1. Exploration du dataset

- **Dataset** : Occupancy Detection (version simplifiée)
- **Nombre de mesures** : environ 8 143 pour le datatraining , 2 664 pour le datatest1 et 9 752 pour le datatest2 (version complète ~20 000)

Colonnes et signification

Colonne	Description
`date`	Horodatage de la mesure
`Temperature`	Température en °C
`Humidity`	Humidité relative en %
`Light`	Intensité lumineuse (lux)
`CO2`	Taux de CO2 en ppm
`HumidityRatio`	Ratio d'humidité (humidité absolue / vapeur d'eau ?)
`Occupancy`	Occupation de la pièce : 1 = occupée, 0 = vide

Cette dernière colonne (`Occupancy`) qu 'on va **prédire**

2. Nettoyage et normalisation

Nettoyage

Suppression de la colonne `date` et le remplacer par `Time_Index` est **une façon de représenter le temps sous forme numérique**, au lieu d'utiliser directement la date et l'heure.

Nous avons pas de doublons ni des valeur manquante

Création de `Time_Index`

Chaque ligne reçoit un numéro croissant (`1, 2, 3, …`) pour représenter l'ordre temporel des mesures

Sélection des features et label

- **Features (X)** : `Temperature, Humidity, Light, CO2, HumidityRatio, Time_Index`
- **Label (y)** : `Occupancy`

`resultats :`

Features (6 colonnes) :

1. `Temperature` : Température en °C
2. `Humidity` : Humidité relative en %

3. `Light` : Intensité lumineuse en lux
4. `CO2` : Taux de CO2 en ppm
5. `HumidityRatio` : Ratio d'humidité
6. `Time_Index` : Indice temporel créé pour remplacer la colonne date

Label : `Occupancy`

7. 0 = inoccupé

`2.1 = occupé`

Analyse statistique du dataset

Colonne	Min	Max	Moyenne	Remarques
Temperature	19	23,18	20,62	Échelle homogène, pas besoin de normalisation
Humidity	16,75	39,12	25,73	Échelle acceptable, pas besoin de normalisation
Light	0	1546,33	119,52	Grande différence → normaliser
CO2	412,75	2028,50	606,55	Grande différence → normaliser
HumidityRatio	0,0027	0,0065	0,00386	Très petite échelle → normaliser
Time_Index	1	8143	4072	Grande échelle → normaliser
Occupancy	0	1	0,21	Label binaire

Observations :

- Les colonnes `Light`, `CO2`, `HumidityRatio` et `Time_Index` ont des échelles très différentes, donc **la normalisation est nécessaire** pour améliorer l'apprentissage.
- Les colonnes `Temperature` et `Humidity` sont déjà sur des échelles comparables, donc **pas besoin de normalisation**.
- `Le dataset contient une colonne temporelle initiale date, remplacée par un` **Time_Index numérique**`, plus adapté pour les modèles de machine learning.`

● Normalisation

- Colonnes normalisées : `Light`, `CO2`, `HumidityRatio`, `Time_Index`
- Colonnes conservées sans modification : `Temperature`, `Humidity`
- Formule appliquée à chaque valeur :Xscaled=(X−moyenne)/ ecart-type

Objectif : mettre toutes les colonnes sur des **échelles comparables**, faciliter l'apprentissage

● Séparation train/test

- `train_test_split` avec `test_size=0.2`
- Stratification pour gérer le déséquilibre des classes (`stratify=y`)

● Gestion du déséquilibre

Calcul des **poids des classes** pour compenser le déséquilibre et améliorer l'apprentissage :

Sans les poids :

- le modèle peut prédire trop souvent **0** ("inoccupé")

- il peut négliger les vrais **1** ("occupé") ⇒ ce qui est mauvais

Avec les poids :

✔️ meilleure précision globale
✔️ meilleur rappel sur la classe 1
✔️ moins de biais vers la classe majoritaire

RESULTATS :

Poids des classes : {0: np.float64(0.6557317952415285), 1: np.float64(2.105324074074074)}

INTERPRETATION :

0 → 0.6557 : la classe inoccupée (majoritaire) reçoit **moins de poids**
1 → 2.1053 : la classe occupée (minoritaire) reçoit **plus de poids**

Pourquoi ces valeurs ?

Le calcul automatique (`class_weight='balanced'`) fait :

poids_classe= N/(n_classe×nombre_de_classes)

- N = total d'échantillons dans `y_train`
- n_classe = nombre d'échantillons de cette classe
- nombre_de_classes = 2

Résultat : les classes minoritaires ont un poids plus élevé pour que le modèle **ne les ignore pas**.

3)Split Train / Validation / Test

Distribution des classes

Ensemble	Classe 0 (inoccupé)	Classe 1 (occupé)
Train	78,77 %	21,23 %
Validation	78,75 %	21,25 %
Test	78,77 %	21,23 %

Interprétation

1. La proportion des classes est **quasi identique dans tous les ensembles**
   la stratification (`stratify=y`) a fonctionné parfaitement.
2. Le dataset est **déséquilibré** :
   - Classe majoritaire = 0 (inoccupé) ≈ 79 %
   - Classe minoritaire = 1 (occupé) ≈ 21 %
3. Cela confirme que l'utilisation des **poids de classes** pour l'entraînement est **très utile**, sinon le modèle risquerait de prédire trop souvent 0.

4) Entraînement des 4 modèles de base

Évaluation des Modèles sur le premier Fichier de Test (Test1):

Dans cette section, nous entraînons quatre modèles de classification classiques utilisés

- **Decision Tree (DT)**
- **Random Forest (RF)**
- **Support Vector Machine (SVM)**
- **k-Nearest Neighbors (kNN)**

L'objectif est de comparer leurs performances sur les données d'occupation des salles afin d'identifier les modèles les plus efficaces pour la détection.

Méthodologie

Les données ont été divisées en trois sous-ensembles :

- 70% pour l'**entraînement (Train)**
- 15% pour la **validation (Val)**
- 15% pour le **test final (Test1)**

Une fonction générique `train_and_evaluate()` a été utilisée pour :
- entraîner chaque modèle
- réaliser les prédictions
- calculer les métriques : précision (accuracy), rappel (recall), F1-score
- afficher la matrice de confusion
- générer le rapport de classification

Cette approche permet d'évaluer la performance des modèles sur les différentes étapes.

Résultats des Modèles

Les performances obtenues sont résumées ci-dessous :

Decision Tree

- **Train accuracy : 100%** : surapprentissage (overfitting)
- **Validation accuracy : 99,26%**
- **Test accuracy : 84,94%**

L’arbre de décision surapprend fortement et généralise mal sur le test.

Points importants :

- Rappel (classe 1) = 65% : le modèle rate beaucoup de salles réellement occupées.
- Très forte variance.

Random Forest

- **Train accuracy : 100%**
- **Validation accuracy : 99,63%**
- **Test accuracy : 95,78%**

Très bonne performance générale.

Points importants :

- F1-score = 0.94
- Très bon équilibre biais/variance
- Mieux que l’arbre seul grâce à l’agrégation de plusieurs arbres

Support Vector Machine (SVM) – Lineaire

- **Train accuracy : 98,8%**
- **Validation accuracy : 98,6%**
- **Test accuracy : 97,79%**

Meilleur modèle sur Test1

Points importants :

- Rappel pour la classe occupée = 99,8% (!!)
- Très bon pour éviter les faux négatifs (mettre "non occupée" alors que la salle est occupée).
- Très robuste au bruit et aux données déséquilibrées.

k-Nearest Neighbors (kNN)

- **Train accuracy : 99,5%**
- **Validation accuracy : 99,5%**
- **Test accuracy : 91,28%**

Bon modèle mais moins performant que SVM et RF.

Points importants :

- Rappel = 83%
- Sensible à l'échelle des données et au bruit
- Coût élevé en temps à l'inférence

Comparaison Globale

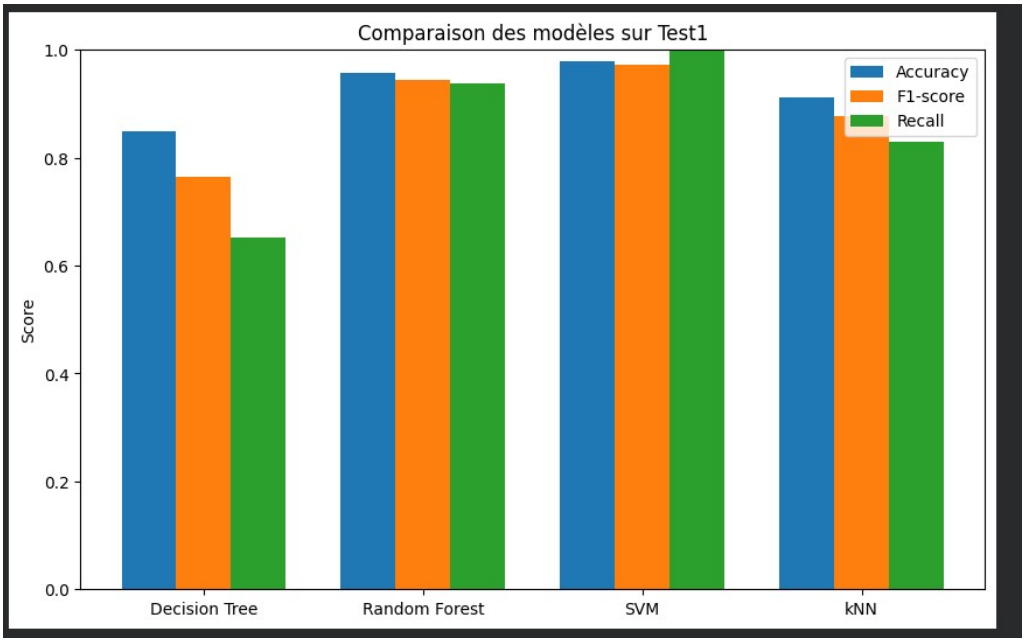
Modèle	Train Acc	Val Acc	Test Acc	Observations
Decision Tree	100%	99.26%	**84.94%**	Surapprentissage, mauvaise généralisation
Random Forest	100%	99.63%	**95.78%**	Très performant et stable
SVM (linéaire)	98.80%	98.65%	**97.79%**	**Meilleur modèle**, excellent rappel
kNN	99.52%	99.50%	**91.29%**	Bon mais limité, sensible au bruit

Conclusion:

Le **SVM** est le modèle le plus performant sur l'ensemble des métriques, suivi par le **Random Forest**.

Le Decision Tree présente un fort overfitting.

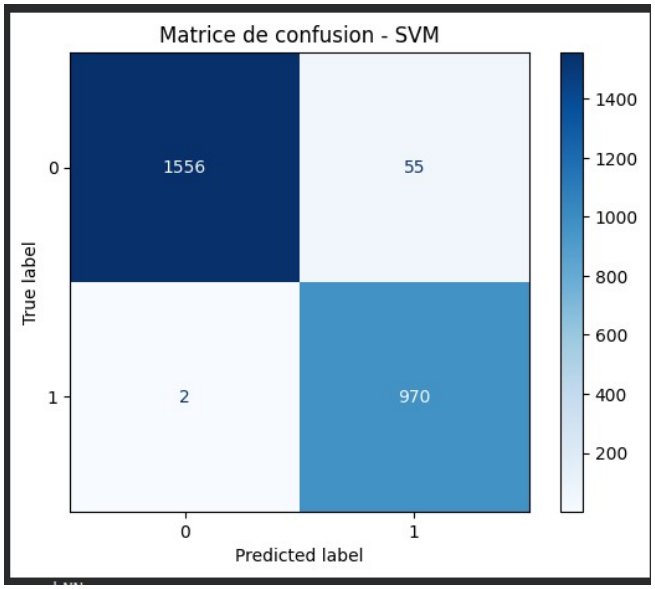
Le kNN est correct mais moins compétitif.



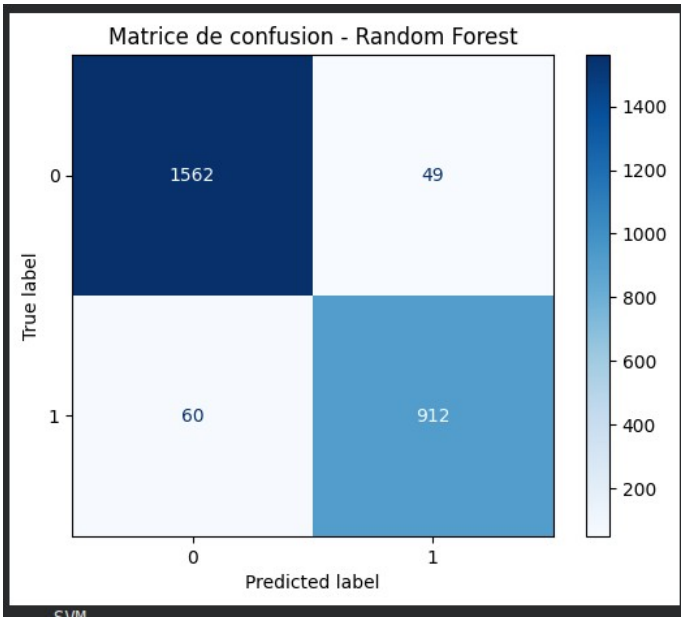
Visualisation : Matrices de Confusion

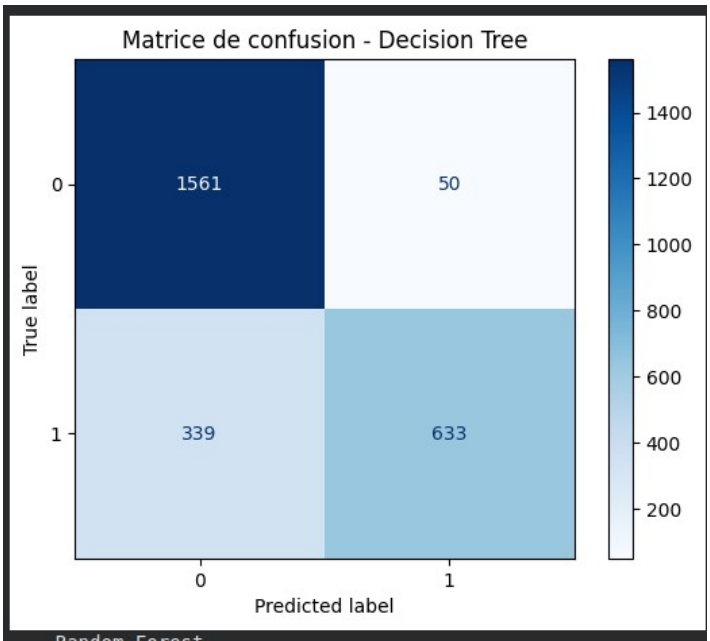
Les matrices de confusion affichées dans le notebook montrent visuellement les erreurs de classification :

- **SVM : très peu d'erreurs : excellent modèle**



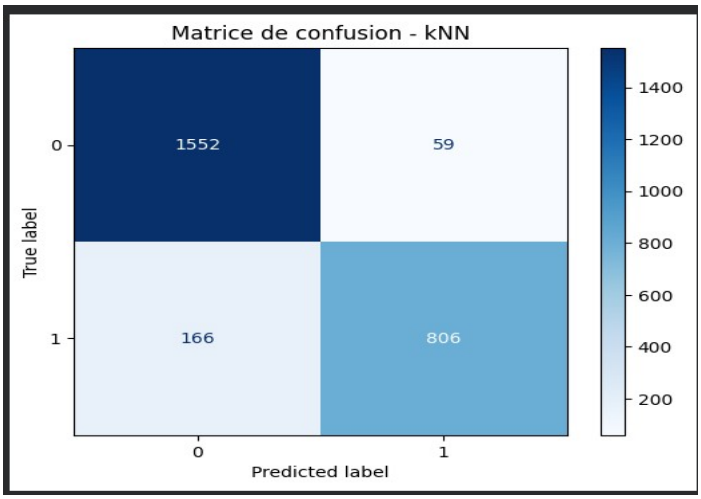
RF : quelques erreurs de la classe 1





- **DT : beaucoup de confusion entre classes**

- **kNN : bonne séparation mais erreurs sur la classe occupée**



Au vu des résultats obtenus, le SVM linéaire apparaît comme le meilleur candidat pour servir de modèle de base dans l'architecture de meta-learning, en raison de sa capacité à généraliser efficacement sur les données de test tout en maintenant une excellente précision et un rappel élevé.

Évaluation des Modèles sur le Deuxième Fichier de Test (Test2)

Après l'évaluation initiale réalisée sur le premier fichier de test (Test1), nous avons procédé à une seconde évaluation indépendante en utilisant le fichier Test2, constitué de 9264 observations.
L'objectif est de vérifier la capacité de généralisation des modèles dans un contexte plus large et plus varié.

Interprétation des Résultats

Cette seconde évaluation confirme les tendances déjà observées sur Test1 :

- Le SVM reste le meilleur modèle

Il offre une précision et un rappel extrêmement élevés.
Il minimise les fausses prédictions, ce qui est essentiel pour une application d'occupation de salles.

- Le Random Forest se place en deuxième position

Très bon compromis entre précision, stabilité et rapidité.

- Le Decision Tree est le plus faible

Même si l'accuracy reste correcte, son rappel sur la classe *occupée* reste bas : ce modèle manque de fiabilité.

- Le kNN est performant mais inférieur

Il fait beaucoup plus d'erreurs dans la détection de la classe *occupée*.

En conclusion l'évaluation sur Test2 renforce la conclusion précédente : le modèle **SVM** présente les meilleures performances globales et constitue le meilleur candidat pour servir de modèle de base dans le système de meta-learning.
Le Random Forest suit de très près, tandis que le Decision Tree et le kNN restent moins adaptés aux exigences du problème.

5) Sauvegarde des modèles (.pkl)

Les modèles finaux ont été enregistrés pour utilisation ultérieure :