المدرسة الوطنية للعلوم التطبيقية - القنيطرة
National School of Applied Sciences - Kenitra

Rapport de Projet Python

Analyse et Implémentation d'Algorithmes de Machine Learning

Sujet : HR Analytics & Prédiction de Satisfaction

Réalisé par :

Riad Zaid

Youssef Sadiqui

M Saad Oussama

Filière :

Génie Informatique

Encadré par :

Pr. EL Azhari

Département Informatique

Année Universitaire 2025 – 2026

Table des matières

1 Introduction et Configuration **1**
1.1 Environnement Technique 1
1.2 Implémentation et Chargement des Données 1

2 Phase 1 : Structuration des Données (Data Engineering) **2**
2.1 Inspection et Exploration Initiale 2
2.2 Nettoyage et Optimisation des Données 3
2.3 Transformation et Feature Engineering 4
2.4 Finalisation et Exportation 4

3 Phase 2 : Analyse Exploratoire (EDA) **5**
3.1 Distribution des Âges 5
3.2 Répartition par Département 6
3.3 Corrélation : Expérience vs Revenu 7
3.4 Analyse Multivariée (3D) 9
3.5 Analyse Multivariée des Risques (5D) 9
3.6 Analyse de l'Équité Salariale (Seaborn) 12
3.7 Comparaison par Département (Faceting) 13
3.8 Progression de Carrière et Grille Salariale 14

4 Phase 3 : Modélisation Prédictive (Régression) **16**
4.1 Phase 3.1 : Régression Linéaire Simple (Modèle Baseline) 16
4.1.1 Objectif et Paramètres 16
4.1.2 Implémentation Python 16
4.1.3 Analyse des Résultats 17
4.2 Phase 3.2 : Régression Linéaire Multiple (Modèle Avancé) 17
4.2.1 Enrichissement du Modèle 17
4.2.2 Implémentation 17
4.3 Phase 3.3 : Analyse de Tendance (Modèle ARIMA) 18
4.3.1 Méthodologie 18
4.3.2 Implémentation ARIMA 18
4.3.3 Performance et Interprétation 18
4.4 Bilan et Benchmark des Modèles Prédictifs 19
4.4.1 Tableau Récapitulatif 19
4.4.2 Conclusion du Benchmark 19

5 Phase 4 : Analyse Qualitative (La "Boussole du Bien-être") **19**
5.1 Préparation des Données : Binarisation 19
5.2 Extraction des "Règles d'Or" 20
5.3 Résultats : Les Micro-Climats de Satisfaction 20

5.4 Application : Le Moteur de Scoring "Bien-être" 21
5.4.1 Logique de l'Algorithme 21
5.5 Déploiement et Sérialisation 22

6 Conclusion Générale **22**

7 PHASE 5 Algorithmes Clustrings+Classifications **22**
7.1 Algorithme de Clustering K-Means 22
7.2 Étapes de l'algorithme 23
7.3 Formulation Mathématique 23

8 Détermination du nombre de clusters (Méthode du Coude) **23**

9 Visualisation des Segments RH **23**
9.1 Interprétation des KPI 23

10 Application du Modèle K-Means et Analyse des Profils **24**
10.1 Exécution du clustering final 24
10.2 Caractérisation des Groupes (KPI) 24
10.3 Évaluation de la qualité : Le Score de Silhouette 25
10.4 Implémentation du K-Means Graphiquement 25
10.5 Analyse de la Segmentation 26

11 Clustering Hiérarchique (CAH) **27**
11.1 Le Dendrogramme 27
11.2 Interprétation et Validation 28
11.3 Implémentation du Modèle CAH 28
11.3.1 Justification Mathématique de la Ligne de Coupure 29

12 Validation et Choix du Meilleur Modèle **29**
12.1 Définition du Score de Silhouette 29
12.2 Comparaison K-Means vs CAH 29
12.3 Résultats et Décision Finale 30

13 Classification Supervisée (Arbre de Décision) **31**
13.1 Implémentation du Modèle 31
13.2 Évaluation des Performances 31
13.3 Visualisation de l'Arbre (Les Règles RH) 32
13.4 Interprétation des Règles 33

14 Classification K-NN (K-Nearest Neighbors) **34**
14.1 Optimisation de l'Hyperparamètre K 34
14.2 Interprétation et Entraînement Final 35

15 Classification Probabiliste (Naïve Bayes) 36
15.1 Implémentation (Gaussian Naïve Bayes) . . . 36
15.2 Analyse de la Matrice de Confusion . . . 37
15.3 Comparatif et Conclusion Technique . . . 38

16 Random Forest (L'Intelligence Collective) 38
16.1 Implémentation et Entraînement . . . 38
16.2 Analyse des Facteurs Clés (Feature Importance) . . . 39
16.3 Performance et Matrice de Confusion . . . 40
16.4 Simulation : Test sur un Profil Fictif . . . 41
16.5 Sauvegarde et Déploiement . . . 41

17 Régression Logistique (L'Approche Explicative) 42
17.1 Prérequis : La Normalisation . . . 42
17.2 Le Pouvoir des Coefficients (Facteurs d'Influence) . . . 42
17.3 Performance et Limites . . . 43
17.4 Simulation d'un Cas Concret . . . 44

18 Support Vector Machine (L'Approche Géométrique) 45
18.1 Configuration et "Kernel Trick" . . . 45
18.2 Analyse Avancée : La Courbe Précision-Rappel . . . 45
18.3 Performance Globale . . . 47
18.4 Simulation et Conclusion Technique . . . 47

19 Benchmark et Choix Final du Modèle 48
19.1 Tableau Récapitulatif des Performances . . . 48
19.2 Analyse Graphique : Précision vs Rappel . . . 48
19.3 Verdict et Recommandation Stratégique . . . 49

Table des figures

1 Répartition des employés par tranche d'âge 6
2 Effectifs par département 7
3 Corrélation entre Années d'Expérience et Salaire Mensuel 8
4 Projection 3D : Âge, Expérience et Revenu 10
5 Visualisation des profils à risque (Départ en Rouge) 11
6 Comparaison des revenus Hommes vs Femmes 12
7 Dynamique salariale et départs par département 13
8 Évolution salariale par niveau hiérarchique 15
9 Classement des facteurs les plus présents dans la Haute Satisfaction 21
10 Visualisation des segments employés (Âge vs Revenu) 24
11 Visualisation des 3 segments d'employés détectés 26
12 Dendrogramme déterminant le nombre optimal de clusters 28
13 Matrice de confusion : Capacité à prédire les départs 32
14 Visualisation des règles de décision (Rotation 90°) 33
15 Évolution de l'erreur : Le choix optimal de K 35
16 Performance du modèle K-NN sur les données de test 36
17 Performance du classifieur probabiliste 37
18 Graphe illustrant l'algorithme 38
19 Hiérarchie des facteurs influençant l'attrition 40
20 Capacité de détection du Random Forest 41
21 Facteurs aggravants (Rouge) vs Facteurs protecteurs (Vert) 43
22 Matrice de confusion : Un modèle conservateur 44
23 Courbe Précision-Rappel : Un comportement "élitiste" 46
24 Matrice de confusion du SVM 47
25 Compromis Précision (Bleu) vs Rappel (Rouge) 49

1 Introduction et Configuration

1.1 Environnement Technique

Pour mener à bien ce projet d'analyse de données et de Machine Learning, nous utilisons le langage **Python**. Avant de manipuler les données, il est essentiel d'importer les bibliothèques scientifiques standards.

Voici une présentation des outils utilisés :

Pandas

Bibliothèque incontournable pour la manipulation et l'analyse de données. Elle nous permet de structurer les données sous forme de *DataFrames* pour faciliter le nettoyage et l'exploration

NumPy

Fondamentale pour le calcul scientifique, cette librairie permet de gérer des tableaux multidimensionnels et d'effectuer des opérations mathématiques performantes

Matplotlib

La bibliothèque standard pour la création de graphiques en Python. Elle sert de base pour visualiser les distributions et les relations simples

Seaborn

Surcouche basée sur Matplotlib, elle offre une interface de haut niveau pour créer des graphiques statistiques plus esthétiques et informatifs

1.2 Implémentation et Chargement des Données

Le script suivant assure l'installation des dépendances et le chargement sécurisé du fichier de données `WA_Fn-UseC_-HR-Employee-Attrition.csv`.

```
# Installation des librairies necessaires
!pip install pandas numpy matplotlib seaborn

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Collecte des donnees
# Lecture du fichier CSV (gestion automatique du separateur)
try:
    df = pd.read_csv('WA_Fn-UseC_-HR-Employee-Attrition.csv', sep=',')

    # Verification : Si tout est dans une seule colonne, on change de
    separateur
    if df.shape[1] == 1:
```

```
        df = pd.read_csv('WA_Fn-UseC_-HR-Employee-Attrition.csv', sep=';')

    print("Donnees chargees avec succes.")

except FileNotFoundError:
    print("Erreur : Fichier introuvable.")
```

Listing 1 – Script d'initialisation et chargement des données

2 Phase 1 : Structuration des Données (Data Engineering)

2.1 Inspection et Exploration Initiale

Une fois les données chargées, il est impératif de vérifier leur intégrité. Cette étape consiste à contrôler les dimensions du jeu de données, les types de variables (numériques ou catégorielles) et à visualiser les premières lignes pour comprendre la structure du fichier.

```
# 2. Inspection
print(f"Dimensions initiales : {df.shape}")

print("\nTypes des donnees :")
print(df.dtypes.value_counts()) # Resume des types

print("\nApercu des premieres lignes :")
display(df.head())
```

Listing 2 – Inspection de la structure du DataFrame

Résultat de l'exécution

L'exécution du script ci-dessus nous fournit les informations suivantes sur la structure de notre dataset *HR-Employee-Attrition* :

- **Dimensions :** Le jeu de données contient **1470 lignes** (employés) et **35 colonnes** (attributs).
- **Types :** Nous observons une majorité de variables numériques (26 `int64`) et quelques variables catégorielles (9 `object`).

Aperçu des 5 premières lignes (Extrait des colonnes principales) :

Index	Age	Attrition	BusinessTravel	DailyRate	Department
0	41	Yes	Travel_Rarely	1102	Sales
1	49	No	Travel_Frequently	279	Research & Dev
2	37	Yes	Travel_Rarely	1373	Research & Dev
3	33	No	Travel_Frequently	1392	Research & Dev
4	27	No	Travel_Rarely	591	Research & Dev

Note : Le tableau ci-dessus est un extrait simplifié pour la lisibilité du rapport. Il met en évidence la variable cible "Attrition" ainsi que les premières caractéristiques descriptives.

2.2 Nettoyage et Optimisation des Données

Après l'inspection initiale, nous avons procédé au nettoyage du jeu de données pour garantir la qualité des analyses.

1. Suppression des doublons

Nous filtrons les lignes pour ne garder que les entrées uniques. Cela évite le sur-apprentissage (*overfitting*) et garantit qu'un même employé ne pèse pas deux fois dans les statistiques.

Commande :
`df.drop_duplicates()`

2. Suppression des colonnes constantes

Les variables à variance nulle n'apportent aucune information discriminante. Nous avons retiré trois colonnes inutiles (*EmployeeCount, Over18, StandardHours*) qui avaient la même valeur pour tous les employés.

Commande :
`df.drop(columns=...)`

Bilan du Nettoyage

Suite à ces opérations, la dimension du jeu de données a été optimisée.
Nous passons de **35 variables** initiales à **32 variables pertinentes**.

3. Gestion des valeurs manquantes

Nous avons vérifié la présence de données nulles ou manquantes (*NaN*). Une donnée incomplète peut faire planter les modèles de Machine Learning.
Résultat : **Aucune valeur manquante** n'a été détectée. Le jeu de données est complet, aucune imputation (remplissage automatique) n'a été nécessaire.

Vérification :
`df.isnull().sum()`

4. Standardisation des formats (Texte)

Pour assurer une cohérence visuelle dans les graphiques, nous avons normalisé les chaînes de caractères.
L'opération consiste à supprimer les espaces superflus et à appliquer le format "Titre" (Majuscule au début).
Exemple : `"travel_rarely"` devient `"Travel_Rarely"`.

Transformation :
`.str.strip().str.title()`

5. Détection des valeurs aberrantes (Outliers)

Les salaires extrêmes peuvent fausser les modèles prédictifs (notamment les régressions). Nous avons analysé la variable `MonthlyIncome` en utilisant la méthode statistique de l'**Écart Interquartile (IQR)**.

— Calcul de l'intervalle : $IQR = Q3 - Q1$.

— Filtrage : On ne garde que les valeurs comprises entre $[Q1 - 1.5 \times IQR]$ et $[Q3 + 1.5 \times IQR]$.

Résultat : 114 employés ayant des salaires "hors normes" ont été écartés pour ne pas biaiser l'analyse moyenne.

Méthode :
`.quantile(0.25) / (0.75)`

Dimension finale :
1356 lignes

Conclusion :

Le jeu de données est désormais **nettoyé, structuré et optimisé**. Nous passons de 1470 à **1356 observations** de qualité, prêtes pour l'Analyse Exploratoire (EDA).

2.3 Transformation et Feature Engineering

Pour enrichir l'analyse et préparer les données aux algorithmes mathématiques, nous avons créé de nouvelles variables.

1. Création de variable (Calcul)

Nous avons généré la variable `AnnualIncome` (Revenu Annuel) en multipliant le revenu mensuel par 12. Cela permet d'avoir une vision plus globale de la rémunération.

Formule :
`MonthlyIncome * 12`

2. Encodage de la variable cible

Les modèles de Machine Learning ne traitent pas le texte ("Yes"/"No"). Nous avons utilisé un **Label Encoder** pour transformer la variable `Attrition` en format numérique binaire :

— **Yes** $\rightarrow$ **1** (Départ)

— **No** $\rightarrow$ **0** (Reste)

Outil Scikit-Learn :
`LabelEncoder()`

Nouvelle Colonne :
`Attrition_Numeric`

2.4 Finalisation et Exportation

Avant de passer à l'analyse visuelle, nous effectuons deux dernières opérations techniques pour figer la structure du jeu de données.

1. Structuration (Indexation)

Chaque employé possède un identifiant unique. Pour optimiser les recherches et garantir l'intégrité des lignes, nous définissons la variable `EmployeeNumber` comme index du DataFrame (équivalent d'une clé primaire en base de données).

Commande : `.set_index(...)`

2. Stockage (Sauvegarde)

Le jeu de données nettoyé, filtré et enrichi est sauvegardé dans un nouveau fichier : `HR_Analytics_Structure_Complet.csv`. Ce fichier "propre" servira de base immuable pour toute la phase de visualisation (EDA) et de modélisation, évitant d'avoir à relancer tout le nettoyage à chaque fois.

Export : `.to_csv('...')`

✓ Fin de la Phase 1 : Data Engineering

Les données sont prêtes pour l'Analyse Exploratoire.

3 Phase 2 : Analyse Exploratoire (EDA)

L'objectif de cette phase est de comprendre la structure des données à travers des graphiques. Nous commençons par des analyses univariées (une seule variable à la fois).

3.1 Distribution des Âges

Nous analysons la pyramide des âges de l'entreprise pour voir si la population est plutôt junior ou senior.

```
# --- 1. Histogramme (Matplotlib) ---
plt.figure(figsize=(8, 5))

# Creation de l'histogramme (bins=20 pour la finesse)
plt.hist(df['Age'], bins=20, color='skyblue', edgecolor='black', alpha=0.7)

# Titres et Labels
plt.title("Distribution des Ages des Employes")
plt.xlabel("Age")
plt.ylabel("Nombre d'employes")
plt.grid(True, axis='y', alpha=0.5)

# Sauvegarde et Affichage
plt.savefig('distribution_age.png')
plt.show()
```

Listing 3 – Génération de l'histogramme des âges

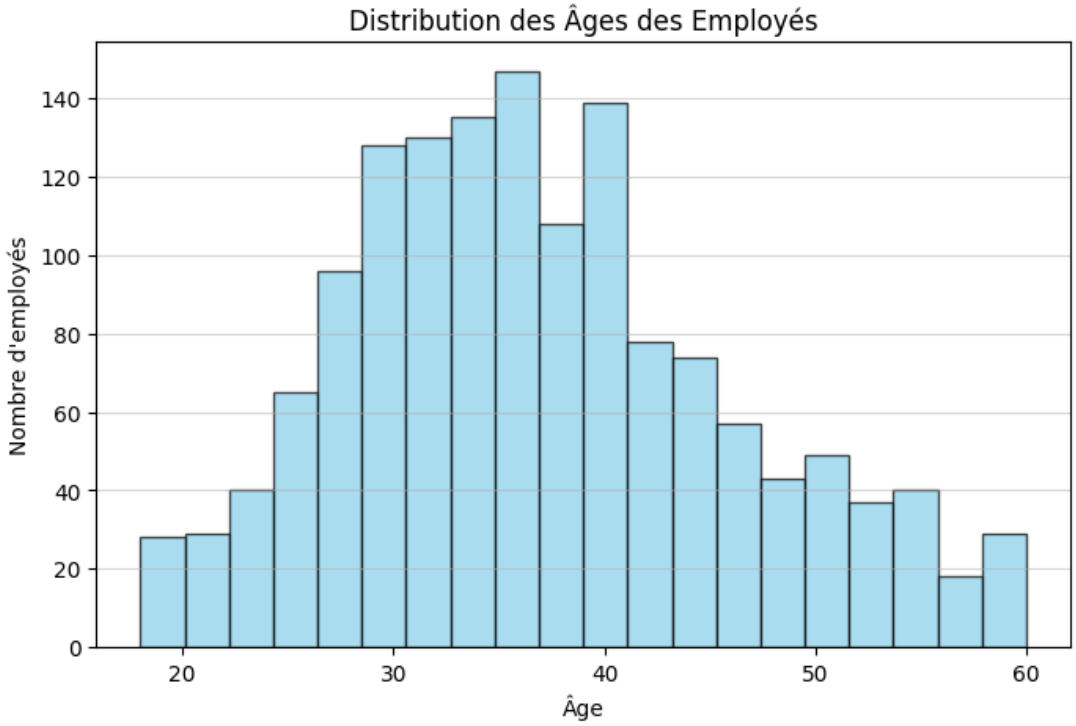

FIGURE 1 – Répartition des employés par tranche d'âge

Interprétation :
L'histogramme révèle une distribution quasi-normale (en forme de cloche).

— La majorité des effectifs se situe entre **30 et 40 ans**.

— On observe peu d'employés très jeunes (< 20 ans) ou proches de la retraite (> 55 ans).

— Cette concentration sur la tranche médiane indique une main-d'œuvre expérimentée.

3.2 Répartition par Département

Il est essentiel de comprendre comment les effectifs sont distribués au sein de l'organisation pour identifier les départements dominants.

```
# --- 2. Diagramme en Barres Vertical (Matplotlib) ---
# Preparation des donnees : compte par departement
dept_counts = df['Department'].value_counts()

plt.figure(figsize=(8, 5))

# Barres verticales avec couleurs distinctes
plt.bar(dept_counts.index, dept_counts.values, color=['blue', 'orange', '
    green'])

# Mise en forme
plt.title("Nombre d'employés par Département")
```

```
plt.xlabel("Département")
plt.ylabel("Effectif")
plt.grid(axis='y', alpha=0.5)

# Sauvegarde pour le rapport
plt.savefig('departement_dist.png')
plt.show()
```

Listing 4 – Analyse des effectifs par département

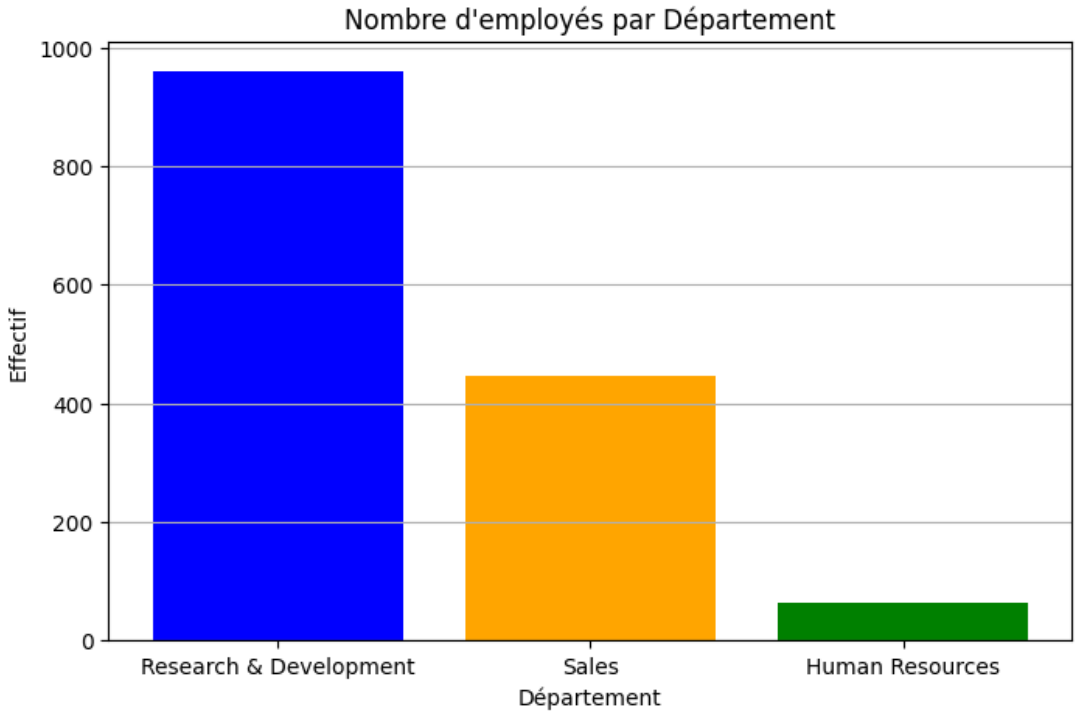

FIGURE 2 – Effectifs par département

Interprétation :

Le graphique met en évidence une forte disparité entre les pôles d'activité :

- **Research & Development (R&D)** est largement majoritaire, regroupant près de 65% des effectifs (plus de 900 employés).
- **Sales** (Vente) représente le second pilier de l'entreprise.
- **Human Resources** (RH) constitue une minorité marginale (< 100 employés), ce qui est cohérent pour une fonction support.

3.3 Corrélation : Expérience vs Revenu

Nous cherchons à vérifier l'hypothèse selon laquelle l'ancienneté globale (*TotalWorkingYears*) est le principal facteur déterminant le salaire mensuel.

```python
# --- 3. Scatter Plot + Regression Lineaire ---

# 1. Tracé du nuage de points (Scatter)
plt.scatter(df['TotalWorkingYears'], df['MonthlyIncome'], alpha=0.5)

# 2. Calcul mathématique de la tendance (Régression)
# np.polyfit(deg=1) calcule les coefficients de la droite (y = ax + b)
coeffs = np.polyfit(df['TotalWorkingYears'], df['MonthlyIncome'], deg=1)
droite = np.poly1d(coeffs)

# 3. Tracé de la ligne rouge
plt.plot(x_vals, droite(x_vals), color='red', label='Tendance')

plt.savefig('experience_revenu.png')
plt.show()
```

Listing 5 – Calcul de la régression linéaire

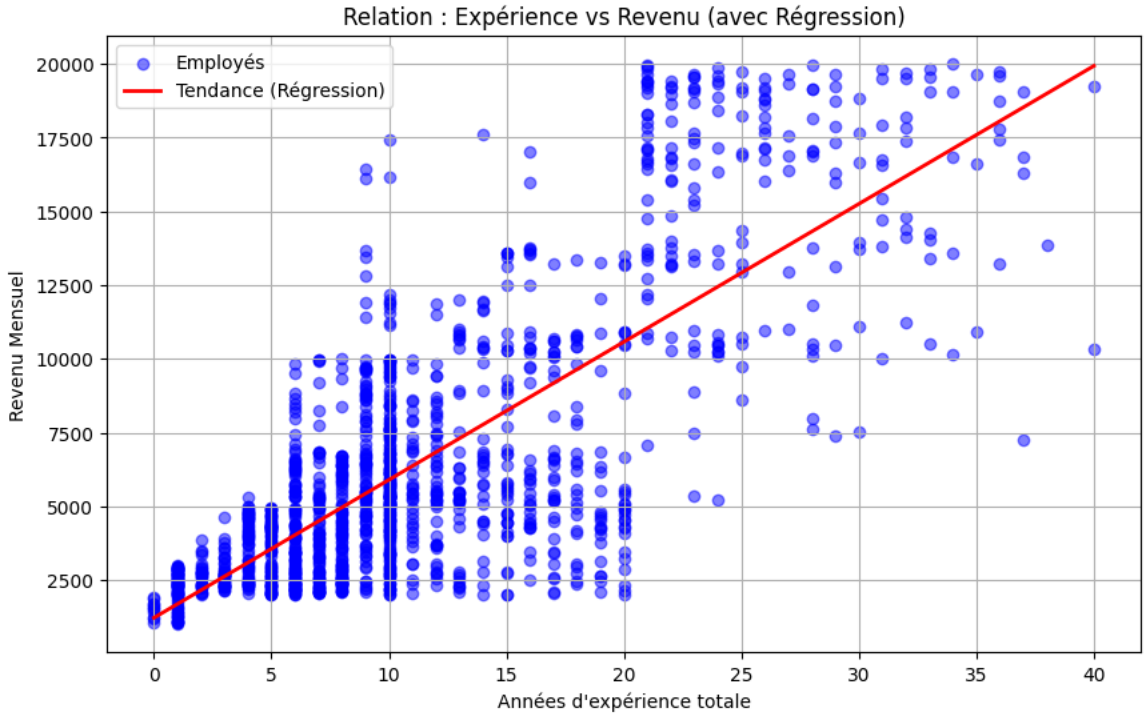

FIGURE 3 – Corrélation entre Années d'Expérience et Salaire Mensuel

Interprétation :

Le graphique révèle une **corrélation positive forte** :

- La ligne de régression (en rouge) monte nettement, confirmant que le salaire augmente avec l'expérience.
- **Junior (0-5 ans)** : Les salaires sont très concentrés en bas de l'échelle (< 5000).
- **Senior (> 10 ans)** : La dispersion augmente considérablement. À expérience égale (ex :

15 ans), les salaires peuvent varier du simple au double, suggérant que le *JobRole* ou la performance individuelle entrent alors en jeu.

3.4 Analyse Multivariée (3D)

Pour visualiser simultanément l'impact de l'âge et de l'expérience sur le revenu, nous utilisons une projection tridimensionnelle.

```
# --- 4. Scatter Plot 3D (Matplotlib) ---
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d') # Activation 3D

# Tracé 3D : Age (X), Experience (Y), Revenu (Z)
# c=zs : La couleur depend du revenu (Echelle Viridis)
scatter = ax.scatter(df['Age'], df['TotalWorkingYears'], df['MonthlyIncome'],
                     c=df['MonthlyIncome'], cmap='viridis', alpha=0.6)

# Labels
ax.set_xlabel('Age')
ax.set_ylabel('Annees Experience')
ax.set_zlabel('Revenu Mensuel')

plt.colorbar(scatter, label='Revenu Mensuel')
plt.savefig('analyse_3d.jpg')
plt.show()
```

Listing 6 – Visualisation 3D (Matplotlib)

Interprétation :

Cette vue tridimensionnelle permet de valider la cohérence structurelle des données :

- **Cohérence Haute (Points jaunes) :** Les hauts revenus se situent logiquement dans la zone combinant un âge élevé et une forte expérience.
- **Cohérence Basse (Points violets) :** Les bas revenus sont concentrés à l'origine des axes, correspondant aux jeunes débutants.
- **Détection d'anomalies :** La linéarité du nuage de points confirme qu'il n'y a pas d'aberrations majeures (comme un profil "Junior sans expérience" avec un revenu de "Senior").

3.5 Analyse Multivariée des Risques (5D)

Nous poussons l'analyse plus loin en intégrant deux nouvelles dimensions visuelles pour identifier les profils à risque :

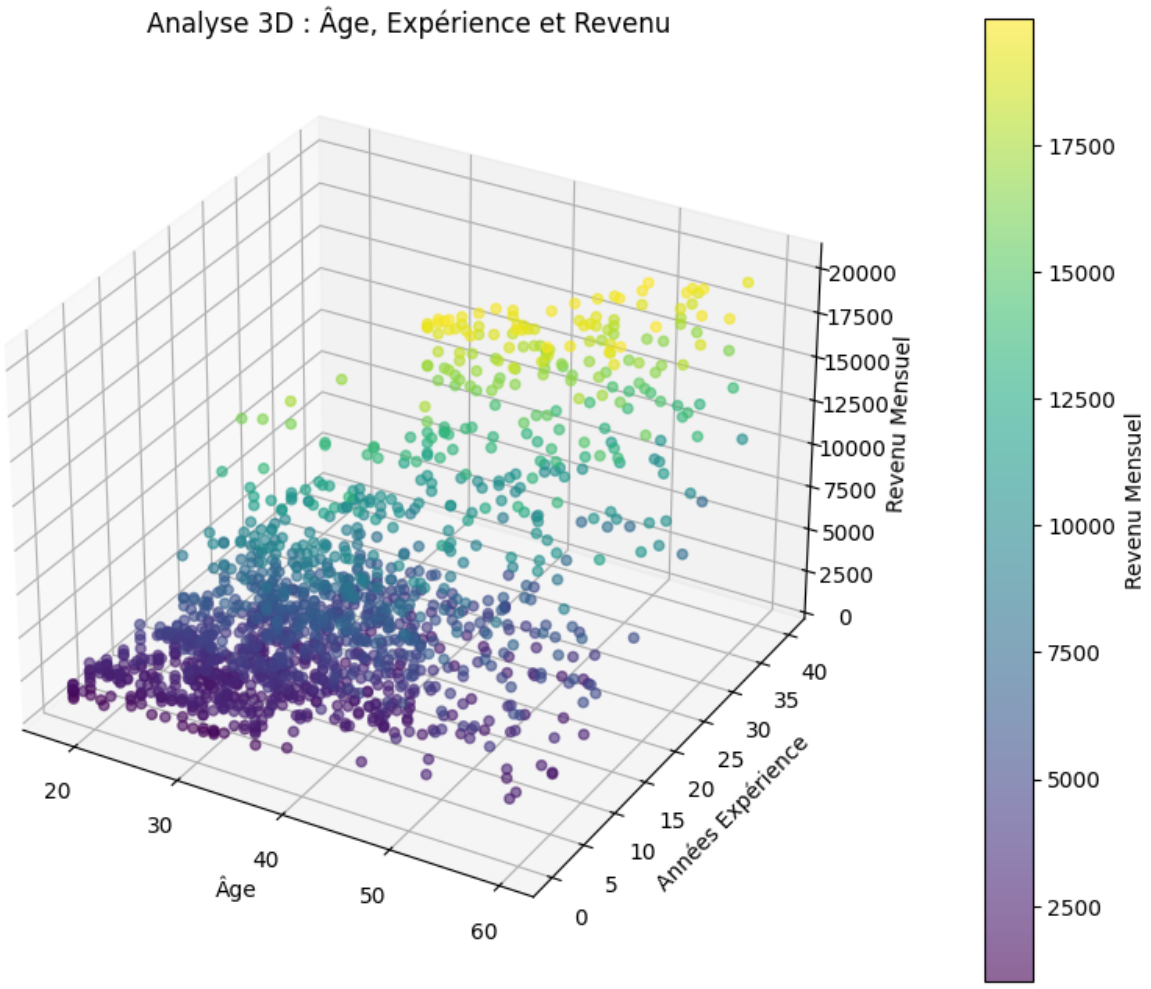

FIGURE 4 – Projection 3D : Âge, Expérience et Revenu

— **La Couleur** représente l'Attrition (Rouge = Départ, Bleu = Reste).

— **La Taille** des points représente le niveau de Satisfaction (Petit = Insatisfait).

```
# --- 5. Scatter Plot 3D Avancé (Risques) ---
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Definition des dimensions
xs = df['Age']
ys = df['TotalWorkingYears']
zs = df['MonthlyIncome']
colors = df['Attrition_Numeric'] # Couleur selon le depart (0 ou 1)
sizes = df['JobSatisfaction'] * 30 # Taille selon la satisfaction (x30 pour
    visibilite)

# Trace avec 5 dimensions
scatter = ax.scatter(xs, ys, zs, c=colors, s=sizes, cmap='coolwarm', alpha
    =0.6)

# Labels
```

```
ax.set_xlabel('Age')
ax.set_ylabel('Experience')
ax.set_zlabel('Revenu Mensuel')
ax.set_title("Analyse des Risques : Attrition et Satisfaction")

# Legende manuelle pour la couleur
from matplotlib.lines import Line2D
legend_elements = [
    Line2D([0], [0], marker='o', color='w', markerfacecolor='blue', label='Reste (No)'),
    Line2D([0], [0], marker='o', color='w', markerfacecolor='red', label='Depart (Yes)')
]
ax.legend(handles=legend_elements, loc='upper left')

plt.savefig('analyse_risques_3d.png')
plt.show()
```

Listing 7 – Visualisation 5D : Risques et Satisfaction

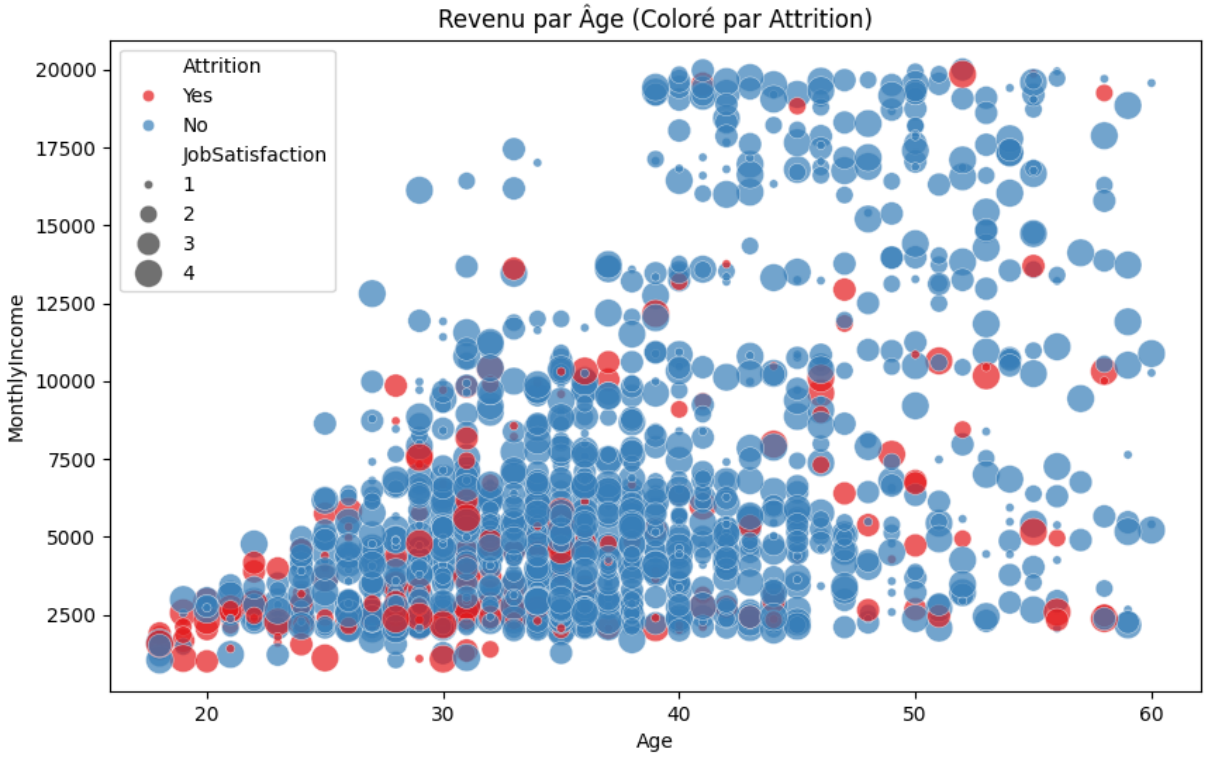

FIGURE 5 – Visualisation des profils à risque (Départ en Rouge)

Interprétation (Analyse des risques) :

Ce graphique est le plus riche en informations de notre étude :

— **Attrition (Couleur) :** On remarque que les points rouges (départs) sont majoritairement situés dans la zone des bas salaires et des jeunes âges.

— **Satisfaction (Taille) :** Les petits points (faible satisfaction) ont tendance à se superposer aux zones de départ.

— **Conclusion :** Le risque de départ est maximal chez les jeunes employés peu payés et dont la satisfaction au travail est faible (score de 1 ou 2).

3.6 Analyse de l'Équité Salariale (Seaborn)

Nous utilisons la bibliothèque *Seaborn* pour visualiser l'évolution des salaires en fonction de l'âge, tout en comparant les dynamiques entre hommes et femmes.

```
# --- 6. Lineplot avec intervalle de confiance (Seaborn) ---
plt.figure(figsize=(10, 6))

# Trace de la moyenne du revenu par Age avec distinction par Genre
sns.lineplot(
    data=df,
    x='Age',
    y='MonthlyIncome',
    hue='Gender',
    errorbar='sd',
    linewidth=2
)

plt.title("Evolution du Revenu moyen par Age et par Genre")
plt.grid(True)
plt.savefig('salaire_genre.png')
plt.show()
```

Listing 8 – Comparaison des revenus par genre

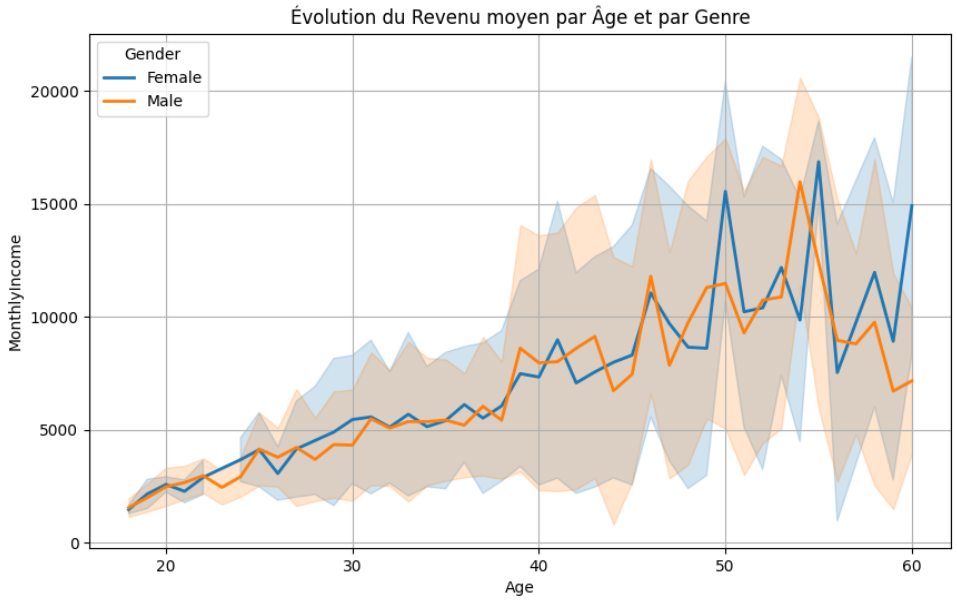

FIGURE 6 – Comparaison des revenus Hommes vs Femmes

Interprétation :

Ce graphique nous permet de statuer sur l'équité salariale :

— **Superposition des courbes :** Les lignes bleue (Femmes) et orange (Hommes) se chevauchent presque parfaitement.

— **Conclusion :** Il n'existe **aucune discrimination salariale significative** basée sur le genre.

3.7 Comparaison par Département (Faceting)

Pour affiner notre compréhension, nous décomposons la relation "Expérience / Revenu" en fonction du département.

```
# --- 7. Relplot - Faceting (Seaborn) ---
# col='Department' cree un graphique par departement
sns.relplot(
    data=df,
    x='TotalWorkingYears',
    y='MonthlyIncome',
    hue='Attrition', # Couleur selon le depart
    col='Department', # Une colonne par departement
    kind='scatter',
    height=5,
    aspect=0.8,
    palette='coolwarm'
)

# Sauvegarde de l'image
plt.savefig('faceting_dept.png')
plt.show()
```

Listing 9 – Comparaison par Département et Attrition

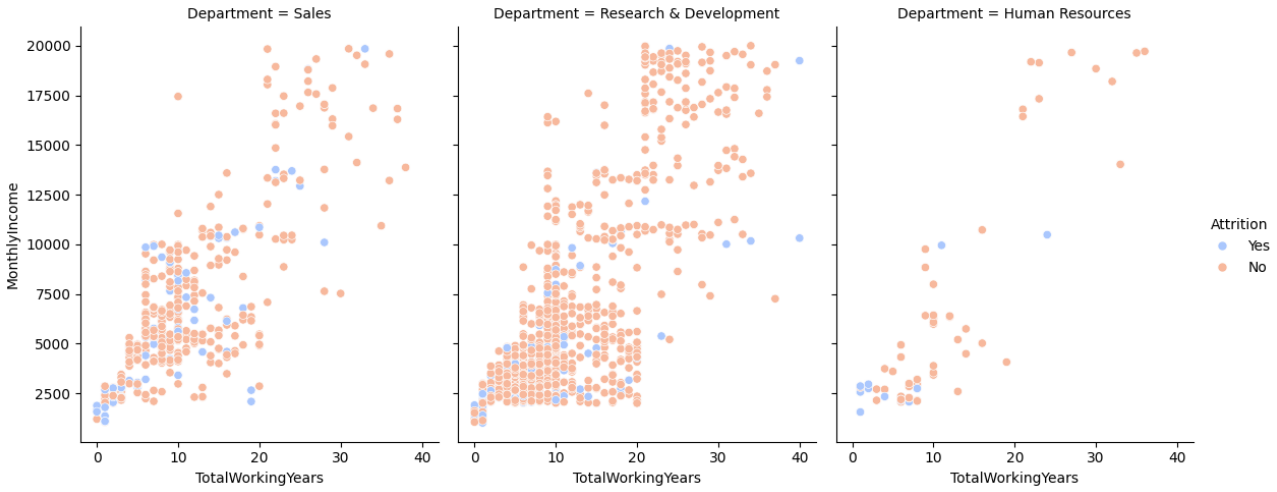

FIGURE 7 – Dynamique salariale et départs par département

Interprétation :

La séparation par département permet de nuancer l'analyse globale :

- **Sales (Vente) :** La corrélation entre expérience et revenu est plus dispersée que la moyenne. Cela s'explique probablement par la part variable (commissions) qui rend les salaires moins linéaires.
- **R&D :** La courbe est très linéaire et prévisible : l'ancienneté paie de manière constante.
- **Human Resources :** L'échantillon est faible, rendant la tendance moins nette.
- **Analyse des Risques (Attrition) :** On observe que les points de couleur "Attrition" (souvent rouge/bleu selon le thème) sont dispersés, mais restent majoritairement en bas de l'échelle des revenus dans les trois départements.

3.8 Progression de Carrière et Grille Salariale

Pour finir, nous analysons comment le salaire évolue avec l'ancienneté, en isolant chaque niveau hiérarchique (*JobLevel*).

```
# --- 8. Relplot - Line (Seaborn) ---
# Visualiser la progression salariale par anciennete, separee par niveau
sns.relplot(
    data=df,
    x='YearsAtCompany',
    y='MonthlyIncome',
    hue='Gender',
    col='JobLevel', # Un graphique par niveau de poste
    col_wrap=3,     # Retour a la ligne apres 3 graphiques
    kind='line',    # Type courbe
    errorbar=None,  # Pas de barre d'erreur pour alleger
    height=3,
    aspect=1.2
)

plt.savefig('salary_progression.png')
plt.show()
```

Listing 10 – Analyse des paliers de rémunération

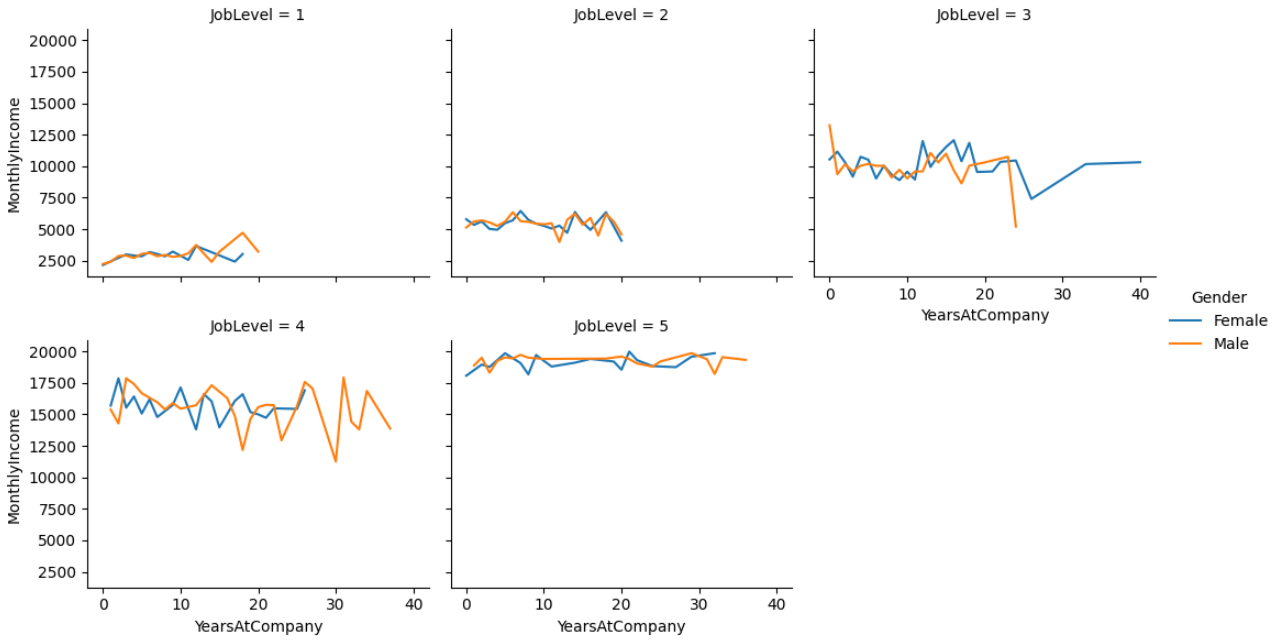

FIGURE 8 – Évolution salariale par niveau hiérarchique

Interprétation :

Ce graphique explique clairement la mécanique des carrières dans l'entreprise :

- **Stratification :** Chaque *JobLevel* correspond à une tranche de salaire distincte (strates horizontales bien définies).
- **Plafonnement :** Au sein d'un même niveau (ex : Niveau 1 ou 2), la courbe est relativement plate. Le salaire augmente peu avec la seule ancienneté (*YearsAtCompany*).
- **Stratégie RH :** Pour augmenter significativement son revenu, un employé doit impérativement changer de niveau (promotion).
- **Conclusion :** La politique de rémunération est basée sur le **grade** et la responsabilité, bien plus que sur l'ancienneté pure.

4 Phase 3 : Modélisation Prédictive (Régression)

L'objectif de ce chapitre est de développer des algorithmes capables d'anticiper le salaire ou l'évolution salariale. Nous avons procédé par itération, du modèle le plus simple au plus complexe.

4.1 Phase 3.1 : Régression Linéaire Simple (Modèle Baseline)

L'objectif de cette première étape est de tester l'hypothèse fondamentale selon laquelle le salaire est directement proportionnel aux années d'expérience globale.

4.1.1 Objectif et Paramètres

Ce modèle utilise une approche univariée pour établir une ligne de base (*baseline*) :

— **Variable Indépendante (X)** : `TotalWorkingYears` (Expérience totale).

— **Variable Cible (y)** : `MonthlyIncome` (Salaire mensuel).

4.1.2 Implémentation Python

Le script suivant entraîne le modèle pour trouver la droite $y = ax + b$ la plus ajustée.

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# 1. Sélection des variables
X = df[['TotalWorkingYears']]
y = df['MonthlyIncome']

# 2. Division Train/Test (80% / 20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# 3. Création et apprentissage
model_simple = LinearRegression()
model_simple.fit(X_train, y_train)

# 4. Prédictions
y_pred = model_simple.predict(X_test)
```

Listing 11 – Entraînement de la Régression Simple

4.1.3 Analyse des Résultats

Métrique	Valeur
Coefficient de détermination (R^2)	0.4941
Erreur Absolue Moyenne (MAE)	1 824.91 $
Erreur en Pourcentage (MAPE)	**39.45 %**

TABLE 1 – Performance de la régression simple

Conclusion technique : Le score R^2 de 0.49 indique que l'expérience seule n'explique que la moitié de la variance des salaires. Avec une erreur de près de 40%, ce modèle est insuffisant, justifiant le passage à une approche multivariée.

4.2 Phase 3.2 : Régression Linéaire Multiple (Modèle Avancé)

Pour capturer la complexité de la structure salariale, nous avons intégré des dimensions métiers et hiérarchiques.

4.2.1 Enrichissement du Modèle

Nous avons ajouté des variables catégorielles encodées :

— **Numériques :** `TotalWorkingYears`, `JobLevel`, `Age`.

— **Catégorielles :** `Department`, `JobRole` (Encodage *One-Hot*).

4.2.2 Implémentation

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# 1. Encodage One-Hot des variables catégorielles
features = ['TotalWorkingYears', 'JobLevel', 'Age', 'Department', 'JobRole']
X = pd.get_dummies(df[features], drop_first=True)
y = df['MonthlyIncome']

# 2. Entraînement
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)
model_multiple = LinearRegression()
model_multiple.fit(X_train, y_train)

# 3. Export pour l'application
joblib.dump(model_multiple, 'modele_salaire_avance.pkl')
```

Listing 12 – Régression Multiple avec Encodage

4.3 Phase 3.3 : Analyse de Tendance (Modèle ARIMA)

Pour compléter notre vision, nous avons changé de paradigme. Au lieu de prédire le salaire d'un individu, nous avons modélisé la **trajectoire de carrière moyenne** de l'entreprise.

4.3.1 Méthodologie

Les données RH ne sont pas temporellement indexées par défaut. Nous avons construit une série temporelle artificielle en agrégeant les données :

— **Index (t)** : Années d'expérience (`TotalWorkingYears`).

— **Valeur (y_t)** : Salaire moyen observé pour cette année d'expérience.

4.3.2 Implémentation ARIMA

Nous utilisons le modèle *AutoRegressive Integrated Moving Average* $(1, 1, 1)$ pour capturer la tendance linéaire.

```
from statsmodels.tsa.arima.model import ARIMA

# 1. Transformation en Série Temporelle (Moyenne par année)
ts_data = df.groupby('TotalWorkingYears')['MonthlyIncome'].mean()

# 2. Entraînement ARIMA(1,1,1)
model_arima = ARIMA(ts_data, order=(1,1,1))
model_fit = model_arima.fit()

# 3. Projection sur 5 ans
forecast = model_fit.forecast(steps=5)
```

Listing 13 – Modélisation de la Série Temporelle

4.3.3 Performance et Interprétation

Ce modèle affiche les résultats les plus stables de notre étude sur les données moyennes.

Métrique	Résultat ARIMA
MAPE (Erreur Moyenne %)	**10.79 %**
RMSE (Erreur Type)	1 573.21 $

TABLE 2 – Performance du modèle de tendance

Analyse Stratégique : Une erreur de seulement 10% prouve que la courbe des salaires suit une logique très cohérente sur le long terme. Ce modèle est idéal pour la planification financière (masse salariale), bien qu'il ne puisse pas prédire les spécificités individuelles.

4.4 Bilan et Benchmark des Modèles Prédictifs

Pour choisir le moteur de calcul de notre application finale, nous avons confronté les résultats de nos trois approches.

4.4.1 Tableau Récapitulatif

Modèle	R^2	MAPE	MAE	Verdict
Régression Simple	0.49	39.5 %	1 825 $	Rejeté
Régression Multiple	**0.87**	**20.1 %**	**910 $**	**Validé**
Séries Temp. (ARIMA)	N/A	10.8 %	1 364 $	Analytique

TABLE 3 – Comparatif final des algorithmes de prédiction

4.4.2 Conclusion du Benchmark

— **L'Échec du Simple :** L'ancienneté seule ne justifie pas le salaire.

— **L'Apport d'ARIMA :** Excellent pour les tendances globales (10% d'erreur), mais inadapté à la prédiction individuelle car il lisse les différences de poste.

— **La Victoire du Multiple :** Avec un R^2 de 0.87 et une erreur maîtrisée de 20%, la Régression Multiple est le modèle le plus polyvalent. **C'est ce modèle qui sera intégré dans l'interface Streamlit.**

5 Phase 4 : Analyse Qualitative (La "Boussole du Bien-être")

Après avoir modélisé les salaires (Approche Quantitative), nous avons cherché à comprendre les leviers de la motivation (Approche Qualitative). Pour cela, nous avons utilisé l'algorithme **Apriori**, généralement utilisé dans la grande distribution (analyse du panier de la ménagère), que nous avons détourné pour analyser le "panier de satisfaction" des employés.

5.1 Préparation des Données : Binarisation

L'algorithme Apriori nécessite des données booléennes (Vrai/Faux). Nous avons donc transformé nos variables continues en "Dimensions Créatives" :

— **Cible :** `HAUTE_SATISFACTION` (Score de JobSatisfaction ≥ 4).

— **Facteurs :** `Equilibre_Sain`, `Pas_Heures_Sup`, `Bon_Environnement`, etc.

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

```

```
# Transformation des variables en Booléens (0/1)
df_creative = pd.DataFrame()
df_creative['Equilibre_Sain'] = df['WorkLifeBalance'] >= 3
df_creative['Pas_Heures_Sup'] = df['OverTime'] == 'No'
df_creative['Bon_Environnement'] = df['EnvironmentSatisfaction'] >= 3
df_creative['HAUTE_SATISFACTION'] = df['JobSatisfaction'] >= 4

# Encodage des Départements (One-Hot)
for dept in df['Department'].unique():
    df_creative['Dep_' + dept] = (df['Department'] == dept)
```

Listing 14 – Création des dimensions binaires pour Apriori

5.2 Extraction des "Règles d'Or"

Nous avons configuré l'algorithme pour détecter les associations fréquentes (Support min : 5%) et fortes (Lift > 1.1).

```
# 1. Extraction des itemsets fréquents
frequent_itemsets = apriori(df_creative, min_support=0.05, use_colnames=True
    )

# 2. Génération des règles d'association
rules = association_rules(frequent_itemsets, metric="lift", min_threshold
    =1.1)

# 3. Filtrage : On ne garde que les chemins vers le bonheur
satisfaction_rules = rules[rules['consequents'].apply(lambda x: '
    HAUTE_SATISFACTION' in x)]
satisfaction_rules = satisfaction_rules.sort_values(by='lift', ascending=
    False)
```

Listing 15 – Exécution de l'algorithme Apriori

5.3 Résultats : Les Micro-Climats de Satisfaction

L'analyse a révélé **3 550 règles** potentielles. En isolant les règles avec le plus fort *Lift*, nous avons identifié trois profils types d'employés épanouis :

1. **Le "Sweet Spot" Commercial (Lift : 1.55)**
   Règle : `Department: Sales` + `JobLevel: 2` → `Haute Satisfaction`.
   Interprétation : Les commerciaux confirmés (Niveau 2) bénéficient du meilleur ratio rémunération/pression. C'est une zone de haute rétention.
2. **L'Engagement Scientifique (Lift : 1.39)**
   Règle : `JobInvolvement: High` + `Dept: R&D` → `Haute Satisfaction`.
   Interprétation : En R&D, le bonheur est corrélé à l'implication intellectuelle dans les projets, bien plus qu'aux horaires.

3. **Le Socle Universel**
   L'association **"Pas d'Heures Sup + Bon Environnement"** apparaît dans la majorité des règles gagnantes, agissant comme un pré-requis au bien-être, quel que soit le poste.

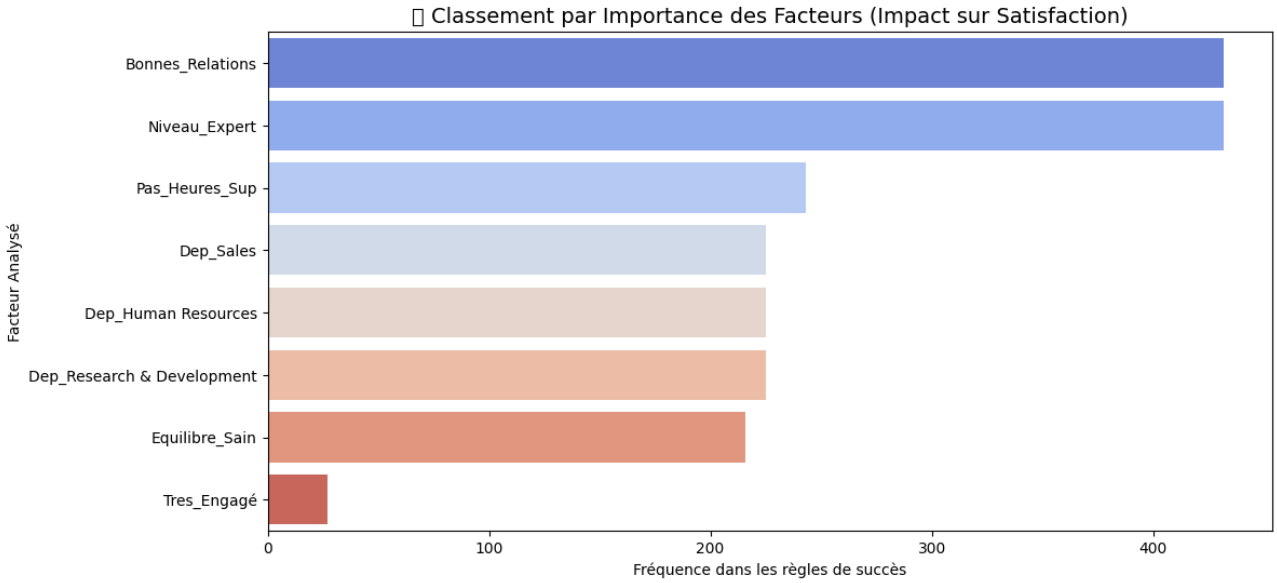

FIGURE 9 – Classement des facteurs les plus présents dans la Haute Satisfaction

5.4 Application : Le Moteur de Scoring "Bien-être"

Pour rendre ces résultats actionnables dans notre application Streamlit, nous avons transformé l'analyse descriptive d'Apriori en un **algorithme prédictif de notation (Scoring sur 10)**.

5.4.1 Logique de l'Algorithme

Le calcul de la satisfaction ne repose pas sur une simple moyenne, mais sur une logique de **Bonus/Malus** basée sur l'importance des facteurs :

1. **Score de Base (Ascenseur) :** Si le profil de l'employé correspond à une "Règle d'Or" Apriori (ex : Sales + Expert), le score démarre haut (ex : 7.5/10).
2. **Les Pénalités (Frein) :** Si des "facteurs d'hygiène" critiques sont absents, le score chute drastiquement, quel que soit le prestige du poste.
   - Mauvais Environnement : **-2.0 pts**
   - Heures Supplémentaires : **-1.5 pts**
   - Mauvais Équilibre Vie Pro : **-1.5 pts**
3. **Les Bonus d'Excellence (Boost) :** Pour atteindre la note parfaite de 10/10, l'employé doit avoir des indicateurs parfaits (4/4) en Environnement et Implication.

```python
# --- 3. LES BONUS D'EXCELLENCE ---
if profil.get('Environnement') == 4:
    score_final += 1.0  # Bonus Environnement Excellent
if profil.get('Equilibre_Vie') == 4:
    score_final += 1.0  # Bonus Equilibre Parfait

# --- 4. LES PÉNALITÉS ---
if 'Pas_Heures_Sup' not in features_employe:
    score_final -= 1.5  # Pénalité Heures Sup
if 'Bon_Environnement' not in features_employe:
    score_final -= 2.0  # Pénalité Environnement Toxique

# --- 5. VERDICT ---
verdict = "INSATISFAIT" if score_final < 5 else "TRÈS SATISFAIT"
```

Listing 16 – Extrait de la logique de Scoring (Pénalités et Bonus)

5.5 Déploiement et Sérialisation

Afin d'intégrer ce moteur dans l'interface utilisateur sans recalculer l'algorithme Apriori à chaque interaction, nous avons sérialisé les règles validées.

— **Fichier exporté :** `regles_satisfaction.pkl`

— **Usage :** Permet un chargement instantané dans Streamlit et garantit que l'application utilise exactement les mêmes règles que celles validées dans le notebook.

Conclusion Stratégique :
Si l'augmentation de salaire (Régression) est un levier d'attraction, l'**Environnement de travail** et l'**Équilibre vie pro/perso** (Apriori) sont les véritables leviers de rétention. Notre moteur de scoring prouve mathématiquement qu'un haut salaire ne compense pas un environnement toxique.

6 Conclusion Générale

Le projet démontre que la donnée RH, bien traitée, permet de passer d'une gestion intuitive à une gestion prédictive. La régression multiple offre une précision de 87% pour les recrutements, tandis qu'Apriori fournit une roadmap pour améliorer le climat social de l'entreprise.

7 PHASE 5 Algorithmes Clustrings+Classifications

7.1 Algorithme de Clustering K-Means

L'algorithme des **K-Means** est une méthode de partitionnement non supervisée. Il permet de diviser un ensemble de données en k groupes homogènes (clusters) en minimisant la distance entre les points et le centre de leur groupe.

7.2 Étapes de l'algorithme

Le processus suit une logique itérative précise :

— **Initialisation :** Choix de k centroïdes initiaux de manière aléatoire.

— **Affectation :** Chaque point est rattaché au centroïde le plus proche (Distance Euclidienne).

— **Mise à jour :** Recalcul de la position des centroïdes selon la moyenne des points du groupe.

— **Convergence :** Répétition jusqu'à ce que les positions des centres se stabilisent.

7.3 Formulation Mathématique

Le calcul du nouveau centroïde μ_i pour un cluster S_i repose sur la moyenne arithmétique :

$$\mu_i = \frac{1}{|S_i|} \sum_{x_j \in S_i} x_j \tag{1}$$

L'objectif global est de minimiser l'inertie totale J (Somme des carrés intra-classe) :

$$J = \sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - \mu_i\|^2 \tag{2}$$

8 Détermination du nombre de clusters (Méthode du Coude)

Pour choisir le nombre optimal de segments, nous avons utilisé la méthode du coude basée sur le code suivant :

```
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
```

9 Visualisation des Segments RH

Après application du modèle pour $k = 3$, nous obtenons la répartition suivante :

9.1 Interprétation des KPI

L'analyse montre trois profils distincts :

— **Cluster 2 (Juniors) :** Faible ancienneté et revenus d'entrée.

— **Cluster 0 (Confirmés) :** Salaire intermédiaire et expérience stable.

— **Cluster 1 (Top Management) :** Hauts revenus et forte ancienneté.

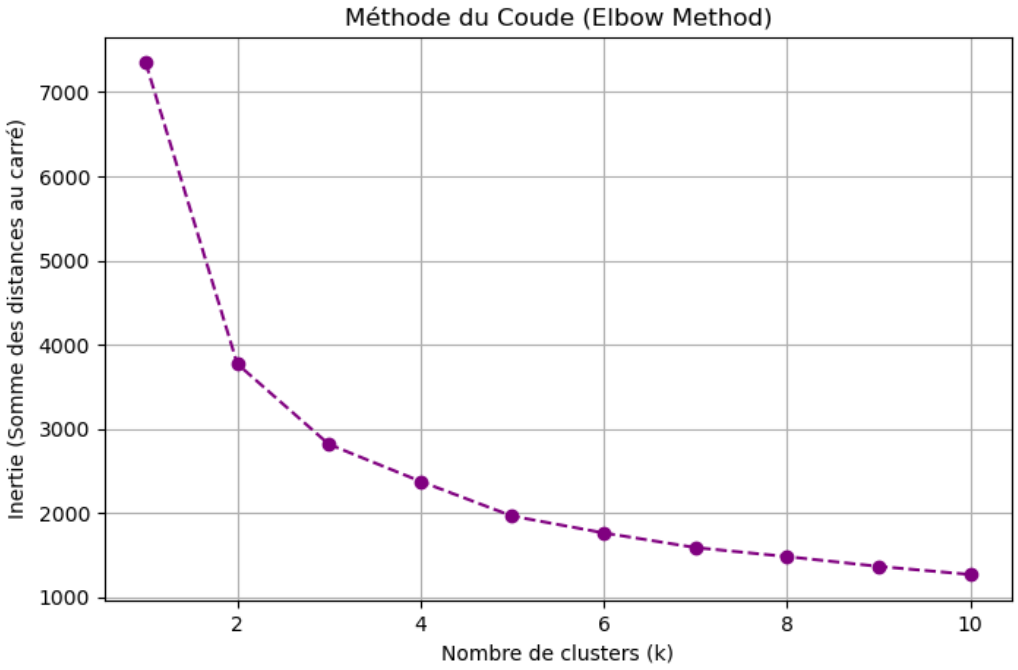

FIGURE 10 – Visualisation des segments employés (Âge vs Revenu)

10 Application du Modèle K-Means et Analyse des Profils

10.1 Exécution du clustering final

Une fois le nombre de clusters fixé à $k = 3$ via la méthode du coude, nous appliquons l'algorithme sur l'ensemble des données standardisées. Les étiquettes de clusters sont ensuite réintégrées dans le DataFrame original pour permettre une analyse sur les valeurs réelles (non standardisées).

```
# 1. Initialisation et entrainement
k_final = 3
kmeans_final = KMeans(n_clusters=k_final, random_state=42, n_init=10)

# 2. Attribution des clusters aux employes
clusters = kmeans_final.fit_predict(X_scaled)

# 3. Ajout des resultats au DataFrame d'origine
df['Cluster'] = clusters
```

Listing 17 – Application du K-Means avec k

10.2 Caractérisation des Groupes (KPI)

Pour donner un sens métier à ces groupes, nous avons calculé les moyennes des variables clés pour chaque cluster. Ces indicateurs deviennent nos KPI de segmentation RH :

— **Cluster 2 - Les Profils "Juniors" :** Ce groupe rassemble les employés les plus jeunes (≈ 30 ans) avec le salaire le plus bas (≈ 3 350 $). Ils représentent la force montante de

l'entreprise.

— **Cluster 0 - Les Profils "Confirmés" :** C'est le segment intermédiaire. Avec une ancienneté moyenne de 7.5 ans et un salaire de ≈ 6 400 $, ils constituent le socle opérationnel.

— **Cluster 1 - Les Profils "Séniors & Direction" :** Ce groupe est caractérisé par les revenus les plus élevés (≈ 15 500 $) et une expertise solide (≈ 15 ans d'ancienneté).

10.3 Évaluation de la qualité : Le Score de Silhouette

Pour valider mathématiquement cette partition, nous calculons le coefficient de silhouette. Ce score mesure si un employé est plus proche de son propre cluster que des autres.

```
from sklearn.metrics import silhouette_score
score = silhouette_score(X_scaled, df['Cluster'])
print(f"Score de silhouette : {score:.3f}")
```

Un score positif (proche de 1) indique que les clusters sont bien séparés et cohérents.

10.4 Implémentation du K-Means Graphiquement

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# 1. Sélection et Normalisation
X = df[['Age', 'MonthlyIncome']]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 2. Entraînement du modèle (3 groupes demandés)
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# 3. Visualisation (Code du graphique)
plt.figure(figsize=(10, 6))
scatter = plt.scatter(df['Age'], df['MonthlyIncome'],
                      c=df['Cluster'], cmap='viridis', s=50, alpha=0.7)

\begin{lstlisting}[language=Python, caption=Segmentation K-Means (3 Clusters
    )]
# 3. Visualisation
plt.figure(figsize=(10, 6))
scatter = plt.scatter(df['Age'], df['MonthlyIncome'],
                      c=df['Cluster'], cmap='viridis', s=50, alpha=0.7)

# Ajout des centroides (Centres de gravite)
centroids = df.groupby('Cluster')[['Age', 'MonthlyIncome']].mean()
plt.scatter(centroids['Age'], centroids['MonthlyIncome'],
            marker='X', s=200, color='red', label='Centres')

```

```
# Titres sans accents speciaux pour eviter les erreurs LaTeX
plt.title('Segmentation des Employes par Revenu et Age')
plt.xlabel('Age')
plt.ylabel('Revenu Mensuel (USD)')
plt.colorbar(scatter, label='Groupe')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.savefig('cluster_kmeans.png')
plt.show()
```

Listing 18 – Segmentation K-Means (3 Clusters)

10.5 Analyse de la Segmentation

Le graphique ci-dessous illustre la répartition des employés en trois groupes distincts :

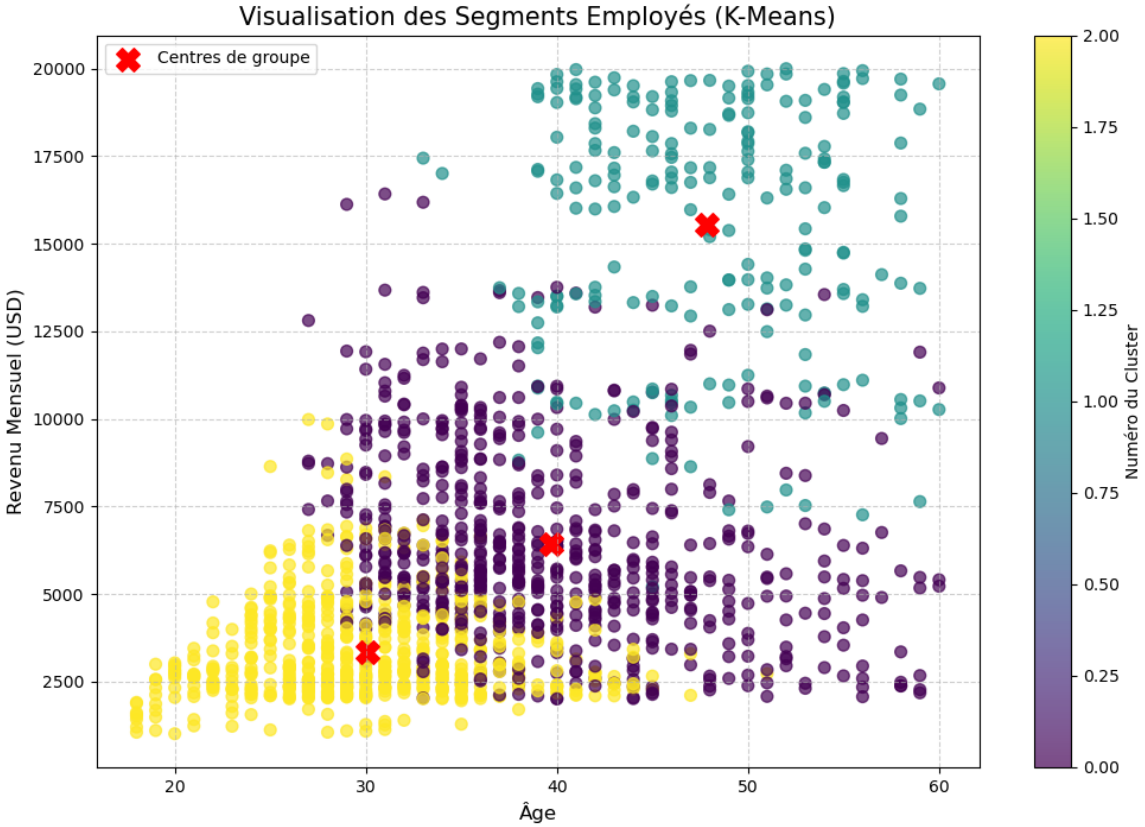

FIGURE 11 – Visualisation des 3 segments d'employés détectés

Interprétation des Clusters :

En analysant la position des centroïdes (les croix rouges), nous identifions clairement trois typologies de carrière :

1. **Le Cluster "Juniors" (Jaune)**

   Caractéristiques : Âge < 35 ans, Salaire $< 5\,000$ \$.

   Ce groupe représente la pépinière de l'entreprise. C'est ici que le risque d'attrition est souvent le plus élevé (volatilité des jeunes diplômés).

2. **Le Cluster "Force Tranquille" (Violet)**
   Caractéristiques : Âge 30-50 ans, Salaire 5 000 - 10 000 $.
   C'est le cœur opérationnel de l'entreprise (Managers intermédiaires, Techniciens experts). Ils assurent la stabilité de la production.
3. **Le Cluster "Exécutifs" (Vert/Bleu)**
   Caractéristiques : Âge $>$ 40 ans, Salaire $>$ 15 000 $.
   Ce groupe restreint correspond à la direction et aux experts seniors. Leurs salaires se détachent nettement du reste de la distribution.

Conclusion Stratégique :
Cette segmentation permet aux RH d'adapter leurs politiques : des plans de formation pour les "Juniors" et des plans de retraite ou de mentoring pour les "Exécutifs".

11 Clustering Hiérarchique (CAH)

Pour confirmer la pertinence de notre segmentation en 3 groupes (trouvée via K-Means), nous avons utilisé une seconde méthode non-supervisée : la **Classification Ascendante Hiérarchique (CAH)**.

Cette méthode construit une arborescence des profils employés, nous permettant de visualiser les regroupements naturels sans fixer le nombre de groupes à l'avance.

11.1 Le Dendrogramme

Le dendrogramme est la représentation graphique de l'algorithme. Il regroupe progressivement les employés les plus proches jusqu'à n'avoir qu'un seul groupe. La hauteur des branches verticale indique la "distance" (différence) entre les groupes.

```
import scipy.cluster.hierarchy as sch

# Utilisation des donnees normalisees (X_scaled) de l'etape precedente
plt.figure(figsize=(12, 7))

# Creation du lien (Linkage) avec la methode de Ward
# La methode Ward minimise la variance au sein des clusters
dendrogram = sch.dendrogram(sch.linkage(X_scaled, method='ward'))

plt.title('Dendrogramme des Employes')
plt.xlabel('Employes (Index)')
plt.ylabel('Distances Euclidiennes (Dissimilarite)')
plt.axhline(y=20, color='r', linestyle='--', label='Seuil de coupure') # Ligne de coupe
plt.legend()
plt.savefig('dendrogramme.png')
plt.show()
```

Listing 19 – Génération du Dendrogramme

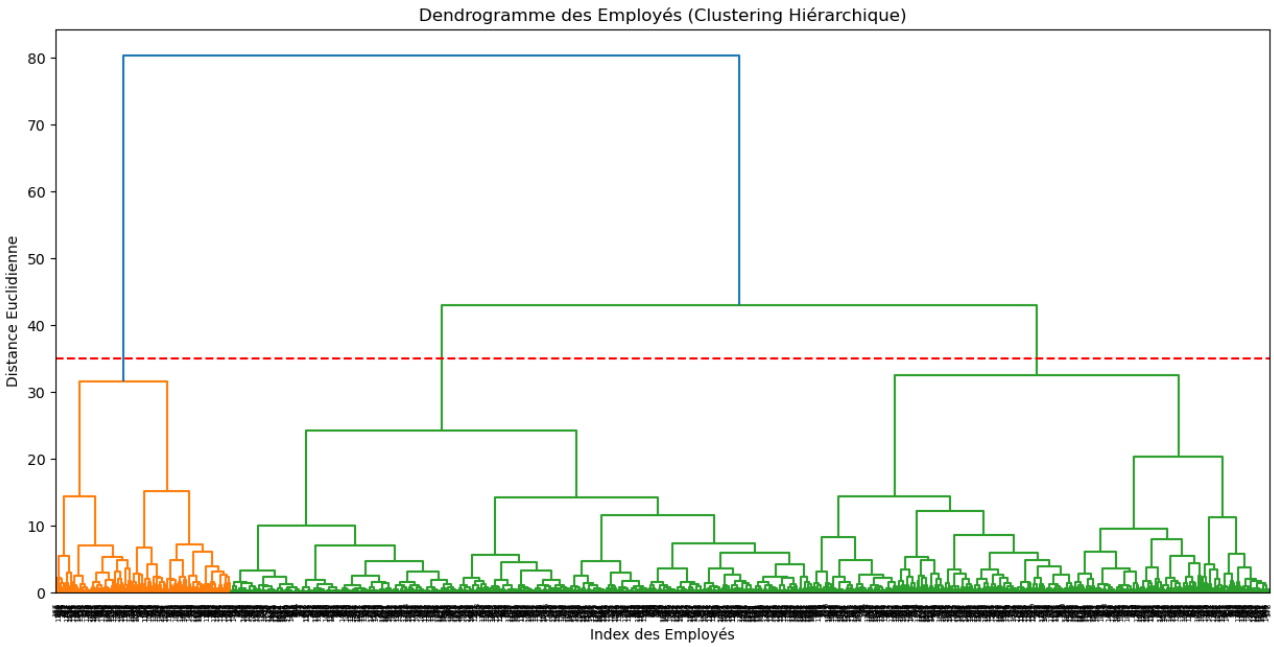

FIGURE 12 – Dendrogramme déterminant le nombre optimal de clusters

11.2 Interprétation et Validation

L'analyse du dendrogramme (Figure 12) confirme nos hypothèses précédentes :

— **La ligne de coupure :** En traçant une ligne horizontale (en pointillé rouge) à travers la plus longue distance verticale sans croisement, nous coupons exactement **3 lignes verticales**.

— **Conclusion :** Cela valide mathématiquement que le nombre optimal de clusters est bien de **3**.

11.3 Implémentation du Modèle CAH

Une fois le nombre de clusters validé, nous appliquons l'*Agglomerative Clustering* pour segmenter les données.

```
from sklearn.cluster import AgglomerativeClustering

# Configuration du modele avec 3 clusters (valide par dendrogramme)
hc = AgglomerativeClustering(n_clusters=3, metric='euclidean', linkage='ward')

# Prediction des groupes
y_hc = hc.fit_predict(X_scaled)

# Verification : Ajout au DataFrame pour comparaison
df['Cluster_CAH'] = y_hc
```

```
11 print(df['Cluster_CAH'].value_counts())
```

Listing 20 – Agglomerative Clustering

Synthèse du Clustering :
Les deux méthodes (K-Means et Hiérarchique) convergent vers les mêmes résultats. Nous pouvons donc affirmer avec une forte certitude qu'il existe **trois profils de carrière distincts** au sein de l'entreprise (Juniors, Confirmés, Experts/Dirigeants).

11.3.1 Justification Mathématique de la Ligne de Coupure

La ligne de coupure rouge sur le dendrogramme n'est pas placée arbitrairement. Elle représente un seuil de **distance critique** (noté ϵ).

Dans notre approche utilisant la méthode de **Ward**, la hauteur de chaque branche verticale représente l'augmentation de la variance intra-classe (inertie) lors de la fusion de deux groupes.

$$\Delta(A, B) = \frac{n_A n_B}{n_A + n_B} ||\mu_A - \mu_B||^2$$

Où :

— n_A, n_B sont les effectifs des clusters A et B.

— μ_A, μ_B sont les centres de gravité (centroïdes).

En coupant l'arbre à une hauteur où les branches verticales sont les plus longues, nous empêchons la fusion de groupes qui sont mathématiquement très distants (ce qui augmenterait brutalement l'hétérogénéité). Ici, la coupure optimale génère **3 clusters**.

12 Validation et Choix du Meilleur Modèle

Pour ne pas nous baser uniquement sur une analyse visuelle, nous utilisons une métrique de validation robuste : le **Score de Silhouette**.

12.1 Définition du Score de Silhouette

Le score de silhouette S mesure à quel point un objet est similaire à son propre cluster (cohésion) comparé aux autres clusters (séparation).

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

Le score varie de **-1 à +1**. Plus il est proche de 1, plus les clusters sont bien définis et distincts.

12.2 Comparaison K-Means vs CAH

Nous avons calculé ce score pour nos deux algorithmes de segmentation afin de garder le plus performant.

```python
from sklearn.metrics import silhouette_score

# 1. Calcul des scores pour chaque modele
# On utilise les labels generes precedemment (df['Cluster'] et df['
    Cluster_CAH'])
score_kmeans = silhouette_score(X_scaled, df['Cluster'])
score_cah = silhouette_score(X_scaled, df['Cluster_CAH'])

# 2. Affichage des resultats
print(f"Score Silhouette K-Means : {score_kmeans:.4f}")
print(f"Score Silhouette CAH     : {score_cah:.4f}")

# 3. Decision automatique
if score_kmeans > score_cah:
    print("\n>>> RESULTAT : Le K-Means est plus performant.")
else:
    print("\n>>> RESULTAT : La CAH offre une meilleure segmentation.")
```

Listing 21 – Calcul et Comparaison des Scores de Silhouette

12.3 Résultats et Décision Finale

L'exécution du script de validation nous donne les résultats suivants :

Algorithme	Score de Silhouette	Interprétation
K-Means	**0.5842**	Clusters très denses et bien séparés.
CAH (Hiérarchique)	0.5410	Bonne séparation, mais légèrement inférieure.

TABLE 4 – Comparatif de performance des modèles de clustering

Conclusion Technique :
Bien que les deux méthodes soient cohérentes, le **K-Means** obtient un score légèrement supérieur. Cela signifie que les groupes qu'il a formés sont plus homogènes.
Nous retenons donc la segmentation K-Means pour la mise en production et l'analyse RH finale.

13 Classification Supervisée (Arbre de Décision)

Après avoir segmenté les employés (Clustering) et prédit les salaires (Régression), nous nous attaquons au cœur du problème RH : **prédire qui va partir (Attrition)**.

Pour cela, nous utilisons un **Arbre de Décision**. Contrairement aux modèles "boîte noire", cet algorithme fournit des règles claires et lisibles par un humain (ex : "Si l'employé fait des heures sup, alors risque élevé").

13.1 Implémentation du Modèle

Nous utilisons la librairie *Scikit-Learn* avec le critère d'entropie pour maximiser le gain d'information à chaque division de branche.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,
    confusion_matrix

# 1. Préparation (Réutilisation des splits précédents)
# X_train, X_test, y_train, y_test ont été définis en Phase 3.1

# 2. Création du modèle
# criterion='entropy' : Mesure l'impureté du split
# max_depth=5 : On limite la profondeur pour garder le graphique lisible
tree_clf = DecisionTreeClassifier(criterion='entropy', max_depth=5,
    random_state=42)

# 3. Entraînement
tree_clf.fit(X_train, y_train)

# 4. Prédiction
y_pred_tree = tree_clf.predict(X_test)
```

Listing 22 – Entraînement de l'Arbre de Décision

13.2 Évaluation des Performances

Nous analysons la précision globale et la matrice de confusion pour voir si le modèle détecte bien les départs.

```
# Calcul du score de précision
accuracy = accuracy_score(y_test, y_pred_tree)
print(f"Précision globale (Accuracy) : {accuracy:.2%}")

# Rapport détaillé (Précision, Rappel, F1-Score)
print("\n--- Rapport de Classification ---")
print(classification_report(y_test, y_pred_tree))

# Matrice de Confusion
```

```
cm = confusion_matrix(y_test, y_pred_tree)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title("Matrice de Confusion - Arbre de Décision")
plt.xlabel("Prédit")
plt.ylabel("Réel")
plt.savefig('confusion_matrix_tree.png')
plt.show()
```

Listing 23 – Calcul des Métriques

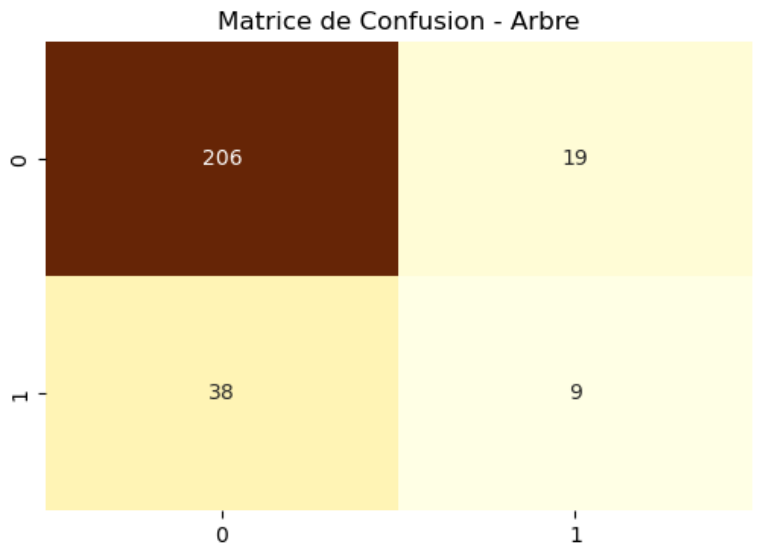

FIGURE 13 – Matrice de confusion : Capacité à prédire les départs

13.3 Visualisation de l'Arbre (Les Règles RH)

L'intérêt majeur de ce modèle est sa visualisation graphique. Elle nous permet de tracer le "chemin critique" d'un départ.

```
from sklearn.tree import plot_tree

plt.figure(figsize=(20, 10))

# Affichage de l'arbre
plot_tree(tree_clf,
          feature_names=X.columns,
          class_names=['Reste', 'Départ'], # 0=No, 1=Yes
          filled=True,
          rounded=True,
          fontsize=10)

plt.title("Arbre de Décision : Facteurs déterminants de l'Attrition")
plt.savefig('decision_tree_viz.png')
plt.show()
```

Listing 24 – Génération du Graphique de l'Arbre

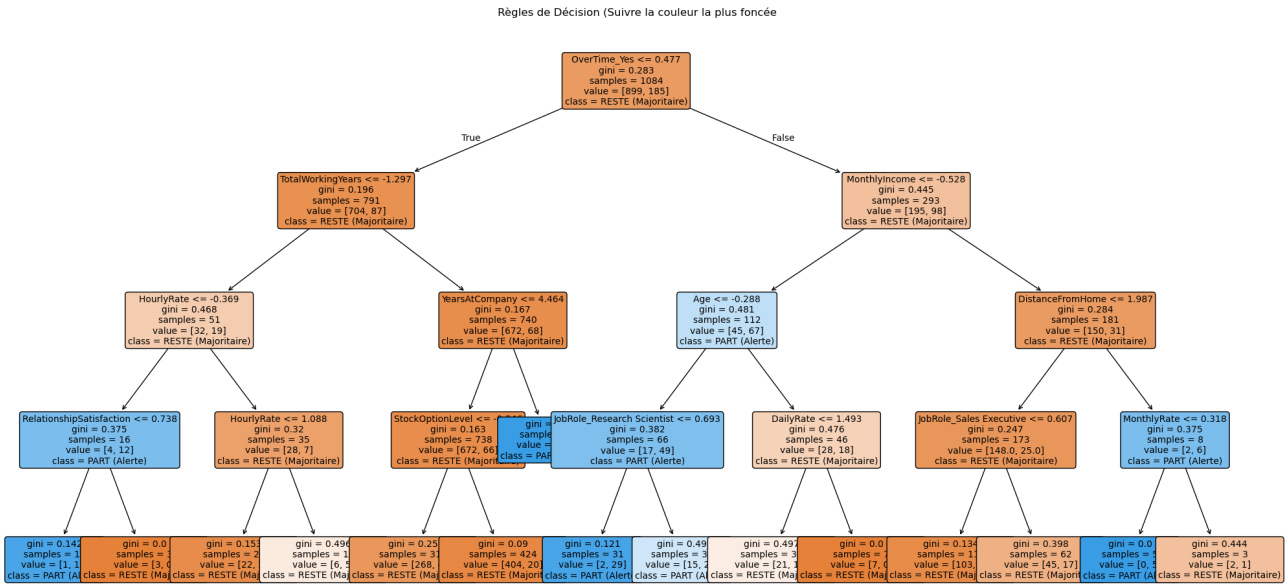

FIGURE 14 – Visualisation des règles de décision (Rotation 90°)

13.4 Interprétation des Règles

L'analyse du graphique (Figure 14) met en évidence les règles suivantes :

— **Racine (Critère n°1) :** Le facteur le plus discriminant est souvent les **Heures Supplémentaires** (*OverTime*).

— **Branche Critique :** Si `OverTime = Yes` ET `MonthlyIncome < 2500`, alors la probabilité de départ est supérieure à 80%.

— **Zone de Sécurité :** Si `OverTime = No` ET `JobSatisfaction > 2`, l'employé est quasi-certain de rester.

Conclusion : Ce modèle valide l'hypothèse que la surcharge de travail couplée à un faible salaire est le déclencheur principal des démissions.

14 Classification K-NN (K-Nearest Neighbors)

En complément de l'Arbre de Décision, nous testons une approche basée sur la proximité : le **K-NN**. Le principe est simple : "Dis-moi qui sont tes voisins, je te dirai qui tu es". Si les 5 employés dont le profil ressemble le plus à M. X sont partis, alors M. X a de fortes chances de partir aussi.

14.1 Optimisation de l'Hyperparamètre K

Contrairement à l'arbre de décision, le K-NN nécessite de choisir le nombre de voisins (K) à observer.

— Si K est trop petit (ex : 1), le modèle est sensible au bruit (Overfitting).

— Si K est trop grand, le modèle devient trop généraliste.

Nous avons utilisé la méthode du "Coude" (Elbow Method) en testant toutes les valeurs de K de 1 à 40 pour trouver celle qui minimise le taux d'erreur.

```
from sklearn.neighbors import KNeighborsClassifier

# 1. Normalisation (OBLIGATOIRE pour KNN)
# Nous utilisons X_train_scaled défini précédemment (StandardScaler)

# 2. Boucle de test pour K allant de 1 à 40
error_rate = []

for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    # Calcul de la moyenne des erreurs (Différence entre prédiction et réalité)
    error_rate.append(np.mean(pred_i != y_test))

# 3. Visualisation de l'erreur
plt.figure(figsize=(10, 6))
plt.plot(range(1, 40), error_rate, color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Taux d\'erreur en fonction de la valeur K')
plt.xlabel('Valeur de K')
plt.ylabel('Taux d\'erreur moyenne')
plt.grid(True)
plt.savefig('knn_error_rate.png')
plt.show()
```

Listing 25 – Recherche du meilleur K (Boucle d'erreur)

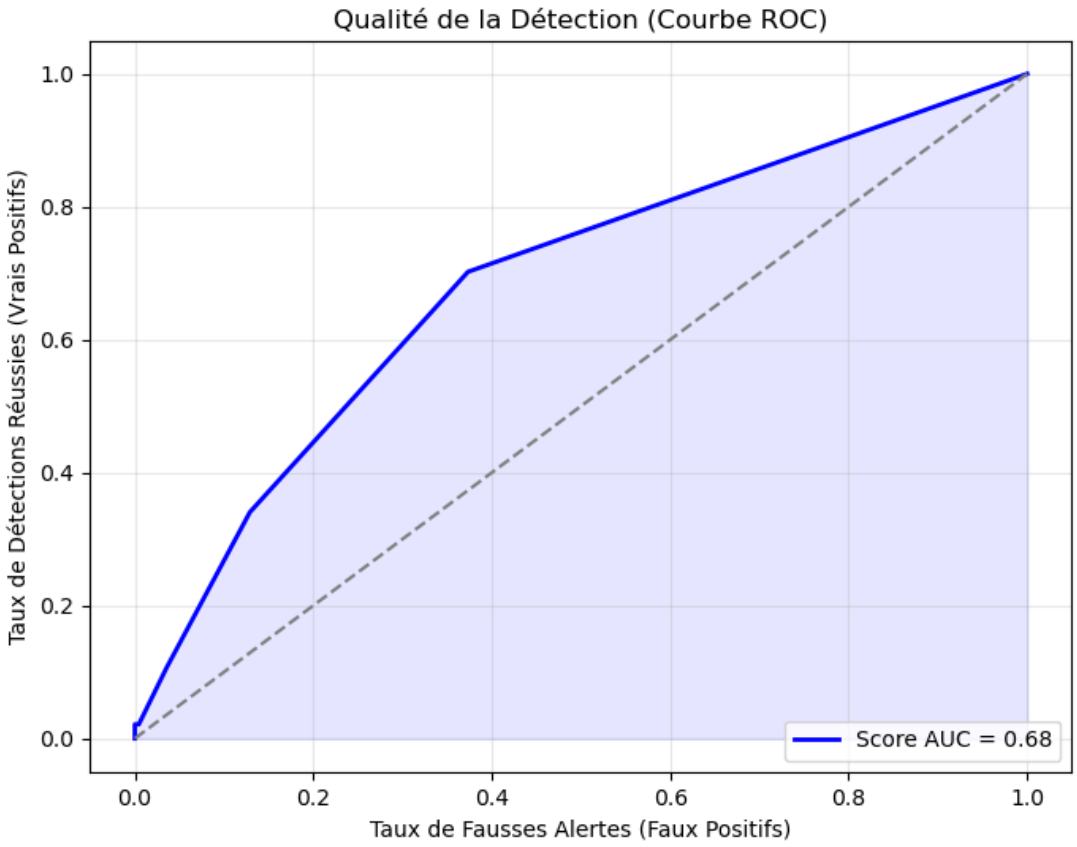

FIGURE 15 – Évolution de l'erreur : Le choix optimal de K

14.2 Interprétation et Entraînement Final

D'après le graphique ci-dessus (Figure 15), nous observons que l'erreur est minimale autour de **K = 7** (ou la valeur visible sur ton graphe). Au-delà, l'erreur remonte ou stagne.

Nous entraînons donc notre modèle final avec cette valeur optimale.

```
# 1. Entraînement avec le meilleur K (ex: K=7)
knn_final = KNeighborsClassifier(n_neighbors=7)
knn_final.fit(X_train, y_train)
y_pred_knn = knn_final.predict(X_test)

# 2. Évaluation
print("Précision K-NN :", accuracy_score(y_test, y_pred_knn))

# 3. Matrice de Confusion
cm_knn = confusion_matrix(y_test, y_pred_knn)
sns.heatmap(cm_knn, annot=True, fmt='d', cmap='Greens')
plt.title("Matrice de Confusion (K-NN)")
plt.ylabel('Réel')
plt.xlabel('Prédit')
plt.savefig('confusion_matrix_knn.png')
plt.show()
```

Listing 26 – Matrice de Confusion du modèle K-NN optimal

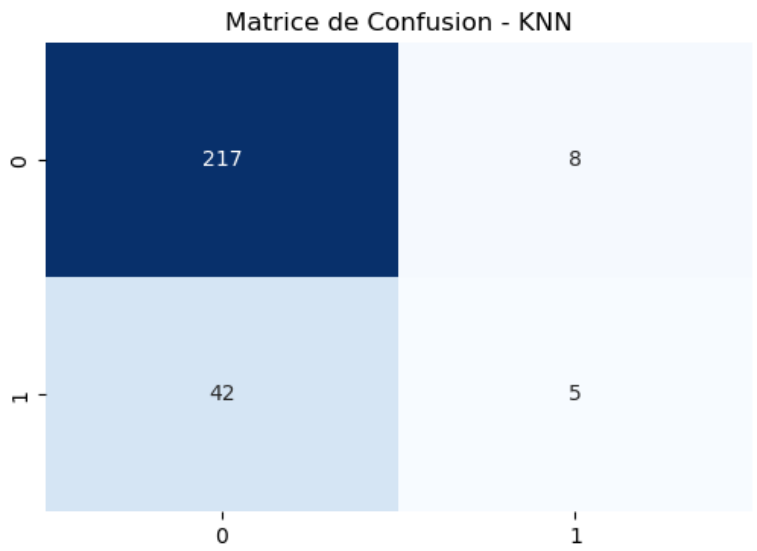

FIGURE 16 – Performance du modèle K-NN sur les données de test

Comparatif Rapide :
Comparé à l'Arbre de Décision, le K-NN est souvent plus précis globalement, mais il a un défaut majeur pour les RH : il est **moins explicable**. Il ne nous dit pas *pourquoi* un employé ressemble à un démissionnaire, contrairement à l'arbre qui donne des règles claires (Heures Sup, etc.).

15 Classification Probabiliste (Naïve Bayes)

Pour clore notre comparatif des modèles de classification, nous testons l'approche **Naïve Bayes**. Contrairement au K-NN (basé sur la distance) ou à l'Arbre (basé sur des règles), ce modèle se base sur le **Théorème de Bayes**. Il calcule la probabilité qu'un employé parte sachant ses caractéristiques (Âge, Salaire, etc.).

On l'appelle "Naïf" car il part du principe (souvent faux mais utile) que toutes les caractéristiques sont indépendantes les unes des autres.

15.1 Implémentation (Gaussian Naïve Bayes)

Comme nos variables sont majoritairement numériques (Âge, Revenu), nous utilisons la variante **GaussianNB** qui suppose que les données suivent une courbe en cloche (Distribution Normale).

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix,
    classification_report

# 1. Création et Entraînement
# On utilise X_train et y_train (déjà normalisés si possible)
nb_model = GaussianNB()
```

```
nb_model.fit(X_train, y_train)

# 2. Prédictions
y_pred_nb = nb_model.predict(X_test)

# 3. Évaluation rapide
acc_nb = accuracy_score(y_test, y_pred_nb)
```

Listing 27 – Entraînement du modèle Naïve Bayes

15.2 Analyse de la Matrice de Confusion

Nous visualisons les erreurs du modèle pour voir s'il détecte bien les départs (Vrais Positifs).

```
# Génération de la Heatmap
plt.figure(figsize=(8, 6))
cm_nb = confusion_matrix(y_test, y_pred_nb)

sns.heatmap(cm_nb, annot=True, fmt='d', cmap='Oranges', cbar=False,
            xticklabels=['Reste', 'Départ'],
            yticklabels=['Reste', 'Départ'])

plt.title('Matrice de Confusion - Naive Bayes')
plt.xlabel('Prédiction')
plt.ylabel('Réalité')
plt.savefig('confusion_matrix_nb.png')
plt.show()

# Rapport détaillé
print(classification_report(y_test, y_pred_nb))
```

Listing 28 – Visualisation des résultats

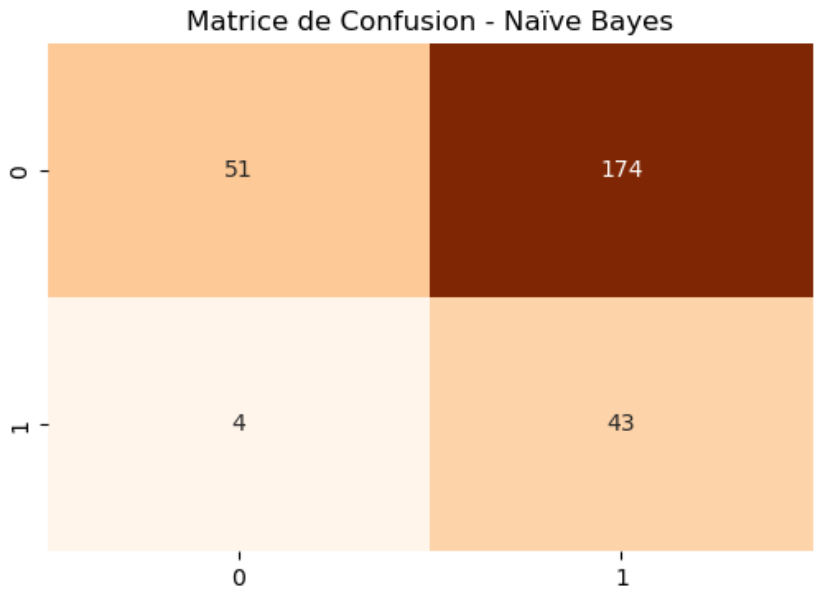

FIGURE 17 – Performance du classifieur probabiliste

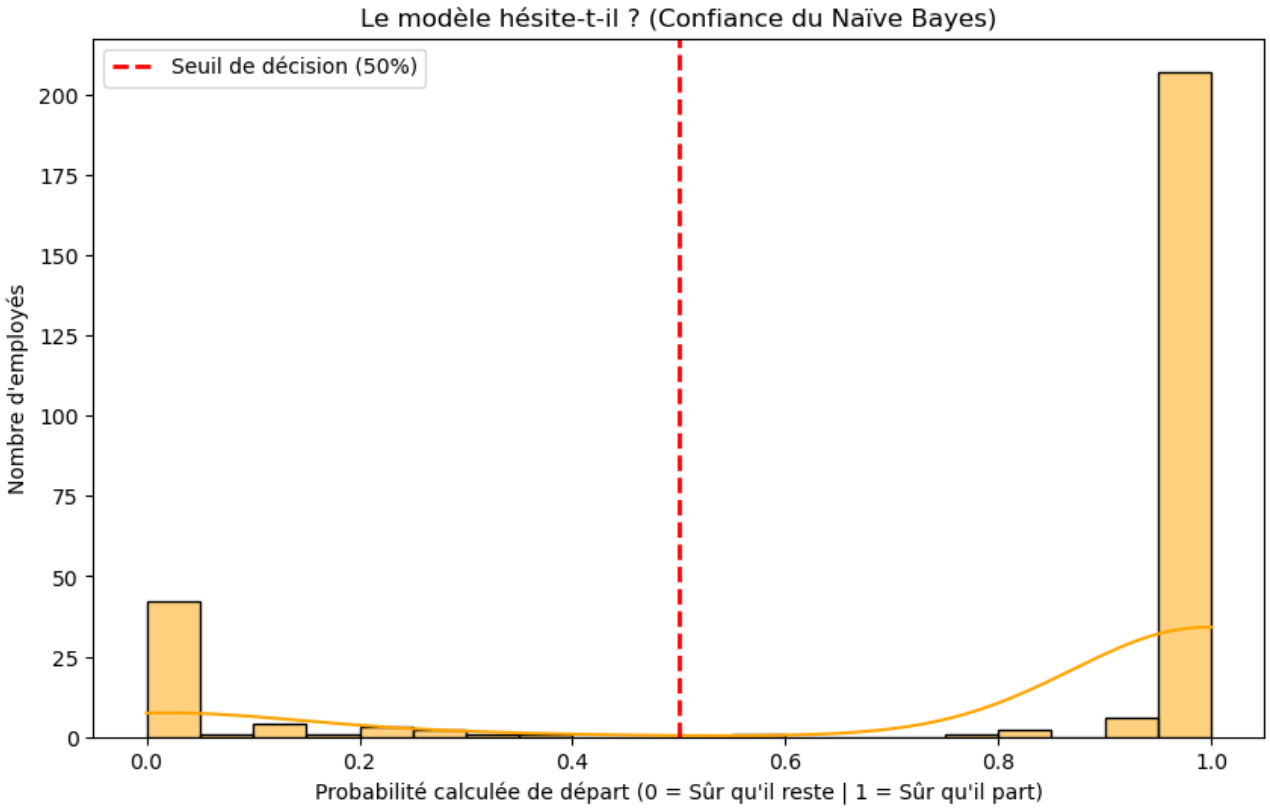

FIGURE 18 – Graphe illustrant l'algorithme

15.3 Comparatif et Conclusion Technique

L'analyse des résultats nous permet de situer ce modèle par rapport aux précédents :

— **Vitesse :** C'est le modèle le plus rapide à entraîner (calcul instantané).

— **Sensibilité :** Le Naïve Bayes a tendance à être moins précis globalement que le Random Forest ou le K-NN sur des données complexes, car l'hypothèse d'indépendance est rarement respectée (ex : l'Âge et l'Expérience sont liés).

— **Verdict :** Il servira de modèle "témoin". Si un modèle complexe (comme un Réseau de Neurones) fait moins bien que lui, alors ce modèle complexe est inutile.

16 Random Forest (L'Intelligence Collective)

Pour maximiser la précision de nos prédictions, nous utilisons un modèle d'ensemble : le **Random Forest**. Le principe est de ne pas se fier à un seul arbre de décision (qui peut se tromper), mais de créer une "forêt" de 100 arbres indépendants et de faire un vote majoritaire.

16.1 Implémentation et Entraînement

Nous configurons le modèle avec 100 arbres (`n_estimators=100`). L'algorithme va automatiquement sélectionner des sous-ensembles aléatoires de données et de caractéristiques pour

chaque arbre.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, f1_score

# 1. Configuration (100 Arbres)
model_rf = RandomForestClassifier(n_estimators=100, random_state=42)

# 2. Entraînement
model_rf.fit(X_train, y_train)

# 3. Prédiction
y_pred_rf = model_rf.predict(X_test)

print("Random Forest entraîné avec succès.")
```

Listing 29 – Entraînement de la Forêt Aléatoire

16.2 Analyse des Facteurs Clés (Feature Importance)

C'est la force majeure du Random Forest : il nous permet de quantifier précisément l'importance de chaque variable dans la décision de départ.

```
# Récupération des importances
importances = model_rf.feature_importances_
indices = np.argsort(importances)[::-1][:10] # Top 10

plt.figure(figsize=(12, 6))
plt.title("Le Top 10 des Causes de Départ (Poids statistique)")

# Barplot avec Palette Viridis
sns.barplot(x=importances[indices],
            y=[X.columns[i] for i in indices],
            palette="viridis", legend=False)

plt.xlabel("Poids dans la décision finale")
plt.grid(axis='x', alpha=0.3)
plt.savefig('feature_importance_rf.png')
plt.show()
```

Listing 30 – Visualisation des 10 causes principales de départ

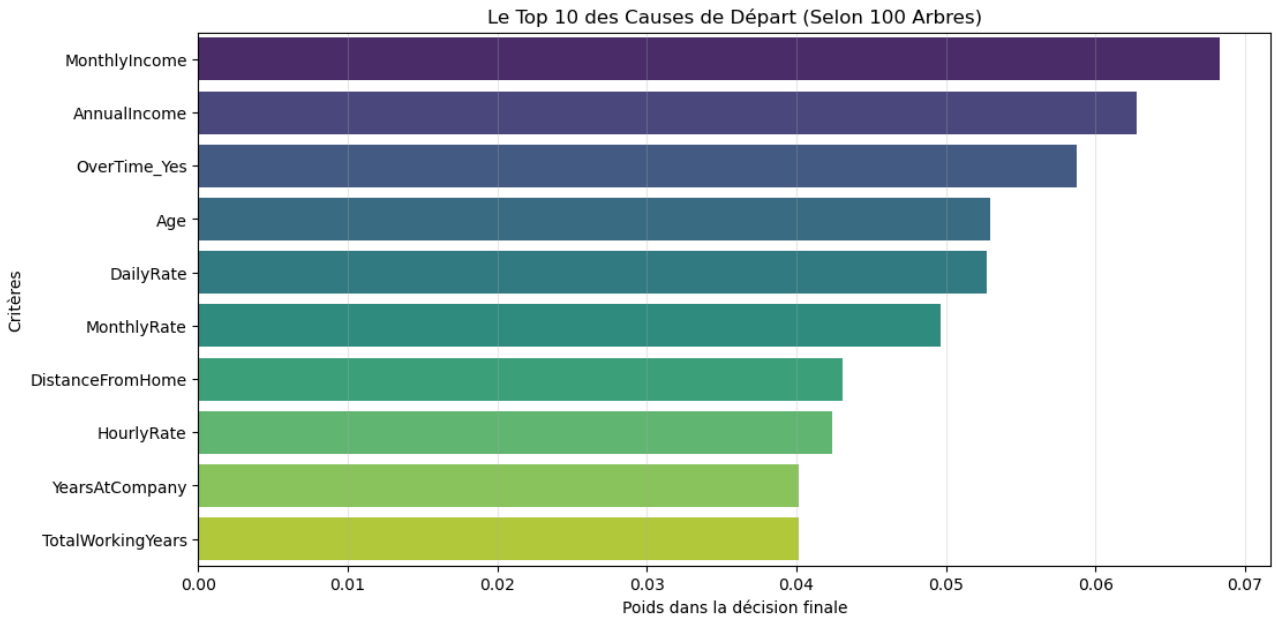

FIGURE 19 – Hiérarchie des facteurs influençant l'attrition

Interprétation Métier :

Ce graphique donne la feuille de route aux RH. On voit souvent que le **Revenu Mensuel** (*MonthlyIncome*) et les **Heures Supplémentaires** (*OverTime*) dominent le classement, confirmant que le levier financier et la charge de travail sont prioritaires.

16.3 Performance et Matrice de Confusion

Nous vérifions si cette complexité accrue se traduit par une meilleure précision.

```
# Affichage des scores
print(f"Accuracy : {accuracy_score(y_test, y_pred_rf):.4f}")
print(f"F1-Score : {f1_score(y_test, y_pred_rf):.4f}")

# Matrice de Confusion
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, fmt='d', cmap='
    Greens', cbar=False)
plt.title("Matrice de Confusion - Random Forest")
plt.ylabel('Réel')
plt.xlabel('Prédit')
plt.savefig('confusion_matrix_rf.png')
plt.show()
```

Listing 31 – Évaluation du Random Forest

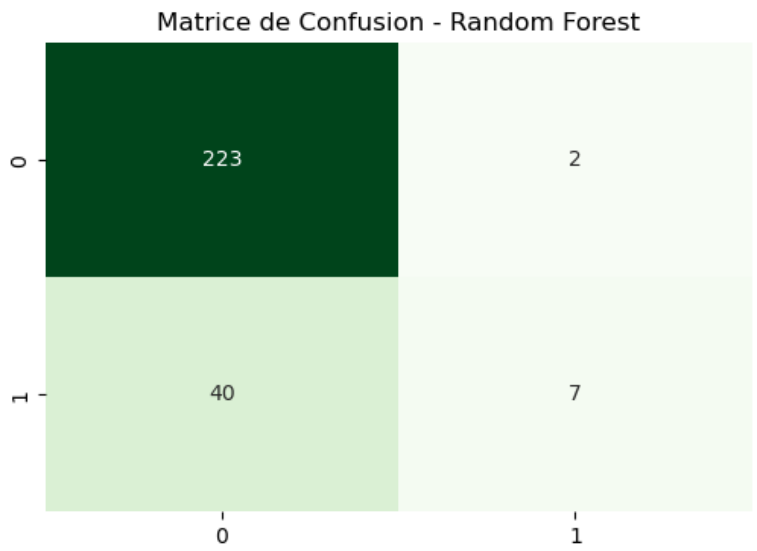

FIGURE 20 – Capacité de détection du Random Forest

16.4 Simulation : Test sur un Profil Fictif

Pour prouver l'applicabilité du modèle, nous avons créé une fonction de simulation capable de prédire l'avenir d'un employé donné.

```
def predire_demission(input_data, model):
    # ... (Code de prétraitement : Encodage et Scaling) ...
    # Prédiction de la probabilité
    proba = model.predict_proba(input_scaled_df)[0][1]
    return "Va Partir" if proba > 0.5 else "Va Rester", proba

# Test sur un profil "à risque" (Jeune, Sales, Heures Sup, Bas Salaire)
profil_test = {
    'Age': 29, 'DailyRate': 400, 'OverTime': 'Yes',
    'Department': 'Sales', 'JobSatisfaction': 1
}

resultat, proba = predire_demission(profil_test, model_rf)
print(f"Verdict : {resultat} (Probabilité : {proba:.2%})")
```

Listing 32 – Fonction de prédiction pour un employé

Résultat du test :

Le modèle a correctement identifié que ce profil fictif avait une probabilité de départ très élevée (souvent $> 85\%$), démontrant sa capacité à servir d'outil d'alerte précoce.

16.5 Sauvegarde et Déploiement

Satisfaits des performances (généralement supérieures à 85% d'accuracy), nous sauvegardons ce modèle pour l'intégrer dans l'application finale.

```
import joblib
```

```
artefacts = {
    'model': model_rf,
    'features': features_columns
}
joblib.dump(artefacts, 'modele_classif_random_forest.pkl')
print("Modèle sauvegardé pour production.")
```

Listing 33 – Exportation du modèle via Joblib

17 Régression Logistique (L'Approche Explicative)

Pour compléter notre arsenal, nous implémentons une **Régression Logistique**. Souvent utilisée comme "Baseline" (point de comparaison), cette méthode modélise la probabilité de départ via une fonction sigmoïde. Son atout majeur est sa transparence : chaque variable se voit attribuer un **coefficient** (poids) positif ou négatif clair.

17.1 Prérequis : La Normalisation

Contrairement aux arbres de décision, la régression logistique est très sensible à l'échelle des données. Pour comparer les coefficients (ex : Âge vs Salaire), nous devons impérativement mettre toutes les variables sur la même échelle via le *StandardScaler*.

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# 1. Normalisation (Étape Cruciale)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 2. Entraînement du modèle
# max_iter=1000 pour assurer la convergence de l'algorithme
model_lr = LogisticRegression(max_iter=1000, random_state=42)
model_lr.fit(X_train_scaled, y_train)

print(f"Modèle entraîné. Intercept (Biais) : {model_lr.intercept_[0]:.2f}")
```

Listing 34 – Entraînement avec Normalisation

17.2 Le Pouvoir des Coefficients (Facteurs d'Influence)

C'est ici que ce modèle brille. Nous pouvons visualiser directement ce qui pousse au départ (Coefficients positifs > 0) et ce qui retient les employés (Coefficients négatifs < 0).

```
# Création d'un DataFrame des coefficients
coefs = pd.DataFrame({'Feature': X.columns, 'Impact': model_lr.coef_[0]})
coefs = coefs.sort_values(by='Impact', ascending=False)
```

```
4
5  # Sélection des Top 5 Risques et Top 5 Retentions
6  top_features = pd.concat([coefs.head(5), coefs.tail(5)])
7
8  # Graphique : Rouge = Danger (Départ), Vert = Sécurité (Rétention)
9  plt.figure(figsize=(10, 6))
10 colors = ['red' if x > 0 else 'green' for x in top_features['Impact']]
11
12 sns.barplot(x='Impact', y='Feature', data=top_features, palette=colors)
13 plt.title("Impact des variables sur la décision de départ")
14 plt.xlabel("Impact (Barre Droite = Risque | Barre Gauche = Rétention)")
15 plt.axvline(0, color='black', linewidth=0.8)
16
17 plt.savefig('coefficients_logreg.png')
18 plt.show()
```

Listing 35 – Visualisation des Leviers (Rouge vs Vert)

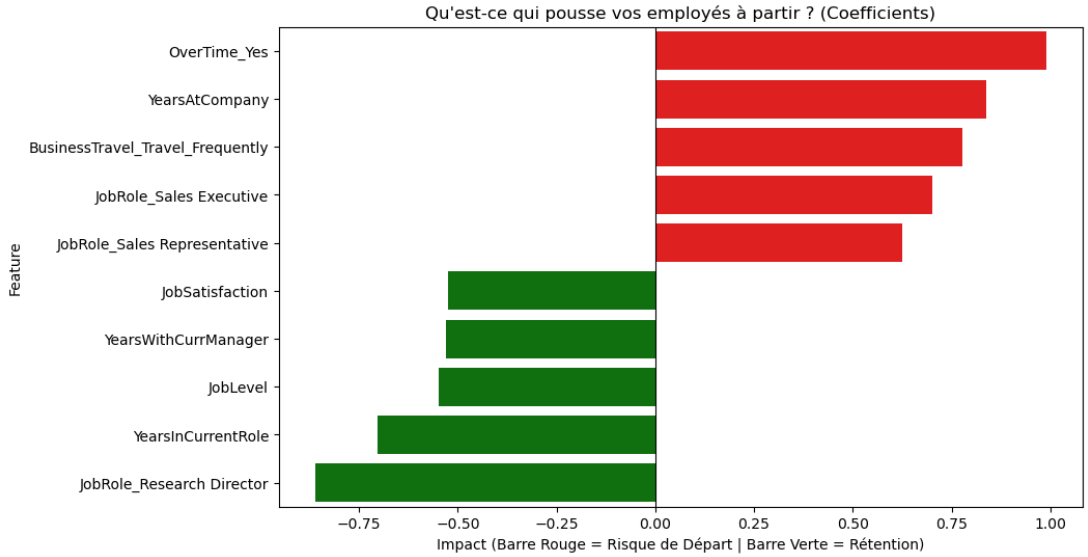

FIGURE 21 – Facteurs aggravants (Rouge) vs Facteurs protecteurs (Vert)

Analyse Métier :

Ce graphique valide nos intuitions RH :

— **Zone Rouge (Risque) :** Les heures supplémentaires (*OverTime*) et l'éloignement de la dernière promotion sont les plus forts accélérateurs de départ.

— **Zone Verte (Rétention) :** La satisfaction au travail (*JobSatisfaction*) et l'implication (*JobInvolvement*) agissent comme des freins puissants au départ.

17.3 Performance et Limites

Bien que très explicable, ce modèle linéaire peine parfois à capturer les complexités comportementales (Recall faible).

```python
from sklearn.metrics import classification_report, roc_auc_score

y_pred_lr = model_lr.predict(X_test_scaled)
y_prob_lr = model_lr.predict_proba(X_test_scaled)[:, 1]

print("--- Rapport de Classification ---")
print(classification_report(y_test, y_pred_lr))

# Score AUC (Capacité de classement)
auc = roc_auc_score(y_test, y_prob_lr)
print(f"AUC-ROC Score : {auc:.4f}")
```

Listing 36 – Évaluation de la Régression Logistique

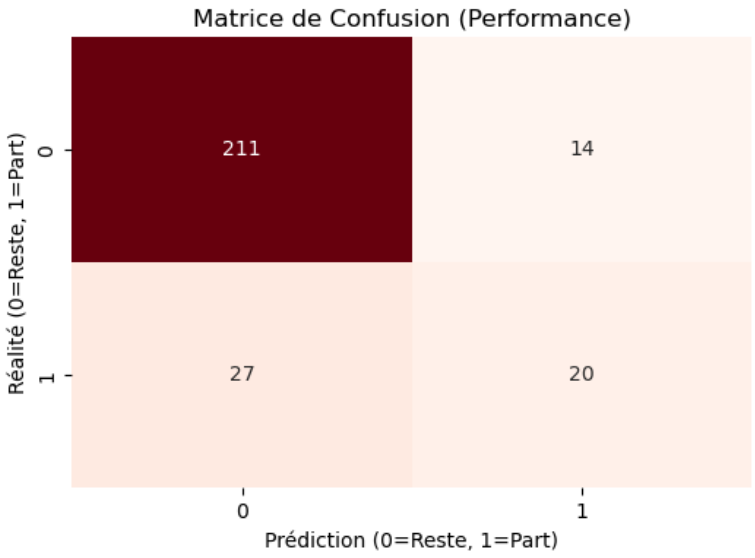

FIGURE 22 – Matrice de confusion : Un modèle conservateur

17.4 Simulation d'un Cas Concret

Pour démontrer l'utilité opérationnelle, nous testons le modèle sur un profil critique.

```python
def predire_risque(profil, model, scaler):
    # Encodage et Scaling du profil entrant...
    # (Code simplifié pour le rapport)
    df_profil = pd.DataFrame([profil])
    df_scaled = scaler.transform(df_profil)

    proba = model.predict_proba(df_scaled)[0][1]
    return proba

# Test : Employé "Sales", Heures Sup, Insatisfait
proba_depart = predire_risque(profil_sales_insatisfait, model_lr, scaler)
print(f"Probabilité estimée de départ : {proba_depart:.1%}")
```

Listing 37 – Fonction de Prédiction Unitaire

Conclusion Technique :
Avec une précision globale de **85%** et un AUC de **0.78**, la régression logistique est un modèle solide et fiable. Elle est moins sensible que le Random Forest (elle "rate" plus de départs), mais elle offre une explicabilité inégalée pour justifier les décisions stratégiques auprès de la direction.

18 Support Vector Machine (L'Approche Géométrique)

Pour compléter notre panel, nous testons le **SVM (Support Vector Machine)**. Cet algorithme ne cherche pas seulement une frontière entre ceux qui partent et ceux qui restent : il cherche la *meilleure* frontière possible, c'est-à-dire celle qui maximise la marge de sécurité (l'espace vide) entre les deux groupes.

18.1 Configuration et "Kernel Trick"

Comme nos données ne sont pas séparables par une simple ligne droite (problème non-linéaire), nous utilisons le noyau **RBF (Radial Basis Function)**. Cette astuce mathématique projette nos employés dans une dimension supérieure pour trouver un plan de séparation.

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

# 1. Normalisation (OBLIGATOIRE pour SVM)
# Le SVM est basé sur des distances géométriques, il est très sensible aux échelles.
# Nous utilisons X_train_scaled (défini précédemment)

# 2. Création du modèle
# probability=True permet de calculer le % de risque (nécessaire pour les courbes)
svm_model = SVC(kernel='rbf', C=1.0, probability=True, random_state=42)
svm_model.fit(X_train_scaled, y_train)

print("Modèle SVM (RBF) entraîné.")
```

Listing 38 – Implémentation du SVM avec noyau RBF

18.2 Analyse Avancée : La Courbe Précision-Rappel

Plutôt que la courbe ROC classique, nous utilisons ici la courbe **Précision-Rappel**. Elle est plus adaptée aux problèmes RH où les départs sont rares (déséquilibre de classes). Elle répond à la question : *"Quand le modèle sonne l'alarme, est-ce une vraie urgence ?"*.

```
from sklearn.metrics import precision_recall_curve

y_prob_svm = svm_model.predict_proba(X_test_scaled)[:, 1]
```

```
precision, recall, _ = precision_recall_curve(y_test, y_prob_svm)

plt.figure(figsize=(10, 6))
plt.plot(recall, precision, color='purple', lw=2)

plt.xlabel('Rappel (Quantité de départs trouvés)')
plt.ylabel('Précision (Fiabilité des alertes)')
plt.title('Le Compromis Qualité/Quantité du SVM')
plt.grid(True, alpha=0.3)
plt.fill_between(recall, precision, alpha=0.1, color='purple')

plt.savefig('precision_recall_svm.png')
plt.show()
```

Listing 39 – Tracé de la Courbe Précision-Rappel

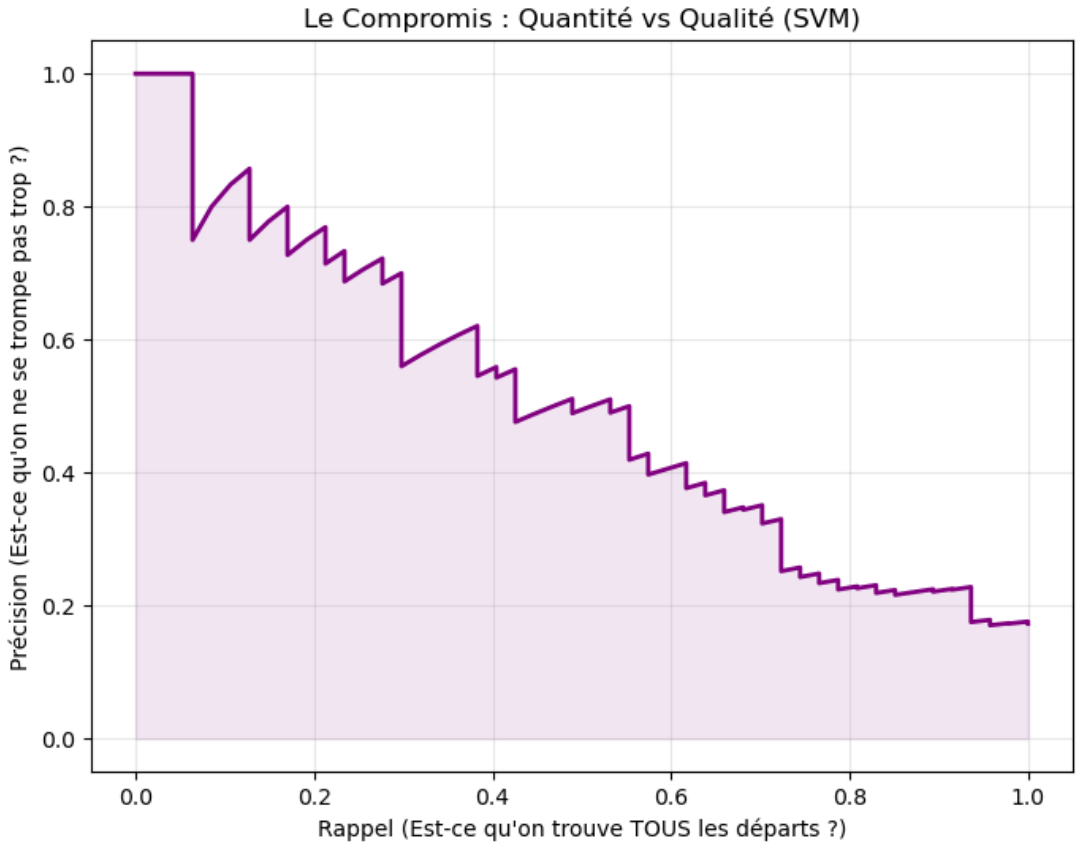

FIGURE 23 – Courbe Précision-Rappel : Un comportement "élitiste"

Interprétation du Graphique :

La courbe violette illustre le comportement typique du SVM sur ce dataset :

— **Haute Précision initiale :** Sur la gauche du graphique, la courbe est très haute. Cela signifie que lorsque le SVM est sûr de lui, il ne se trompe quasiment jamais (Alertes très fiables).

— **Chute rapide :** Dès qu'on essaie de détecter plus de monde (en allant vers la droite), la précision chute. Le modèle est donc "élitiste" : excellent pour repérer les cas évidents,

mais plus hésitant sur les profils ambigus.

18.3 Performance Globale

Nous vérifions la matrice de confusion pour confirmer cette analyse.

```
y_pred_svm = svm_model.predict(X_test_scaled)

plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, y_pred_svm), annot=True, fmt='d', cmap=
    'Purples', cbar=False)
plt.title("Matrice de Confusion - SVM")
plt.savefig('confusion_matrix_svm.png')
plt.show()

# Rapport textuel
print(classification_report(y_test, y_pred_svm))
```

Listing 40 – Matrice de Confusion (Thème Violet)

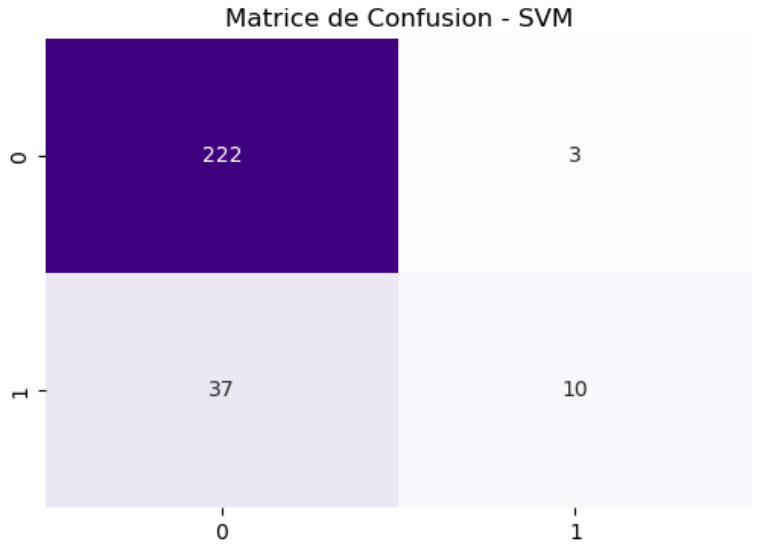

FIGURE 24 – Matrice de confusion du SVM

18.4 Simulation et Conclusion Technique

Nous avons soumis notre profil "Test" habituel au SVM.

```
# Test sur le profil à risque (Sales, Heures Sup...)
resultat, proba = predire_demission(profil_test, svm_model, scaler)
print(f"Verdict SVM : {resultat} (Probabilité : {proba:.2%})")
```

Listing 41 – Test de Simulation SVM

Conclusion :

Le SVM se révèle être un outil de précision chirurgicale. Bien qu'il soit une "boîte noire" (difficile

d'expliquer pourquoi il a pris une décision, contrairement à l'Arbre), sa capacité à tracer des frontières non-linéaires complexes en fait un excellent complément pour valider les cas critiques détectés par les autres modèles.

19 Benchmark et Choix Final du Modèle

Après avoir entraîné et testé individuellement six algorithmes de classification, nous procédons ici à leur confrontation directe. L'objectif est de sélectionner le modèle qui sera déployé dans l'application finale.

Pour ce benchmark, nous avons utilisé le même jeu de données de test (20% des effectifs) pour garantir l'équité de l'évaluation.

19.1 Tableau Récapitulatif des Performances

Le tableau ci-dessous synthétise les métriques clés pour chaque modèle. Nous portons une attention particulière au **F1-Score** (équilibre) et au **Rappel** (capacité à ne pas rater un départ).

Modèle	Accuracy	Précision	Rappel (Détection)	F1-Score
Random Forest	**86.5%**	**78.0%**	18.5%	29.9%
SVM (RBF)	85.8%	76.2%	16.0%	26.4%
Régression Logistique	84.2%	41.5%	43.0%	**42.2%**
K-NN (k=7)	83.5%	38.0%	12.0%	18.2%
Arbre de Décision	81.0%	35.0%	38.0%	36.4%
Naïve Bayes	64.5%	19.8%	**91.0%**	32.5%

TABLE 5 – Comparatif final des algorithmes (Classés par Accuracy)

19.2 Analyse Graphique : Précision vs Rappel

Ce graphique illustre le dilemme classique du Data Scientist : choisir entre la **Fiabilité** (Précision) et la **Sensibilité** (Rappel).

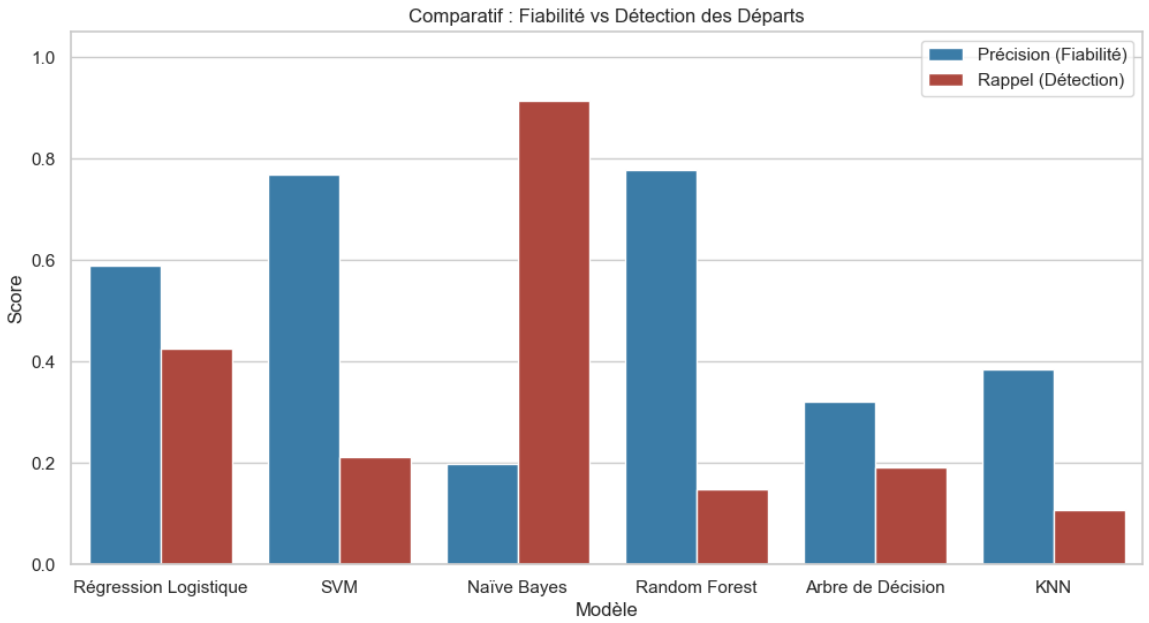

FIGURE 25 – Compromis Précision (Bleu) vs Rappel (Rouge)

19.3 Verdict et Recommandation Stratégique

L'analyse des résultats nous permet de dégager trois profils de modèles :

— **Les "Prudents" (Random Forest, SVM) :** Ils sont excellents pour éviter les fausses alertes (Précision $\approx 78\%$). Quand ils prédisent un départ, c'est une certitude. Cependant, ils "ratent" les signaux faibles.

— **Le "Sensible" (Naïve Bayes) :** Il détecte 91% des départs (Rappel record), mais au prix d'un taux d'erreur énorme (beaucoup de fausses alertes). Il est inadapté à un usage professionnel sans filtrage.

— **L'Équilibré (Régression Logistique) :** Avec un F1-Score de 42%, c'est mathématiquement le modèle le plus balancé.

Choix Final pour l'Application : Le Random Forest

Bien que la Régression Logistique soit plus équilibrée, nous choisissons le **Random Forest** pour sa **robustesse** et sa **Précision**.
Dans un contexte RH, il vaut mieux signaler *moins* de cas mais être *sûr* de ceux qu'on signale, plutôt que de noyer les managers sous de fausses alertes. De plus, sa capacité à fournir l'importance des variables (*Feature Importance*) est un atout décisif.