

# Travaux Pratiques: Regression & Pipeline ML

Louis Fippo Fitime, Claude Tinku, Kerolle Sonfack  
Département Génie Informatique, ENSPY

11 octobre 2025

## Objectif Général

- L'objectif de ce TP est de vous guider à travers toutes les étapes de la construction d'un modèle de Machine Learning, de la collecte des données à l'évaluation finale.
  - Nous utiliserons le jeu de données **California Housing**, qui contient des informations agrégées sur des blocs de recensement en Californie et dont la variable cible est le prix médian des maisons (en centaines de milliers de dollars). Il s'agit d'un problème de **régression**.
- 

## 1 Phase I : Processing, Wrangling & Visualizing Data

### 1.1 Data Collection et Description

- **Tâche 1.1** : Chargez le jeu de données California Housing en utilisant `scikit-learn`. Créez un `DataFrame Pandas` pour faciliter la manipulation et incluez la variable cible (`MedHouseVal`).

Code Python 1.1: Chargement et description initiale

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_california_housing

Charger les données
housing = fetch_california_housing(as_frame=True)
df = housing.frame
df.rename(columns={'MedHouseVal': 'Target'}, inplace=True)

print("Description du DataFrame:")
print(df.info())
```

```
print("\nPremières lignes:")
print(df.head())
```

## 1.2 Data Summarization et Wrangling

- **Tâche 1.2** : Affichez les statistiques descriptives de base (`describe()`). Identifiez les valeurs manquantes (s'il y en a) et proposez un plan pour les gérer.
- **Tâche 1.3** : Séparez les caractéristiques (X) de la variable cible (y).

## 1.3 Data Visualization

- **Tâche 1.4** : Visualisez la distribution de la variable cible (`Target`) à l'aide d'un histogramme. Commentez sa forme (symétrique, biaisée, etc.).
- **Tâche 1.5** : Affichez la matrice de corrélation pour visualiser les relations linéaires entre les caractéristiques et avec la cible. Quel est l'impact de `MedInc` (Revenu Médian) sur `Target` ?

Code Python 1.4 & 1.5: Visualisation

Tâche 1.4

```
plt.figure(figsize=(8, 6))
sns.histplot(df['Target'], bins=50, kde=True)
plt.title('Distribution de la valeur médiane des maisons (Target)')
plt.show()
```

Tâche 1.5

```
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Matrice de Corrélation')
plt.show()
```

---

## 2 Phase II : Feature Engineering and Selection

### 2.1 Feature Extraction and Engineering

- **Tâche 2.1** : Le jeu de données fournit des totaux (pièces, chambres, population) par bloc. Créez les trois nouvelles caractéristiques dérivées suivantes (ratios par foyer) pour capturer des informations plus pertinentes :
  1. `RoomsPerHousehold`
  2. `BedroomsPerRoom`
  3. `PopulationPerHousehold`

Code Python 2.1: Feature Engineering

```
X = df.drop('Target', axis=1)
y = df['Target']
```

```
X['RoomsPerHousehold'] = X['AveRooms'] / X['AveOccup']
X['BedroomsPerRoom'] = X['AveBedrms'] / X['AveRooms']
X['PopulationPerHousehold'] = X['Population'] / X['AveOccup']
```

Séparation des données

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=)
```

## 2.2 Feature Scaling

- **Tâche 2.2 :** Étant donné que les caractéristiques ont des échelles très différentes, appliquez une normalisation (**StandardScaler**) aux ensembles d'entraînement et de test. Expliquez pourquoi il est crucial d'appliquer *fit* *transform* *uniquement sur l'ensemble d'entraînement* et *transform* *sur l'ensemble de test*.

Code Python 2.2: Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)
```

```
print("\nÉchelle des données après standardisation (premières lignes):")
print(X_train_scaled.head())
```

## 2.3 Feature Selection

- **Tâche 2.3 :** Utilisez la méthode de sélection basée sur l'importance des caractéristiques (**Feature Importance**) d'un modèle de forêt aléatoire (**RandomForestRegressor**) pour identifier les 5 caractéristiques les plus importantes.

# 3 Phase III : Building, Tuning Deploying Models

## 3.1 Model Building

- **Tâche 3.1 :** Entraînez un modèle de **Régression Linéaire** (**LinearRegression**) et un modèle de **Forêt Aléatoire** (**RandomForestRegressor**) en utilisant les données normalisées.
- **Tâche 3.2 :** Expliquez brièvement les avantages théoriques de l'utilisation d'un modèle non-linéaire (Random Forest) par rapport à un modèle linéaire (Régression Linéaire) dans ce contexte.

Code Python 3.1: Entraînement des modèles

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
```

Modèle 1: Régression Linéaire

```
lin_reg = LinearRegression()
lin_reg.fit(X_train_scaled, y_train)
```

Modèle 2: Forêt Aléatoire (hyperparamètres de base)

```
forest_reg = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
forest_reg.fit(X_train_scaled, y_train)
```

```
print("\nModèles entraînés.")
```

## 3.2 Model Evaluation

- **Tâche 3.3 :** Évaluez les deux modèles sur l'ensemble de test (`X_test_scaled`, `y_test`) en utilisant les métriques suivantes (plus les faibles valeurs sont bonnes) :
  1. **Mean Squared Error (MSE)**
  2. **Root Mean Squared Error (RMSE)**
- Quelle est le meilleur modèle sur ces métriques ?

Code Python 3.3: Évaluation des modèles

```
from sklearn.metrics import mean_squared_error
```

Prédictions

```
y_pred_lin = lin_reg.predict(X_test_scaled)
y_pred_forest = forest_reg.predict(X_test_scaled)
```

Évaluation Régression Linéaire

```
mse_lin = mean_squared_error(y_test, y_pred_lin)
rmse_lin = np.sqrt(mse_lin)
```

Évaluation Forêt Aléatoire

```
mse_forest = mean_squared_error(y_test, y_pred_forest)
rmse_forest = np.sqrt(mse_forest)
```

```
print(f"\n[Régression Linéaire] RMSE: {rmse_lin:.4f}")
print(f"[Forêt Aléatoire] RMSE: {rmse_forest:.4f}")
```

## 3.3 Model Interpretation

- **Tâche 3.4 :** Visualisez et analysez l'importance des caractéristiques (`feature_importances_`) du modèle `RandomForestRegressor` (Modèle 2). Corroborez ces résultats avec l'étape de sélection de caractéristiques (Tâche 2.3).

### 3.4 Model Deployment (Simulé)

- **Tâche 3.5** : En supposant que le modèle Random Forest soit le meilleur, utilisez la bibliothèque `joblib` pour sauvegarder le modèle entraîné sur le disque. C'est l'étape de **déploiement** de base qui permet de réutiliser le modèle sans le ré-entraîner.

Code Python 3.5: Sauvegarde/Déploiement du modèle

```
import joblib
```

Sauvegarder le meilleur modèle (ici, le Random Forest)

```
joblib.dump(forest_reg, 'random_forest_regressor.pkl')
```

```
print("\nModèle 'random_forest_regressor.pkl' sauvegardé.")
```

Test de chargement

```
loaded_model = joblib.load('random_forest_regressor.pkl')
```

```
print(f"Modèle chargé. Test de prédiction sur le premier élément du test: {loaded_model.predict(test_data[0])}")
```

---

## 4 Conclusion et Perspectives

- **Tâche 4.1** : Rédigez un court paragraphe de synthèse décrivant les étapes clés du pipeline ML que vous avez exécuté et le résultat principal de l'évaluation du modèle.
- **Tâche 4.2 (Bonus/Approfondissement)** : Proposez une stratégie d'amélioration pour ce pipeline (e.g., Optimisation des Hyperparamètres avec `GridSearchCV`, utilisation d'un autre modèle comme le Gradient Boosting, gestion des outliers).