Master Big Data Analytics & Smart Systems

RAPPORT

Final project of Boston House Price Prediction

Réaliser par :

Elhagouchi Halima

Encadrer par :

Pr. Afaf BOUHOUTE

Année universitaire 2023/2024

TABLE DE MATIÈRE

Introduction

1 Data mining

1.1 Définition:

L'exploration de données (ou Data Mining en anglais) est le processus de découverte de modèles, de tendances et d'informations précieuses à partir de vastes ensembles de données en utilisant diverses techniques telles que l'apprentissage automatique, l'analyse statistique et les systèmes de bases de données. L'objectif de l'exploration de données est d'extraire des connaissances précieuses et des informations exploitables à partir de données brutes, pouvant être utilisées pour la prise de décisions, la prédiction et l'optimisation dans divers domaines.

1.2 Étapes :

Le processus d'exploration de données implique généralement plusieurs étapes clés:

- **Collecte des données :** La première étape consiste à collecter des données pertinentes à partir de sources multiples, y compris des bases de données, des feuilles de calcul, des fichiers texte et des API externes. Ces données peuvent être structurées ou non structurées et peuvent être présentées sous différentes formes.

- **Prétraitement des données :** Une fois les données collectées, elles doivent être nettoyées et prétraitées pour garantir leur qualité et leur utilisabilité. Cette étape implique des tâches telles que la gestion des valeurs manquantes, la suppression des doublons, la normalisation des formats et la transformation des variables.

- **Analyse exploratoire des données (AED) :** À cette étape, les analystes explorent les données visuellement et statistiquement pour obtenir des

informations sur leurs caractéristiques, leur distribution et les relations entre les variables. Les techniques d'AED incluent les statistiques descriptives, la visualisation des données et l'analyse de corrélation.

- **Ingénierie des caractéristiques :** L'ingénierie des caractéristiques implique la sélection, la création ou la transformation des caractéristiques (variables) pour améliorer les performances des modèles d'apprentissage automatique. Cette étape peut inclure la réduction de la dimensionnalité, la mise à l'échelle des caractéristiques et la génération de nouvelles caractéristiques basées sur des connaissances du domaine.

- **Sélection de modèle :** Ensuite, les analystes choisissent des algorithmes d'apprentissage automatique ou des modèles statistiques appropriés en fonction de la nature des données et de la tâche à accomplir. Les algorithmes courants incluent les arbres de décision, les modèles de régression, les algorithmes de regroupement et les réseaux neuronaux.

- **Entraînement du modèle** : À cette étape, les modèles sélectionnés sont entraînés sur un sous-ensemble des données, appelé ensemble d'entraînement. Pendant l'entraînement, les modèles apprennent des modèles et des relations à partir des données, ajustant leurs paramètres pour minimiser les erreurs ou maximiser les métriques de performance.

- **Évaluation du modèle :** Après l'entraînement, les modèles sont évalués à l'aide d'un ensemble de données distinct, appelé ensemble de validation ou ensemble de test. Des métriques d'évaluation telles que l'exactitude, la précision, le rappel, le score F1 et l'aire sous la courbe ROC sont utilisées pour évaluer les performances des modèles et les comparer.

- **Déploiement du modèle :** Une fois un modèle satisfaisant identifié, il peut être déployé pour une utilisation réelle. Cela peut impliquer l'intégration du

modèle dans des systèmes existants, l'automatisation des prédictions et la surveillance de ses performances dans le temps.

- **Processus itératif :** L'exploration de données est souvent un processus itératif, où les analystes affinent et améliorent continuellement les modèles en fonction des commentaires et des nouvelles données. Cela peut impliquer le réentraînement des modèles, la mise à jour des caractéristiques ou l'exploration de différents algorithmes pour obtenir de meilleurs résultats.

En suivant ces étapes, les analystes peuvent utiliser efficacement les techniques d'exploration de données pour extraire des informations précieuses et des connaissances à partir de vastes ensembles de données, ce qui permet de prendre des décisions éclairées et d'optimiser les résultats dans divers domaines

2 Problématique

La problématique réside dans la prédiction des prix des maisons dans la région de Boston.

Avec des facteurs économiques et sociaux en constante évolution, il est essentiel de comprendre les déterminants des prix immobiliers pour les acheteurs, les vendeurs et les investisseurs.

La question clé est de savoir comment utiliser les données disponibles pour développer un modèle précis de prédiction des prix des maisons

3 Objective

L'objectif principal est de développer un modèle de prédiction des prix des maisons à Boston qui soit précis et fiable.

Ce modèle doit être capable de prendre en compte une variété de facteurs, tels que la taille de la propriété, la qualité de l'environnement, la proximité des commodités

et d'autres caractéristiques pertinentes, pour fournir des estimations de prix précises.

4 Solution

Nous allons utiliser les données sur les prix des maisons dans la région de Boston, disponibles dans le célèbre ensemble de données "Boston Housing Dataset".

En utilisant des techniques d'apprentissage automatique, nous allons explorer, nettoyer et analyser ces données pour identifier les facteurs les plus influents sur les prix des maisons.

Ensuite, nous allons développer et évaluer plusieurs modèles de prédiction des prix des maisons, tels que les régressions linéaires, les forêts aléatoires et les réseaux de neurones, pour trouver celui qui offre les meilleures performances prédictives.

4.1 Etapes :

4.1.1 Lire les données :

"Boston House Prices" (ou "Boston Housing Dataset") contient des informations sur différents aspects qui pourraient influencer les prix des maisons dans la région de Boston.

Voici une explication de chaque colonne (variable) dans ce jeu de données :

nom	Explication
CRIM	Taux de criminalité par habitant dans la ville où se trouve la maison.
ZN	Proportion de terrains résidentiels zonés pour des lots de plus de 25 000 pieds carrés (environ 2322 mètres carrés)
INDUS	Proportion de superficies commerciales non commerciales par ville
CHAS	Variable factice (dummy) indiquant si la maison est située sur la rive du fleuve Charles (1 si oui, 0 sinon)

NOX	Concentration d'oxydes d'azote (parties par million) dans l'air
RM	Nombre moyen de pièces par logement
AGE	Proportion de logements occupés par leur propriétaire construits avant 1940
DIS	Distances pondérées vers cinq centres d'emploi de Boston
RAD	Indice d'accessibilité aux autoroutes radiales
TAX	Taux d'imposition foncière plein d'une valeur de 10 000 dollars
PTRATIO	Ratio élèves-professeurs par ville
B	Proportion de personnes afro-américaines par ville
LSTAT	Pourcentage de statut inférieur de la population
PRICE	Valeur médiane des maisons occupées par leur propriétaire en milliers de dollars

4.1.2 Importer le Dataset

Cette étape consiste à importer les bibliothèques Python nécessaires, telles que Pandas et NumPy…, pour manipuler les données,

```
# Load libraries
import numpy
from numpy import arange
from matplotlib import pyplot
from pandas import read_csv
from pandas import set_option
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error

```

Ainsi que le dataset lui-même. Le dataset peut être chargé à partir d'un fichier CSV, d'une base de données ou d'une autre source de données

```
1
2 dataset=read_csv('boston.csv')
3
```

```
1 dataset
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273.0	21.0	396.90	7.88	11.9

506 rows × 14 columns

Autre méthode pour charger le dataset à partir du **load_boston**

```
from sklearn.datasets import load_boston
import pandas as pd

# Load the Boston dataset
boston_dataset = load_boston()

# Create a DataFrame with the data
boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)

# Add the target variable to the DataFrame
boston['PRICE'] = boston_dataset.target

# Display the first few rows of the DataFrame
boston.head()

```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

4.1.3 Définie le type et le rôles de ces variables (input ou output)

4.1.3.1 Type :

the type of every variable

```
data.dtypes
```

```
CRIM       float64
ZN         float64
INDUS      float64
CHAS       float64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD        float64
TAX        float64
PTRATIO    float64
B          float64
LSTAT      float64
PRICE      float64
dtype: object
```

```
# Identifying the unique number of values in the dataset
data.nunique()
```

```
CRIM       504
ZN          26
INDUS       76
CHAS         2
NOX         81
RM         446
AGE        356
DIS        412
RAD          9
TAX         66
PTRATIO     46
B          357
LSTAT      455
PRICE      229
dtype: int64
```

On Remarque qu'on a deux variable CHAS et RAD qui ont de type categorienne et l'autre sont des variables numerique

On va stocker les valeurs numériques dans un variable nommé **numeric**

numeric data

```
numeric=boston.drop(['CHAS','RAD'],axis=1)
numeric
```

	CRIM	ZN	INDUS	NOX	RM	AGE	DIS	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.538	6.575	65.2	4.0900	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.469	6.421	78.9	4.9671	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.469	7.185	61.1	4.9671	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.458	6.998	45.8	6.0622	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.458	7.147	54.2	6.0622	222.0	18.7	396.90	5.33	36.2

et les valeurs categrorriennes dans un variable nomme **categorienne**

```
categorienne=boston[['CHAS','RAD']]
categorienne
```

	CHAS	RAD
0	0	1
1	0	2
2	0	2
3	0	3
4	0	3
...	...	...
501	0	1
502	0	1
503	0	1
504	0	1
505	0	1

4.1.3.2 Le Rôle :

Dans l'ensemble de données sur le logement à Boston, les variables peuvent être divisées en deux catégories principales : les variables d'entrée (input variables) et les variables de sortie (output variables).

Variables d'entrée (input variables) :

CRIM, ZN, INDUS, CHAS, NOX, RM, AGE, DIS, RAD, TAX, PTRATIO, B LSTAT

Ces variables sont utilisées comme entrées dans un modèle prédictif pour estimer la valeur médiane des maisons occupées par leur propriétaire (PRICE). Elles décrivent diverses caractéristiques des quartiers de Boston qui pourraient influencer le prix des maisons.

Variable de sortie (output variable) :

PRICE

La variable PRICE est la variable de sortie ou la cible que le modèle tente de prédire. Elle représente la valeur médiane des maisons occupées par leur propriétaire en milliers de dollars.

4.1.4 Statistiques sur chaque attribut numérique(mean, quartile, min, max, …)

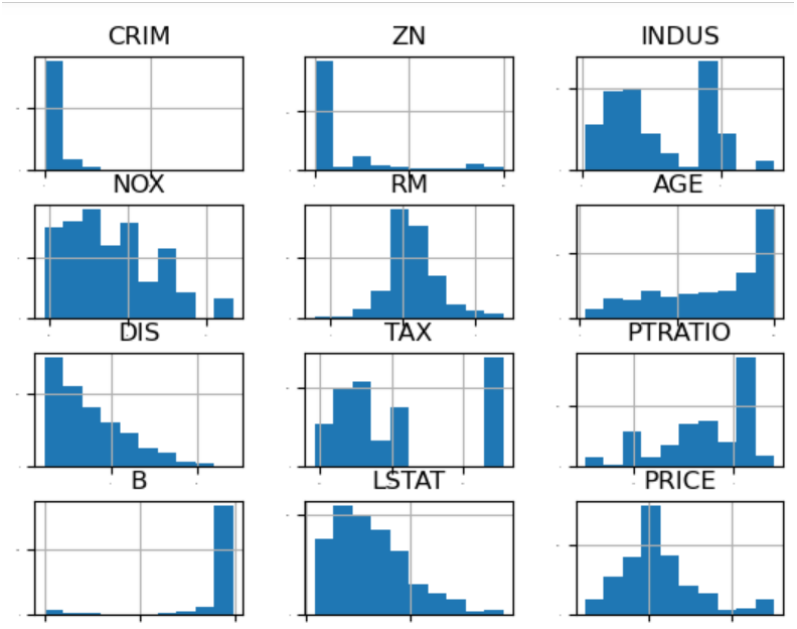
Statistics over each numerical attribute (mean, quartile, min, max, …)

```
# Viewing the data statistics
numeric.describe()
```

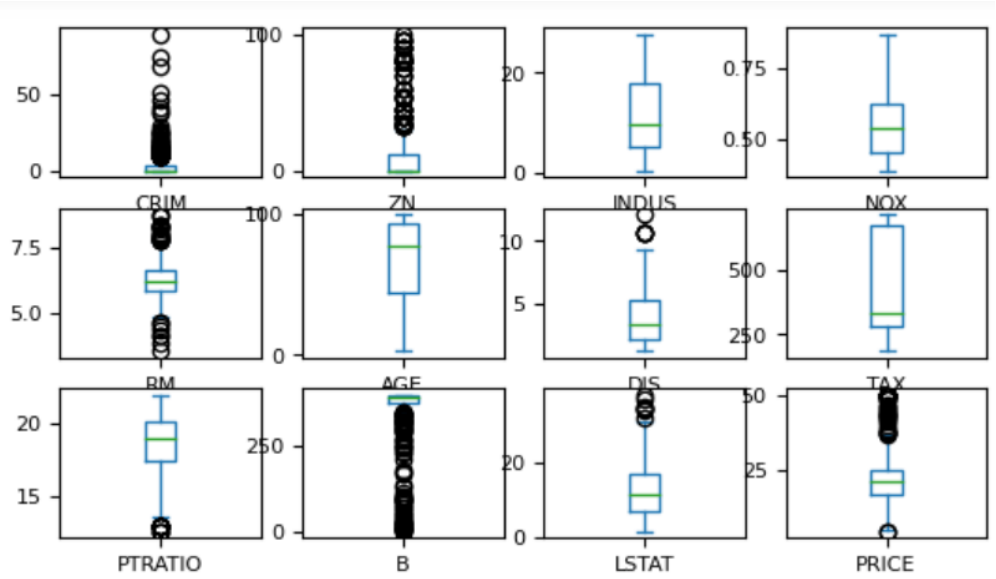
	CRIM	ZN	INDUS	NOX	RM	AGE	DIS	TAX	PTRATIO	E
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.554695	6.284634	68.574901	3.795043	408.237154	18.455534	356.674032
std	8.601545	23.322453	6.860353	0.115878	0.702617	28.148861	2.105710	168.537116	2.164946	91.294864
min	0.006320	0.000000	0.460000	0.385000	3.561000	2.900000	1.129600	187.000000	12.600000	0.320000
25%	0.082045	0.000000	5.190000	0.449000	5.885500	45.025000	2.100175	279.000000	17.400000	375.377500
50%	0.256510	0.000000	9.690000	0.538000	6.208500	77.500000	3.207450	330.000000	19.050000	391.440000
75%	3.677083	12.500000	18.100000	0.624000	6.623500	94.075000	5.188425	666.000000	20.200000	396.225000

4.1.5 Data Visualizations

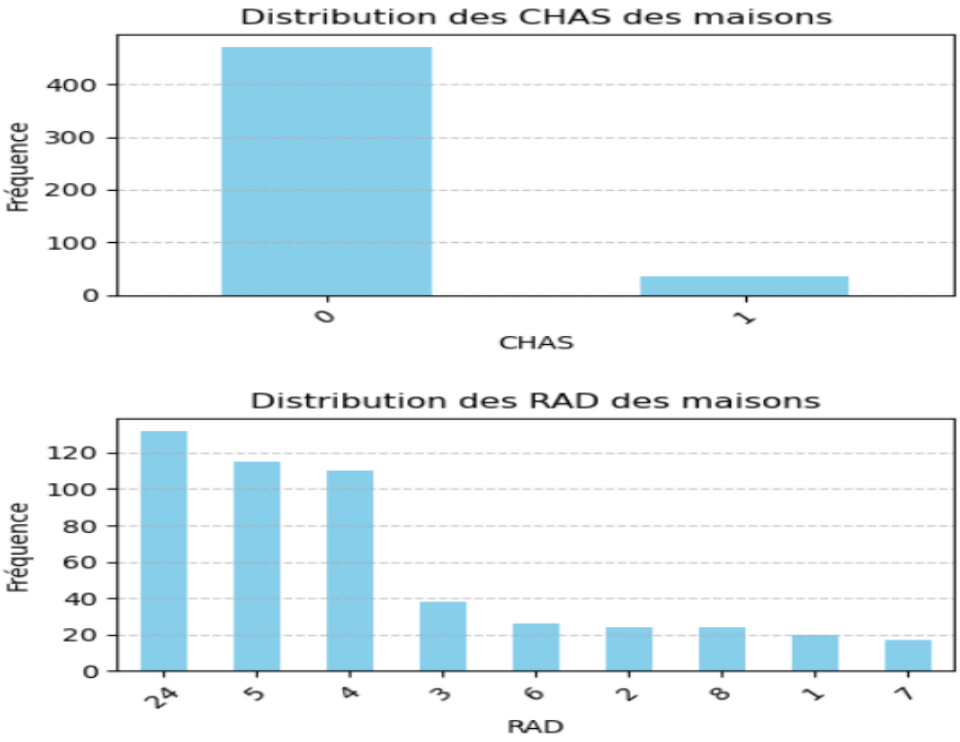
4.1.5.1 Histograms



4.1.5.2 BoxPlot

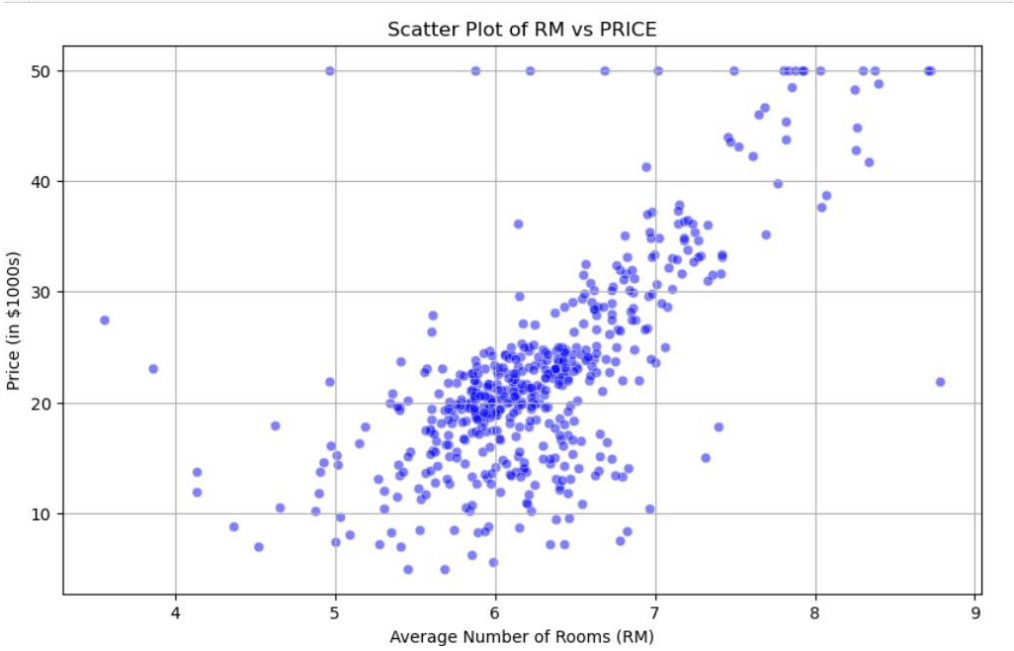


4.1.5.3 BarChart

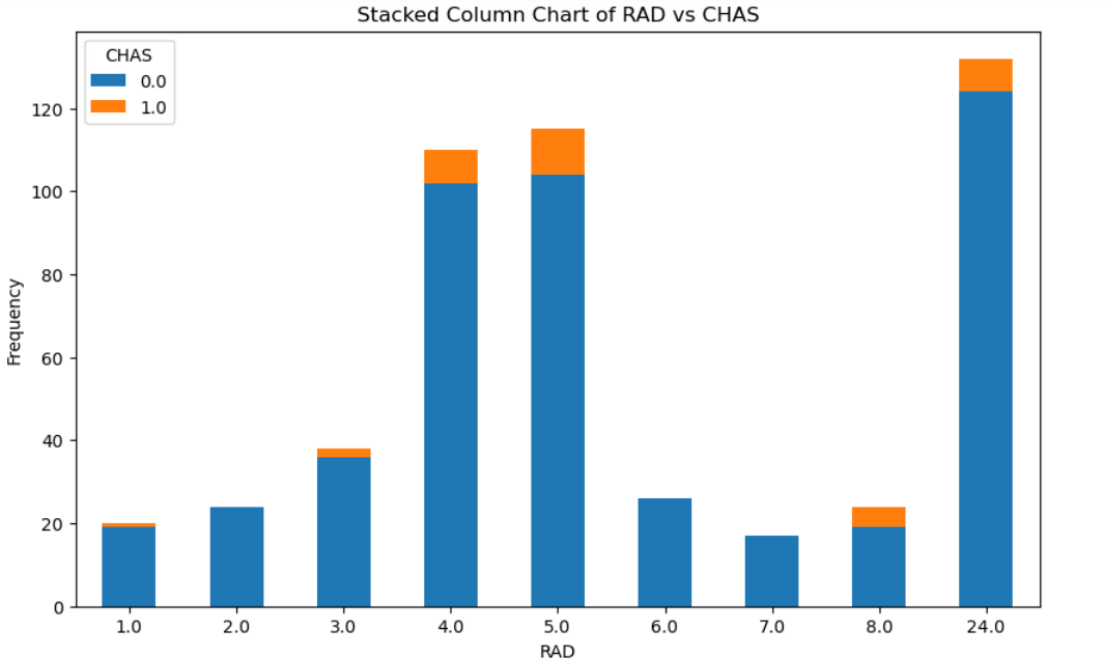


4.1.6 Analyse bi-variables

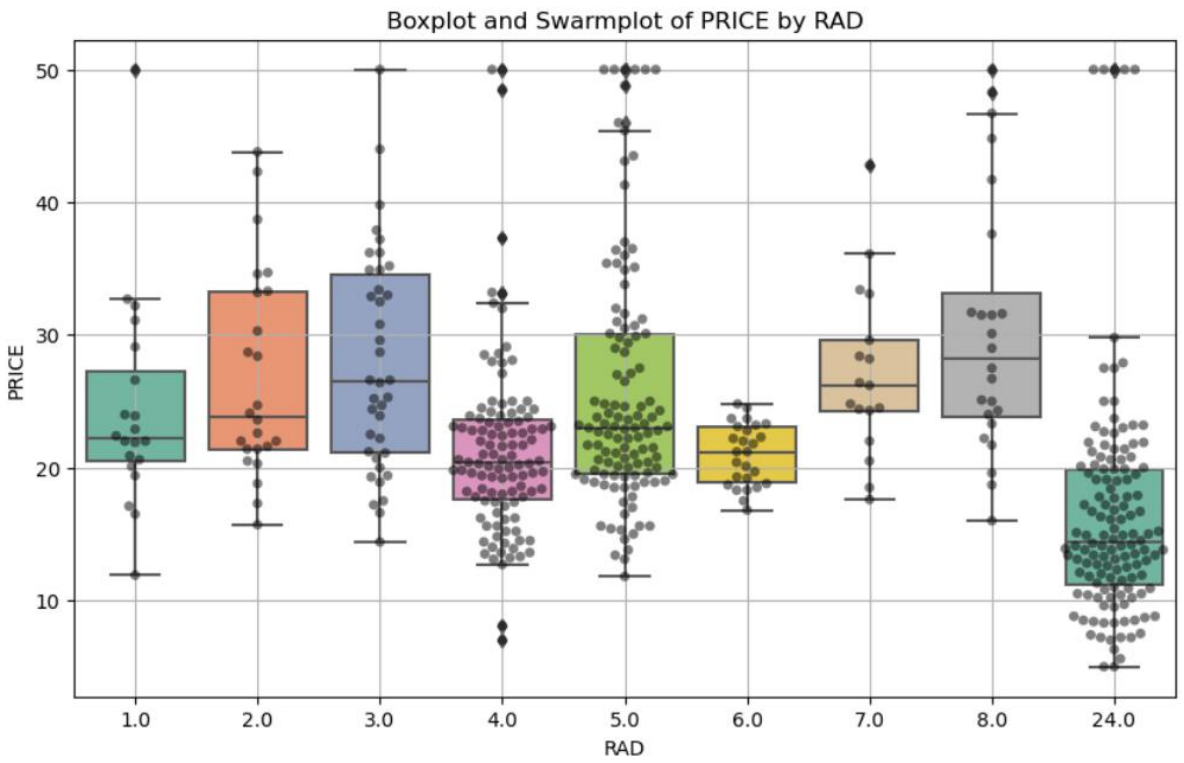
4.1.6.1 Continu et Continu (Matrice)



4.1.6.2 Catégorique et catégorique



4.1.6.3 Catégorique et continu



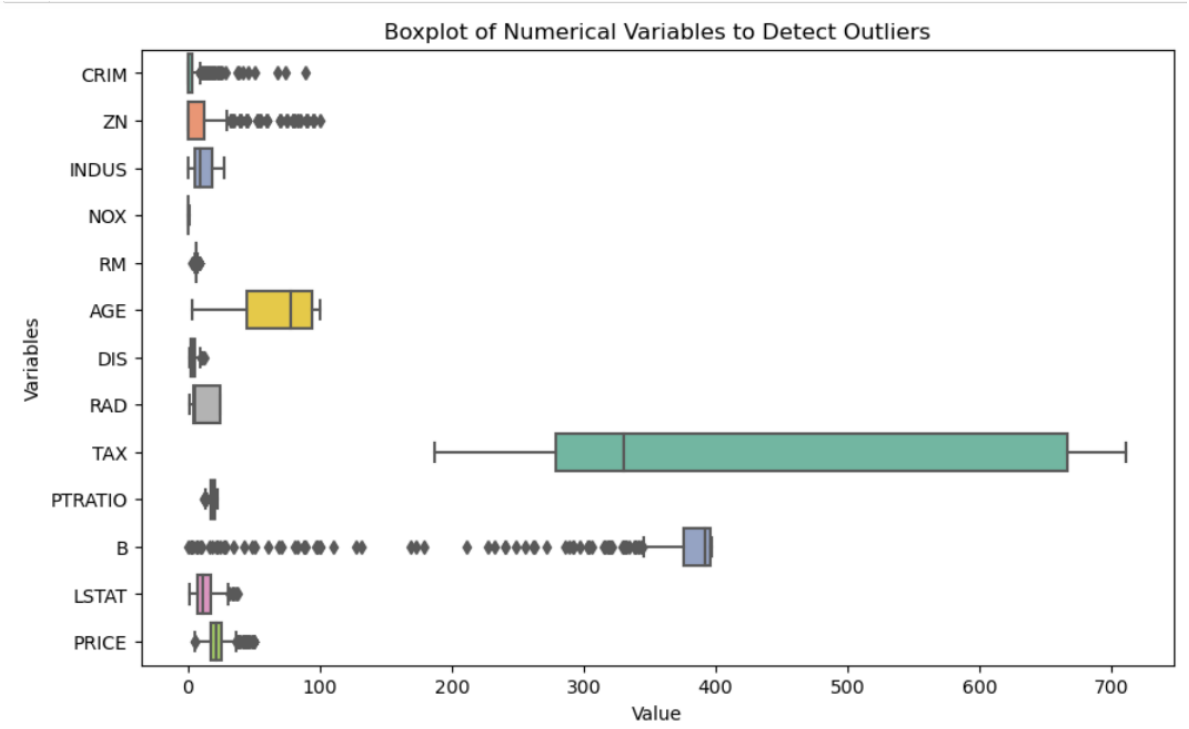
4.1.7 Détection / Traitement des valeurs manquantes

```
1  # Check for missing values
2  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
B          0
LSTAT      0
PRICE      0
dtype: int64
```

Pas de values manquantes alors on va contenu notre etude

4.1.8 Détection / Traitement des valeurs aberrantes



On a beaucoup des outliers, maintenant Si on choisir de

4.1.8.1 Supprimer les outliers

```
# Calculer les quartiles
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1

# Définir un seuil de suppression (par exemple, 1,5 fois l'écart interquartile)
threshold = 1.5

# Identifier les outliers
outliers = ((data < (Q1 - threshold * IQR)) | (data > (Q3 + threshold * IQR))).any(axis=1)

# Supprimer les outliers
clean_data = data[~outliers]

# Afficher les dimensions du jeu de données avant et après suppression des outliers
print("Dimensions du jeu de données original :", data.shape)
print("Dimensions du jeu de données après suppression des outliers :", clean_data.shape)

```

```
Dimensions du jeu de données original : (506, 14)
Dimensions du jeu de données après suppression des outliers : (268, 14)
```

Comme vous remarquer on à perdue beaucoup d'information alors ce n'est pas le bon choix

4.1.8.2 Imputation

L'imputation est un processus utilisé en statistiques et en analyse de données pour remplacer les valeurs manquantes ou aberrantes par des estimations basées sur les valeurs existantes dans un ensemble de données. L'objectif principal de l'imputation est de conserver autant d'informations que possible tout en minimisant les distorsions potentielles dans les analyses ultérieures.

Il existe plusieurs méthodes d'imputation, chacune avec ses propres avantages et limitations.

Voici quelques-unes des méthodes d'imputation les plus couramment utilisées :

- **Imputation par la moyenne :** Remplacer les valeurs manquantes ou aberrantes par la moyenne des valeurs existantes pour la même variable.
- **Imputation par la médiane :** Remplacer les valeurs manquantes ou aberrantes par la médiane des valeurs existantes pour la même variable. Cette méthode est moins sensible aux valeurs aberrantes que l'imputation par la moyenne.
- **Imputation par la mode :** Remplacer les valeurs manquantes ou aberrantes par le mode (la valeur la plus fréquente) des valeurs existantes pour la même variable. Cette méthode est principalement utilisée pour les variables catégorielles.

L'imputation est souvent un bon choix pour manipuler les valeurs aberrantes (outliers) dans les données pour plusieurs raisons :

- **Préserve les données existantes :** Supprimer simplement les valeurs aberrantes peut entraîner une perte d'informations potentiellement importantes dans les données. L'imputation permet de conserver ces données tout en les ajustant pour réduire leur impact sur l'analyse.

- **Maintient la taille de l'échantillon :** Si vous supprimez les valeurs aberrantes, vous pouvez réduire la taille de votre échantillon de données, ce qui peut affecter la précision de votre analyse. En imputant les valeurs aberrantes, vous pouvez maintenir la taille de l'échantillon et donc la robustesse de vos résultats.
- **Préserve la distribution des données :** L'imputation permet de remplacer les valeurs aberrantes par des estimations basées sur les autres valeurs présentes dans les données. Cela peut aider à préserver la distribution des données, ce qui est important pour de nombreuses analyses statistiques.

Application de l'imputation par la médian

```
1  from sklearn.impute import SimpleImputer
2
3  # Remplacer les valeurs aberrantes par la médiane
4  imputer = SimpleImputer(strategy='median')
5  data_imputed = imputer.fit_transform(boston)
6
```

```
1  data_imputed.shape
```

```
(506, 14)
```

4.1.9 Feature Engineering (Ingénierie des fonctionnalités)

La feature engineering, ou génie des caractéristiques en français, est le pro cessus de création et de sélection des caractéristiques (features) les plus per tinentes à partir des données brutes afin d'améliorer les performances des modèles d'apprentissage automatique.

C'est une étape cruciale dans le développement de modèles prédictifs effic aces. Voici quelques aspects importants de la feature engineering :

- **Sélection des caractéristiques:**Il s'agit de choisir les caractéristique s les plus informatives pour le modèle. Cela peut impliquer l'élimina

tion des caractéristiques redondantes ou peu utiles, ainsi que la sélec tion des caractéristiques qui ont le plus d'impact sur la variable cible.

- **Création de nouvelles caractéristiques** : Parfois, les caractéristique s existantes peuvent être combinées ou transformées pour créer de n ouvelles caractéristiques plus informatives. Par exemple, dans le cas de données temporelles, des caractéristiques telles que le jour de la s emaine, le mois ou la saison peuvent être extraites de la date.

- **Gestion des valeurs manquantes :** Les valeurs manquantes peuven t être préjudiciables aux performances des modèles. La façon dont le s valeurs manquantes sont traitées peut avoir un impact significatif s ur les résultats. Cela peut impliquer l'imputation des valeurs manqua ntes ou la création de variables indicatives pour indiquer si une vale ur est manquante ou non.

- **Normalisation et mise à l'échelle** : Les caractéristiques peuvent êtr e normalisées ou mises à l'échelle pour garantir qu'elles sont compar ables et qu'elles ont des ordres de grandeur similaires. Cela est partic ulièrement important pour les algorithmes sensibles à l'échelle, tels q ue les méthodes basées sur la distance.

- **Encodage des variables catégorielles :** Les variables catégorielles doivent souvent être encodées sous forme numérique pour être utilis ées dans les modèles d'apprentissage automatique. Cela peut impliqu er l'utilisation d'encodage one-hot, d'encodage ordinal ou d'autres mé thodes en fonction de la nature des données.

- **Extraction de caractéristiques à partir de données non structuré es :** Si les données incluent des informations non structurées telles q ue du texte, de l'image ou du son, des techniques spécifiques peuven t être utilisées pour extraire des caractéristiques pertinentes à partir d e ces données.

Pour créer une nouvelle colonne qui pourrait aider à améliorer les performances de votre modèle dans l'ensemble de données sur les prix des maisons à Boston, vous

pouvez envisager différentes techniques d'ingénierie des caractéristiques. Voici quelques idées :

- **Interaction entre les caractéristiques :** Créez de nouvelles caractéristiques en combinant des caractéristiques existantes. Par exemple, vous pourriez multiplier le nombre moyen de pièces (RM) par le pourcentage de population de statut inférieur (LSTAT) pour capturer l'interaction entre ces deux variables.
- **Caractéristiques polynomiales :** Générez des caractéristiques polynomiales à partir de caractéristiques existantes. Cela peut aider à capturer les relations non linéaires. Par exemple, vous pourriez créer des termes au carré (CRIM^2, RM^2, etc.) ou des termes d'interaction (CRIM * RM, AGE * DIS, etc.).
- **Discrétisation :** Convertissez les caractéristiques continues en caractéristiques catégorielles en les divisant en intervalles discrets. Cela peut parfois capturer des motifs qui ne sont pas évidents avec des variables continues.
- **Encodage de la cible :** Encodez les variables catégorielles en fonction de la moyenne de la variable cible. Par exemple, si vous avez une variable catégorielle CHAS (variable factice de la rivière Charles), vous pourriez l'encoder en fonction de la moyenne des PRIX pour chaque catégorie.

Voici un exemple de création d'une nouvelle caractéristique en multipliant deux caractéristiques existantes :

```
# Créer une nouvelle caractéristique 'RM_LSTAT' comme le produit de 'RM' et 'LSTAT'
boston['RM_LSTAT'] = boston['RM'] * boston['LSTAT']

# Afficher le DataFrame mis à jour
boston.head()

```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE	RM_LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0	32.74350
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6	58.68794
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7	28.95555
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4	20.57412
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2	38.09351

4.1.10 Modélisation et évaluation

```
# Spliting target variable and independent variables
X = boston.drop(['PRICE'], axis = 1)
y = boston['PRICE']
```

```
# Splitting to training and testing data

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random_state = 4)
```

4.1.10.1 Régression linéaire :

Forces :

- Simplicité : La régression linéaire est simple et facile à comprendre.
- Interprétabilité : Les coefficients de la régression linéaire fournissent des informations sur les relations entre les variables.
- Rapidité : L'entraînement et la prédiction sont généralement très rapides, surtout avec de grands ensembles de données.

Faiblesses :

- Hypothèse de Linéarité : La régression linéaire suppose que la relation entre les variables indépendantes et dépendantes est linéaire, ce qui n'est pas toujours le cas.
- Complexité Limitée : La régression linéaire peut ne pas capturer les relations complexes entre les variables.

```
# Import library for Linear Regression
from sklearn.linear_model import LinearRegression
# Create a Linear regressor
lm = LinearRegression()
# Train the model using the training sets
lm.fit(X_train, y_train)
```

```
LinearRegression()
```

```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
# Define k-fold cross-validation
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
accuracy = cross_val_score(lm, X, y, cv=kfold, scoring='accuracy')
precision = cross_val_score(lm, X, y, cv=kfold, scoring='precision')
recall = cross_val_score(lm, X, y, cv=kfold, scoring='recall')
f1 = cross_val_score(lm, X, y, cv=kfold, scoring='f1')
roc_auc = cross_val_score(lm, X, y, cv=kfold, scoring='roc_auc')
    # Print average scores
print("Average Accuracy:", accuracy.mean())
print("Average Precision:", precision.mean())
print("Average Recall:", recall.mean())
print("Average F1-score:", f1.mean())
print("Average ROC-AUC:", roc_auc.mean())
```

Entraîne le modèle de régression linéaire et faire le cross validation par le calcule de accuracy,precision,recel,roc_auc,f1-score

Warning :

```
Average Accuracy: nan
Average Precision: nan
Average Recall: nan
Average F1-score: nan
Average ROC-AUC: nan

C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: UserWarning: Scoring failed. The scor
e on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87, in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 242, in _score
```

Le message d'avertissement que vous avez reçu indique que la notation a é choué car la métrique de notation utilisée (précision) n'est pas prise en char ge pour les variables cibles continues.

vous devez choisir des mesures d'évaluation appropriées pour les tâches de régression

- **Erreur absolue moyenne (MAE) :** mesure les différences absolues moyennes entre les valeurs prédites et les valeurs réelles.

- **Erreur quadratique moyenne (MSE) :** mesure la moyenne des car rés des erreurs, en accordant plus de poids aux erreurs plus importan tes.
- **Erreur quadratique moyenne (RMSE)** : racine carrée de la MSE, qui est dans les mêmes unités que la variable cible.
- **R au carré (R2) :** mesure la proportion de la variance de la variable dépendante qui est prévisible à partir des variables indépendantes.
- **Adjusted *R*^2** :The adjusted R-squared compares the explanatory power of regression models that contain different numbers of predictors.
- **MAE :** C'est la moyenne de la valeur absolue des erreurs. Il mesure la différence entre deux variables continues, ici les valeurs réelles et prédites de y.

Evaluation

```
# Model prediction on train data
y_pred = lm.predict(X_train)
```

```
# Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.8135987749393635
Adjusted R^2: 0.8059007892436441
MAE: 2.6301912378340084
MSE: 14.030574860441543
RMSE: 3.74574089606336
```

```
# Predicting Test data with the model
y_test_pred = lm.predict(X_test)
# Model Evaluation
acc_linreg = metrics.r2_score(y_test, y_test_pred)
acc_linreg
```

```
0.7769282842909614
```

[redacted]Comparaison

```
# Visualizing the differences between actual prices and predicted values
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```

4.1.10.2 XGBoost Regressor

Forces :

- Performance : XGBoost dépasse souvent les autres algorithmes en termes de précision prédictive.

- Régularisation : XGBoost fournit des techniques de régularisation intégrées pour éviter la suradaptation.
- Flexibilité : XGBoost peut gérer différents types de données et est efficace pour les tâches de régression et de classification.

Faiblesses :

- Complexité de l'Ajustement : XGBoost a de nombreux hyperparamètres qui doivent être réglés correctement, ce qui peut être chronophage.
- Ressources Informatiques : XGBoost peut être intensif en ressources, surtout lorsqu'il s'agit de grands ensembles de données et de modèles complexes.
- Manque d'Interprétabilité : Tout comme la Forêt Aléatoire, les modèles XGBoost ne sont pas facilement interprétables.

[redacted]Evaluation

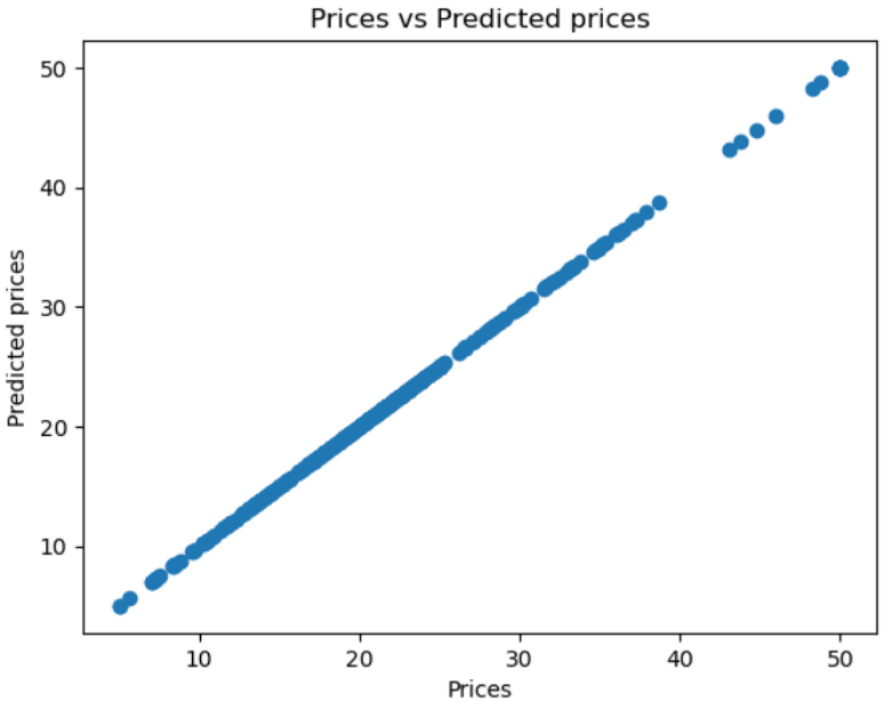
```
# Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.9999987286107582
Adjusted R^2: 0.9999986761050078
MAE: 0.00679692306087521
MSE: 9.569852305189996e-05
RMSE: 0.009782562192590443
```

```
#Predicting Test data with the model
y_test_pred = reg.predict(X_test)
# Model Evaluation
acc_xgb = metrics.r2_score(y_test, y_test_pred)
acc_xgb
```

```
0.876173581127551
```

[redacted]Compariason



4.1.10.3 SVM Regressor

Forces :

- Efficace dans les Espaces de Grande Dimension : SVM fonctionne bien même dans des espaces de grande dimension, ce qui le rend adapté aux ensembles de données avec de nombreuses caractéristiques.
- Noyaux Polyvalents : SVM peut utiliser différentes fonctions de noyau pour gérer différents types de données et de frontières de décision.
- Robuste à la Suradaptation : SVM est moins sujet à la suradaptation, surtout dans les espaces de grande dimension.

Faiblesses :

- Intensif en Calcul : SVM peut être lent à s'entraîner, surtout sur de grands ensembles de données.
- Paramètres d'Ajustement : SVM nécessite une sélection minutieuse des hyperparamètres et du choix du noyau, ce qui peut être difficile.

[redacted]Evaluation

```
# Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.650954001423209
Adjusted R^2: 0.6365391224259374
MAE: 2.9094521202061956
MSE: 26.272981903290262
RMSE: 5.125717696409963
```

```
# Predicting Test data with the model
y_test_pred = reg.predict(X_test)
acc_svm = metrics.r2_score(y_test, y_test_pred)
acc_svm
```

```
0.5978533914659678
```

[redacted]Comparison

4.1.10.4 Forêts aléatoires :

Forces :

- Apprentissage par Ensemble : La Forêt Aléatoire est une méthode d'ensemble qui combine plusieurs arbres de décision, ce qui conduit à une meilleure généralisation et robustesse.

- Gère les Relations Non Linéaires : La Forêt Aléatoire peut capturer des relations complexes et non linéaires entre les caractéristiques et la variable cible.
- Robuste à la Suradaptation : La Forêt Aléatoire est moins sujette à la suradaptation par rapport aux arbres de décision individuels.

Faiblesses :

- Manque d'Interprétabilité : Les modèles de Forêt Aléatoire ne sont pas facilement interprétables par rapport aux modèles linéaires.
- Temps d'Entraînement : La Forêt Aléatoire peut être coûteuse en calcul, surtout avec un grand nombre d'arbres et de caractéristiques.
- Consommation de Mémoire : Les modèles de Forêt Aléatoire peuvent consommer beaucoup de mémoire, surtout avec de grands ensembles de données et de nombreux arbres.

[redacted]Evaluation

```
# Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.9768064822174118
Adjusted R^2: 0.9760333649579922
MAE: 0.885163366336634
MSE: 2.0148998366336652
RMSE: 1.419471675178362
```

```
# Predicting Test data with the model
y_test_pred = reg.predict(X_test)
# Model Evaluation
acc_rf = metrics.r2_score(y_test, y_test_pred)
acc_rf
```

```
0.8797507762456317
```

[redacted]comapaison

4.1.10.5 Conclusion

Evaluation and comparision of all the models

```
models = pd.DataFrame({
    'Model': ['Linear Regression', 'Random Forest', 'XGBoost', 'Support Vector Machines'],
    'R-squared Score': [acc_linreg*100, acc_rf*100, acc_xgb*100, acc_svm*100]})
models.sort_values(by='R-squared Score', ascending=False)
```

	Model	R-squared Score
1	Random Forest	87.975078
2	XGBoost	87.617358
0	Linear Regression	77.692828
3	Support Vector Machines	59.785339

Comme conclusion le model Random Forest est le plus performant avec notre dataset des maison à boston

4.1.11 L'apprentissage ensembliste (Ensemble Learning)

Ensemble Learning est une technique d'apprentissage automatique qui combine plusieurs modèles d'apprentissage pour améliorer les performances prédictives et la stabilité de l'algorithme. Plutôt que de se fier à un seul modèle pour prendre des décisions, l'ensemble learning combine les prédictions de plusieurs modèles pour obtenir une prédiction plus robuste et précise. Les méthodes d'ensemble les plus couramment utilisées comprennent le bagging, le boosting et les méthodes de stacking.

4.1.11.1 Bagging (Bootstrap Aggregating)

Le bagging consiste à entraîner plusieurs modèles indépendants sur des sous-ensembles aléatoires de données, généralement obtenus par bootstrap (échantillonnage avec remplacement). Chaque modèle produit une prédiction et les prédictions de tous les modèles sont agrégées (par moyenne ou majorité) pour obtenir une prédiction finale. Les exemples les plus populaires de bagging sont les forêts aléatoires, où les modèles de base sont des arbres de décision.

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import BaggingRegressor

model = BaggingRegressor(base_estimator=KNeighborsRegressor(),
                         n_estimators=100)
model.fit(X_train, y_train)
model.score(X_test, y_test)

```

```
0.6514014064924735
```

```
model5=RandomForestRegressor(n_estimators=100 ,max_depth=10)
model5.fit(X_train,y_train)
model5.score(X_test,y_test)
```

```
0.8815263747440475
```

On va essayez différentes valeurs de max_depth

```
max_depth_values = [5, 10, 15, 20]
best_max_depth = None
best_score = 0

for max_depth in max_depth_values:
    model = RandomForestRegressor(n_estimators=100, max_depth=max_depth)
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    print(f"Score avec max_depth={max_depth}: {score}")
    if score > best_score:
        best_score = score
        best_max_depth = max_depth

print(f"Meilleur score obtenu avec max_depth={best_max_depth}: {best_score}")
```

```
Score avec max_depth=5: 0.85938142458748
Score avec max_depth=10: 0.8819748052401782
Score avec max_depth=15: 0.8824138703628316
Score avec max_depth=20: 0.8763804485423938
Meilleur score obtenu avec max_depth=15: 0.8824138703628316
```

Meilleur score 0.88 avec max depth 15

4.1.11.2 Boosting

Le boosting est une technique qui construit une séquence de modèles, où chaque modèle tente de corriger les erreurs des modèles précédents. Les modèles sont généralement faibles (appelés "apprenants faibles"), ce qui signifie qu'ils sont légèrement meilleurs que le hasard. À chaque itération, le modèle est entraîné sur un sous-ensemble pondéré des données, en mettant davantage l'accent sur les exemples mal classés. Des exemples courants de boosting incluent AdaBoost, Gradient Boosting, et XGBoost.

```
from sklearn.ensemble import AdaBoostRegressor,GradientBoostingRegressor
```

```
model_1=AdaBoostRegressor(n_estimators=100)
model_1.fit(X_train,y_train)
model_1.score(X_test,y_test)
```

```
0.8464214538524473
```

4.1.11.3 Stacking (Stacked Generalization)

Le stacking combine les prédictions de plusieurs modèles de base en utilisant un modèle métier (méta-modèle) pour apprendre comment combiner les prédictions de ces modèles. Les prédictions des modèles de base sont utilisées comme caractéristiques d'entrée pour le modèle métier. Le modèle métier peut être n'importe quel algorithme d'apprentissage supervisé, comme une régression linéaire, une forêt aléatoire, ou un réseau de neurones. Le stacking est plus

complexe à mettre en œuvre que le bagging ou le boosting, mais il peut offrir de meilleures performances prédictives dans certains cas.

```
from sklearn.ensemble import StackingRegressor
```

```
from sklearn.linear_model import SGDRegressor
from sklearn.tree import DecisionTreeRegressor
model1=RandomForestRegressor(random_state=0)
model2=XGBRegressor(random_state=0)
model3=KNeighborsRegressor(n_neighbors=2)
model_=StackingRegressor([('SGD',model1),
                          ('Tree',model2),
                          ("KNN",model3)],final_estimator=KNeighborsRegressor())
model_.fit(X_train,y_train)
model_.score(X_test,y_test)
```

```
0.8681242848620463
```

4.1.12 hyperparameter tuning

est le processus de recherche des meilleures valeurs pour les hyperparamètres d'un modèle d'apprentissage automatique afin d'optimiser sa performance sur un ensemble de données spécifique.

Voici comment le processus de tuning des hyperparamètres fonctionne généralement :

- **Sélection des hyperparamètres à optimiser :** Identifiez les hyperparamètres les plus importants à ajuster pour votre modèle. Ceux-ci peuvent varier selon le type de modèle que vous utilisez. Par exemple, pour un algorithme de machine learning comme les machines à vecteurs de support (SVM), les hyperparamètres incluent le noyau, le paramètre de régularisation (C), etc. Pour les réseaux de neurones, les hyperparamètres comprennent le taux d'apprentissage, le nombre de couches cachées, le nombre de neurones par couche, etc.

- **Définir l'espace de recherche :** Pour chaque hyperparamètre sélectionné, déterminez la plage de valeurs possibles à tester. Cela crée un "espace de recherche" pour chaque hyperparamètre.

- **Choisir une méthode de recherche :** Il existe plusieurs approches pour rechercher les meilleures combinaisons d'hyperparamètres :

  - *Recherche aléatoire :* Sélectionnez aléatoirement des combinaisons d'hyperparamètres dans l'espace de recherche et évaluez-les.

  - *Recherche en grille :* Évaluez systématiquement toutes les combinaisons possibles d'hyperparamètres dans l'espace de recherche.

  - *Recherche bayésienne :* Utilisez des méthodes probabilistes pour choisir les prochaines combinaisons d'hyperparamètres à évaluer, en se basant sur les performances des évaluations précédentes.

- **Évaluation de la performance :** Pour chaque combinaison d'hyperparamètres, entraînez le modèle sur un ensemble d'entraînement et évaluez sa performance sur un ensemble de validation. Vous pouvez utiliser des mesures telles que l'exactitude, la précision, le rappel, le F1-score, etc.
- **Sélection du meilleur modèle :** Identifiez la combinaison d'hyperparamètres qui donne les meilleures performances sur l'ensemble de validation.

- **Évaluation finale :** Évaluez le modèle avec les meilleurs hyperparamètres sur un ensemble de test indépendant pour obtenir une estimation impartiale de sa performance.

- **Validation croisée :** Dans certains cas, la validation croisée peut être utilisée pour évaluer plus rigoureusement les performances du modèle avec différentes combinaisons d'hyperparamètres.

4.1.12.1 Recherche en grille (Grid Search)

- La recherche en grille consiste à définir une grille d'hyperparamètres, où chaque dimension de la grille correspond à un hyperparamètre à régler.

- Pour chaque combinaison possible d'hyperparamètres dans la grille, le modèle est entraîné et évalué à l'aide d'une validation croisée.

- C'est une méthode exhaustive qui teste toutes les combinaisons spécifiées dans la grille.

- Bien qu'elle puisse être efficace pour un espace de recherche relativement petit, la recherche en grille peut devenir très coûteuse en calcul lorsque le nombre d'hyperparamètres et les valeurs à tester augmentent.

Le choix des valeurs de la grille

dépend de plusieurs facteurs, notamment de la nature de l'algorithme, du domaine de données et de la ressource informatique disponible. Voici quelques conseils pour choisir les valeurs de la grille :

- **Compréhension de l'algorithme :** Il est important de comprendre comment chaque hyperparamètre affecte le comportement de l'algorithme. Par exemple, pour un arbre de décision, la profondeur maximale de l'arbre (max_depth) contrôle la complexité de l'arbre, tandis que le nombre d'arbres dans une forêt aléatoire (n_estimators) détermine la diversité de la forêt.

- **Recherche préliminaire** : Effectuez une recherche préliminaire en utilisant des valeurs de départ courantes ou recommandées pour chaque hyperparamètre. Vous pouvez trouver ces valeurs dans la documentation de la bibliothèque ou du modèle que vous utilisez, ou dans des études de recherche sur des ensembles de données similaires.

- **Espace de recherche raisonnable :** Limitez l'espace de recherche en choisissant un ensemble de valeurs qui sont susceptibles de bien fonctionner pour votre problème. Par exemple, si vous savez que la

profondeur maximale de l'arbre ne devrait pas dépasser 10 en raison de la taille de votre ensemble de données, vous pouvez limiter la grille à des valeurs de 1 à 10 pour max_depth.

- **Ressources informatiques :** Considérez les ressources informatiques disponibles, telles que la puissance de calcul et le temps de calcul. Une grille trop dense avec des valeurs de paramètres peut prendre beaucoup de temps à explorer, alors assurez-vous que votre grille est gérable en termes de temps et de ressources.

- **Validation croisée** : Utilisez la validation croisée pour évaluer les performances de chaque combinaison de paramètres. Cela vous donnera des informations sur la manière dont chaque combinaison se comporte sur différentes partitions de votre ensemble de données, vous aidant ainsi à choisir les valeurs les plus prometteuses.

[redacted]Application

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
param_grid = {
    'n_estimators': [10, 50, 100],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]}
rf = RandomForestRegressor()
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=3)
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
best_params
```

```
{'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 100}
```

Explication :

- Définit une grille de valeurs pour les hyperparamètres que nous voulons optimiser. Dans cet exemple, nous spécifions différentes valeurs pour n_estimators, max_depth, et min_samples_split (ontrôle le nombre minimum

d'échantillons requis pour diviser un nœud interne dans un arbre de décision dans l'algorithme RandomForestRegressor)

- Initialise un objet GridSearchCV avec l'estimateur (estimator) RandomForestRegressor, la grille de paramètres (param_grid) définie précédemment, et le nombre de folds de validation croisée (cv=3) (La validation croisée (cross-validation en anglais) est une technique essentielle en apprentissage automatique pour évaluer les performances d'un modèle et pour sélectionner les meilleurs hyperparamètres de manière robuste. Elle consiste à diviser l'ensemble de données en sous-ensembles, à entraîner le modèle sur une partie de ces sous-ensembles (ensemble d'entraînement) et à le tester sur le reste (ensemble de validation ou de test).
- Exécute la recherche sur la grille en ajustant le modèle pour chaque combinaison d'hyperparamètres et en utilisant la validation croisée pour évaluer les performances.
- Récupère les meilleurs hyperparamètres trouvés par la recherche sur la grille.

```
{'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 100}
```

4.1.12.2 Recherche aléatoire (Random Search)

- Contrairement à la recherche en grille, la recherche aléatoire sélectionne aléatoirement des combinaisons d'hyperparamètres à évaluer, sans suivre une grille prédéfinie.

- Cette approche permet d'explorer un espace de recherche beaucoup plus large de manière plus efficace, car elle n'est pas limitée par une grille fixe.

- En raison de sa nature aléatoire, la recherche aléatoire peut souvent trouver de bonnes solutions avec moins d'itérations que la recherche en grille.

- Elle peut être particulièrement utile lorsque la relation entre les hyperparamètres et la performance du modèle est complexe ou lorsque certaines hyperparamètres sont plus importants que d'autres.

[redacted]Application

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
param_dist = {
    'n_estimators': randint(10, 100),
    'max_depth': [None, 10, 20],
    'min_samples_split': randint(2, 10)}
rf = RandomForestRegressor()
random_search = RandomizedSearchCV(estimator=rf, param_distributions=param_dist, n_iter=10, cv=3)
random_search.fit(X_train, y_train)
best_params = random_search.best_params_
best_params

```

```
{'max_depth': None, 'min_samples_split': 5, 'n_estimators': 98}
```

Explication :

- Définit une distribution pour chaque hyperparamètre que nous voulons optimiser. Ici, nous utilisons la fonction randint pour générer des valeurs aléatoires entre des bornes spécifiées
- Exécute la recherche aléatoire en ajustant le modèle pour chaque combinaison d'hyperparamètres générée aléatoirement et en utilisant la validation croisée pour évaluer les performances.
- Récupère les meilleurs hyperparamètres trouvés par la recherche aléatoire.

```
{'max_depth': None, 'min_samples_split': 5, 'n_estimators': 98}
```

[redacted]Comparaison de Performance

```python
# Modèle avec les hyperparamètres de la recherche sur la grille
model_grid = RandomForestRegressor(max_depth=10, min_samples_split=2, n_estimators=100)
model_grid.fit(X_train, y_train)
y_pred_grid = model_grid.predict(X_test)
# Modèle avec les hyperparamètres de la recherche aléatoire
model_random = RandomForestRegressor(max_depth=8, min_samples_split=8, n_estimators=71)
model_random.fit(X_train, y_train)
y_pred_random = model_random.predict(X_test)
#score
acc_grille = metrics.r2_score(y_test, y_pred_grid)
acc_grille
acc_random = metrics.r2_score(y_test, y_pred_random)
acc_random
#affichage
print("Performance du modèle avec Grid Search:")
print("Grillesearch__Srore= ",acc_grille)
print("\nPerformance du modèle avec Random Search:")
print("random_seach__score",acc_random)
```

```
Performance du modèle avec Grid Search:
Grillesearch__Srore=  0.8276302230753972

Performance du modèle avec Random Search:
random_seach__score 0.829358741567555
```

Explication :

- Utilise le modèle entraîné pour faire des prédictions sur l'ensemble de données de test X_test et stocke les prédictions dans y_pred_grid.
- Entraîne le modèle aléatoire sur les mêmes données d'entraînement.
- Calcule le coefficient de détermination (R^2) pour les prédictions du modèle de grille et aléatoire et les affichées.

Conclusion

En conclusion, j'ai mené à bien une analyse complète pour prédire les prix des maisons à Boston. Grâce à l'analyse exploratoire des données, au prétraitement des données, à l'évaluation des modèles et à l'exploration des méthodes d'ensemble, j'ai obtenu des insights précieux sur le jeu de données et développé des modèles prédictifs.

En comparant les performances des différents modèles à l'aide de métriques appropriées, j'ai identifié les points forts et les points faibles, fournissant ainsi une base pour une prise de décision éclairée.

Ce projet me fournit des connaissances et des outils précieux pour prédire efficacement les prix des maisons, contribuant ainsi à une meilleure prise de décision dans le domaine de l'immobilier.

Références

- ***https://www.talend.com/fr/resources/what-is-data-mining/***

- ***https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205***

- ***https://builtin.com/articles/feature-engineering#:~:text=Feature%20engineering%20is%20a%20machine,while%20also%20enhancing%20model%20accuracy.***