LE CAS

BOSTON HOUSING DATASET

ANTONIN LEMULLOIS

LE DATASET

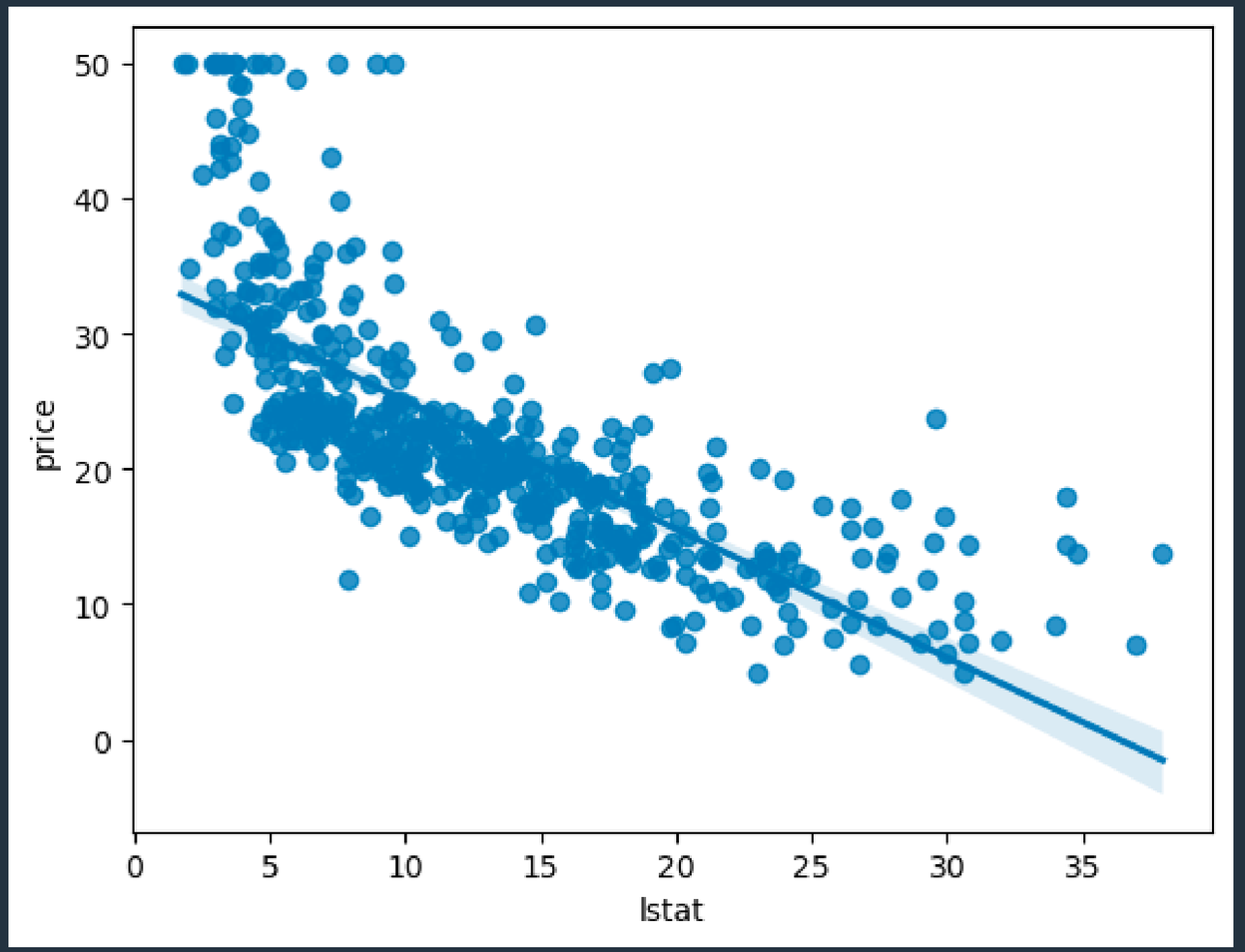
INFORMATIONS UTILES

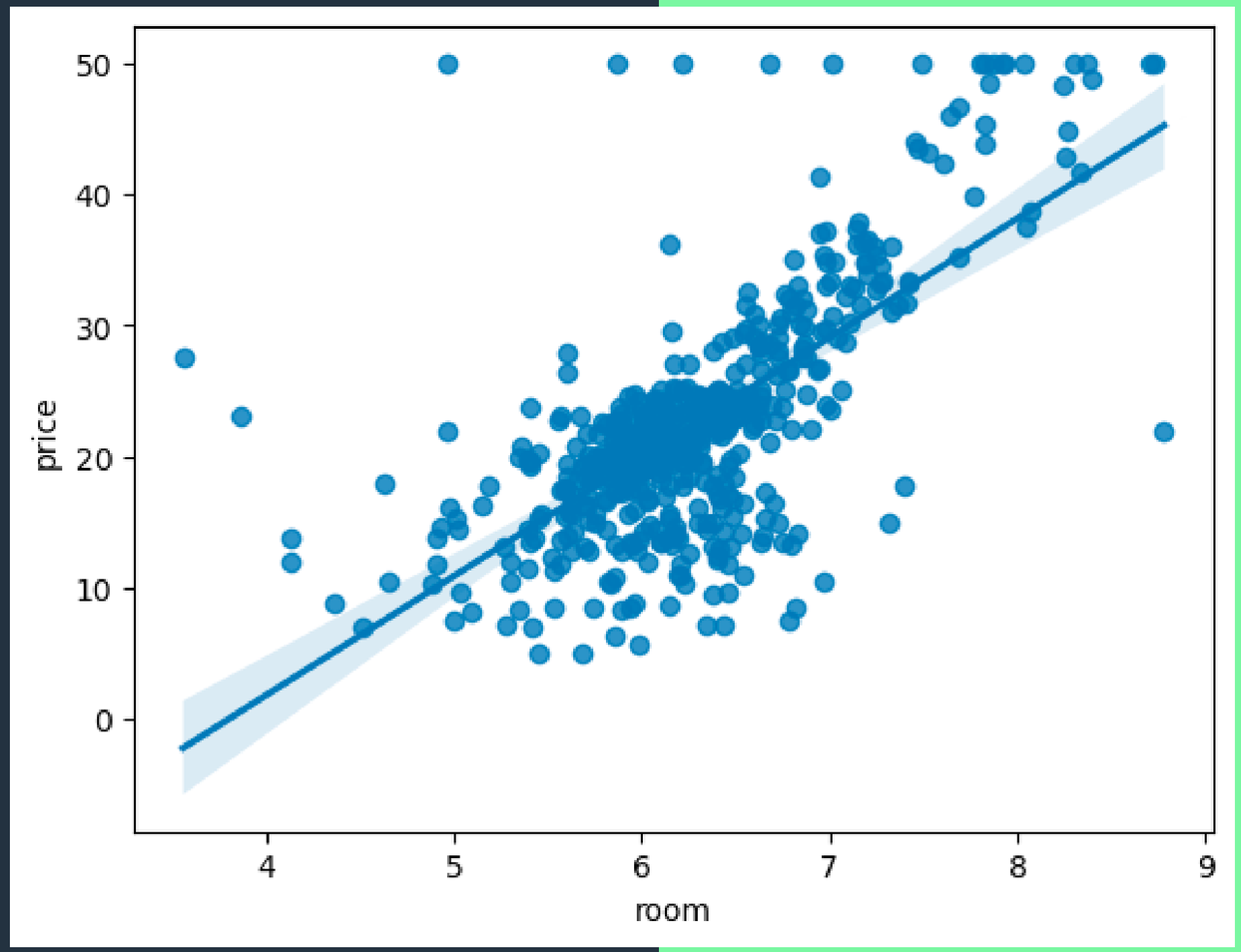
Miscellaneous Details

- **Origin**
  The origin of the boston housing data is **Natural**.
- **Usage**
  This dataset may be used for **Assessment**.
- **Number of Cases**
  The dataset contains a total of **506** cases.
- **Order**
  The order of the cases is **mysterious**.
- **Variables**
  There are **14** attributes in each case of the dataset. They are:

1. CRIM - per capita crime rate by town
2. ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS - proportion of non-retail business acres per town.
4. CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
5. NOX - nitric oxides concentration (parts per 10 million)
6. RM - average number of rooms per dwelling
7. AGE - proportion of owner-occupied units built prior to 1940
8. DIS - weighted distances to five Boston employment centres
9. RAD - index of accessibility to radial highways
10. TAX - full-value property-tax rate per $10,000
11. PTRATIO - pupil-teacher ratio by town
12. B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
13. LSTAT - % lower status of the population
14. MEDV - Median value of owner-occupied homes in $1000's

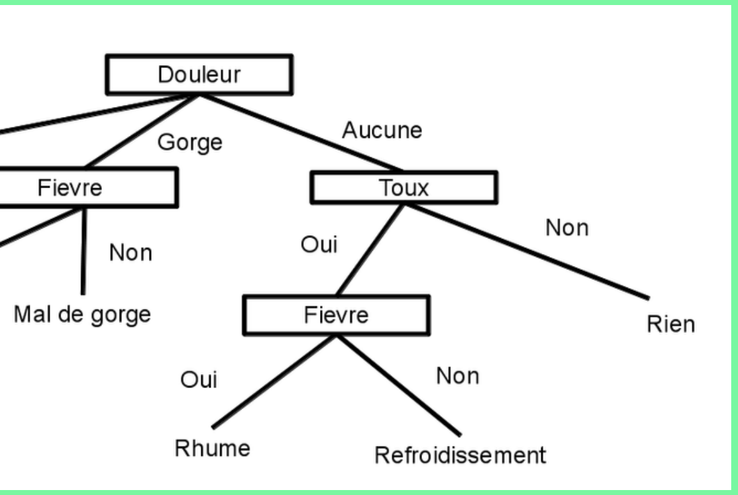
LE DATASET – GRAPHIQUES





LE MODÈLE UTILISÉ

GRADIENT BOOSTING REGRESSOR



Le Gradient Boosting Regressor est un algorithme d'apprentissage supervisé puissant et flexible, connu pour sa **précision** élevée. Il crée un modèle prédictif sous forme d'un **ensemble de modèles faibles**, généralement des arbres de décision. En utilisant la technique du boosting, chaque nouvel arbre tente de **corriger** les erreurs faites par l'ensemble des modèles précédents.

Cet algorithme offre plusieurs avantages, comme une grande précision, la capacité à gérer des variables de différents types et la prévention de l'overfitting grâce à l'approche de "**shrinkage**". Celle-ci consiste à réduire l'impact de chaque nouvel arbre en le multipliant par un "taux d'apprentissage", ce qui ralentit le processus d'apprentissage et rend le modèle plus robuste.

Hyperparamètres sensibles.

HYPERPARAMÈTRES

learning_rate : Imaginez ce paramètre comme le rythme auquel notre modèle apprend. Plus le chiffre est petit, plus le modèle prend son temps pour apprendre de chaque arbre avant de passer au suivant.

n_estimators : C'est le nombre d'arbres que nous laissons à notre modèle pour apprendre. Plus on a d'arbres, plus on peut avoir de possibilités.
Cependant, trop d'arbres peut aussi conduire notre modèle à l'overfitting.

max_depth : C'est la quantité d'information que chaque arbre peut apprendre. Une profondeur plus élevée signifie que chaque arbre peut apprendre plus de détails, mais risque aussi de trop apprendre des données d'entraînement. Nous avons choisi une profondeur de 6, pour équilibrer entre la capture des détails dans les données et la prévention de l'overfitting.

min_samples_split & min_samples_leaf : Ces paramètres sont comme des garde-fous qui empêchent chaque arbre de grandir trop.

HYPERPARAMÈTRES

```
model = GradientBoostingRegressor(
    learning_rate=0.01915003072140585 7,
    n_estimators=500,
    max_depth=6,
    min_samples_split=2,
    min_samples_leaf=1
)
```

Ces hyperparamètres ont été choisis par Optuna (alternative au grid search).
Qui permet de choisir les meilleures hyperparamètres au niveau d'une métric (r2).

PRÉPROCESSING

Application d'un RobustScaler, car les données contiennent pas mal de disparités entre les quartiles & les min/max.

	crime	zone	indus	river	nox	room	age	district	rad	tax	ptratio	black_population	lstat	price
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.653063	22.532806
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.141062	9.197104
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.730000	5.000000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.950000	17.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.360000	21.200000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.955000	25.000000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.970000	50.000000

FEATURES IMPORTANCE

```
room: 0.5174642347874622
lstat: 0.2750747773782461
district: 0.07523545594965506
crime: 0.04216574093669256
nox: 0.01990420492121404
black_population: 0.019889913500679184
tax: 0.015605903429430789
ptratio: 0.014825369334410895
age: 0.011362448049438305
indus: 0.004274518708121225
rad: 0.0032604277607313223
river: 0.0005898859886744348
zone: 0.00034711925524386804
```

Suite à l'évaluation des features sur mon modèle, j'ai sélectionné quelques features qui ne vont pas influencer la prédiction et permettre plus ou moins de maintenir la véracité de mon modèle.

```python
df_dropped = df.drop(columns=["price", "zone", "river", "rad"])

X = df_dropped
y = df["price"]
```

SCORE FINAL

```
Model R² Score: 0.942
```

Un score plus que correct, qui peut toutefois révéler un problème (des analyses plus poussées doivent être réalisées).
Les essais avec divers modèles peuvent être assez longs, PyCaret permet de choisir le meilleur modèle mais cause des problèmes d'environnement.