Objectif du TP :

Dans ce TP, les étudiants mettront en œuvre un algorithme d'apprentissage supervisé : **la régression linéaire**. En particulier, on s'intéressera à deux types de régression linéaire:

- **la régression linéaire simple**: utilise une seule variable indépendante (ou explicative) pour prédire une variable dépendante (cible).
- **la régression linéaire multiple**: utilise plusieurs variables indépendantes pour prédire une variable dépendante.

La régression linéaire

- est l'une des branches les plus fondamentales de l'apprentissage supervisé.
- est un outil statistique puissant permettant d'examiner la relation entre deux ou plusieurs variables d'intérêt.
- est utilisée pour estimer la relation entre une variable dépendante et diverses variables indépendantes.
- son principe est de tracer une ligne qui correspond le mieux aux données. Cette ligne est connue sous le nom de `ligne de régression` et représente les valeurs prédites générées par l'algorithme. Elle est une droite dans le cas de régression linéaire simple et plus généralement un hyperplan dans le cas de régression linéaire multiple.

Rappel

C'est quoi l'apprentissage supervisé :

- L'apprentissage supervisé, également appelé apprentissage automatique supervisé, est une sous-catégorie de l'apprentissage automatique et de l'intelligence artificielle.
- Il se définit par l'utilisation d'ensembles de données labelisées pour former des algorithmes permettant de classer des données ou de prédire des résultats avec précision.
- Au fur et à mesure que les données d'entrée sont introduites dans le modèle, celui-ci ajuste ses poids jusqu'à ce que le modèle soit correctement ajusté.

Comment fonctionne l'apprentissage supervisé :

- L'apprentissage supervisé utilise un ensemble d'apprentissage pour apprendre aux modèles à produire les résultats souhaités.
- Cet ensemble de données d'apprentissage comprend des entrées et des sorties correctes, ce qui permet au modèle d'apprendre au fil du temps.

- L'algorithme mesure sa précision en utilisant des formules mathématiques, et s'ajuste jusqu'à ce que l'erreur soit suffisamment minimisée.

a. Partie pour l'apprentissage et partie pour le test (train test split):

- Une approche consiste à diviser notre dataset en deux parties : une partie pour le `training` et l'autre pour le `testing`.

- L'idée consiste à entraîner des différents modèles à l'aide de l'ensemble d'entraînement, ensuite on passe à l'étape de l'évaluation sur la liste du test pour choisir le modèle le plus performant.

Exemple d'application :

- Nous allons utiliser la fonction `train_test_split()` pour choisir aléatoirement une liste pour le training et une liste pour le testing.
- Cette fonction accepte un ou plusieurs tableaux (arrays) (tels que des listes, des tableaux de NumPy ou des DataFrames de pandas) en entrée et divise le(s) tableau(x) en sous-ensembles de training et de test. Après la division, elle renvoie deux ou plusieurs tableaux contenant la répartition train-test du ou des tableaux d'entrée.

```
X=[1,2,3,4,5,6,7,8,9,10]
y=[23,11,31,45,12,65,43,90,13,12]
```

- La fonction divise aléatoirement les deux tableaux et renvoie quatre tableaux. Si nous effectuons une répartition $80-20$, la fonction peut retourner les tableaux suivants :

```
# training subset :
X=[1,2,4,5,6,7,9,10]
y=[23,11,45,12,65,43,13,12]

# test subset :
X=[3,8]
y=[31,90]
```

b. validation croisée à K couches (K-Fold Cross-Validation):

- La validation croisée à k couches implique de diviser notre dataset en k sous-ensembles, appelés k couches.

- Sur les k couches, $k-1$ d'entre eux sont utilisés pour l'apprentissage, tandis que la couche restante est utilisée pour la validation.

- L'algorithme est entraîné et testé k fois, en utilisant à chaque fois une nouvelle couche comme ensemble de validation.

- Enfin, le résultat du processus de validation croisée est la moyenne des résultats obtenus à chaque passage.

```
X=[1,2,3,4,5,6,7,8,9,10]
y=[23,11,31,45,12,65,43,90,13,12]
```

- On utilise, par exemple, une validation croisée à 3 couches.

Couche 1 :

```
X=[1,5,9]
y=[23,12,13]
```

Couche 2 :

```
X=[2,6,8]
y=[11,65,90]
```

Couche 3:

```
X=[3,4,7]
y=[31,45,43]
```

Régression linéaire simple

- Pour appliquer la validation croisée, on fait le training du modèle sur les couches 1 et 2 et on fait le test en utilisant la couche 3.
- On fait le même process en changeant les couches de test et de training : 2 et 3 pour le training et 1 pour le test, ensuite 3 et 1 pour le training et 2 pour le test.
- Chaque evaluation va donner un score, exemple, 0.97, 0.93, 0.99.
- On prend la moyenne de ces scores $(0.97+0.93+0.99)/3=0.963$.

Comment peut on choisir la meilleure droite :

- La régression linéaire $\overset{\text{simple}}{\text{¿}}$ utilise une droite pour définir la relation entre deux variables, X et Y et elle utilise le modèle mathématique suivant :

$$Y=\beta_0+\beta_1 X$$

.

- La régression linéaire $\overset{\text{multiple}}{\text{¿}}$ utilise un hyperplan pour définir la relation entre plusieur variables indépendantes $X_1, X_2, \dots X_n$ et la variable dépendante Y et elle utilise le modèle mathématique suivant :

$$Y=\beta_0+\beta_1 X_1+\beta_2 X_2+\dots+\beta_n X_n=\beta_0+\begin{pmatrix}\beta_1 & \beta_2 & \dots & \beta_n\end{pmatrix}\begin{pmatrix}X_1\\X_2\\\vdots\\X_n\end{pmatrix}=\beta_0+\beta . X$$

.

- $\beta = \begin{pmatrix} \beta_1 & \beta_2 & \dots & \beta_n \end{pmatrix}$ et $X = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{pmatrix}$

- $\beta_0, \beta_1, \beta_2, \dots, \beta_n$ sont les paramètres du modèles, et l'objectif principal de l'algorithme est de déterminer les meilleures valeurs pour ces paramètres.

- L'erreur d'un point de données est la distance entre sa valeur prédite et sa valeur réelle. Cette erreur est techniquement connue comme le résidu.

- L'objectif de l'algorithme de régression linéaire est de trouver les meilleures valeures $\beta_0, \beta_1, \beta_2, \dots, \beta_n$ qui donne le plus petit Mean Squared Error : une fonction construite par tous les résidus et avec des variables $\beta_0, \beta_1, \beta_2, \dots, \beta_n$.

⇒ La régression linéaire multiple est plus performante que la La régression linéaire simple.

Les étapes pour appliquer **la régression linéaire:**

1. Importer des modules spécifiques (linear_model, model_selection) à partir des bibliothèques (sklearn) nécessaires pour le modèle.
2. Construire la dataframe.
3. Examiner la data.
4. Diviser la data en deux parties : training and testing.
5. Visualiser la data.
6. Entrainer le modèle.
7. Evaluer le modèle.

1. Importer les bibliothèques nécessaires pour le modèle.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
```

2. Construire la dataframe.

```
#Charger le dataframe 'house_area_price.csv' et le nommer housing
housing=pd.read_csv('house_area_price.csv')
```

3. Examiner la data.

```
#Number of rows and columns
print(housing.shape)
```

```
(1000, 2)
```

```
#Show the first 5 rows of the dataset
print(housing.head())
```

```
   Area         Price
0  1360  2.623829e+05
1  4272  9.852609e+05
2  3592  7.779774e+05
3   966  2.296989e+05
4  4926  1.041741e+06
```

```
#Get information about columns
print(housing.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Area    1000 non-null   int64  
 1   Price   1000 non-null   float64
dtypes: float64(1), int64(1)
memory usage: 15.8 KB
None
```

```
# Generate descriptive statistics.
print(housing.describe())
```

```
              Area         Price
count  1000.000000  1.000000e+03
mean   2815.422000  6.188610e+05
std    1255.514921  2.535681e+05
min     503.000000  1.116269e+05
25%    1749.500000  4.016482e+05
50%    2862.500000  6.282673e+05
75%    3849.500000  8.271413e+05
max    4999.000000  1.108237e+06
```

```
#check for duplicate rows using the duplicated() method
print(housing.duplicated().sum())
```

```
0
```

```
#Check for missing values
print(housing.isnull().sum())
```

```
Area     0
Price    0
dtype: int64
```

4. Diviser la data en deux parties : training and testing.

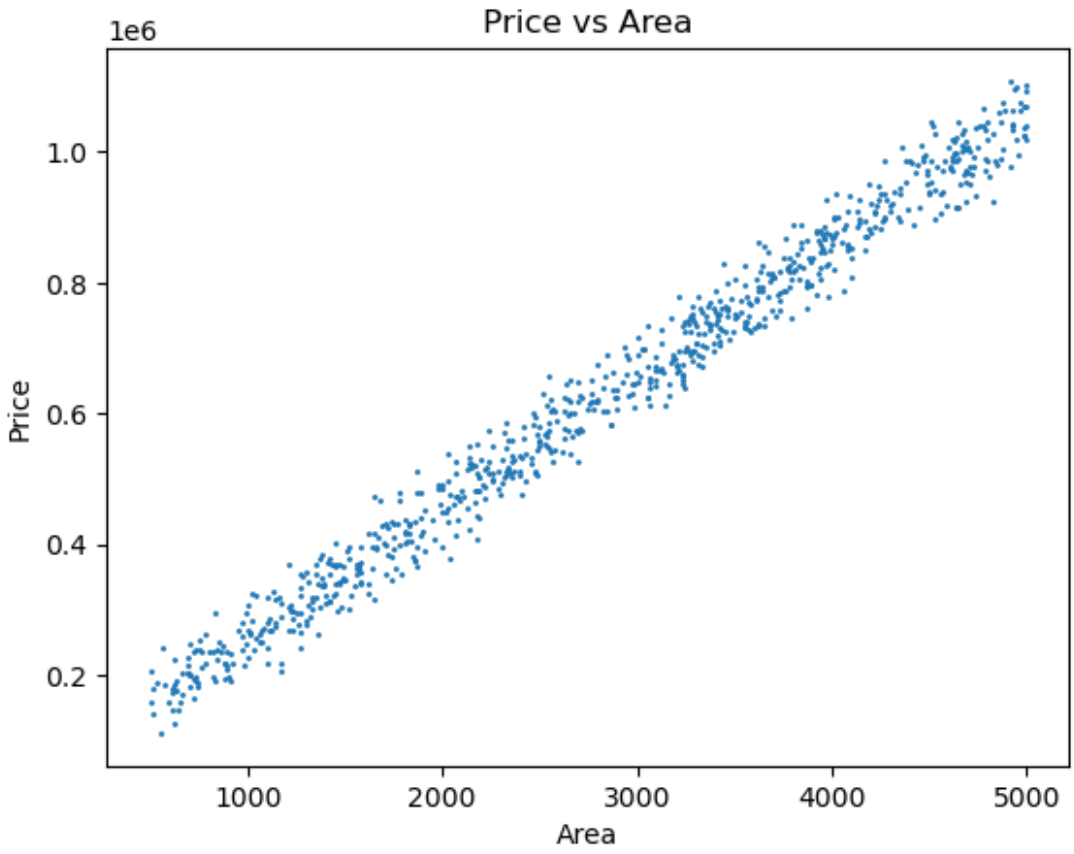
```
X = housing[['Area']]
y = housing['Price']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=True)
```

5. Visualiser la data.

```
plt.scatter(X_train['Area'], y_train, s=1)
plt.title('Price vs Area')
plt.xlabel('Area')
plt.ylabel('Price')

Text(0, 0.5, 'Price')
```



6. Entrainer le modèle.

```
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
LinearRegression()
```

```
# Afficher les paramètres du modèle
print(lr.intercept_)
print(lr.coef_)
```

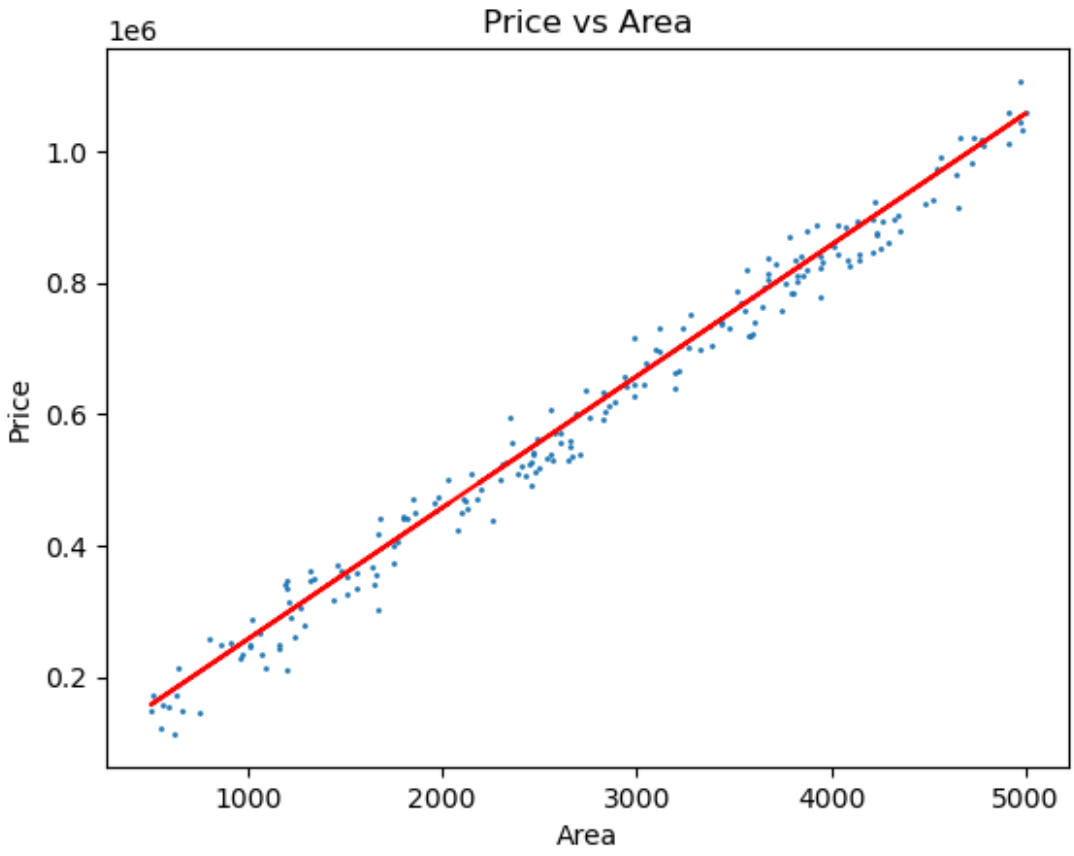
```
57087.17915647873
[200.21142071]
```

```
plt.scatter(X_test['Area'], y_test, s=1)

y_pred = lr.predict(X_test)
plt.plot(X_test['Area'], y_pred, color='red')

plt.title('Price vs Area')
plt.xlabel('Area')
plt.ylabel('Price')
```

```
Text(0, 0.5, 'Price')
```



7. Evaluer le modèle.

Discutons des critères d'évaluation des modèles de régression. Les mesures couramment utilisées sont:

- L'erreur absolue moyenne (MAE)
- l'erreur quadratique moyenne racine (Root Mean Squared Error)
- le R-carré (R squared).

7.1 L'erreur absolue moyenne (MAE):

- Cette valeur mesure l'ampleur moyenne des erreurs entre les valeurs prédites et les valeurs réelles, sans tenir compte de la direction de ces erreurs.

$$MAE = \frac{1}{n}\sum \vee y_{vrai} - y_{pred} \vee ¿$$

- Supposons que nous avons deux listes: **pred** et **true** qui stockent les valeurs réelles et les valeurs prédites :

```
pred=[2.1,1.4,5.6,7.9]
true=[2.5,1.6,5.1,6.8]
```

Pour calculer le MAE, on commence par

- Calculer la différence entre les valeurs des deux listes $\Rightarrow [-0.4, -0.2, 0.5, 1.1)$.
- On cherche les valeurs absolues des valeurs obtenues $\Rightarrow [0.4, 0.2, 0.5, 1.1)$.
- On Calcule la somme $\Rightarrow 0.4+0.2+0.5+1.1=2.2$.

⇒ MAE = 2.2.

7.2 L'erreur quadratique moyenne racine (Root Mean Squared Error): RMSE

- Cette valeur est déterminée en calculant la racine carrée de la moyenne des différences au carré entre les valeurs réelles et les valeurs prédites.

$$RMSE = \sqrt{\frac{e_1^2+e_2^2+\ldots+e_n^2}{n}}, \text{avec } e_i = y_i - \tilde{y}_i$$

- Dans l'exemple précédent:

```
pred=[2.1,1.4,5.6,7.9]
true=[2.5,1.6,5.1,6.8]
```

Pour calculer le MSE, on commence par

- Calculer la différence entre les valeurs des deux listes $\Rightarrow [-0.4, -0.2, 0.5, 1.1)$.
- On cherche les carrés des valeurs obtenues $\Rightarrow [0.16, 0.4, 0.25, 1.21)$.

- On cherche la moyenne des valeurs trouvées $\Rightarrow \frac{(0.16+0.4+0.25+1.21)}{4}=0.505.$

⇒ RMSE = la racine carrées de la moyenne = $\sqrt{0.505}=0.71.$

7.3 le R-carré (R squared): R^2

- Aussi connue sous le nom score R^2, c'est une mesure statistique de la proximité des données par rapport à la ligne de régression ajustée. Il est également connu sous le nom de coefficient de détermination, ou de coefficient de détermination multiple pour la régression multiple.

$$R^2 \text{ score} = 1 - \frac{\text{Sum of Squared Differences between predicted and true values}}{\text{Sum of Squared Differences between true values and mean of true values}} = 1 - \frac{\sum\left(y_{p_i} - y_{t_i}\right)^2}{\sum\left(y_{t_i} - \acute{y}_t\right)^2}$$

- R^2 est entre 0 % et 100 % **dans le contexte de la régression linéaire** :

  0 % indique que le modèle n'explique aucune des variabilités des données de réponse autour de sa moyenne.

  100 % indique que le modèle explique toute la variabilité des données de réponse autour de sa moyenne.

  Cependant, il est important de noter que dans certains cas particuliers, comme les modèles de régression non linéaire ou les modèles mal spécifiés, les valeurs de R^2 peuvent être négatives.

On prend les mêmes listes :

```
pred=[2.1,1.4,5.6,7.9]
true=[2.5,1.6,5.1,6.8]
```

- On commence par calculer la différence entre les valeurs des deux listes $\Rightarrow[-0.4,-0.2,0.5,1.1).$
- On cherche le carrés des valeurs de la liste $\Rightarrow[0.16,0.4,0.25,1.21)=¿$ Squared Differences between predicted and true values $= (y_{p_i} - y_{t_i})^2 $
- On va calculer la somme: $\sum\left(y_{p_i} - y_{t_i}\right)^2=0.16+0.4+0.25+1.21=1.66.$
- On cherche la moyenne des valeurs de la liste true $\Rightarrow \frac{(2.5+1.6+5.1+6.8)}{4}=4.$
- On cherche la liste de la difference entre les valeurs de true et leurs moyenne $\left(y_{t_i} - \acute{y}_t\right)^2\ [-1.5,-2.4,1.1,2.8) \Rightarrow$ les carrés$=[2.25,5.76,1.21,7.84).$
- On fait la somme $\sum\left(y_{t_i} - \acute{y}_t\right)^2$ ¿$2.25+5.76+1.21+7.84=17.06.$

$$\Rightarrow R^2 = 1 - \frac{\sum \left(y_{p_i} - y_{t_i}\right)^2}{\sum \left(y_{t_i} - \acute{y}_t\right)^2} = 1 - \frac{1.66}{17.06} = 0.902696$$

```
MAE = mean_absolute_error(y_test, y_pred)
RMSE = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
print(MAE)
print(RMSE)
print(r2)
```

```
27085.216196781043
32737.020858082935
0.982164362898043
```

```
c:\Users\yassi\anaconda3\Lib\site-packages\sklearn\metrics\
_regression.py:492: FutureWarning: 'squared' is deprecated in version
1.4 and will be removed in 1.6. To calculate the root mean squared
error, use the function'root_mean_squared_error'.
  warnings.warn(
```

Régression linéaire multiple

1. Importer les bibliothèques nécessaires pour le modèle.

2. Construire la dataframe.

```
#Charger le dataframe 'boston.xls' et le nommer boston
boston=pd.read_csv('boston.xls')
```

3. Examiner la data.

```
#Number of rows and columns
print(boston.shape)
```

```
(506, 13)
```

```
#Show the first 5 rows of the dataset
print(boston.head())
```

```
      CRIM    ZN   INDUS   CHAS    NOX     RM   AGE     DIS  RAD  TAX
PTRATIO  \
0  0.00632  18.0    2.31     0  0.538  6.575  65.2  4.0900    1  296
15.3
1  0.02731   0.0    7.07     0  0.469  6.421  78.9  4.9671    2  242
17.8
```

```
2  0.02729   0.0    7.07     0  0.469  7.185  61.1  4.9671    2  242
17.8
3  0.03237   0.0    2.18     0  0.458  6.998  45.8  6.0622    3  222
18.7
4  0.06905   0.0    2.18     0  0.458  7.147  54.2  6.0622    3  222
18.7
```

```
   LSTAT  MEDV
0   4.98  24.0
1   9.14  21.6
2   4.03  34.7
3   2.94  33.4
4   5.33  36.2
```

```
#Get information about columns
print(boston.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 13 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1    ZN      506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    int64
 10  PTRATIO  506 non-null    float64
 11  LSTAT    506 non-null    float64
 12  MEDV     506 non-null    float64
dtypes: float64(10), int64(3)
memory usage: 51.5 KB
None
```

```
# Generate descriptive statistics.
```

```
#check for duplicate rows
print(boston.duplicated().sum())
```

```
0
```

```
#Check for missing values
print(boston.isnull().sum())
```

```
CRIM       0
 ZN        0
```

```
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
LSTAT      0
MEDV       0
dtype: int64
```

4. Diviser la base de données en deux tableaux Xm (tableau d'observations) et ym (la variable cible)

```
Xm = boston.iloc[:,0:12]
ym = boston['MEDV']

#Diviser les données: 80% pour l'entrainement et 20% pour le test
Xm_train, Xm_test, ym_train, ym_test = train_test_split(Xm, ym,
test_size=0.2, random_state=True)
```

5. Entrainer le modèle.

The following code standardizes the training and test datasets so that they have a mean of 0 and a standard deviation of 1, which is a common preprocessing step for many machine learning algorithms (Standardization (Standard Scaler)):

```
from sklearn.preprocessing import StandardScaler

# Initialize StandardScaler
scaler = StandardScaler()

# Fit and transform Xm_train
Xm_train_sc = scaler.fit_transform(Xm_train)

# Transform Xm_test
Xm_test_sc = scaler.transform(Xm_test)
```

Entraîner le modèle et prédire les labels de Xm_test

```
lr = LinearRegression()

# Entraîner le modèle sur les données standardisées
lr.fit(Xm_train_sc, ym_train)

# Prédire les valeurs sur les données test standardisées
```

```
ym_pred = lr.predict(Xm_test_sc)

# Afficher les prédictions
print("Valeurs prédites :", ym_pred)
```

```
Valeurs prédites : [32.64466143 28.02150176 17.93076308 21.47077322
18.37241408 19.79975584
 32.47139863 18.18333705 24.29980803 27.08683319 26.52602064
28.71672856
 21.02348155 26.81486817 23.34624829 20.54731508 16.6184764
38.14830438
 30.53816017  8.08514565 20.75301561 17.4362114  25.16022455
24.81878297
 31.38795475 12.63506377 14.71026648 16.69424041 36.35941986
13.9093412
 21.26372736 13.82332917 43.12224578 17.23143855 21.91087943
20.35781087
 17.36410126 27.23464372  8.91577615 19.19775256 24.3682084
21.19482648
 29.48943018 16.22742769 18.70461114 16.09908675 39.16913201
17.35340826
 26.83421734 22.68655592 24.62483054 24.46622381 25.18382827
27.13192096
  4.63858386 24.04569016 10.28669942 26.87883919 16.84352902
35.83259937
 19.35772057 27.60399835 15.8495098  20.34466255 11.02365005
32.42148514
 36.64521008 21.93153487 24.45188707 24.92312881 23.37338305
5.95465182
 16.58075591 20.45373431 20.80582949 21.01263024 33.52240491
27.95056848
 25.02383385 34.69197064 18.38241071 23.89069166 34.53680886
14.45790881
 20.78124347 30.22893348 16.97541424 24.25866036 19.0771969
16.7237838
 27.07250003 41.61001001 13.81650069 23.0807691  16.48376412
21.68689822
 23.03640927 29.04669378 37.04331453 20.44053576 19.37988172
16.92154023]
```

Parameters & Predictions

```
print('Intercept = ', lr.intercept_)

print('Coefficients : ', lr.coef_)

print("Predictions: ", ym_pred)
```

```
Intercept =  22.522277227722775
Coefficients :  [-1.09395657  1.35443117  0.0824158   0.60540243 -
2.3758176   2.06581518
  0.18015689 -3.2108662   2.49269931 -1.90690775 -2.10712714 -
4.07166415]
Predictions:  [32.64466143 28.02150176 17.93076308 21.47077322
18.37241408 19.79975584
 32.47139863 18.18333705 24.29980803 27.08683319 26.52602064
28.71672856
 21.02348155 26.81486817 23.34624829 20.54731508 16.6184764
38.14830438
 30.53816017  8.08514565 20.75301561 17.4362114  25.16022455
24.81878297
 31.38795475 12.63506377 14.71026648 16.69424041 36.35941986
13.9093412
 21.26372736 13.82332917 43.12224578 17.23143855 21.91087943
20.35781087
 17.36410126 27.23464372  8.91577615 19.19775256 24.3682084
21.19482648
 29.48943018 16.22742769 18.70461114 16.09908675 39.16913201
17.35340826
 26.83421734 22.68655592 24.62483054 24.46622381 25.18382827
27.13192096
  4.63858386 24.04569016 10.28669942 26.87883919 16.84352902
35.83259937
 19.35772057 27.60399835 15.8495098  20.34466255 11.02365005
32.42148514
 36.64521008 21.93153487 24.45188707 24.92312881 23.37338305
5.95465182
 16.58075591 20.45373431 20.80582949 21.01263024 33.52240491
27.95056848
 25.02383385 34.69197064 18.38241071 23.89069166 34.53680886
14.45790881
 20.78124347 30.22893348 16.97541424 24.25866036 19.0771969
16.7237838
 27.07250003 41.61001001 13.81650069 23.0807691  16.48376412
21.68689822
 23.03640927 29.04669378 37.04331453 20.44053576 19.37988172
16.92154023]
```

6. Evaluer le modèle.

```python
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
import numpy as np

# Prédire les valeurs sur les données test
y_pred = lr.predict(Xm_test_sc)
```

```python
# Calculer la racine de l'erreur quadratique moyenne (RMSE)
rmse = np.sqrt(mean_squared_error(ym_test, ym_pred))
print("Root Mean Squared Error (RMSE):", rmse)

# Calculer l'erreur absolue moyenne (MAE)
mae = mean_absolute_error(ym_test, ym_pred)
print("Mean Absolute Error (MAE):", mae)

# Calculer le coefficient de détermination (R²)
r2 = r2_score(ym_test, ym_pred)
print("R-squared (R²):", r2)
```

```
Root Mean Squared Error (RMSE): 4.9566149825653225
Mean Absolute Error (MAE): 3.8688225202456725
R-squared (R²): 0.7514046235740633
```