INTODUCTION A l'IA ET LA ROBOTIQUE
2A SIE

REDIGE PAR:

Arnold NGNOUBA NDIE TCHEGOU

RAPPORT DE PROEJET APPRENTISSAGE MACHINE (ML)

Prédiction des prix des maisons à partir des caractéristiques disponibles dans le jeu de données "California Housing", à l'aide des algorithmes d'apprentissage Supervisé.

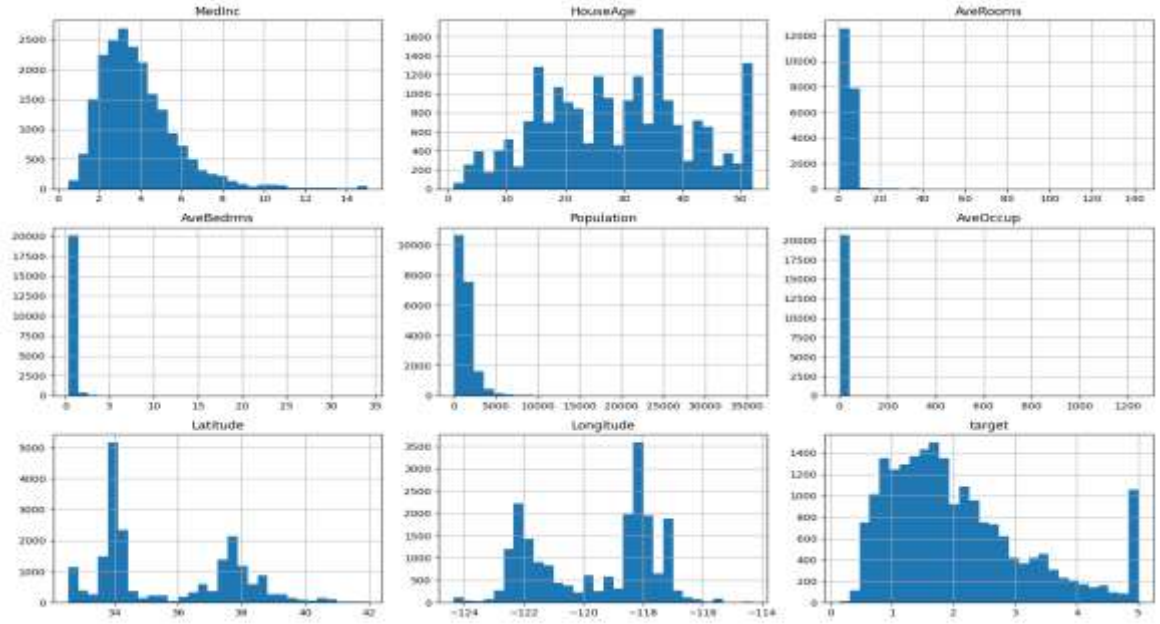
1. Définition du problème et méthodologie

Ce projet vise à prédire les prix des maisons dans la région de Californie en utilisant des techniques d'apprentissage supervisé. Cette tâche est formulée comme un problème de régression, où l'objectif est de prédire des valeurs continues représentant les prix des maisons en fonction de diverses caractéristiques. Pour ce fait, nous avons utilisé la méthodologie suivante :

- *Collecte des Données*
- *Exploration et Prétraitement des Données*
- *Sélection des algorithmes de régression*
- *Entraînement et Comparaison des modèles*
- *Sélection du meilleur modèle et optimisation de ses hyper paramètres*
- *Explication de l'importance des caractéristiques (features) pour la prédiction des prix.*

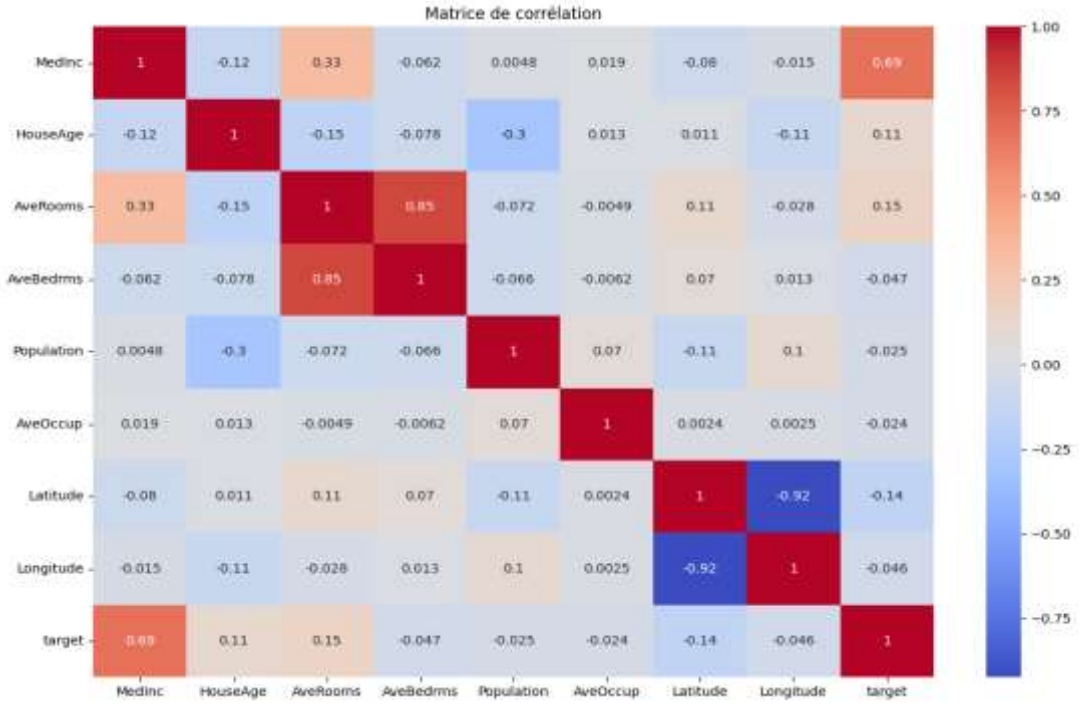
2. exploration statistique des données

2.1.Histogrammes de caractéristiques

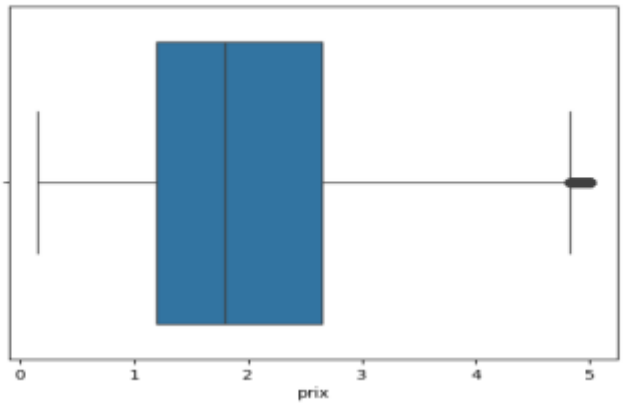


Dans un histogramme de caractéristiques en machine learning, les pics les plus élevés représentent les valeurs les plus fréquentes ou les occurrences les plus importantes pour une caractéristique donnée. En d'autres termes, ils indiquent les valeurs qui sont les plus représentatives ou les plus dominantes dans l'ensemble des données pour cette caractéristique spécifique. Ces pics peuvent fournir des informations précieuses sur la distribution et la tendance des données, ce qui peut aider à comprendre les modèles et les relations entre les caractéristiques et la variable cible dans un modèle d'apprentissage automatique.

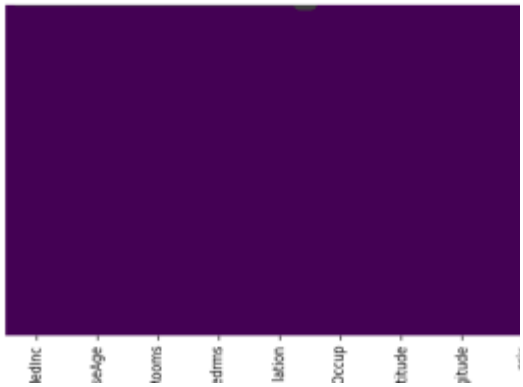
2.2.Matrice de corrélation



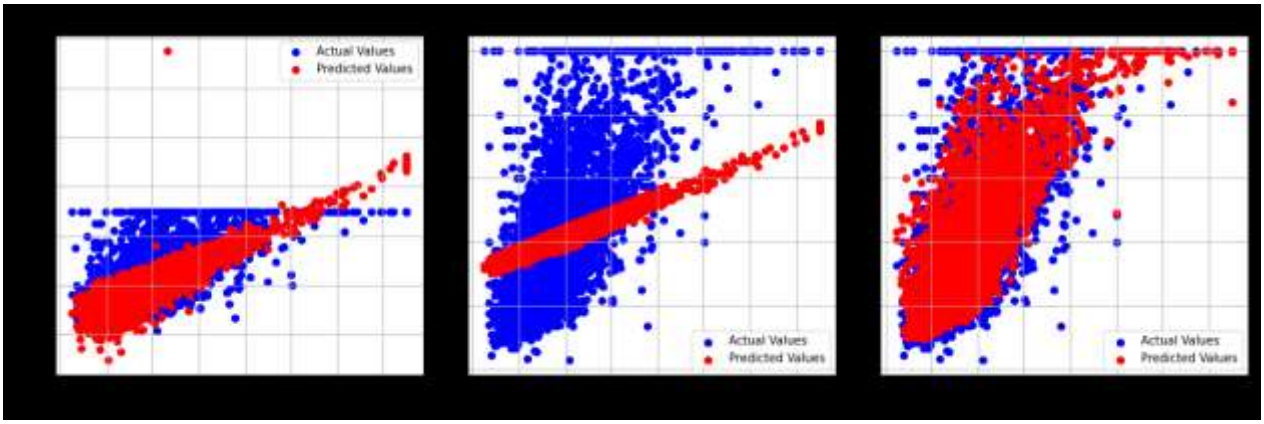
2.3.Digramme boite à moustache de la variable cible



2.4.Cartes de chaleur



3. **mettre au point plusieurs modèles de régression (LinearRegression, Lasso, RandomForestRegressor) en estimant leurs capacités de généralisation en veillant à bien séparer le jeu de données en un ensemble d'entrainement et de test.**



Afin d'observer la capacité de généralisation de chacun des algorithmes d'apprentissage automatique, j'ai tracé un graphique des valeurs réelles à prédire (y_test) par rapport aux valeurs prédites par rapport à la variable la plus fortement corrélée, qui est le revenu médian des ménages. Nous pouvons remarquer que l'algorithme ***RandomForest*** présente une meilleure capacité de généralisation que les autres algorithmes.

4. **Mesurer les performances des modèles sur l'ensemble de test en utilisant la métrique du coefficient de détermination (R^2)**

Les performances des trois modèles de régression évalués varient significativement :

- La régression linéaire a obtenu un coefficient de détermination (R^2) de 57.58%, indiquant une capacité modérée à expliquer la variance des données. Cependant, une partie importante de la variance reste inexploitée.
- La régression Lasso a donné un R^2 de 28.42%, suggérant une performance inférieure à celle de la régression linéaire.
- En revanche, la régression par RandomForest a produit un R^2 de 80.61%, démontrant une capacité supérieure à expliquer la variance des données. Cette performance élevée est due à la nature non linéaire et à la capacité de généralisation de l'algorithme.

5. Comparer les performances des différents modèles, sélectionner le meilleur et faire varier ses paramètres pour améliorer sa performance en utilisant par exemple la fonction GridSearchCV présente dans scikit-learn

Les résultats précédents indiquent que la régression par RandomForest surpasse les autres modèles en termes de capacité à généraliser sur le jeu de donnée fourni. Nous allons explorer en détail l'algorithme RandomForest afin de développer le meilleur modèle possible pour prédire les valeurs de la variable cible.

5.1.Visualiser les paramètres par défauts

Entrée [11]:
```
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(random_forest_reg.get_params())
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

5.2.Sélection des hyper paramètres à optimiser

Nous allons essayer d'ajuster l'ensemble suivant d'hyperparamètres :

n_estimateurs = nombre d'arbres de décision à construire.

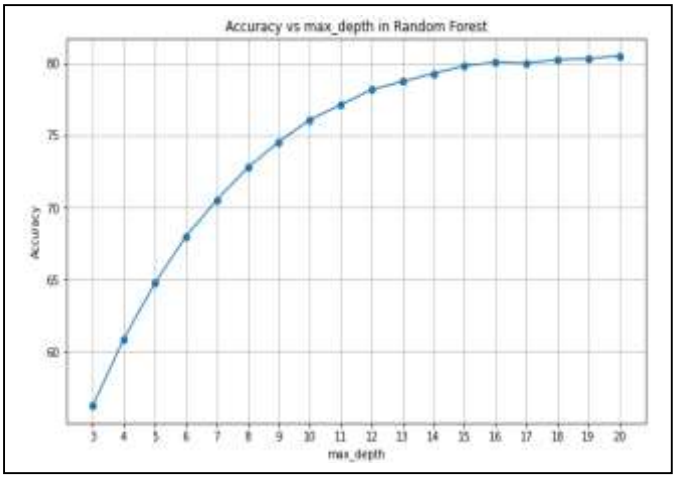
max_features = nombre maximal de caractéristiques prises en compte pour la division d'un nœud

max_depth = nombre maximal de niveaux dans chaque arbre de décision

bootstrap = méthode d'échantillonnage de points de données (avec ou sans remplacement)

5.2.1. Faire Varier les paramètres et observer le graphique de performance

Nous avons essayé de faire varier la profondeur maximale des arbres de décision afin d'observer l'évolution de la métrique de performance basée sur la moyenne des erreurs dans chacun des cas. Les résultats sont présentés dans le graphique suivant.



Dans ce cas, on constate que la performance reste quasiment constante à partir d'une certaine valeur.

5.2.2. En utilisant la méthode RandomSearchCV

Pour utiliser RandomizedSearchCV, nous devons d'abord créer une grille de paramètres à partir de laquelle nous effectuerons l'échantillonnage lors de l'ajustement.

Entrée [12]:
```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(3, 25, num =5, endpoint=False)]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'bootstrap': bootstrap}
pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [3, 7, 11, 16, 20],
 'max_features': ['auto', 'sqrt'],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Nous allons ensuite instancier la recherche aléatoire et l'ajuster comme n'importe quel modèle Scikit-Learn. Les arguments les plus importants de RandomizedSearchCV sont n_iter, qui contrôle le nombre de combinaisons différentes à essayer, et cv qui est le nombre de plis à utiliser pour la validation croisée (nous utilisons 100 et 3 respectivement). Nous pouvons visualiser les meilleurs paramètres :

Entrée [13]:
```
from sklearn.model_selection import RandomizedSearchCV
# Use the random grid to search for best hyperparameters
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 5 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=
# Fit the random search model
rf_random.fit(X_train, Y_train)
rf_random.best_params_
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

Out[13]:
```
{'n_estimators': 1600,
 'max_features': 'sqrt',
 'max_depth': 20,
 'bootstrap': False}
```

Ensuite nous comparons le modèle de base avec le meilleur modèle issu de la recherche aléatoire.

Entrée [14]:
```
def evaluate(model, X_test, Y_test):
    predictions = model.predict(X_test)
    errors = abs(predictions - Y_test)
    mape = 100 * np.mean(errors / Y_test)
    accuracy = 100 - mape
    print('Average Error: {:0.4f}'.format(np.mean(errors)))
    print('Accuracy = {:0.2f}%.'.format(accuracy))
    return accuracy

model_de_base = RandomForestRegressor(n_estimators = 100, random_state = 42)
model_de_base.fit(X_train, Y_train)
print('Performance of model_de_base')
base_accuracy = evaluate(model_de_base, X_test, Y_test)

model_random = rf_random.best_estimator_
print('Performance of model_random')
random_accuracy = evaluate(model_random, X_test, Y_test)

print('Improvement in performance after Hyperparamter tuning : {:0.2f}%.'.format( 100 * (random_accuracy - base_accuracy) / base_
```

Performance of model_de_base
Average Error: 0.3281
Accuracy = 81.07%.
Performance of model_random
Average Error: 0.3167
Accuracy = 81.51%.
Improvement in performance after Hyperparamter tuning : 0.54%.

5.2.3. Affiner la recherche en utilisant la méthode GridSearchCV

La recherche aléatoire nous a permis de réduire la plage de chaque hyper paramètre. Maintenant que nous savons où concentrer notre recherche, nous pouvons spécifier explicitement toutes les combinaisons de paramètres à essayer. C'est ce que nous faisons avec GridSearchCV, une méthode qui, au lieu d'échantillonner aléatoirement à partir d'une

distribution, évalue toutes les combinaisons que nous définissons. Pour utiliser la recherche par grille, nous créons une autre grille basée sur les meilleures valeurs fournies par la recherche aléatoire :

```
from sklearn.model_selection import GridSearchCV
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [False],
    'max_depth': [20, 22, 24, 26, 28],
    'max_features': ['sqrt'],
    'n_estimators': [1500, 1600, 1700, 1800, 1900]
}
# Create a based model
rf = RandomForestRegressor()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)

# Fit the grid search to the data
grid_search.fit(X_train, Y_train)
grid_search.best_params_

best_grid = grid_search.best_estimator_
grid_accuracy = evaluate(best_grid, X_test, Y_test)

print('Improvement of {:0.2f}%.'.format( 100 * (grid_accuracy - base_accuracy) / base_accuracy))
```

Fitting 3 folds for each of 25 candidates, totalling 75 fits
Average Error: 0.3148
Accuracy = 81.62%.
Improvement of 0.67%.

6. **Analyser l'importance des caractéristiques dans la prédiction des prix des maisons en utilisant la bibliothèque SHAP puis proposer une explication littérale.**

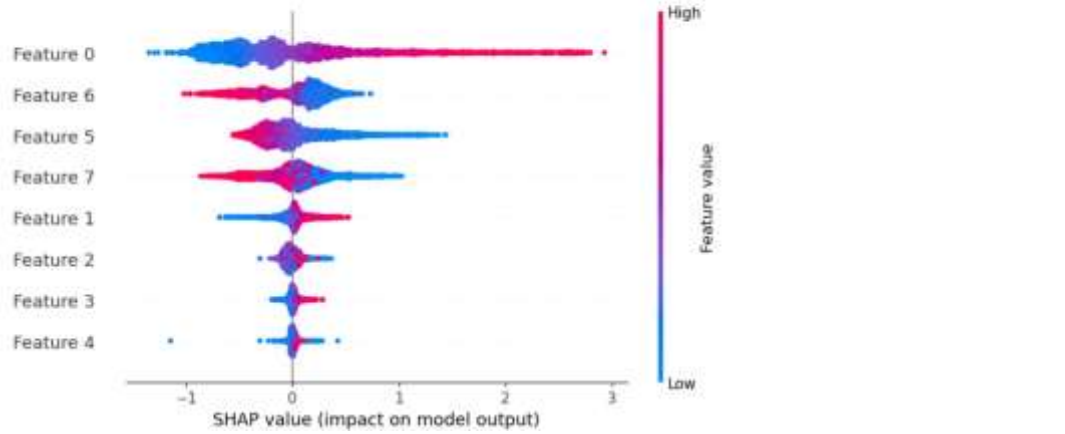
```
#!conda install -c conda-forge shap
import shap
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

# Load dataset
california = fetch_california_housing()
X, y = california.data, california.target

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Random Forest Regressor model
random_forest_reg = RandomForestRegressor()
random_forest_reg.fit(X_train, y_train)

explainer = shap.Explainer(random_forest_reg)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
shap.summary_plot(shap_values[0], X_test)
```



Pour le modèle entrainée en utilisant l'algorithme d'apprentissage randomForest, on observe les points suivants qui révèlent l'importance de chaque caractéristique dans la prédiction du prix des maisons. Ici nous tentons d'expliquer l'importance des trois caractéristiques les plus importantes ***(Feature0→MedInc, Feature6→longitude et Feature5→latitude)***:

- Le revenu médian a le plus grand impact sur la prédiction du prix des maisons. Les valeurs élevées de MedInc (à droite de 0) sont associé à un prix élevé, tandis que les valeurs faibles (à gauche de 0) sont associées à un prix moins élevée.
- Les valeurs élevées de Latitude ont un impact négatif sur le prix prédit tandis que les valeurs faibles ont un impact positif.
- La longitude a également un impact significatif sur le prix prédit. Cela suggère que les maisons situées dans les zones à longitude positif ont tendance à être prédites comme étant moins chères, tandis que les maisons situées dans les zones à longitudes négatifs sont prédites comme étant plus chères.

CONCLUSION ET PERSPECTIVES

Parvenu au terme de notre projet d'apprentissage machine visant à pour prédire les prix des maisons à partir du jeu de données "California Housing", plusieurs constats significatifs ont émergé. Tout d'abord, nous avons observé que l'algorithme RandomForest a surpassé les autres modèles, démontrant une capacité supérieure à généraliser sur le jeu de données fourni. En effet, son coefficient de détermination (R^2) élevé de 80.61% souligne sa capacité à expliquer la variance des données. De plus, nous avons effectué une optimization des hyperparamètres du modèle de base obtenu, utilisant des techniques telles que RandomizedSearchCV et GridSearchCV pour optimiser ses performances.

Pour améliorer davantage les performances du modèle, une perspective prometteuse serait d'enrichir le jeu de données avec des variables supplémentaires pertinentes, telles que la proximité des écoles, des centres commerciaux ou encore des infrastructures de transport public. Cette approche permettrait de capturer davantage de facteurs influençant les prix des maisons, ce qui pourrait conduire à des prédictions plus précises. Parallèlement, l'exploration de techniques d'ensemble supplémentaires, comme Gradient Boosting ou AdaBoost, représente une autre piste intéressante à explorer. Ces méthodes offrent de nouvelles possibilités pour renforcer la capacité du modèle à généraliser.

REFERENCES

Gradient Boosting & AdaBoost (site de AWS):
https://aws.amazon.com/fr/what-is/boosting/#:~:text=Le%20boosting%20adaptatif%20(AdaBoost)%20est,%C3%A0%20chaque%20ensemble%20de%20donn%C3%A9es.