

# Jour 41 : Housing Boston

L'objectif de ce cas pratique est de **prédir le prix médian (medv) de logements à Boston**. Le Boston Housing Dataset est un jeu de données classique en machine learning et statistique. Il décrit le marché immobilier de Boston, en ciblant les facteurs clés des prix (criminalité, taxes, éducation, accès aux autoroutes, etc.). Basé sur des données de recensement, il permet d'analyser des tendances, de créer des modèles prédictifs et de comprendre les dynamiques des prix des maisons occupées par leur propriétaire en milliers de dollars (medv). On se propose ici de construire un modèle de régression linéaire multiple pour répondre à l'objectif fixé.

## \$\bigoplus\$ Signification des variables

- crim : Taux de criminalité par habitant dans la ville.
- zn : Proportion de lots résidentiels de grande taille (supérieurs à 25 000 pieds carrés).
- indus : Proportion de terrains occupés par des entreprises non commerciales dans la ville.
- chas : Variable binaire indiquant si la propriété est proche de la rivière Charles (1 pour oui, 0 pour non).
- nox : Concentration d'oxydes d'azote dans l'air.
- rm : Nombre moyen de pièces par logement.
- age : Proportion de logements occupés par des propriétaires et construits avant 1940.
- dis : Distances pondérées aux centres d'emploi de Boston.
- rad : Indice d'accessibilité aux autoroutes radiales.
- tax : Taux d'imposition foncière pour 10 000 \$.
- ptratio : Ratio du nombre d'élèves par enseignant.
- b : Indice de proportion de la population afro-américaine dans chaque quartier
- lstat: Pourcentage de la population appartenant à une catégorie socio-économique inférieure.
- medv : Valeur médiane des maisons occupées par leurs propriétaires

## 0. Chargement des librairies

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error

# Pour désactiver les warnings
```

```
import warnings  
warnings.filterwarnings("ignore")
```

## 1. Chargement de la base de données

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

```
Out[2]:
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5

## 2. Identification et traitement des valeurs manquantes

```
In [3]: data.isnull().mean() * 100
```

```
Out[3]:
```

crim	0.000000
zn	0.000000
indus	0.000000
chas	0.000000
nox	0.000000
rm	0.988142
age	0.000000
dis	0.000000
rad	0.000000
tax	0.000000
ptratio	0.000000
b	0.000000
lstat	0.000000
medv	0.000000
dtype: float64	

Notre base de données ne contiennent aucune valeur manquante, à l'exception du nombre moyen de pièces par logement (rm) qui contient 0.98% de valeurs manquantes. Ce taux de valeur manquante est assez faible et on pourrait décider de la suppression des individus concernés. Mais, essayons d'imputer ces valeurs par la moyenne.

```
In [4]: # Imputation par La moyenne  
data["rm"] = data["rm"].fillna(data["rm"].mean())
```

```
In [5]: # Vérification  
data.isnull().mean() * 100
```

```
Out[5]: crim      0.0
         zn       0.0
         indus    0.0
         chas     0.0
         nox      0.0
         rm       0.0
         age      0.0
         dis      0.0
         rad      0.0
         tax      0.0
         ptratio   0.0
         b        0.0
         lstat    0.0
         medv    0.0
         dtype: float64
```

Nous n'avons à présent plus de valeurs manquantes.

## 3. Analyse descriptive

### 3.1. Statistiques descriptives

```
In [6]: data.describe().transpose()
```

	<b>count</b>	<b>mean</b>	<b>std</b>	<b>min</b>	<b>25%</b>	<b>50%</b>	<b>75%</b>
<b>crim</b>	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.677083
<b>zn</b>	506.0	11.363636	23.322453	0.00000	0.000000	0.00000	12.500000
<b>indus</b>	506.0	11.136779	6.860353	0.46000	5.190000	9.69000	18.100000
<b>chas</b>	506.0	0.069170	0.253994	0.00000	0.000000	0.00000	0.000000
<b>nox</b>	506.0	0.554695	0.115878	0.38500	0.449000	0.53800	0.624000
<b>rm</b>	506.0	6.284341	0.702085	3.56100	5.885500	6.21000	6.618750
<b>age</b>	506.0	68.574901	28.148861	2.90000	45.025000	77.50000	94.075000
<b>dis</b>	506.0	3.795043	2.105710	1.12960	2.100175	3.20745	5.188425
<b>rad</b>	506.0	9.549407	8.707259	1.00000	4.000000	5.00000	24.000000
<b>tax</b>	506.0	408.237154	168.537116	187.00000	279.000000	330.00000	666.000000
<b>ptratio</b>	506.0	18.455534	2.164946	12.60000	17.400000	19.05000	20.200000
<b>b</b>	506.0	356.674032	91.294864	0.32000	375.377500	391.44000	396.225000
<b>lstat</b>	506.0	12.653063	7.141062	1.73000	6.950000	11.36000	16.955000
<b>medv</b>	506.0	22.532806	9.197104	5.00000	17.025000	21.20000	25.000000

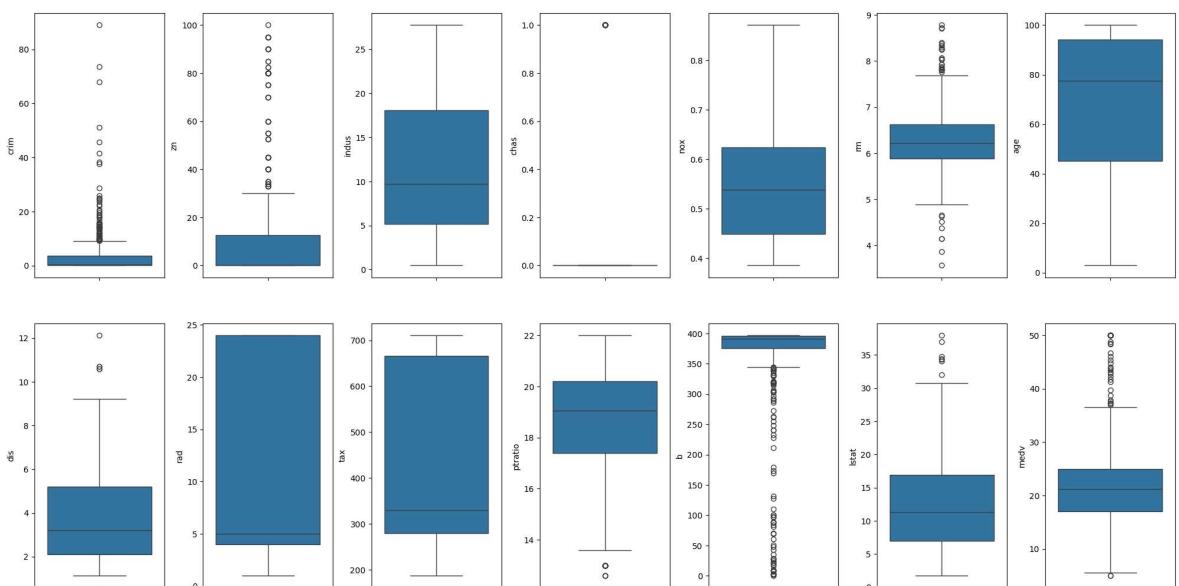


Le jeu de données contient 506 agglomérations de Boston. Entre autres, si tous les logements dans ces agglomérations devraient avoir le même prix médian, ce prix serait de 22.53 mille dollars environ.

### 3.2. Analyse des distributions des données

```
In [7]: # Création de boxplots
fig, ax = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
ax = ax.flatten()

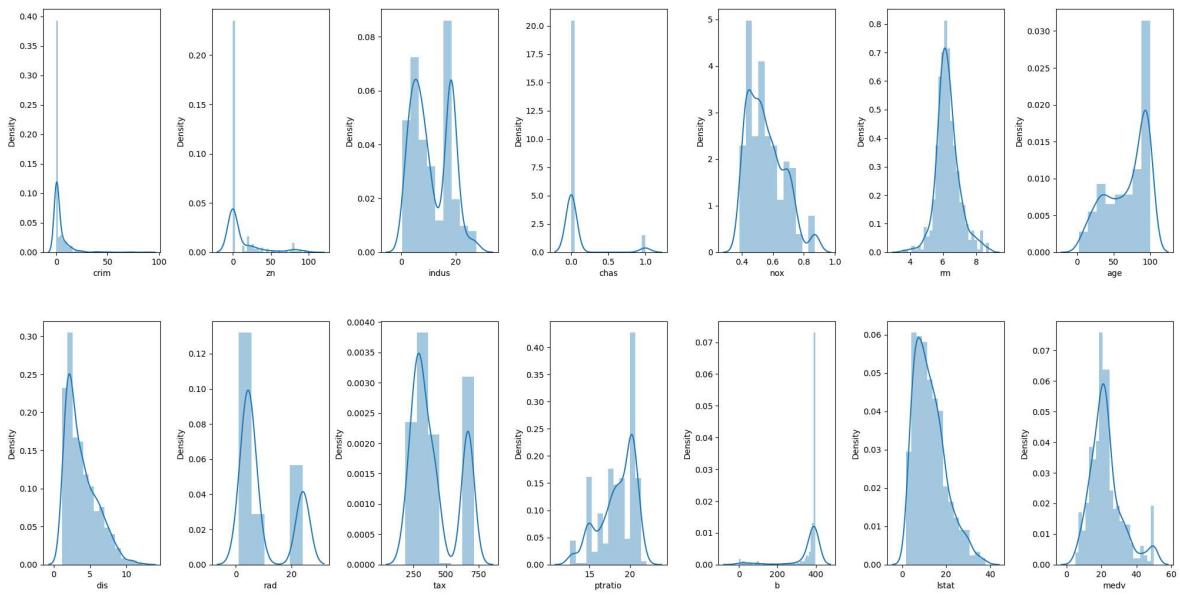
for col, value in data.items():
    sns.boxplot(y=col, data=data, ax=ax[index])
    index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



Sur ces boîtes à moustache, les petits ronds représentent les outliers. Pour une analyse rigoureuse, les variables concernées doivent être traitées particulièrement ou on préférera un autre type de modèle, au modèle de régression linéaire classique. Les colonnes contenant de nombreuses valeurs aberrantes ne suivent pas une distribution normale. Nous pouvons réduire l'influence des valeurs aberrantes en appliquant une transformation logarithmique. Nous pouvons également supprimer les colonnes contenant des valeurs aberrantes ou supprimer les lignes qui en contiennent.

```
In [8]: # Densités de distribution des données
fig, ax = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
ax = ax.flatten()

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

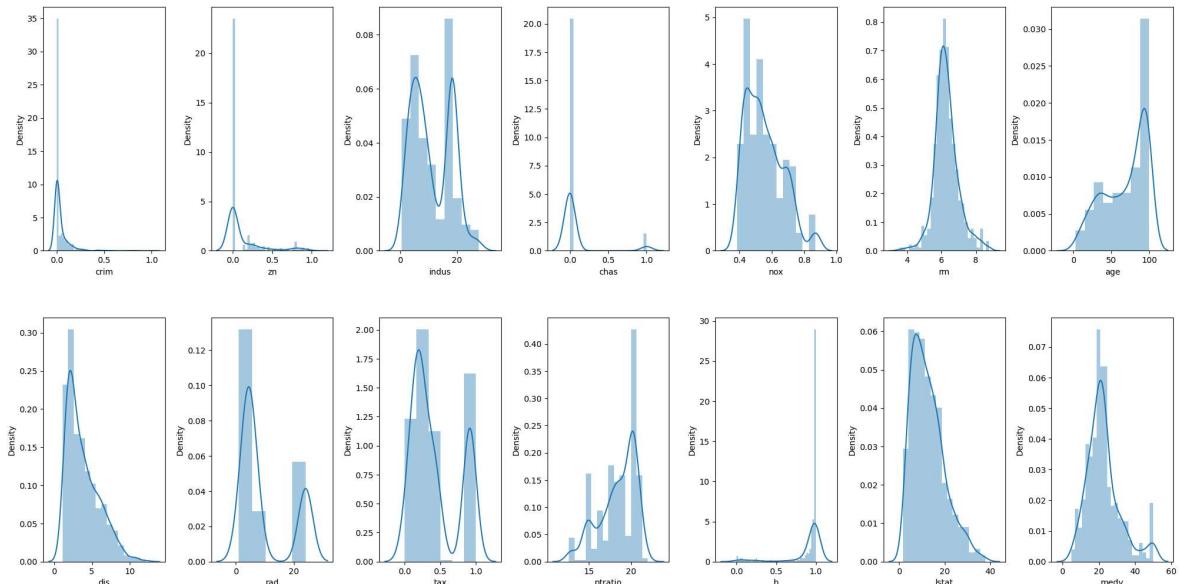


Nous pouvons observer des graphiques asymétriques à droite et à gauche pour '**crim**', '**zn**', '**tax**' et '**b**'. Par conséquent, nous devons normaliser ces données. Une normalisation du type min-max sera appliquée à ces variables afin de ramener leurs valeurs entre 0 et 1.

```
In [9]: cols = ['crim', 'zn', 'tax', 'b']
for col in cols:
    # Recherche du min et du max de chaque colonne
    minimum = min(data[col])
    maximum = max(data[col])
    data[col] = (data[col] - minimum) / (maximum - minimum)
```

```
In [10]: fig, ax = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
ax = ax.flatten()

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



Désormais, les valeurs de ces colonnes sont comprises entre 0 et 1. La normalisation Min-Max a transformé la valeur maximale en '1' et la valeur minimale en '0'.

Si on souhaitais standardiser les données, on pourra procéder comme suit.

```
In [11]: # standardisation
from sklearn import preprocessing
scalar = preprocessing.StandardScaler()

# Application aux données
scaled_cols = scalar.fit_transform(data[cols])
scaled_cols = pd.DataFrame(scaled_cols, columns=cols)
scaled_cols.head()
```

```
Out[11]:
```

	crim	zn	tax	b
0	-0.419782	0.284830	-0.666608	0.441052
1	-0.417339	-0.487722	-0.987329	0.441052
2	-0.417342	-0.487722	-0.987329	0.396427
3	-0.416750	-0.487722	-1.106115	0.416163
4	-0.412482	-0.487722	-1.106115	0.441052

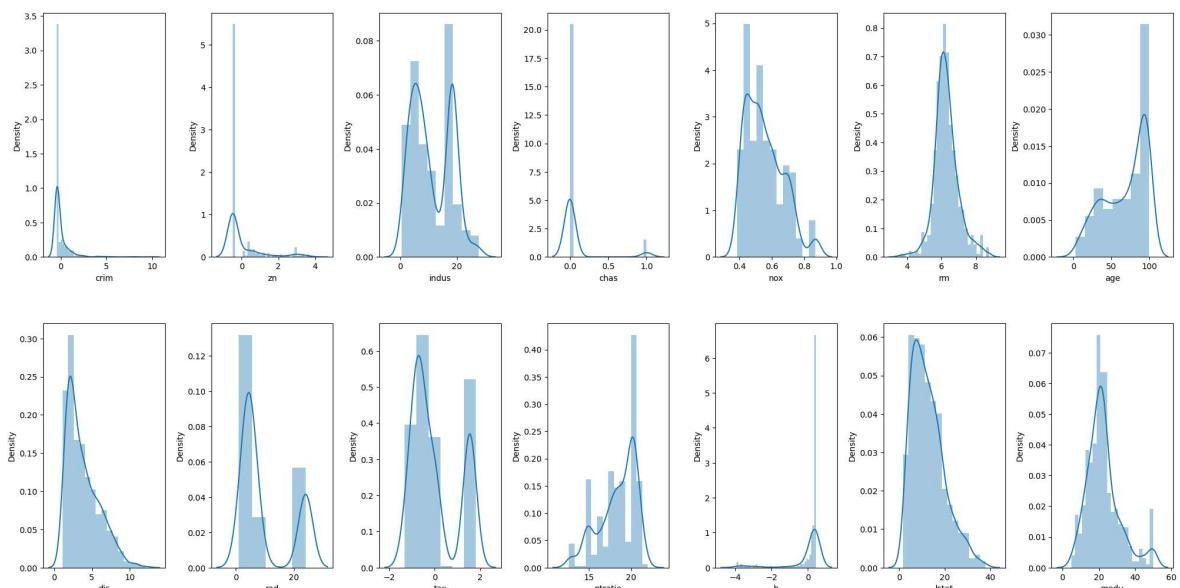
On peut revenir à la base de données initiale en y intégrant les données standardisées.

```
In [12]: for col in cols:
    data[col] = scaled_cols[col]
```

Reprendons la représentation graphique des distributions des données.

```
In [13]: fig, ax = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
ax = ax.flatten()

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



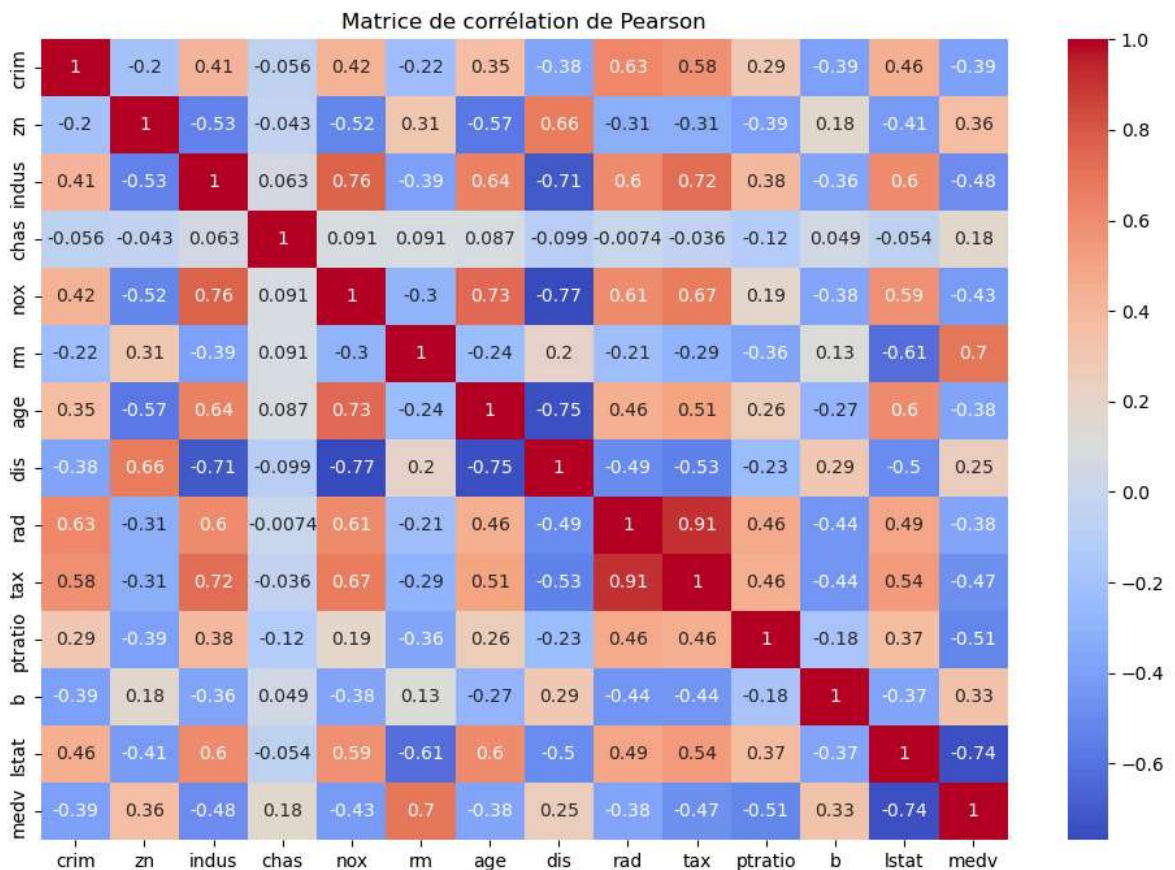
Même après normalisation, les colonnes '**crim**', '**zn**', '**tax**' et '**b**' ne suivent pas une distribution normale parfaite. Cependant, la standardisation de ces colonnes améliorera légèrement les performances du modèle.

### 3.3. Matrice de corrélation linéaire de Pearson

In [14]: `correlation_matrix = data.corr()`

```
plt.figure(figsize=(12, 8))
sns.heatmap(
    correlation_matrix,
    annot=True, # Afficher les valeurs
    cmap='coolwarm', # Palette de couleurs
)

plt.title("Matrice de corrélation de Pearson")
plt.show()
```



Nous nous concentrons principalement sur la variable cible, car il s'agit d'un problème de régression.

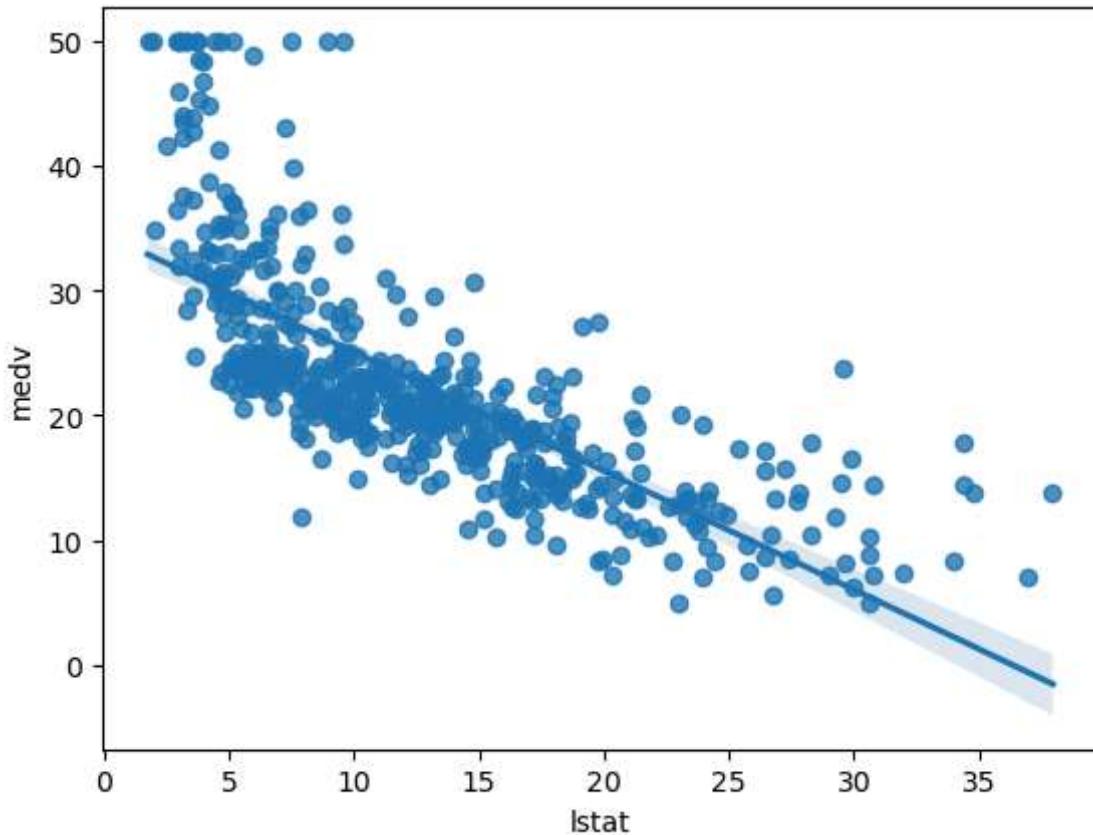
Cependant, nous pouvons également observer d'autres attributs fortement corrélés, notamment les colonnes 'tax' et 'rad'.

Nous éliminerons cette corrélation plus tard en ignorant l'une de ces variables.

De plus, nous afficherons 'lstat' et 'rm' afin de mettre en évidence leur corrélation avec la variable cible 'medv'.

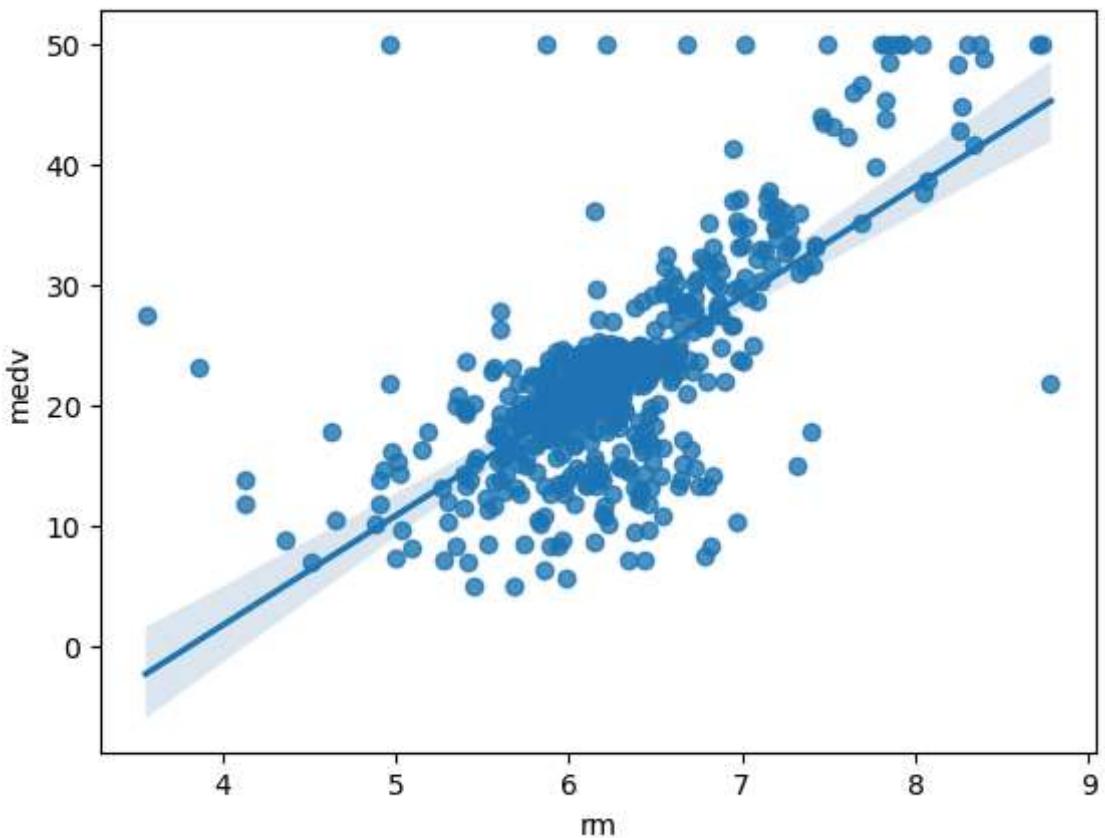
### 3.3. Relations entre la variable cible et les variables qui lui sont fortement corrélées

```
In [15]: sns.regplot(y=data['medv'], x=data['lstat'])
plt.show()
```



Ici, le prix des maisons diminue avec l'augmentation du pourcentage de la population appartenant à une catégorie socio-économique inférieure (*lstat*). Il y a donc une corrélation négative..

```
In [16]: sns.regplot(y=data['medv'], x=data['rm'])
plt.show()
```



Ici, le prix des maisons augmente avec l'augmentation du nombre moyen de pièces (*rm*). Il y a donc une corrélation positive.

## 4. Modélisation

### 4.1. Préparation des données

```
In [17]: X = data.drop(columns = ['medv', 'rad'], axis=1) # 'rad' est retirée pour éviter
y = data['medv']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

```
In [18]: # Modélisation : on peut définir une fonction qui entraîne le modèle et fait les
def train(model, X, y):
    # Entraînement du modèle
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
    model.fit(X_train, y_train)

    # Prédiction sur les données d'entraînement
    pred = model.predict(X_test)

    # Validation croisée du modèle
    cv_score = cross_val_score(model, X, y, scoring='neg_mean_squared_error', cv=5)
    cv_score = np.abs(np.mean(cv_score))

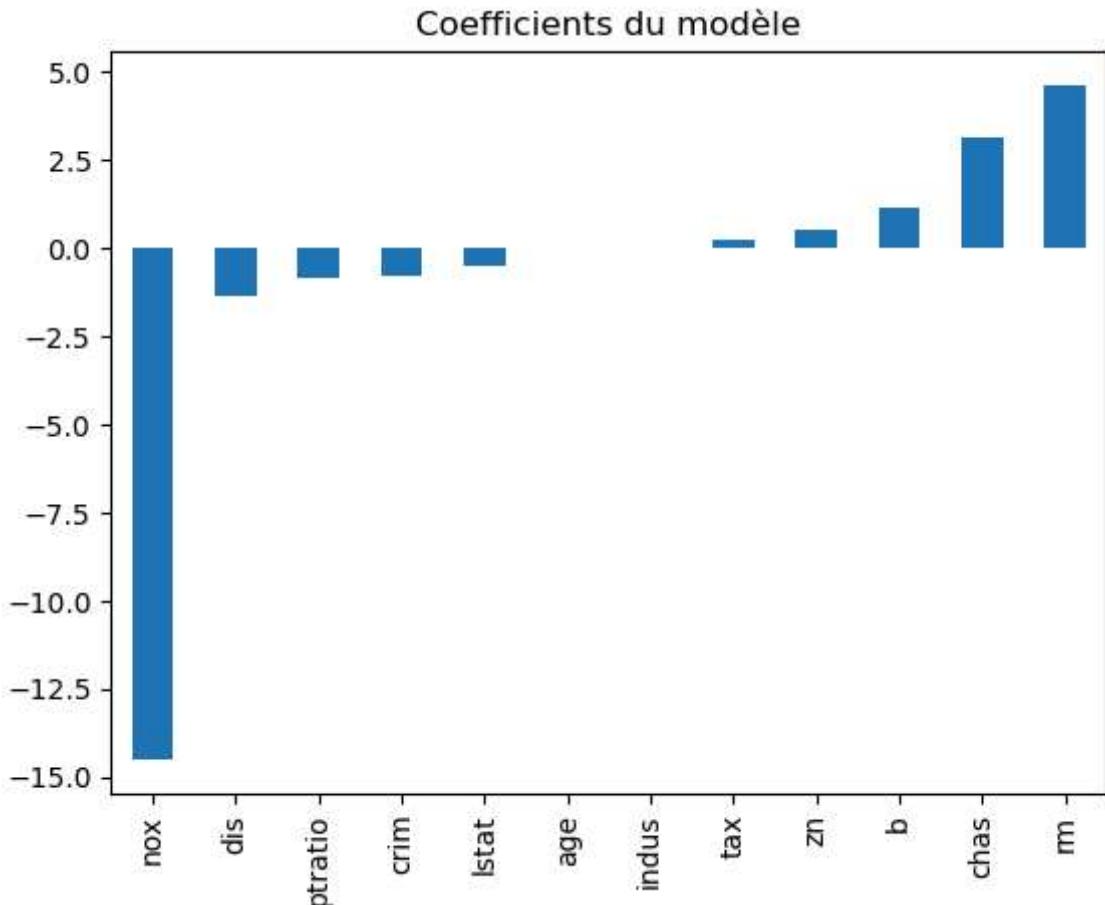
    print("Rapport du modèle")
    print("RMSE :", np.sqrt(mean_squared_error(y_test, pred)))
    print("R2 :", r2_score(y_test, pred))
    print('CV Score :', cv_score)
```

## 4.2. Application de la régression linéaire

```
In [19]: model = LinearRegression()
train(model, X, y)
coef = pd.Series(model.coef_, X.columns).sort_values()
coef.plot(kind='bar', title='Coefficients du modèle')
```

Rapport du modèle  
RMSE : 4.8991673318070905  
R2 : 0.6572494708117895  
CV Score : 35.69582283886881

```
Out[19]: <Axes: title={'center': 'Coefficients du modèle'}>
```



Seules les variables significatives ont leurs coefficients affichés (les variables *age* et *indus* ne sont pas significatives dans le modèle estimé). La variable *rm* présente un coefficient fortement positif, tandis que *nox* affiche un coefficient fortement négatif.

Les variables explicatives utilisées permettent d'expliquer 65.72% de la variabilité du prix médian des logements. Le score de validation croisée est de 35.69%. Il permet d'éviter les problèmes d'overfitting et d'underfitting. Si d'autres modèles sont entraînés, on pourrait comparer leurs performances et en choisir le meilleur. Cette comparaison sera faite dans un autre cas pratique.

L'objectif initial de cette analyse était de prédire le prix médian des maisons à Boston en utilisant la régression linéaire multiple classique. L'étude a montré que le modèle construit explique 65.72% de la variabilité des prix et offre une erreur moyenne de 4900 dollars environ (RMSE = 4.899). Entre autres, les variables *nombre moyen de pièces (rm)*,

*proportion de la population à faible revenu (lstat) et la concentration d'oxydes d'azote dans l'air (nox) se sont révélées être les plus significatives dans la détermination du prix des maisons.* En se servant de ce modèle, on pourrait prédire le prix d'un logement à Boston à partir de ses caractéristiques.