddings sont utilisés pour simplifier les données d'entrée tout en préservant les relations entre elles, facilitant ainsi l'apprentissage des modèles. Des techniques comme Word2Vec ou GloVe sont couramment utilisées pour générer des embeddings dans les applications de traitement du langage naturel

In [35]:

```
# Importer les bibliothèques nécessaires
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from gensim.models import Word2Vec
from sklearn.naive_bayes import GaussianNB  # Par exemple, GaussianNB pour gérer des données continues
from sklearn.metrics import accuracy_score, classification_report
from scipy.sparse import hstack

# Liste d'emails (spam et non-spam) - À remplacer par tes données
emails = spam_emails + non_spam_emails
labels = [1] * len(spam_emails) + [0] * len(non_spam_emails)  # 1 pour spam, 0 pour non-spam

# Étape 1 : Convertir le texte en vecteurs numériques (TF-IDF)
vectorizer = TfidfVectorizer()
X_tfidf = vectorizer.fit_transform(emails).toarray()  # Convertir en tableau dense

# Vérification des dimensions après transformation
print(f"Dimension de la matrice TF-IDF : {X_tfidf.shape[0]} emails et {X_tfidf.shape[1]} caractéristiques")

# Étape 2 : Entraîner un modèle Word2Vec pour générer des embeddings
tokenized_emails = [email.split() for email in emails]
embedding_dim = 100  # Dimension des embeddings de mots
word2vec_model = Word2Vec(sentences=tokenized_emails, vector_size=embedding_dim, window=5, min_count=1, worl

# Fonction pour obtenir la moyenne des embeddings d'un email
def get_word2vec_embedding(email_tokens, model, embedding_dim):
    embeddings = [model.wv[word] for word in email_tokens if word in model.wv]
    if embeddings:
        return np.mean(embeddings, axis=0)
    else:
        return np.zeros(embedding_dim)

# Créer les embeddings pour chaque email
email_embeddings = np.array([get_word2vec_embedding(email.split(), word2vec_model, embedding_dim) for email

# Vérification des dimensions après la génération des embeddings
print(f"Dimension de la matrice des embeddings : {email_embeddings.shape}")
```

```python
# Étape 3 : Combiner TF-IDF et les embeddings de mots Word2Vec
X_combined = np.hstack([X_tfidf, email_embeddings])  # Combiner les matrices denses

# Étape 4 : Diviser le jeu de données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X_combined, labels, test_size=0.3, random_state=42)

# Étape 5 : Créer et entraîner un modèle Naïve Bayes (par exemple, GaussianNB pour des données continues)
model = GaussianNB()
model.fit(X_train, y_train)

# Étape 6 : Prédire sur les données de test
y_pred = model.predict(X_test)

# Étape 7 : Évaluer le modèle
accuracy = accuracy_score(y_test, y_pred)
print(f"Précision du modèle : {accuracy:.2f}")

# Afficher un rapport de classification
print("\nRapport de classification :\n", classification_report(y_test, y_pred))

# Étape 8 : Tester une prédiction manuelle
new_email = ["Félicitations, vous avez gagné un iPhone gratuit, cliquez ici pour réclamer"]
new_email_tfidf = vectorizer.transform(new_email).toarray()
new_email_tokens = new_email[0].split()
new_email_embedding = get_word2vec_embedding(new_email_tokens, word2vec_model, embedding_dim)
new_email_combined = np.hstack([new_email_tfidf, new_email_embedding.reshape(1, -1)])  # Combiner TF-IDF + E

prediction = model.predict(new_email_combined)
print("\nL'email est classifié comme :", "spam" if prediction[0] == 1 else "non-spam")
```

```
Dimension de la matrice TF-IDF : 102 emails et 335 caractéristiques
Dimension de la matrice des embeddings : (102, 100)
Précision du modèle : 0.90

Rapport de classification :
               precision    recall  f1-score   support

           0       0.81      1.00      0.90        13
           1       1.00      0.83      0.91        18

    accuracy                           0.90        31
   macro avg       0.91      0.92      0.90        31
weighted avg       0.92      0.90      0.90        31


L'email est classifié comme : spam
```

Idem mais avec les embeddings provenant de gloves

In [36]:
```python
# https://nlp.stanford.edu/projects/glove/
# Importer les bibliothèques nécessaires
import os
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer  # Remplacer CountVectorizer par TfidfVectorize
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB  # Utiliser GaussianNB
from sklearn.metrics import accuracy_score, classification_report
from scipy.sparse import hstack

# Charger les embeddings GloVe à partir du fichier texte
def load_glove_embeddings(file_path, embedding_dim=300):
    embeddings_index = {}
    with open(file_path, encoding='utf-8') as f:
        for line in f:
            values = line.split()
            word = values[0]
            coefs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs
    return embeddings_index

# Remplacez le chemin par l'emplacement du fichier GloVe téléchargé
glove_file_path = './glove.6B.300d.txt'  # Chemin vers le fichier GloVe 300d
if not os.path.exists(glove_file_path):
    raise FileNotFoundError(f"Le fichier GloVe n'a pas été trouvé à l'emplacement : {glove_file_path}")

embedding_dim = 300  # Utiliser 300 dimensions pour GloVe 300d
glove_embeddings = load_glove_embeddings(glove_file_path, embedding_dim)

# Fonction pour obtenir la moyenne des embeddings GloVe d'un email
def get_glove_embedding(email_tokens, embeddings_index, embedding_dim):
    embeddings = [embeddings_index[word] for word in email_tokens if word in embeddings_index]
    if embeddings:
        return np.mean(embeddings, axis=0)
    else:
        return np.zeros(embedding_dim)

# Liste d'emails (spam et non-spam) - À remplacer par tes données
emails = spam_emails + non_spam_emails
labels = [1] * len(spam_emails) + [0] * len(non_spam_emails)  # 1 pour spam, 0 pour non-spam
```

```python
# Vérification des tailles
print(f"Nombre d'emails: {len(emails)}")
print(f"Nombre d'étiquettes: {len(labels)}")

# Étape 1 : Convertir le texte en vecteurs numériques (TF-IDF au lieu de Bag of Words)
vectorizer = TfidfVectorizer()  # Remplacer CountVectorizer par TfidfVectorizer
X_tfidf = vectorizer.fit_transform(emails).toarray()  # Convertir en tableau dense

# Vérification des dimensions après transformation
print(f"Dimension de la matrice TF-IDF : {X_tfidf.shape[0]} emails et {X_tfidf.shape[1]} caractéristiques")

# Étape 2 : Tokeniser les emails et générer les embeddings GloVe pour chaque email
tokenized_emails = [email.split() for email in emails]
email_embeddings = np.array([get_glove_embedding(email_tokens, glove_embeddings, embedding_dim) for email_to

# Vérification des dimensions après la génération des embeddings
print(f"Dimension de la matrice des embeddings : {email_embeddings.shape}")

# Étape 3 : Combiner TF-IDF et les embeddings de mots GloVe
X_combined = np.hstack([X_tfidf, email_embeddings])  # Combiner les matrices denses

# Étape 4 : Diviser le jeu de données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X_combined, labels, test_size=0.3, random_state=42)

# Étape 5 : Créer et entraîner le modèle Naïve Bayes (GaussianNB)
model = GaussianNB()
model.fit(X_train, y_train)

# Étape 6 : Prédire sur les données de test
y_pred = model.predict(X_test)

# Étape 7 : Évaluer le modèle
accuracy = accuracy_score(y_test, y_pred)
print(f"Précision du modèle : {accuracy:.2f}")

# Afficher un rapport de classification
print("\nRapport de classification :\n", classification_report(y_test, y_pred))

# Étape 8 : Tester une prédiction manuelle
new_email = ["Félicitations, vous avez gagné un iPhone gratuit, cliquez ici pour réclamer"]
```

```python
new_email_tfidf = vectorizer.transform(new_email).toarray()
new_email_tokens = new_email[0].split()
new_email_embedding = get_glove_embedding(new_email_tokens, glove_embeddings, embedding_dim)
new_email_combined = np.hstack([new_email_tfidf, new_email_embedding.reshape(1, -1)])  # Combiner TF-IDF + 

prediction = model.predict(new_email_combined)
print("\nL'email est classifié comme :", "spam" if prediction[0] == 1 else "non-spam")
```

```
Nombre d'emails: 102
Nombre d'étiquettes: 102
Dimension de la matrice TF-IDF : 102 emails et 335 caractéristiques
Dimension de la matrice des embeddings : (102, 300)
Précision du modèle : 0.90

Rapport de classification :
              precision    recall  f1-score   support

           0       0.81      1.00      0.90        13
           1       1.00      0.83      0.91        18

    accuracy                           0.90        31
   macro avg       0.91      0.92      0.90        31
weighted avg       0.92      0.90      0.90        31


L'email est classifié comme : spam
```

Conclusion :

Naïve Bayes + TF-IDF est la meilleure solution sur cette exemple.

Récapitulation : Régression en apprentissage supervisé

En apprentissage supervisé, la performance des algorithmes de régression dépend de la nature des données, des relations entre les variables et du bruit. Voici un aperçu des forces et faiblesses des principaux algorithmes :

```
1 Régression Linéaire : Bonne pour les relations linéaires, rapide et simple, mais faible sur les
données non linéaires ou avec outliers.

2 Régression Polynomiale : Captures des relations non linéaires mais risque de surapprentissage s
i le degré du polynôme est trop élevé.

3 Ridge (L2) : Limite le surapprentissage en pénalisant les coefficients. Utile pour les données
multicolinéaires, mais ne fait pas de sélection de variables.

4 Lasso (L1) : Similaire à Ridge, mais élimine certaines variables, ce qui aide à la sélection au
tomatique des caractéristiques.

5 Arbres de décision : Excellents pour les relations non linéaires complexes, mais tendance à sur
apprendre sans régularisation.

6 Forêt Aléatoire : Combine plusieurs arbres, performant pour des relations complexes et robuste
contre le surapprentissage.
```

Classement général :

```
1 Forêt aléatoire
2 Arbres de décision
3 Lasso
4 Ridge
5 Régression polynomiale
6 Régression linéaire
```

Conclusion : Les algorithmes d'ensemble (Forêt aléatoire, Arbres) sont les plus performants pour des données complexes, tandis que les modèles linéaires sont préférables pour des données simples ou lorsque l'interprétabilité est importante.

Le gradient boosting

C'est une méthode itérative qui ajoute des modèles à un ensemble. Elle commence par un modèle initial simple, puis à chaque cycle, elle génère des prédictions, calcule une fonction de perte, ajuste un nouveau modèle pour réduire la perte via la descente de gradient, puis l'ajoute à

l'ensemble. Le processus est répété jusqu'à obtenir un modèle performant.

Example de Régression

In [37]:

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Read the data
data = pd.read_csv('./immodata.csv')

# Select subset of predictors
cols_to_use = ['Rooms', 'Distance', 'Landsize', 'BuildingArea', 'YearBuilt']
X = data[cols_to_use]

# Select target
y = data.Price

# Separate data into training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(X, y)
```

Cet exemple utilise la bibliothèque XGBoost, une implémentation du gradient boosting avec des fonctionnalités supplémentaires pour améliorer la performance et la vitesse. En important l'API xgboost.XGBRegressor, vous pouvez construire et ajuster un modèle de la même manière qu'avec scikit-learn, tout en profitant des nombreux paramètres ajustables de XGBoost.

In [38]:
```
from xgboost import XGBRegressor

my_model = XGBRegressor()
my_model.fit(X_train, y_train)
```

Out[38]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

On fait des prédictions et on évalue.

In [39]:
```
from sklearn.metrics import mean_absolute_error

predictions = my_model.predict(X_valid)
print("Mean Absolute Error: " + str(mean_absolute_error(predictions, y_valid)))
```

```
Mean Absolute Error: 239525.9222466403
```

Le paramètre n_estimators dans XGBoost détermine combien de modèles sont inclus dans l'ensemble, influençant la précision et l'ajustement du modèle. Une valeur trop faible provoque un sous-ajustement, tandis qu'une valeur trop élevée entraîne un surajustement. Les valeurs typiques varient entre 100 et 1000, selon le paramètre learning_rate.

In [40]:
```
my_model = XGBRegressor(n_estimators=90)
my_model.fit(X_train, y_train)
```

Out[40]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=90, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [41]:
```
predictions = my_model.predict(X_valid)
print("Mean Absolute Error: " + str(mean_absolute_error(predictions, y_valid)))
```

```
Mean Absolute Error: 239892.2903695692
```

Le paramètre early_stopping_rounds permet d'arrêter les itérations lorsque le score de validation cesse de s'améliorer. Il est conseillé de définir une valeur élevée pour n_estimators et d'utiliser early_stopping_rounds pour identifier le moment optimal d'arrêt. Un réglage courant est early_stopping_rounds=5, ce qui arrête le modèle après 5 tours consécutifs de dégradation des scores. Il faut également définir un ensemble de validation via le paramètre eval_set.

In [42]:
```
my_model = XGBRegressor(n_estimators=90, early_stopping_rounds=5)  # Set eval_metric here

my_model.fit(X_train, y_train,
             eval_set=[(X_valid, y_valid)],  # Validation set
             verbose=False)                  # Suppress verbose output
```

Out[42]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=5,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=90, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [43]:
```
predictions = my_model.predict(X_valid)
print("Mean Absolute Error: " + str(mean_absolute_error(predictions, y_valid)))
```

```
Mean Absolute Error: 243982.6704298601
```

Le taux d'apprentissage (learning_rate) ajuste les prédictions de chaque modèle avant de les additionner, permettant d'ajouter plus d'estimateurs (n_estimators) sans surajustement. Un faible taux d'apprentissage avec un grand nombre d'estimateurs rend les modèles XGBoost plus précis, mais augmente le temps d'entraînement. Par défaut, XGBoost utilise learning_rate=0.1.

In [44]:
```
my_model = XGBRegressor(n_estimators=90, learning_rate=0.15)
my_model.fit(X_train, y_train,
             eval_set=[(X_valid, y_valid)],
             verbose=False)
```

Out[44]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.15, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=90, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [45]:
```
predictions = my_model.predict(X_valid)
print("Mean Absolute Error: " + str(mean_absolute_error(predictions, y_valid)))
```

```
Mean Absolute Error: 239333.96155191457
```

Le paramètre n_jobs permet d'utiliser le parallélisme pour accélérer l'entraînement sur de grands ensembles de données, en le réglant au nombre de cœurs de la machine. Cela n'améliore pas le modèle, mais réduit le temps d'attente pendant l'entraînement, surtout pour les ensembles volumineux. Pour les petits ensembles de données, ce paramètre n'a que peu d'impact.

In [46]:
```
my_model = XGBRegressor(n_estimators=90, learning_rate=0.15, n_jobs= -1)
my_model.fit(X_train, y_train,
             eval_set=[(X_valid, y_valid)],
             verbose=False)
```

Out[46]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.15, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=90, n_jobs=-1,
             num_parallel_tree=None, random_state=None, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [47]:
```
predictions = my_model.predict(X_valid)
print("Mean Absolute Error: " + str(mean_absolute_error(predictions, y_valid)))
```

```
Mean Absolute Error: 239333.96155191457
```

Il existe plusieurs autres outils et bibliothèques pour les modèles de machine learning supervisé, comparables à XGBoost, en particulier pour les tâches de classification et de régression. Ces outils se distinguent par leur performance, leur facilité d'utilisation et leur capacité à traiter de grandes quantités de données. Voici quelques-unes des alternatives les plus populaires :

1. LightGBM (Light Gradient Boosting Machine)

   Description : LightGBM est une bibliothèque de boosting basée sur des arbres de décision comme XGBoost, mais elle est optimisée pour la rapidité et la gestion de grands ensembles de données. Elle utilise des techniques telles que la croissance de l'arbre basée sur les feuilles et une gestion plus efficace de la mémoire. Avantages : Plus rapide que XGBoost sur de nombreux ensembles de données. Supporte les fonctionnalités de catégorisation nativement. Très efficace pour les ensembles de données volumineux. Exemple d'utilisation :

In [48]:
```
import lightgbm as lgb
model1 = lgb.LGBMRegressor(n_estimators=500, learning_rate=0.15)
model1.fit(X_train, y_train, eval_set=[(X_valid, y_valid)])
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000620 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 822
[LightGBM] [Info] Number of data points in the train set: 10185, number of used features: 5
[LightGBM] [Info] Start training from score 1075176.536082
```

Out[48]:
```
LGBMRegressor(learning_rate=0.15, n_estimators=500)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [49]:
```
predictions = model1.predict(X_valid)
print("Mean Absolute Error: " + str(mean_absolute_error(predictions, y_valid)))
```

```
Mean Absolute Error: 234357.93823316114
```

2. CatBoost

   Description : CatBoost est une bibliothèque de gradient boosting développée par Yandex. Elle est conçue pour traiter efficacement des variables catégorielles sans nécessiter de prétraitement explicite, ce qui est un atout majeur pour les données contenant beaucoup de colonnes non numériques. Avantages : Très efficace avec des données catégorielles (sans encodage manuel nécessaire). Moins de risque d'overfitting comparé à d'autres implémentations de boosting. Compatible avec les GPU. Exemple d'utilisation :

In [50]:
```
from catboost import CatBoostRegressor
model2 = CatBoostRegressor(iterations=500, learning_rate=0.15)
model2.fit(X_train, y_train, eval_set=(X_valid, y_valid))
```

```
0:      learn: 601556.7324231   test: 593172.5161437    best: 593172.5161437 (0)        total: 54.1ms   re
maining: 27s
1:      learn: 569341.4200053   test: 558603.4890342    best: 558603.4890342 (1)        total: 56.5ms   re
maining: 14.1s
2:      learn: 544523.3980662   test: 532780.2615109    best: 532780.2615109 (2)        total: 59.9ms   re
maining: 9.92s
3:      learn: 524625.1795890   test: 511602.9003966    best: 511602.9003966 (3)        total: 62ms     re
maining: 7.68s
4:      learn: 507438.7289612   test: 493375.3911134    best: 493375.3911134 (4)        total: 65.3ms   re
maining: 6.46s
5:      learn: 492404.7207419   test: 476888.5531509    best: 476888.5531509 (5)        total: 67.9ms   re
maining: 5.59s
6:      learn: 480791.3306436   test: 464440.5808064    best: 464440.5808064 (6)        total: 70.6ms   re
maining: 4.97s
7:      learn: 470917.0558828   test: 454000.3715802    best: 454000.3715802 (7)        total: 73.2ms   re
maining: 4.5s
8:      learn: 463085.5617689   test: 445537.8029003    best: 445537.8029003 (8)        total: 76.1ms   re
maining: 4.15s
9:      learn: 455630.5649783   test: 438375.1238795    best: 438375.1238795 (9)        total: 78.3ms   re
```

In [51]:
```
predictions = model2.predict(X_valid)
print("Mean Absolute Error: " + str(mean_absolute_error(predictions, y_valid)))
```

```
Mean Absolute Error: 234250.70911688844
```

3. HistGradientBoosting (Scikit-learn)

   Description : Il s'agit d'une implémentation plus récente et plus rapide du gradient boosting dans Scikit-learn, inspirée par LightGBM et XGBoost. Elle utilise des histogrammes pour accélérer la formation du modèle. Avantages : Très rapide, similaire à LightGBM ou XGBoost en termes de performance. Gère automatiquement les données manquantes. Exemple d'utilisation :

In [52]:
```
from sklearn.ensemble import HistGradientBoostingRegressor

# Créer et entraîner le modèle
model3 = HistGradientBoostingRegressor(max_iter=500)
model3.fit(X_train, y_train)
```

Out[52]:
```
HistGradientBoostingRegressor(max_iter=500)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [53]:
```
predictions = model3.predict(X_valid)
print("Mean Absolute Error: " + str(mean_absolute_error(predictions, y_valid)))
```

```
Mean Absolute Error: 233940.54775966224
```

Classification avec Xgboost

Maintenant utilisons XGBoost pour une tache de classification.

Voici un exemple complet de code Python qui montre comment classifier des données tabulaires en utilisant XGBoost.

Dans cet exemple, nous allons utiliser l'ensemble de données classique Iris, qui est un ensemble de données de classification avec des caractéristiques numériques (longueur et largeur des pétales et des sépales) et une variable cible (espèce de fleur). Étapes :

1 Charger et préparer les données tabulaires. 2 Diviser les données en ensembles d'entraînement et de test. 3 Entraîner un modèle de classification XGBoost. 4 Évaluer le modèle.

In [54]:
```python
# Charger et préparer les données, diviser en ensembles d'entraînement et de test,
# entraîner et évaluer un modèle XGBoost

# Import des bibliothèques
import xgboost as xgb
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris

# 1. Charger l'ensemble de données Iris
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)  # Crée un DataFrame à partir des caractéristiques
y = iris.target  # Les étiquettes des classes (0, 1, 2 pour les espèces de fleurs)

# 2. Diviser les données en ensembles 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)

# 3. Initialiser et entraîner le modèle XGBoost
xgb_model = xgb.XGBClassifier(n_estimators=100, eval_metric='mlogloss')  # Modèle avec 100 arbres
xgb_model.fit(X_train, y_train)  # Entraînement sur les données d'entraînement

# 4. Faire des prédictions sur l'ensemble de test
y_pred = xgb_model.predict(X_test)

# 5. Évaluer la performance du modèle
accuracy = accuracy_score(y_test, y_pred)  # Calcul de l'exactitude
print(f"Exactitude du modèle XGBoost: {accuracy:.2f}")

# Explication :
# - load_iris() charge les données Iris (caractéristiques et étiquettes)
# - train_test_split divise les données en 80% pour l'entraînement et 20% pour le test
# - XGBClassifier entraîne un modèle XGBoost avec 100 arbres
# - accuracy_score évalue la proportion d'échantillons correctement classés
```

```
Exactitude du modèle XGBoost: 1.00
```

idem avec une iris au hasard

In [55]:
```python
# Importer les bibliothèques nécessaires
import numpy as np

# 1. Sélectionner une observation au hasard dans l'ensemble de test
random_index = np.random.randint(0, len(X_test))  # Choisit un index aléatoire dans X_test
random_iris = X_test.iloc[random_index]           # Extrait la fleur au hasard
random_iris_actual_class = y_test[random_index]   # Classe réelle de cette fleur

# 2. Faire une prédiction pour cette observation avec le modèle déjà entraîné
random_iris_reshaped = random_iris.values.reshape(1, -1)  # Reshape pour correspondre au modèle
predicted_class = xgb_model.predict(random_iris_reshaped)[0]

# 3. Afficher les résultats
iris_species = iris.target_names  # Noms des espèces d'Iris
print(f"Iris prise au hasard : {random_iris.values}")
print(f"Classe réelle : {iris_species[random_iris_actual_class]}")
print(f"Classe prédite : {iris_species[predicted_class]}")

# Explication :
# - np.random.randint génère un index aléatoire pour extraire une observation de X_test.
# - reshape ajuste l'observation pour qu'elle soit compatible avec XGBoost.
# - predict utilise le modèle pour prédire la classe de l'observation.
# - Les noms des espèces sont affichés pour la classe réelle et prédite.
```

```
Iris prise au hasard : [5.4 3.4 1.5 0.4]
Classe réelle : setosa
Classe prédite : setosa
```

idem avec une iris générée

In [56]:
```python
import numpy as np

# Plages réalistes pour générer une nouvelle Iris (selon l'ensemble de données Iris)
sepal_length_range = (4.3, 7.9)  # Longueur du sépale
sepal_width_range = (2.0, 4.4)   # Largeur du sépale
petal_length_range = (1.0, 6.9)  # Longueur du pétale
petal_width_range = (0.1, 2.5)   # Largeur du pétale

# 1. Générer une nouvelle Iris avec des caractéristiques aléatoires dans les plages définies
new_iris = np.array([[np.random.uniform(*sepal_length_range),
                      np.random.uniform(*sepal_width_range),
                      np.random.uniform(*petal_length_range),
                      np.random.uniform(*petal_width_range)]])

# 2. Utiliser le modèle XGBoost déjà entraîné pour prédire la classe de la nouvelle Iris
predicted_class_new = xgb_model.predict(new_iris)[0]  # Prédire la classe

# 3. Afficher les résultats
print(f"Caractéristiques de la nouvelle Iris : {new_iris}")
print(f"Classe prédite pour la nouvelle Iris : {iris_species[predicted_class_new]}")

# Explication :
# - np.random.uniform génère des valeurs aléatoires pour chaque caractéristique (longueur et largeur).
# - La nouvelle fleur est prédite par le modèle XGBoost, qui retourne la classe correspondante.
# - Les noms des espèces sont affichés en fonction de la classe prédite.
```

```
Caractéristiques de la nouvelle Iris : [[7.75056364 2.66327423 5.07097849 2.1039295 ]]
Classe prédite pour la nouvelle Iris : virginica
```

Classification d'images avec Xgboost

XGBoost est principalement conçu pour des données tabulaires (comme les ensembles de données avec des colonnes et des lignes), et non pour des images directement. Toutefois, il est possible d'utiliser XGBoost pour la classification d'images en combinant des techniques de traitement d'image et d'apprentissage automatique.

In [57]:
```python
# XGBoost est conçu pour les données tabulaires, mais on peut l'utiliser pour la classification d'images en
# traitement d'images et apprentissage automatique.

# Étapes :
# - Extraire les caractéristiques d'images avec un modèle pré-entraîné (par ex. ResNet ou VGG16).
# - Entraîner XGBoost avec ces caractéristiques pour classer les images.

import numpy as np
import xgboost as xgb
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# 1. Charger un modèle pré-entraîné (par ex. VGG16)
vgg_model = VGG16(weights='imagenet', include_top=False, input_shape=(64, 64, 3))

# 2. Préparer les images avec ImageDataGenerator
datagen = ImageDataGenerator(rescale=1./255)

# Indiquer le répertoire des images
train_data = datagen.flow_from_directory(
    '/home/pnln/car-or-truck/train',
    target_size=(64, 64),
    batch_size=128,
    class_mode='binary',  # 'categorical' pour plusieurs classes
    shuffle=False
)

# 3. Extraire les caractéristiques avec VGG16
def extract_features(model, data):
    features = model.predict(data)
    return features.reshape(features.shape[0], -1)  # Aplatir les caractéristiques

# Extraire les caractéristiques des images
features = extract_features(vgg_model, train_data)

# Obtenir les étiquettes
labels = train_data.classes
```

```python
# 4. Diviser les données en ensemble d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)

# 5. Entraîner XGBoost pour la classification
xgb_model = xgb.XGBClassifier(n_estimators=500, eval_metric='logloss')
xgb_model.fit(X_train, y_train)

# 6. Évaluer le modèle sur l'ensemble de test
y_pred = xgb_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Exactitude du modèle XGBoost: {accuracy:.2f}")

# Explication :
# 1. Modèle pré-entraîné (VGG16) pour extraire des caractéristiques d'images au lieu de faire des prédictio
# 2. ImageDataGenerator pour charger et prétraiter les images.
# 3. Les caractéristiques sont aplaties pour être utilisées par XGBoost.
# 4. Entraînement et évaluation de XGBoost sur ces caractéristiques.
# Cette méthode combine deep learning pour l'extraction de caractéristiques et XGBoost pour la classificati
```

```
2024-10-12 20:46:46.180487: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda drive
rs on your machine, GPU will not be used.
2024-10-12 20:46:46.184987: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda drive
rs on your machine, GPU will not be used.
2024-10-12 20:46:46.199348: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:485] Unable to regis
ter cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
2024-10-12 20:46:46.232262: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:8454] Unable to regi
ster cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered
2024-10-12 20:46:46.238583: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1452] Unable to reg
ister cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registere
d
2024-10-12 20:46:46.252282: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is
optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropria
te compiler flags.
2024-10-12 20:46:47.538025: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could
not find TensorRT

Found 5117 images belonging to 2 classes.
```

```
/home/pnln/anaconda3/lib/python3.10/site-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:1
21: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kw
args` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit
()`, as they will be ignored.
  self._warn_if_super_not_called()

40/40 ━━━━━━━━━━━━━━━━━━━━ 113s 3s/step
Exactitude du modèle XGBoost: 0.75
```

In [58]:
```
print(f"Exactitude du modèle XGBoost: {accuracy:.2f}")
```

```
Exactitude du modèle XGBoost: 0.75
```

Récapitulatif des concepts d'apprentissage automatique supervisé et non supervisé pour résoudre des problèmes de classification et de prédiction :

1. Apprentissage Supervisé

1.1 Classification :

Régression Logistique : Simple pour la classification binaire avec des frontières linéaires.
k-NN : Basé sur les voisins proches, sensible aux grandes données.
SVM : Séparation optimale des classes avec noyaux pour la non-linéarité.
Arbres de Décision, Random Forest, Gradient Boosting : Pour des classifications plus complexes, ave c robustesse accrue et prévention du surajustement.

1.2 Régression :

Régression Linéaire, Ridge/Lasso : Pour des relations simples, avec régularisation pour réduire l'o verfitting.
Arbres de Décision, Random Forest, Gradient Boosting : Utilisés pour prédire des valeurs continues avec meilleure gestion des relations non linéaires.

2. Apprentissage Non Supervisé

2.1 Clustering :

k-Means, Clustering Hiérarchique : Solutions classiques pour regrouper les données en clusters.
DBSCAN, GMM : Solutions avancées pour gérer les formes complexes et les outliers.

2.2 Réduction de Dimensions :

```
PCA, t-SNE, Autoencodeurs : Techniques pour réduire le nombre de variables,
avec des solutions adaptées à la visualisation et aux structures complexes.
```

Conclusion :

Les solutions avancées comme les forêts aléatoires, boosting, et réseaux de neurones sont plus performantes pour des problèmes complexes.

En apprentissage non supervisé, il n'y a pas de régression classique comme on la voit en apprentissage supervisé. Cela est dû au fait que la régression, par définition, implique de prédire une variable cible continue (par exemple, le prix d'une maison), ce qui nécessite des données étiquetées. En apprentissage non supervisé, les algorithmes ne disposent pas de cibles à prédire, mais se concentrent plutôt sur la découverte de structures cachées dans les données.

La régression est généralement une tâche supervisée, car elle nécessite des données étiquetées pour apprendre à faire des prédictions continues. En apprentissage non supervisé, les techniques sont plutôt orientées vers la classification, le clustering ou la réduction de dimensions pour découvrir des structures sans utiliser de cibles préexistantes