RAPPORT DE TRAVAUX PRATIQUES

# MACHINE LEARNING

## TP1 – TP2 – TP3

Pipeline ML, Régression Avancée & Clustering

ENSPY

Département Génie Informatique

2025

Enseignants : L. Fippo Fitime, C. Tinku, K. Sonfack

# Table des matières

# Introduction générale

Le **Machine Learning** (apprentissage automatique) constitue l'un des piliers de l'intelligence artificielle moderne. Il permet aux systèmes informatiques d'apprendre à partir de données sans être explicitement programmés, ouvrant la voie à des applications dans des domaines aussi variés que la santé, la finance, le marketing ou encore l'industrie du jeu vidéo.

Ce rapport présente la synthèse de trois travaux pratiques réalisés dans le cadre du cours de Machine Learning au Département de Génie Informatique de l'École Nationale Supérieure Polytechnique de Yaoundé (ENSPY). Ces TPs couvrent un spectre progressif de compétences :

- **TP1 – Pipeline ML complet :** De la préparation des données au déploiement d'un modèle de régression, en passant par le feature engineering et la sélection de caractéristiques.
- **TP2 – Régression avancée et MLOps :** Étude approfondie de la régularisation (Ridge, LASSO), du compromis biais-variance, et introduction aux pratiques MLOps (MLflow, Docker).
- **TP3 – Clustering et apprentissage non supervisé :** Implémentation et comparaison de K-Means, GMM/EM et DBSCAN, suivi MLflow des expériences de clustering, et concepts théoriques avancés (LDA).

La démarche suivie est rigoureusement scientifique : pour chaque TP, nous définissons la problématique, décrivons la méthodologie, présentons l'implémentation et les résultats, puis proposons une analyse critique. L'ensemble s'inscrit dans une perspective **MLOps**, où la reproductibilité, la traçabilité et le déploiement sont au cœur de la démarche.

# TP1 – Introduction au Pipeline ML

## 1.1 Rappel théorique

### 1.1.1 Contexte

Le premier TP introduit le concept de **pipeline de Machine Learning**, c'est-à-dire l'ensemble des étapes nécessaires pour transformer des données brutes en un modèle prédictif déployable. Le jeu de données utilisé est le `California Housing` de scikit-learn, qui contient 20 640 observations décrivant des blocs de logements californiens avec 8 variables prédictives et une variable cible (valeur médiane des maisons).

### 1.1.2 Objectifs

- Maîtriser les étapes de préparation et nettoyage des données.
- Réaliser des visualisations exploratoires pertinentes.
- Appliquer le feature engineering et la sélection de caractéristiques.
- Construire, évaluer et comparer des modèles de régression.
- Sauvegarder le modèle final pour un déploiement ultérieur.

Pipeline ML Un **pipeline ML** est une séquence structurée d'étapes : collecte des données → prétraitement → feature engineering → entraînement → évaluation → déploiement.

## 1.2 Méthodologie

### 1.2.1 Phase I : Traitement et Visualisation des Données

#### Collecte et description

Les données sont chargées via `fetch_california_housing` de scikit-learn. Le DataFrame résultant contient 9 colonnes : `MedInc`, `HouseAge`, `AveRooms`, `AveBedrms`, `Population`, `AveOccup`, `Latitude`, `Longitude` et `Target` (valeur médiane).

#### Analyse exploratoire

Les statistiques descriptives révèlent :

- Aucune valeur manquante dans le jeu de données.
- Des échelles très différentes entre les variables (nécessité de standardisation).
- Quelques valeurs extrêmes (ex. `AveOccup` max = 1243, `AveRooms` max = 141.9).

#### Visualisation

Deux visualisations clés ont été produites :

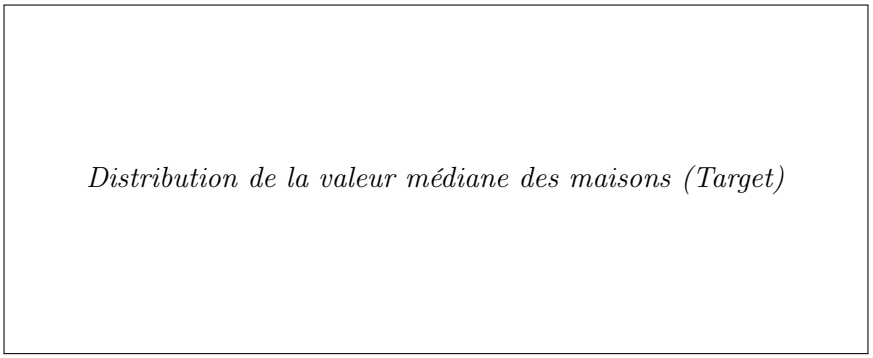


**Figure 1.1.** Distribution de la variable cible (Target) – TP1

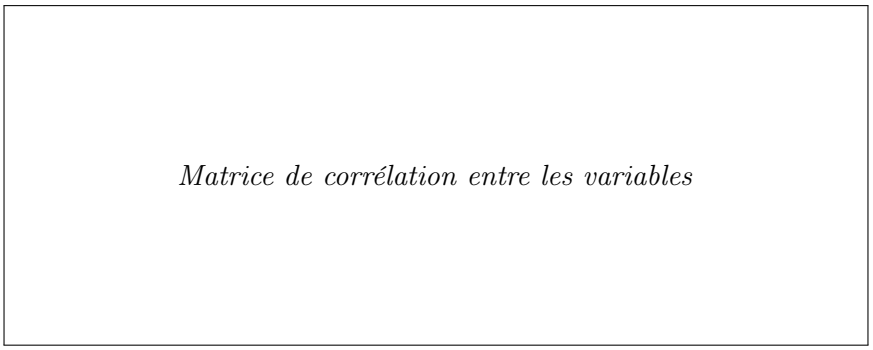


**Figure 1.2.** Matrice de corrélation – TP1

### 1.2.2 Phase II : Feature Engineering et Sélection

#### Création de nouvelles variables

Trois nouvelles variables ont été créées pour enrichir la représentation :

- `RoomsPerHousehold` = AveRooms/AveOccup
- `BedroomsPerRoom` = AveBedrms/AveRooms
- `PopulationPerHousehold` = Population/AveOccup

#### Standardisation

La standardisation `StandardScaler` (moyenne = 0, écart-type = 1) a été appliquée sur l'ensemble d'entraînement, puis transférée sur l'ensemble de test. Cette étape est cruciale car les algorithmes à base de distance ou de gradient sont sensibles à l'échelle des données.

### Sélection de caractéristiques

Un `RandomForestRegressor` a été utilisé pour calculer l'importance des variables. Les 5 caractéristiques les plus importantes ont été identifiées, avec `MedInc` (revenu médian) largement en tête.

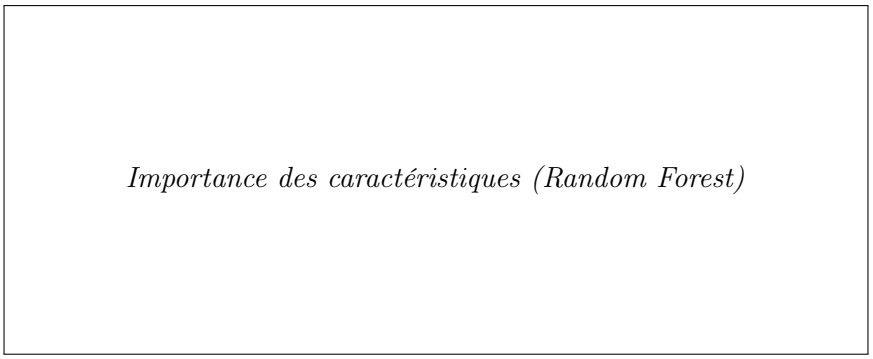


**Figure 1.3.** Importance des features – TP1

## 1.3 Implémentation

### 1.3.1 Phase III : Construction et Évaluation des Modèles

Deux modèles de régression ont été entraînés :

```
# Modele 1: Regression Lineaire
lin_reg = LinearRegression()
lin_reg.fit(X_train_scaled, y_train)

# Modele 2: Random Forest
forest_reg = RandomForestRegressor(
    n_estimators=100, random_state=42, n_jobs=-1)
forest_reg.fit(X_train_scaled, y_train)
```

**Listing 1.1.** Entraînement des modèles – TP1

L'optimisation des hyperparamètres du Random Forest a été réalisée via `GridSearchCV` (validation croisée à 3 folds sur 108 combinaisons).

## 1.4 Résultats

| Modèle | RMSE | Meilleurs hyperparamètres |
|---|---|---|
| Régression Linéaire | 0.6711 | – |
| Random Forest (base) | 0.5114 | `n_estimators=100` |
| Random Forest (optimisé) | 0.5096 | `max_depth=30, min_samples_leaf=2` |

**Table 1.1.** Comparaison des modèles – TP1

Le Random Forest optimisé atteint un RMSE de **0.5096**, soit une amélioration de 24 % par rapport à la régression linéaire. Le modèle final est sauvegardé via `joblib` pour déploiement.

## 1.5 Analyse critique

- La régression linéaire est dépassée par le Random Forest, ce qui suggère des relations non-linéaires dans les données.
- L'optimisation par GridSearchCV n'apporte qu'une amélioration marginale (0.5114 à 0.5096), indiquant que les hyperparamètres par défaut étaient déjà proches de l'optimal.
- Le feature engineering (création de ratios) améliore la qualité des prédictions en capturant des relations latentes.
- **Limite :** le modèle ne gère pas les valeurs aberrantes de façon explicite. Un prétraitement des outliers pourrait améliorer les résultats.

# TP2 – Régression Avancée et MLOps

## 2.1 Rappel théorique

### 2.1.1 Contexte

Le deuxième TP approfondit les techniques de régression en introduisant la **régularisation** comme mécanisme de contrôle du compromis biais-variance. Le jeu de données `Diabetes` de scikit-learn (442 patients, 10 variables) est utilisé.

### 2.1.2 Objectifs

- Comprendre le compromis biais-variance et le phénomène de sur-apprentissage.
- Implémenter et comparer OLS, Ridge (L2) et LASSO (L1).
- Maîtriser l'optimisation des hyperparamètres par validation croisée.
- Introduire les pratiques MLOps : packaging, tracking MLflow, Docker.

### 2.1.3 Fondements théoriques

Compromis Biais-Variance L'erreur de généralisation se décompose en :

$$\text{Erreur totale} = \text{Biais}^2 + \text{Variance} + \text{Bruit irréductible}$$

La régularisation augmente légèrement le biais mais réduit significativement la variance, améliorant la généralisation.

Les fonctions de coût régularisées s'écrivent :

$$\text{Ridge (L2)} \quad : \quad \mathcal{L} = \text{MSE} + \lambda \sum_i w_i^2 \tag{2.1}$$

$$\text{LASSO (L1)} \quad : \quad \mathcal{L} = \text{MSE} + \lambda \sum_i |w_i| \tag{2.2}$$

**Différence clé :** Ridge réduit les coefficients sans les annuler (dérivée $\partial w_i^2/\partial w_i = 2w_i \to 0$), tandis que LASSO produit des zéros exacts (dérivée $\partial|w_i|/\partial w_i = \text{sign}(w_i)$, constante). Géométriquement, la contrainte L1 forme un losange dont les coins se situent sur les axes.

## 2.2 Méthodologie

### 2.2.1 Phase I : Régression linéaire avancée

#### Préparation des données

Le jeu de données Diabetes est divisé en 80 % entraînement / 20 % test, puis standardisé via `StandardScaler`.

#### Modèle OLS (baseline)

La régression OLS sert de référence avec un RMSE de 54.52 et un $R^2$ de 0.44.

#### Ridge Regression (L2)

Le modèle Ridge avec $\alpha = 1.0$ est entraîné et évalué par validation croisée à 5 folds. L'alpha optimal est ensuite déterminé par `RidgeCV` avec 100 valeurs de $\alpha$ dans $[10^{-3}, 10^{2}]$ :

$$\alpha^*_{\text{Ridge}} = 55.91$$

#### LASSO Regression (L1)

Le LASSO est entraîné de manière similaire. L'alpha optimal trouvé par `LassoCV` est :

$$\alpha^*_{\text{LASSO}} = 0.1417$$

Le LASSO élimine automatiquement 1 feature sur 10 (coefficient mis à zéro), démontrant la propriété de **sparsity** (parcimonie).

## 2.3 Implémentation

```
# Ridge Tuning
ridge_cv = RidgeCV(alphas=np.logspace(-3, 2, 100), cv=10)
ridge_cv.fit(X_train_scaled, y_train)

# LASSO Tuning
lasso_cv = LassoCV(alphas=np.logspace(-3, 0, 100),
                   cv=10, max_iter=10000)
lasso_cv.fit(X_train_scaled, y_train)
```

**Listing 2.1.** Optimisation Ridge et LASSO – TP2

### 2.3.1 Phase II : MLOps

#### Packaging du modèle

Le modèle LASSO final et le scaler sont sauvegardés via `joblib`. La sauvegarde du scaler est **cruciale** : en production, les données d'entrée doivent subir exactement la même transformation que lors de l'entraînement.

### Tracking avec MLflow

Les trois modèles (OLS, Ridge, LASSO) sont enregistrés dans MLflow avec :

- **Hyperparamètres :** type de modèle, alpha, type de pénalité.
- **Métriques :** RMSE, $R^2$, nombre de coefficients nuls (LASSO).
- **Artefacts :** modèle sérialisé.

## 2.4 Résultats

| Modèle | RMSE | $R^2$ | Alpha | Features actives |
|---|---|---|---|---|
| OLS | 54.52 | 0.4389 | – | 10 |
| Ridge | 53.88 | 0.4521 | 55.91 | 10 |
| LASSO | 54.23 | 0.4449 | 0.14 | 9 |

**Table 2.1.** Comparaison des modèles de régression – TP2

Ridge obtient le meilleur RMSE (53.88) et $R^2$ (0.4521). LASSO élimine automatiquement une feature tout en conservant des performances comparables, offrant un modèle plus interprétable.

## 2.5 Analyse critique

- La régularisation améliore la généralisation (Ridge : $-1.2\,\%$ RMSE vs OLS).
- Le LASSO confirme sa capacité de sélection automatique de variables, ce qui est précieux en haute dimension.
- Les performances globales restent modestes ($R^2 \approx 0.45$), suggérant que des modèles non-linéaires (Random Forest, XGBoost) seraient plus adaptés à ce jeu de données.
- L'intégration MLOps (MLflow + packaging joblib) pose les bases d'un cycle de vie modèle reproductible et auditable.
- **Limite :** le Dockerfile et l'API Flask sont présentés de manière théorique et n'ont pas été déployés en conditions réelles.

# TP3 – Clustering et Apprentissage Non Supervisé

## 3.1 Rappel théorique

### 3.1.1 Contexte

Le troisième TP aborde l'**apprentissage non supervisé** à travers le clustering, une technique fondamentale pour découvrir des structures latentes dans des données non étiquetées. Les applications visées concernent la segmentation client (marketing) et l'analyse comportementale (gaming).

### 3.1.2 Objectifs

- Implémenter et comparer trois familles de clustering : K-Means, GMM/EM, DBSCAN.
- Maîtriser les métriques d'évaluation non supervisée (inertie, coefficient de silhouette).
- Tracker les expériences de clustering via MLflow.
- Étudier les concepts théoriques avancés : initialisation K-Means++, LDA.

### 3.1.3 Fondements théoriques

Coefficient de Silhouette Pour un point $i$ :

$$s(i) = \frac{b(i) - a(i)}{\max(a(i),\, b(i))}$$

où $a(i)$ est la distance intra-cluster moyenne et $b(i)$ la distance inter-cluster minimale moyenne. $s(i) \in [-1, +1]$.

Algorithme EM pour GMM Le modèle GMM définit :

$$P(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k)$$

L'algorithme EM alterne :

- **Étape E :** calcul des responsabilités $\gamma_{ik} = P(z_i = k \mid x_i)$.
- **Étape M :** mise à jour des paramètres $(\pi_k, \mu_k, \Sigma_k)$.

La vraisemblance augmente à chaque itération (convergence garantie vers un optimum local).

## 3.2 Méthodologie

### 3.2.1 Phase I : K-Means et Segmentation Client

#### Données

Un jeu de données simulé de 14 clients est créé avec deux variables : `Annual_Income (k$)` et `Spending_Score (1-100)`. Les données sont standardisées via `StandardScaler`.

#### Choix du nombre de clusters

La méthode du coude (Elbow Method) et le score de silhouette sont utilisés pour déterminer $K$ optimal. Le coude est observé autour de $K = 3$.

*Méthode du coude (inertie) et score de silhouette*

**Figure 3.1.** Méthode du coude et score de silhouette – TP3

#### Évaluation finale

Le modèle K-Means final avec $K = 3$ obtient un coefficient de silhouette de **0.6018**.

*Segmentation K-Means ($K = 3$) avec centroïdes*

**Figure 3.2.** Segmentation K-Means finale – TP3

### 3.2.2 Phase II : Clustering Avancé et Comparaison

#### Données non-globulaires

Des données synthétiques en forme de croissants de lune (`make_moons`, $n = 200$) sont générées pour tester les limites du K-Means.

#### GMM vs K-Means

Le modèle GMM (`GaussianMixture`) est comparé à K-Means sur les données moons. Les deux algorithmes échouent à capturer la forme non-convexe des clusters, mais GMM offre une affectation probabiliste (soft assignment).

#### DBSCAN

L'algorithme DBSCAN ($\varepsilon = 0.3$, `min_samples` $= 5$) identifie correctement les deux clusters non-convexes grâce à son approche basée sur la densité.

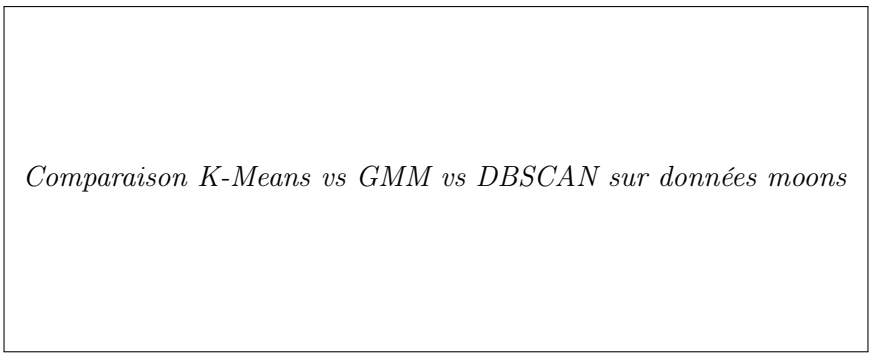


**Figure 3.3.** Comparaison des trois algorithmes de clustering – TP3

| Aspect | K-Means | GMM | DBSCAN |
|---|---|---|---|
| Affectation | Hard | Soft (probabiliste) | Hard + bruit |
| Forme des clusters | Sphérique | Ellipsoïdale | Arbitraire |
| Nombre de clusters | À fixer (K) | À fixer (K) | Automatique |
| Gestion du bruit | Non | Non | Oui (label $-1$) |
| Données moons | Échec | Échec partiel | Succès |

**Table 3.1.** Comparaison des algorithmes de clustering – TP3

## 3.3 Implémentation

### 3.3.1 Phase III : MLOps pour le Clustering

#### Tracking MLflow

Quatre configurations de K-Means ($K = 2, 3, 4, 5$) sont trackées dans MLflow avec :

- **Hyperparamètres :** $K$, algorithme, `n_init`, `random_state`.
- **Métriques :** inertie, silhouette, Calinski-Harabasz.
- **Artefacts :** modèle sérialisé, centroïdes en CSV.

```python
mlflow.set_experiment("TP3_Customer_Clustering")

def track_clustering_run(X_data, k, run_name):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = kmeans.fit_predict(X_data)
    with mlflow.start_run(run_name=run_name):
        mlflow.log_param("K", k)
        mlflow.log_metric("Silhouette_Score",
            silhouette_score(X_data, labels))
        mlflow.log_metric("Inertia", kmeans.inertia_)
        mlflow.sklearn.log_model(kmeans, name="kmeans_model")
```

**Listing 3.1.** Tracking MLflow du clustering – TP3

#### Déploiement et monitoring

Le modèle optimal ($K = 3$) est sauvegardé avec son scaler via `joblib`. Deux indicateurs de monitoring post-production sont identifiés :

- **Dérive des centroïdes :** mesurer la distance euclidienne entre les centroïdes originaux et ceux recalculés périodiquement.
- **Distribution des tailles de clusters :** surveiller si un segment absorbe ou perd anormalement des clients.

### 3.3.2 Phase IV : Concepts Théoriques Avancés

#### Comparaison des initialisations

L'objectif non-convexe de K-Means possède de multiples optima locaux. L'initialisation **K-Means++** sélectionne les centroïdes initiaux avec une probabilité proportionnelle à leur distance au centroïde le plus proche, garantissant une convergence $O(\log K)$-compétitive par rapport à l'optimum global. Les expériences (20 runs) montrent une variance significativement plus faible pour K-Means++ par rapport à l'initialisation aléatoire.

*Comparaison K-Means++ vs Random (boxplot et inerties par run)*

**Figure 3.4.** Comparaison des stratégies d'initialisation – TP3

#### Latent Dirichlet Allocation (LDA)

Le modèle **LDA** introduit la notion d'**appartenance mixte** (mixed membership) : chaque document est un mélange de topics, et chaque topic est un mélange de mots.

$$P(\text{mot} \mid \text{doc}) = \sum_{k=1}^{K} P(\text{topic}_k \mid \text{doc}) \cdot P(\text{mot} \mid \text{topic}_k)$$

Contrairement à K-Means appliqué sur des vecteurs TF-IDF (1 document = 1 cluster), LDA permet à un document de traiter **simultanément** plusieurs thèmes. Une démonstration sur un corpus de 12 documents simulés (sport, technologie, cuisine) confirme cette capacité : les documents à thématique mixte reçoivent des distributions non triviales sur plusieurs topics.

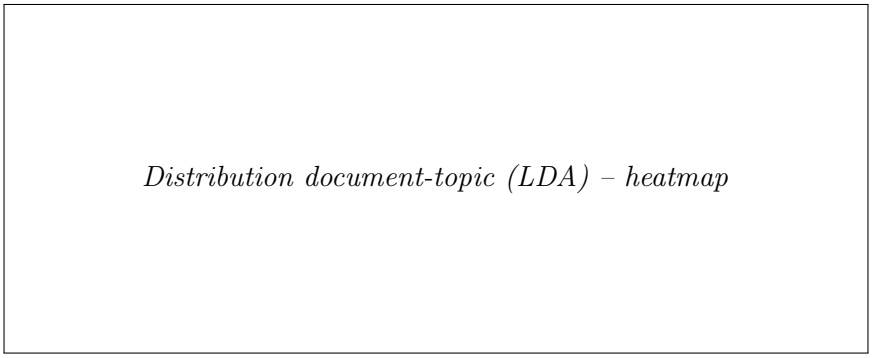


**Figure 3.5.** Distribution des topics par document (LDA) – TP3

## 3.4 Résultats

| K | Inertie | Silhouette | Calinski-Harabasz |
|---|---|---|---|
| 2 | 10.25 | 0.5616 | – |
| 3 | 4.42 | 0.6018 | – |
| 4 | 1.60 | 0.6888 | – |
| 5 | 0.76 | 0.7213 | – |

**Table 3.2.** Résultats du tracking MLflow (K-Means) – TP3

DBSCAN se révèle le plus performant sur des données non-globulaires. K-Means reste pertinent pour des clusters sphériques avec un bon score de silhouette. Le tracking MLflow permet une comparaison systématique et reproductible des configurations.

## 3.5 Analyse critique

- K-Means est simple et efficace pour des clusters globulaires, mais échoue complètement sur des formes non-convexes.
- GMM offre une flexibilité supérieure (clusters ellipsoïdaux, affectation probabiliste) mais reste limité sur des formes arbitraires.
- DBSCAN est le seul algorithme capable de détecter des clusters de forme arbitraire et de gérer le bruit, mais il est sensible au choix de $\varepsilon$ et `min_samples`.
- L'intégration MLflow garantit la traçabilité et facilite la comparaison des configurations.
- **Limite :** le jeu de données client est très petit (14 points). Une évaluation sur des données réelles serait nécessaire pour valider les conclusions.

- La Phase IV démontre que K-Means++ devrait toujours être préféré à l'initialisation aléatoire, et que LDA offre une alternative riche au clustering classique pour le texte.

# Conclusion générale

## Bilan

Ces trois travaux pratiques ont permis de parcourir un spectre représentatif des techniques de Machine Learning, de l'apprentissage supervisé (régression) à l'apprentissage non supervisé (clustering), en passant par les pratiques MLOps indispensables à la mise en production.

**Compétences acquises :**

- **TP1 :** Construction d'un pipeline ML complet, du prétraitement au déploiement (RMSE final : 0.51).
- **TP2 :** Compréhension profonde de la régularisation et intégration des outils MLOps (MLflow, Docker).
- **TP3 :** Maîtrise de trois familles de clustering, tracking d'expériences, et concepts théoriques avancés (EM, LDA).

## Limites

Plusieurs limitations méritent d'être notées :

- Les jeux de données utilisés sont académiques (scikit-learn) ou simulés. Les défis de données réelles (valeurs manquantes massives, déséquilibre, volume) ne sont pas abordés.
- Le déploiement Docker est présenté de manière théorique sans mise en œuvre réelle.
- Les modèles de régression atteignent un $R^2$ plafonné autour de 0.45 (TP2), suggérant que des approches non-linéaires seraient nécessaires.
- Le monitoring post-production est décrit conceptuellement mais non implémenté.

## Perspectives

Pistes d'amélioration et d'approfondissement :

- **Modèles avancés :** intégrer des méthodes de boosting (XGBoost, LightGBM) et des réseaux de neurones.
- **Données réelles :** appliquer les pipelines à des jeux de données industriels avec toutes leurs imperfections.
- **MLOps complet :** implémenter un pipeline CI/CD avec GitHub Actions, tests automatisés et déploiement continu sur Kubernetes.
- **Clustering avancé :** explorer HDBSCAN (version hiérarchique de DBSCAN) et le spectral clustering.
- **NLP et LDA :** appliquer LDA à un vrai corpus de documents avec prétraitement linguistique complet (lemmatisation, suppression des stopwords).
- **Monitoring :** mettre en place un système de détection de drift avec des outils comme Evidently AI ou Whylogs.

Pipeline ML → Régularisation → Clustering → MLOps → Production