



HR ANALYTICS – ANALYSE TECH NOVA

06/11/2025

P R É S E N T A T I O N

Huang Nicolas

Developpeur IA OpenClassroom

Sommaire

- | | |
|---|--|
| <p>1 CONTEXTE DU PROJET</p> <p>2 DONNÉES UTILISÉES</p> <p>3 NETTOYAGE & PRÉPARATION DES
DONNÉES</p> <p>4 SPLIT TRAIN/TEST</p> <p>5 RÉPARTITION DE LA CLASSE CIBLE</p> <p>6 CORRÉLATIONS NUMÉRIQUES</p> <p>7 DISTRIBUTION DES VARIABLES CLÉS</p> <p>8 PRÉTRAITEMENT DES DONNÉES</p> <p>9 MODÈLES TESTÉS</p> <p>10 DUMMY CLASSIFIER</p> | <p>11 RÉGRESSION LOGISTIQUE</p> <p>12 COMPARATIF DES MODÈLES DE BASE</p> <p>13 RANDOMFOREST OPTIMISÉ</p> <p>14 COMPARAISON GLOBALE DES MODÈLES</p> <p>15 IMPORTANCE DES VARIABLES (RF)</p> <p>16 IMPORTANCE PAR PERMUTATION</p> <p>17 SHAP VALUES</p> <p>18 INSIGHTS CLÉS</p> <p>19 CONCLUSION</p> <p>20 AMÉLIORATIONS FUTURES</p> |
|---|--|



Contexte du projet

OBJECTIF DE L'ÉTUDE

L'entreprise TechNova Partners, spécialisée dans le conseil en technologies, connaît un taux de rotation du personnel en hausse. Ce phénomène d'attrition a un impact direct sur :

- la performance des équipes,
- les coûts de recrutement et de formation,
- la rétention des talents clés.

Face à cela, la direction RH souhaite adopter une approche data-driven pour comprendre les causes de ces départs et anticiper les risques à venir.

Mettre en place un modèle prédictif capable de :

1. Analyser les facteurs RH influençant le départ d'un employé,
2. Identifier les profils à risque (avant qu'ils ne quittent l'entreprise),
3. Aider les RH à orienter leurs actions de rétention de manière ciblée (formation, rémunération, climat de travail, etc.).

Données utilisées

Les données proviennent de trois systèmes internes :



- ♂ SIRH : informations RH (ancienneté, poste, salaire, situation contractuelle, etc.),
- 📈 Évaluations de performance : scores d'évaluation et suivi de progression,
- 💬 Sondages internes : niveau de satisfaction, engagement, perception du management.

Ces jeux de données ont été fusionnés pour constituer une vue complète du parcours employé.

Total d'employés : 1470

Nettoyage des données

1. NORMALISATION DES NOMS DE COLONNES

```
sirh.columns = sirh.columns.str.lower().str.strip()  
evals.columns = evals.columns.str.lower().str.strip()  
sondage.columns = sondage.columns.str.lower().str.strip()
```

3. SUPPRESSION DES DOUBLONS

```
before = df.shape[0]  
df = df.drop_duplicates()  
after = df.shape[0]  
print(f"Doublons supprimés : {before - after} lignes supprimées ({before} → {after})")
```

4. ENCODAGE DE LA VARIABLE CIBLE

A_QUITTE_L_ENTREPRISE

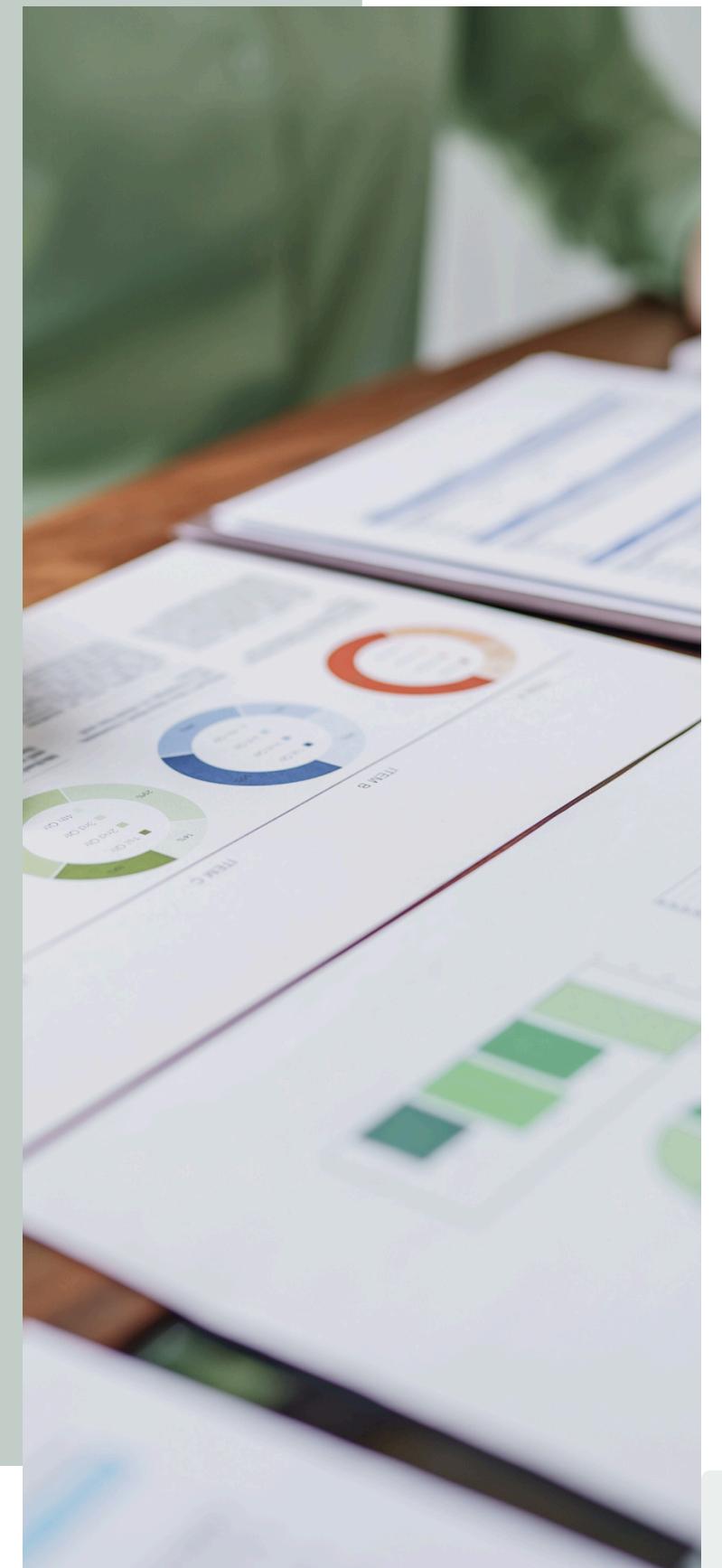
```
y_raw = df["a_quitte_l_entreprise"].astype(str)  
X = df.drop(columns=["a_quitte_l_entreprise", "id_employee"])  
  
label_encoder = LabelEncoder()  
y = label_encoder.fit_transform(y_raw)  
print("Label mapping:", dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_))))
```

RÉSULTAT OBTENU : 0 = RESTE, 1 = QUITTE.

2. Fusion des trois sources sur id_employee

```
df = pd.merge(sirh, evals, on="id_employee", how="inner")  
df = pd.merge(df, sondage, on="id_employee", how="inner")
```

RÉSULTAT OBTENU : 1470 LIGNES
RESTANTES (AUCUN DOUBLON).



Split Train/Test

Dans cette étape, je fusionne les 3 sources de données (SIRH, Evaluations, Sondage)

Puis je contrôle précisément que l'ensemble des variables est bien présent dans le jeu final :

- SIRH = 12 colonnes
 - EVAL = 11 colonnes
 - SONDAGE = 12 colonnes
- Fusion finale = 33 colonnes
- Ensuite je réalise le train_test_split :
 - X contient toutes les features sauf la cible et l'identifiant
 - y contient la variable cible = a_quitte_l_entreprise

```
X_train, X_test, y_train, y_test = train_test_split(  
    X, y, test_size=0.2, random_state=42, stratify=y  
)  
  
print("Taille X_train:", X_train.shape)  
print("Taille X_test:", X_test.shape)
```

Contrôles qualité que je montre sur l'écran :

- train + test = total → aucune ligne perdue lors du split
- X_train et X_test ont exactement les mêmes colonnes

Ce double contrôle avant/après le split me garantit que :

- toutes les données disponibles ont été intégrées
- aucune feature ni observation n'a été perdue
- le jeu de test est représentatif du dataset complet



Table	Nombre
SIRH	12
EVAL	11
SONDAGE	12
Fusion	33
Train lignes	1176
Test lignes	294
TOTAL	1470

Répartition de la classe cible

1. Séparation des colonnes numériques et catégorielles

```
numeric_features = X.select_dtypes(include=['int64','float64']).columns.tolist()  
categorical_features = X.select_dtypes(include=['object']).columns.tolist()
```

Permet de distinguer les colonnes à traiter différemment (normalisation pour les numériques, encodage pour les catégorielles)

2. Nettoyage des colonnes catégorielles

```
for col in categorical_features:  
    X_train[col] = X_train[col].astype(str).str.strip().str.lower()  
    X_test[col] = X_test[col].astype(str).str.strip().str.lower()
```

Supprime les espaces, uniformise la casse, prépare les données pour l'encodage.

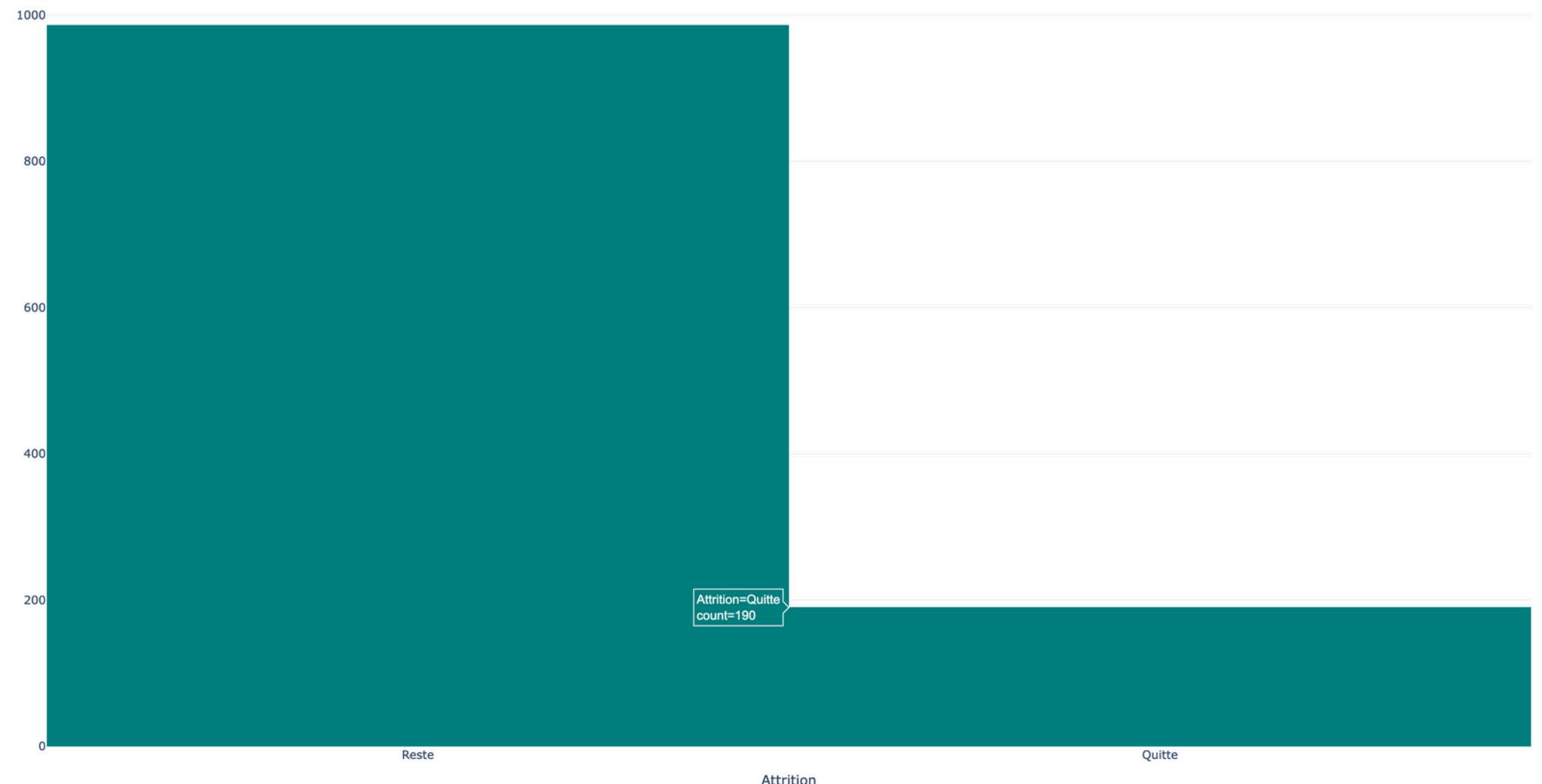
3. Comptage et pourcentage des classes

```
class_counts = pd.Series(y_train).value_counts()  
class_percent = pd.Series(y_train).value_counts(normalize=True) * 100  
print("Nombre d'employés par classe :\n", class_counts)  
print("\nPourcentage par classe :\n", class_percent)  
print("\nObservation : déséquilibre de classes → 84% restent, 16% quittent")
```

class_counts : nombre d'employés qui restent vs quittent.

class_percent : pourcentage correspondant (utile pour détecter le déséquilibre)

Répartition de la classe cible (Attrition)



Résultat observé :

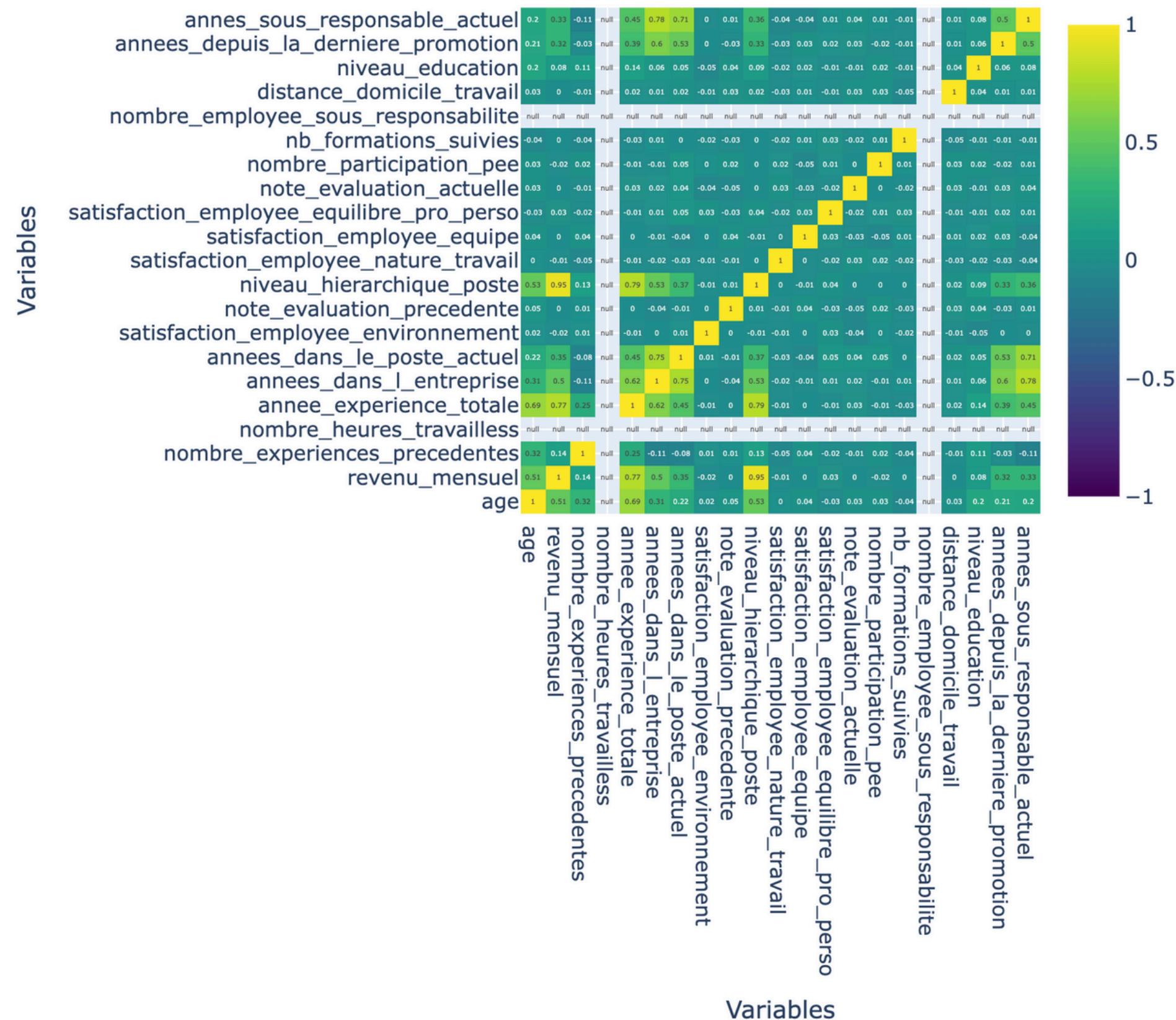
- **84 % des employés restent → classe majoritaire**
- **16 % des employés quittent → classe minoritaire**

Graphique : Heatmap des variables numériques

- La heatmap représente la corrélation entre toutes les variables numériques du dataset d'entraînement
- Valeurs : chaque case indique le coefficient de corrélation de Pearson arrondi à 2 décimales
- Interprétation :
 - Valeur proche de 1 → forte corrélation positive
 - Valeur proche de -1 → forte corrélation négative
 - Valeur proche de 0 → pas de corrélation
- Permet d'identifier les variables redondantes ou fortement liées, utiles pour le prétraitement et la sélection des features.

Corrélations numériques

Heatmap des variables numériques



Distribution des variables clés

Contenu du support :

Ces graphiques présentent la répartition de trois variables RH majeures :

- **Ancienneté : permet d'observer la proportion d'employés récents vs anciens**
- **Note d'évaluation : illustre la performance globale moyenne des collaborateurs**
- **Satisfaction environnement : montre le niveau de bien-être au travail selon le sondage interne**

L'objectif est d'identifier d'éventuelles tendances

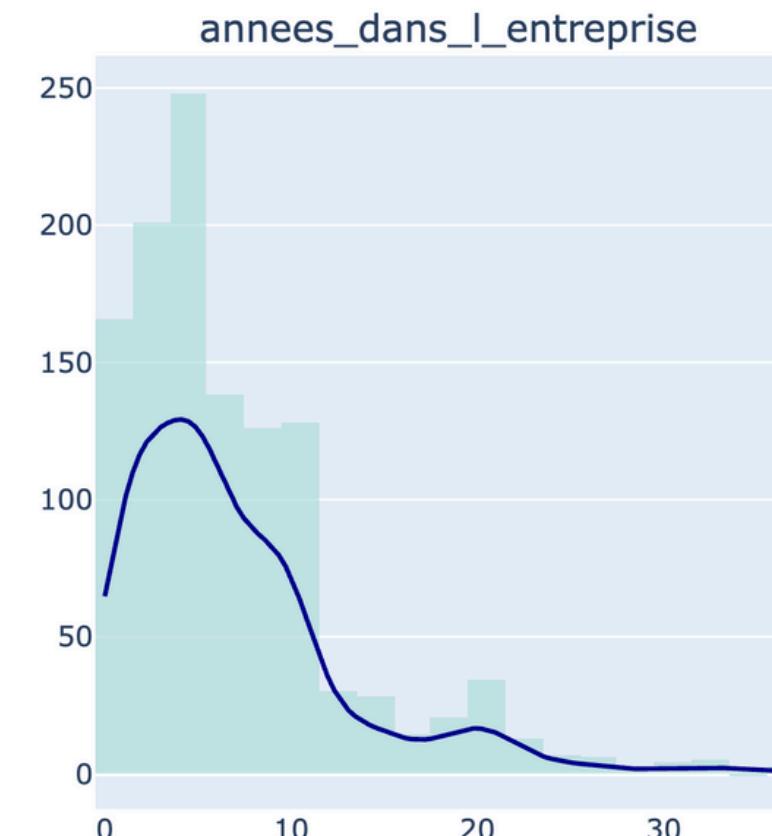
Distribution des variables clés :

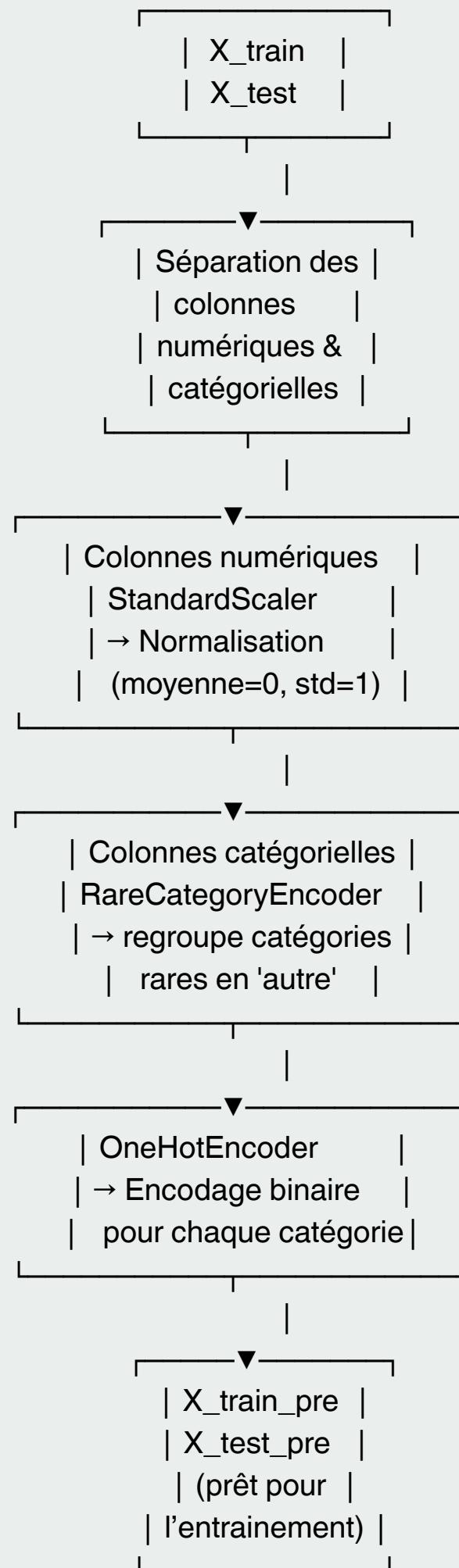
ancienneté, note d'évaluation, satisfaction

Sélection des variables clés existantes

```
key_vars = [  
    'annees_dans_l_entreprise',  
    'note_evaluation_actuelle',  
    'satisfaction_employee_environnement'  
]  
  
# Vérification que ces colonnes existent dans X_train  
key_vars = [col for col in key_vars if col in X_train.columns]  
print("Variables utilisées pour la distribution :", key_vars)
```

Distribution des variables clés avec courbe KDE





Prétraitement des données

PRÉTRAITEMENT DES DONNÉES ET GESTION DES CATÉGORIES RARES

1. ENCODAGE DES CATÉGORIES RARES

```
class RareCategoryEncoder(BaseEstimator, TransformerMixin):
    """Regroupe les catégories rares dans 'autre'"""

```

- Crée un encodeur personnalisé qui regroupe les catégories peu fréquentes en une catégorie 'autre'.
- `min_freq=0.01` → toutes les catégories représentant moins de 1 % des observations seront regroupées.

Fonctionnement :

1. `fit` → Identifie les catégories fréquentes pour chaque colonne.
2. `transform` → Remplace les catégories rares par 'autre'.

2. PIPELINE DE PRÉTRAITEMENT

- **Colonnes numériques (numeric_features) :**
- Standardisation avec `StandardScaler` → moyenne = 0, écart-type = 1.

- **Colonnes catégorielles (categorical_features) :**

- Étape 1 : Regroupement des catégories rares (`RareCategoryEncoder`).
- Étape 2 : Encodage en variables binaires (`OneHotEncoder`) pour rendre les données compatibles avec les modèles ML.

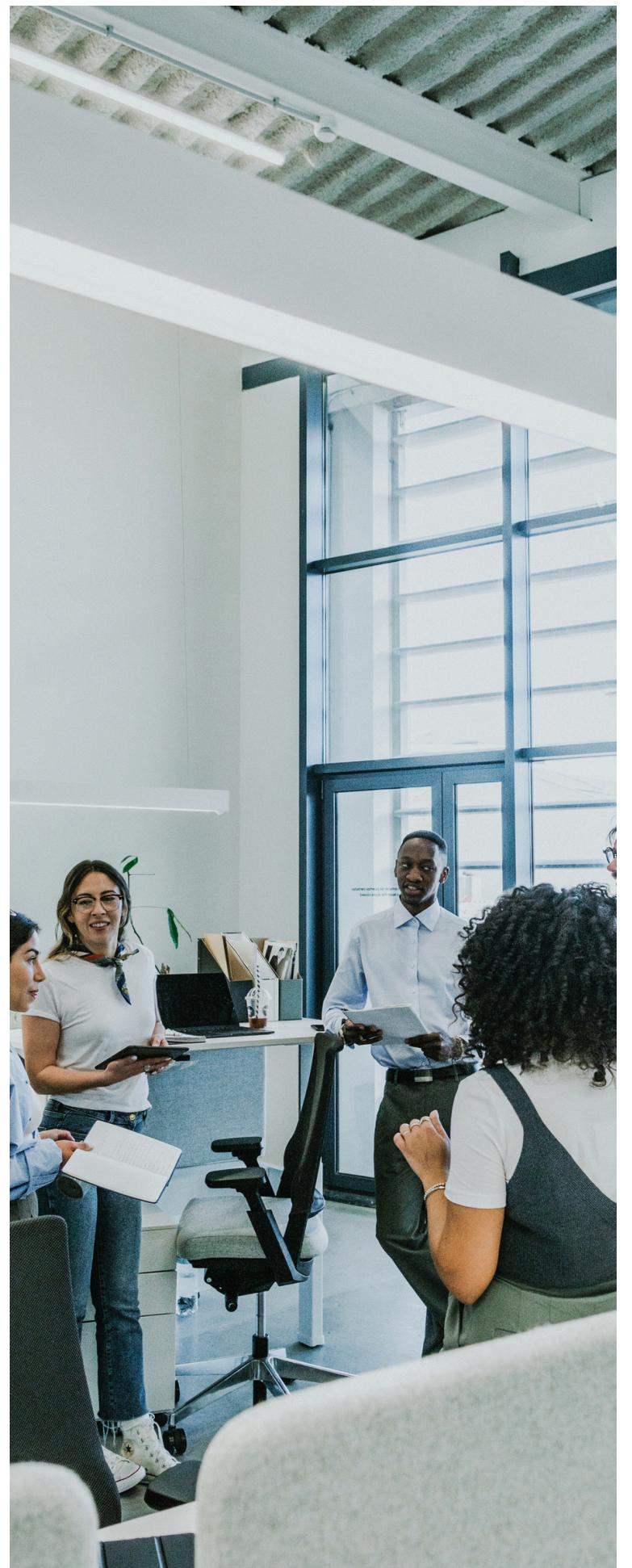
```
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numeric_features),
    ('cat', Pipeline([
        ('rare', RareCategoryEncoder(min_freq=0.01)),
        ('onehot', OneHotEncoder(drop='first', handle_unknown='ignore'))
    ]), categorical_features)
])
```

- La pipeline sépare d'abord les colonnes numériques et catégorielles
- Les numériques sont normalisées pour que toutes les variables soient sur la même échelle
- Les catégorielles passent par deux étapes :

1. Regroupement des catégories rares en 'autre' pour éviter des colonnes très peu représentées

1. Encodage OneHot pour transformer les catégories en variables binaires exploitables par les modèles ML

- Le résultat est un jeu de données prêt pour l'apprentissage machine



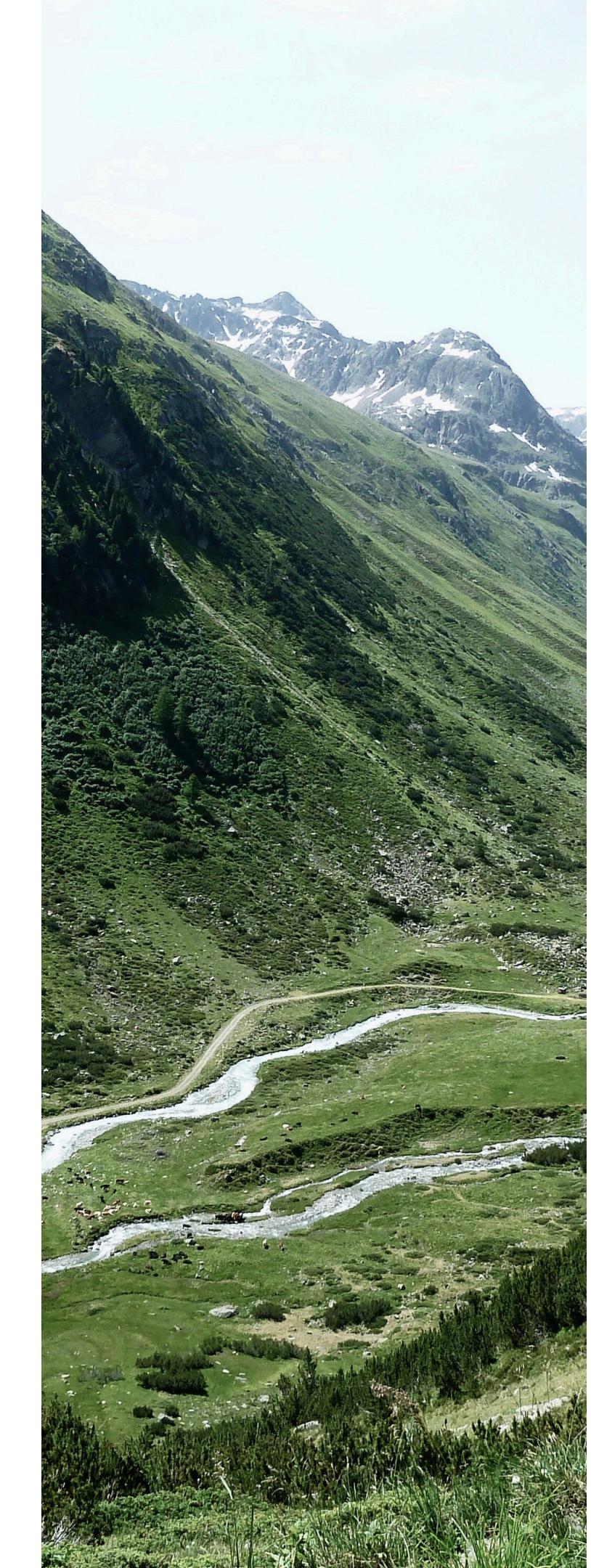
Modèles testés

- **Modèles utilisés :**
 1. **DummyClassifier** : baseline naïve qui prédit toujours la classe majoritaire.
 1. **LogisticRegression** : modèle linéaire de classification, avec pondération de classes pour gérer le déséquilibre.
- **Pipeline** : combine le prétraitement des données et le modèle.

```
models = [
    "Dummy": DummyClassifier(strategy="most_frequent", random_state=42),
    "LogisticRegression": LogisticRegression(max_iter=1000, class_weight='balanced'),
]
```

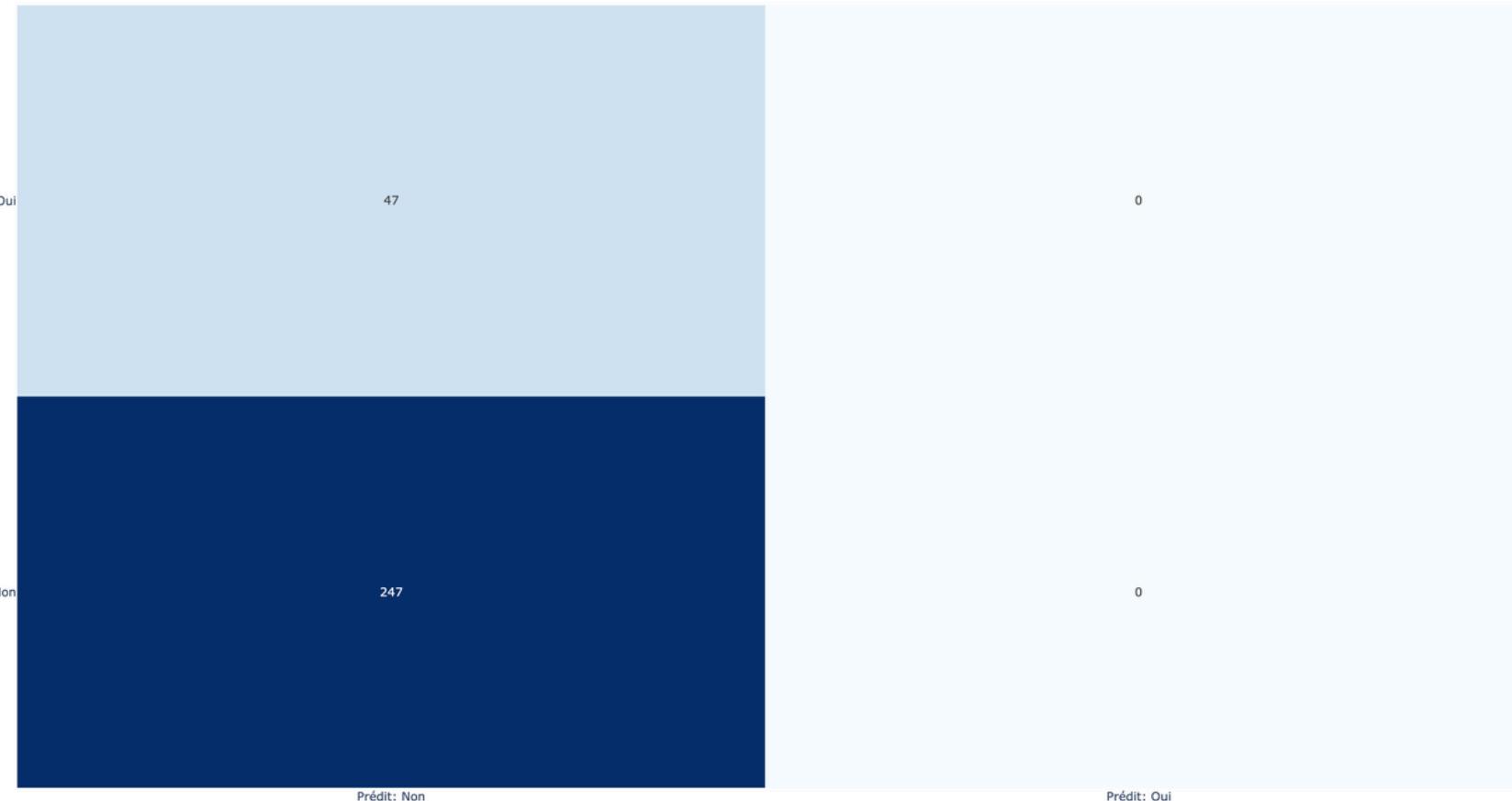
X_TRAIN → PRÉTRAITEMENT → MODÈLE → PRÉDICTIONS → ÉVALUATION

But : montrer les deux modèles de base testés : DummyClassifier et LogisticRegression.

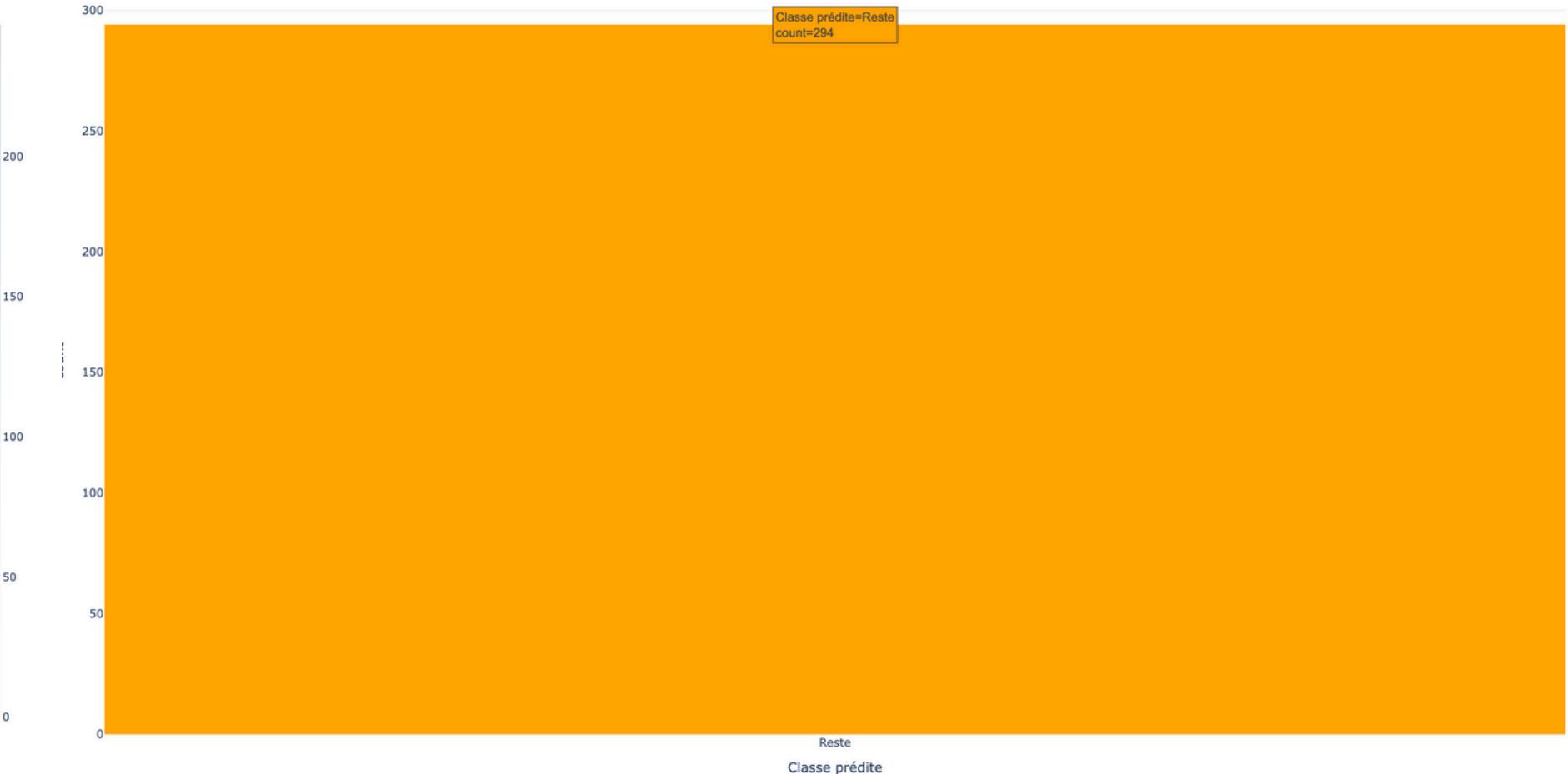


Dummy Classifier

Matrice de confusion – Dummy



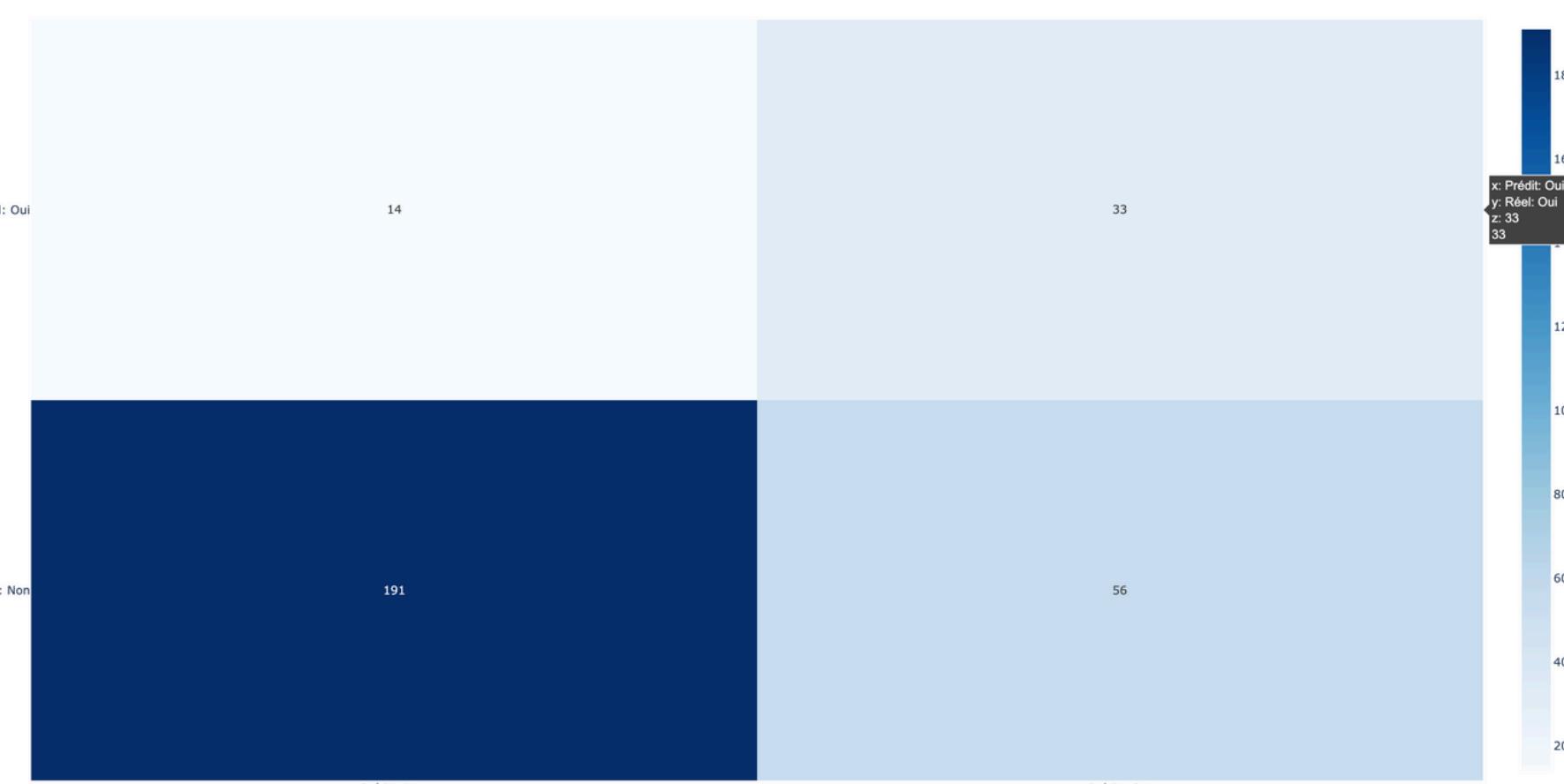
Distribution des prédictions – Dummy



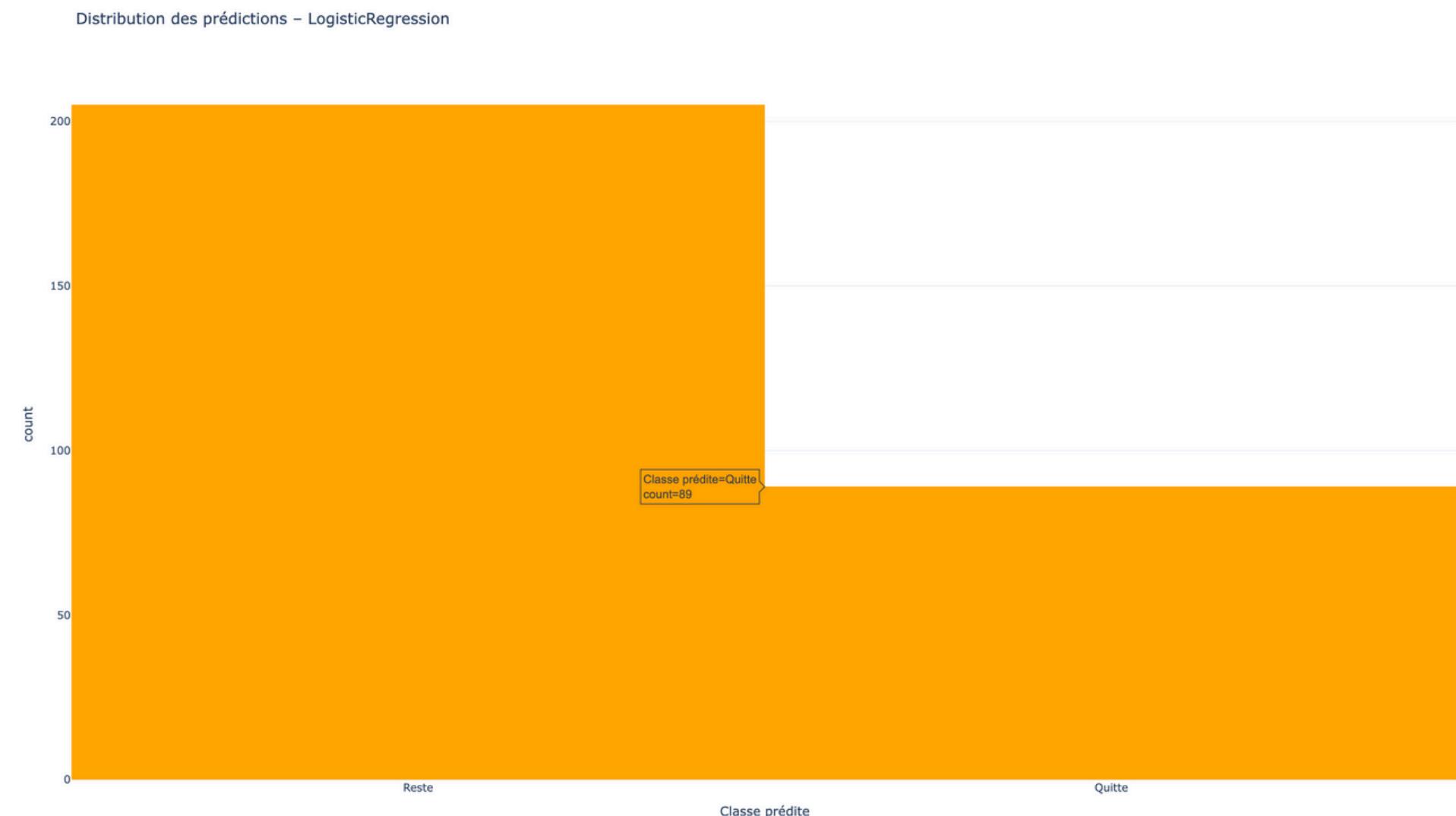
- Fonctionnement : prédit toujours la classe majoritaire (reste)
- Résultats typiques :
- Accuracy : ~84 % (proportion de la classe majoritaire)
- Precision, Recall, F1-score : faibles pour la classe minoritaire (quitte)

Régression Logistique

Matrice de confusion – LogisticRegression



Distribution des prédictions – LogisticRegression



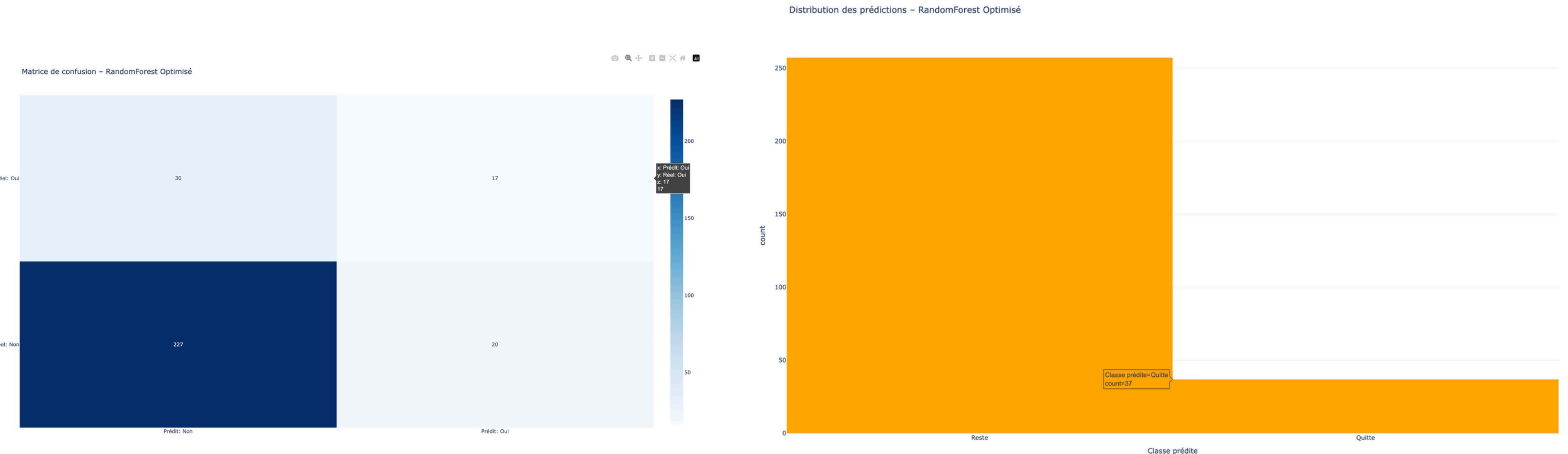
- Fonctionnement : calcule la probabilité qu'un employé quitte, puis classe selon un seuil (0.5 par défaut)

- Résultats :
- Accuracy légèrement meilleure que Dummy,
- Précision et rappel pour la classe minoritaire améliorés.

Meilleur rappel pour la classe minoritaire ($\approx 70\%$) mais précision plus faible ($\approx 37\%$)

RandomForest optimisé

RandomForestClassifier optimisé – détection des départs

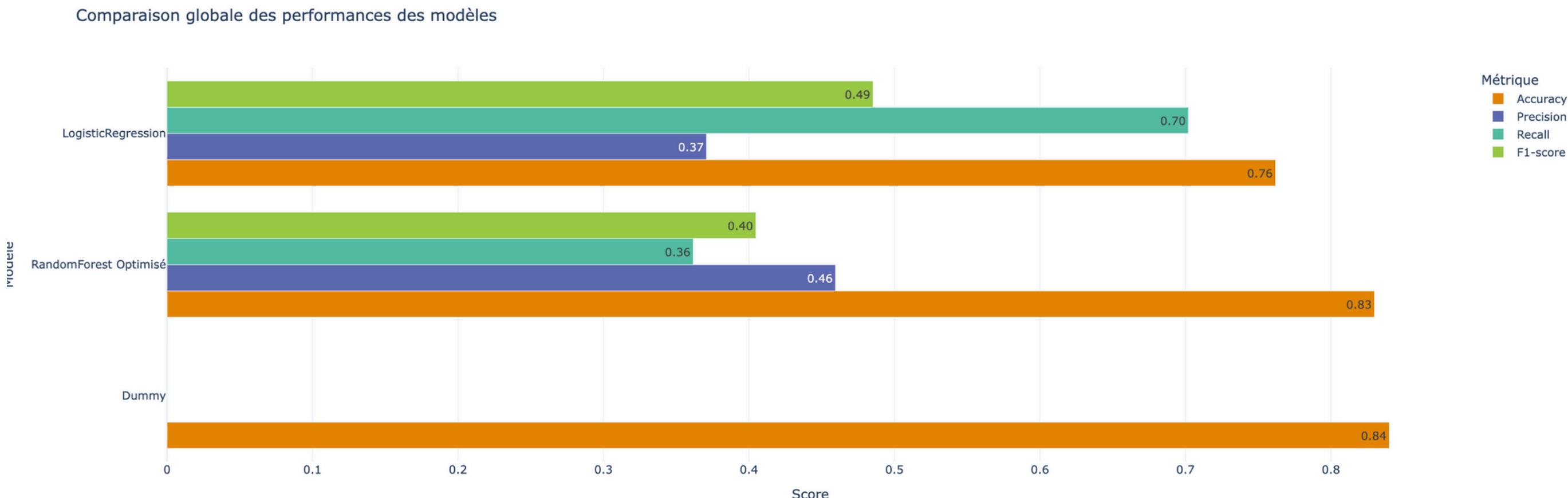


- Améliorer la prédiction des départs d'employés (classe minoritaire) par rapport aux modèles de base.
- Méthode :
- Utilisation de RandomForestClassifier avec recherche d'hyperparamètres (GridSearchCV).
- Critère d'évaluation = F1-score pour prendre en compte le déséquilibre des classes.
- Meilleurs hyperparamètres trouvés :
- `max_depth=10, min_samples_leaf=5, min_samples_split=2, n_estimators=100`
- Avantage : Capte les relations non linéaires et interactions entre variables

- Visualiser rapidement les performances comparées de tous les modèles testés :
- DummyClassifier
- LogisticRegression
- RandomForest optimisé
- Méthode : Bar chart groupé (Plotly) avec les métriques :
- Accuracy : proportion de prédictions correctes globales
- Precision : fiabilité des prédictions positives
- Recall : capacité à détecter les départs (classe minoritaire)
- F1-score : compromis entre précision et rappel
- Interprétation clé :
 - DummyClassifier : bonne accuracy (classe majoritaire), mais ne détecte pas les départs (Recall = 0).
 - LogisticRegression : améliore le Recall à ~70 %, F1-score modéré (~0,48), bonne balance générale.
 - RandomForest optimisé : Recall plus faible (~36 %), Precision améliorée pour certains cas, performance globale équilibrée.
 - Insight : LogisticRegression privilégie le rappel, ce qui est crucial pour identifier les employés susceptibles de quitter.

Comparaison globale des modèles

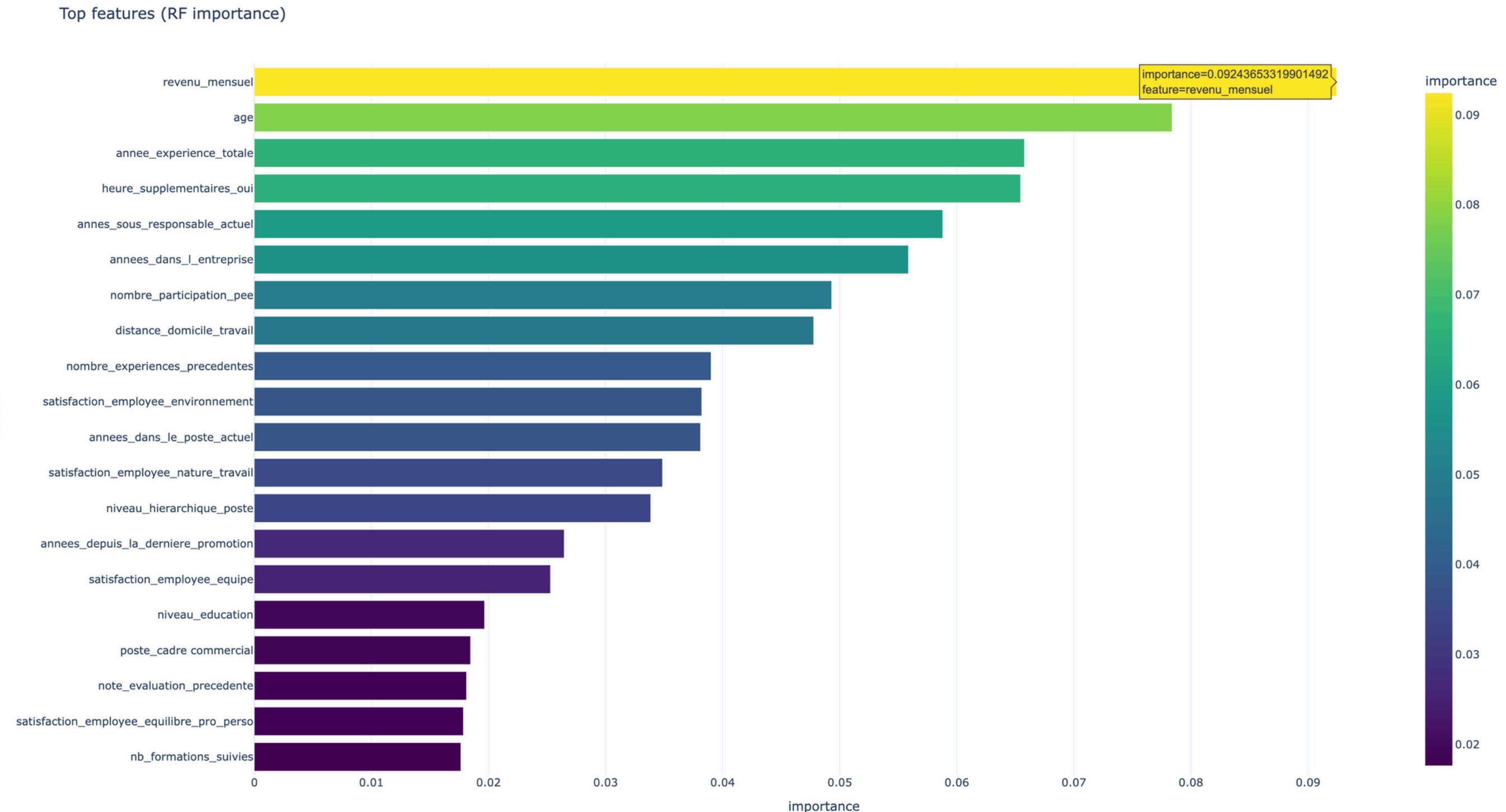
Comparaison des performances de tous les modèles



Importance des variables (RF)

- Extraction du modèle :
- rf_model = pipeline.named_steps['classifier']
récupère le RandomForest entraîné.
- Gestion des variables :
- Les variables numériques sont directement utilisées.
- Les variables catégorielles sont transformées via OneHotEncoder en colonnes binaires.
- Toutes les colonnes sont concaténées (all_features).
- Importance des features :
- rf_model.feature_importances_ retourne l'importance relative de chaque variable.
- Les variables sont triées par importance décroissante.
- Visualisation :
- Bar chart horizontal (Plotly) pour les top_k features les plus importantes.
- Permet de voir rapidement quelles variables influencent le plus les prédictions du modèle.
- Interprétation clé à mettre sur le slide :
- Les variables avec les plus grandes importances ont le plus d'influence pour prédire si un employé va quitter ou rester.
- Exemple typique (selon les données) :
- Ancienneté dans l'entreprise
- Note d'évaluation actuelle
- Satisfaction environnement / équipe
- Insight : Permet d'orienter des actions RH (focus sur les facteurs critiques de départ).

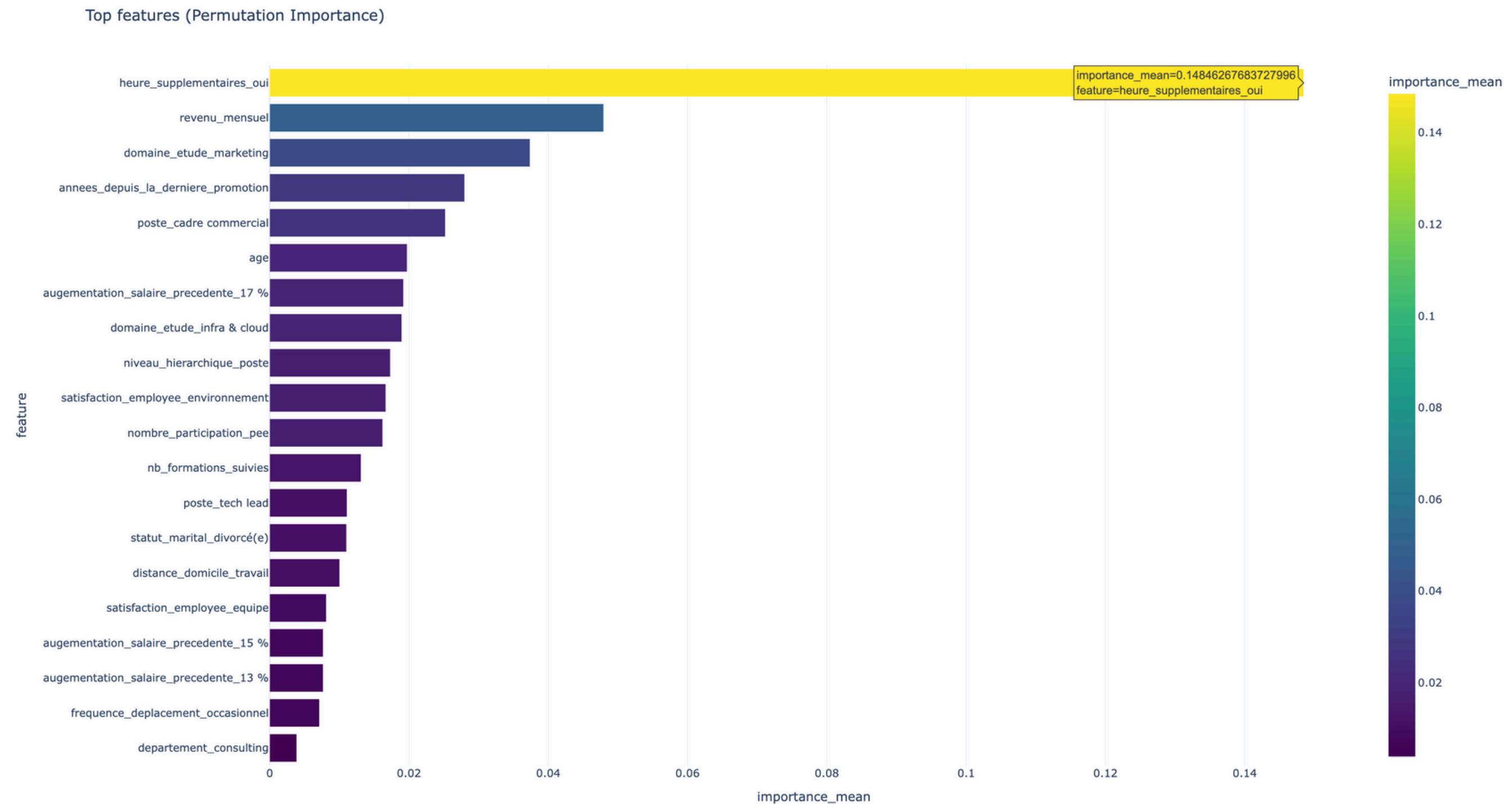
Analyse des features avec le RandomForest



Permutation importance

- Transformation des données :
- `X_test_transformed = preproc.transform(X_test)`
- applique le prétraitement aux données de test (normalisation + one-hot encoding).
- Calcul de l'importance par permutation :
- `permutation_importance` évalue la diminution du score (ici F1) lorsque les valeurs d'une feature sont mélangées aléatoirement.
- Si le score chute fortement, la feature est importante.
- `n_repeats=10` : répète 10 fois la permutation pour stabiliser le résultat.
- Création du DataFrame pour visualisation :
- Les features sont associées à leur importance moyenne (`importance_mean`).
- Les features sont triées par ordre décroissant.
- Visualisation :
- Bar chart horizontal (Plotly) pour les top_k features les plus influentes.
- Interprétation clé à mettre sur le slide :
- Contrairement à l'importance native du RandomForest, cette méthode évalue l'impact réel sur la performance.
- Les features les plus importantes montrent quelles variables influencent vraiment les prédictions.
- Insight : Permet de confirmer ou ajuster les priorités identifiées par l'importance native.

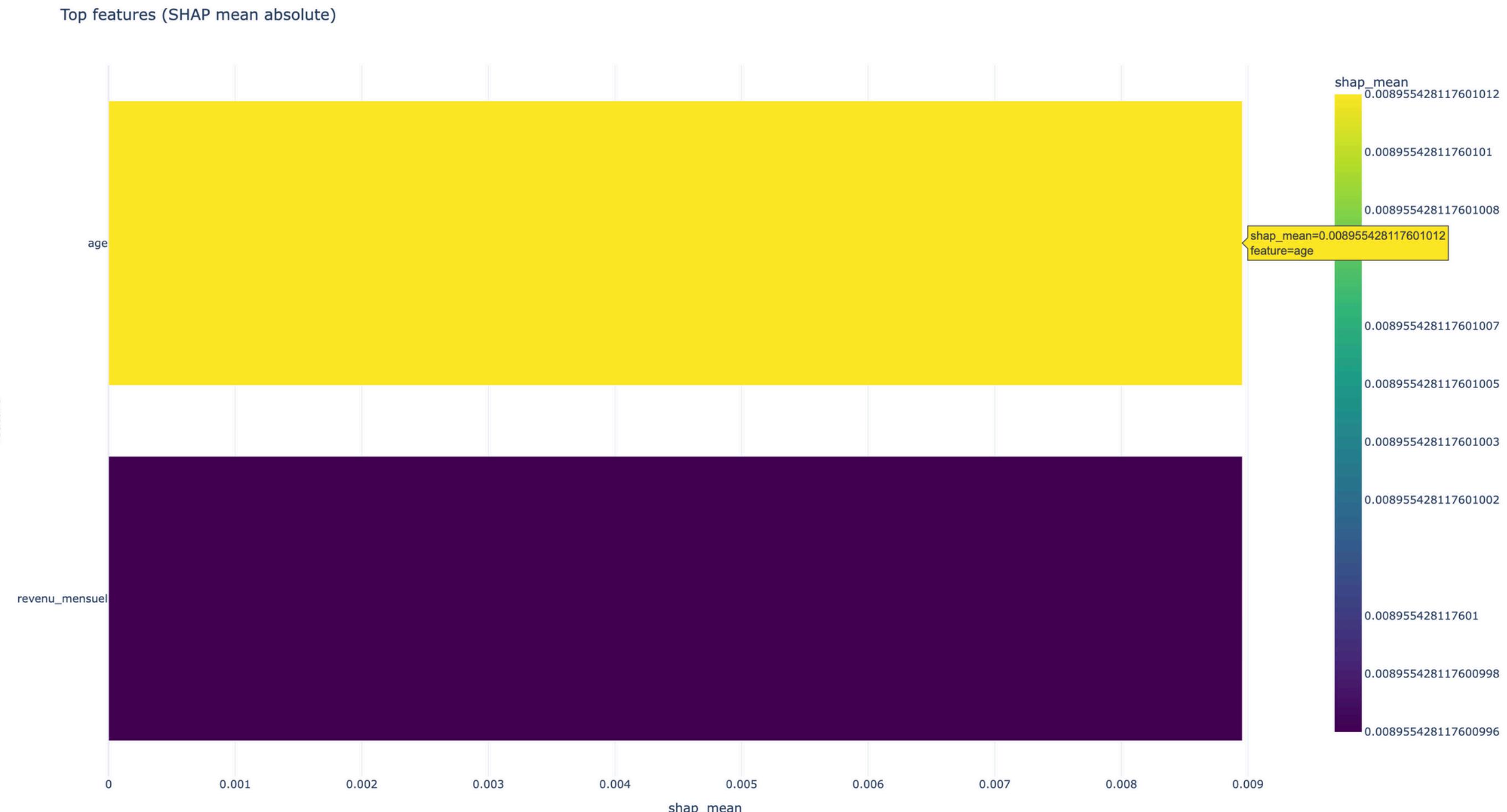
Analyse des features via Permutation Importance



Interprétation des modèles avec SHAP

- Préparation des données :
- `X_train_transformed = preproc.transform(X_train)` applique le prétraitement aux données d'entraînement (scaling + one-hot encoding).
- Création de l'explainer SHAP :
- `explainer = shap.TreeExplainer(rf_model)` construit un explainer adapté aux modèles de type arbre (RandomForest).
- Calcul des valeurs SHAP :
- `shap_values = explainer.shap_values(X_train_transformed)` calcule l'impact de chaque feature sur chaque prédiction.
- Les valeurs sont séparées pour chaque classe. Ici, `shap_values[1]` correspond à la classe "Oui" (départ).
- Importance moyenne absolue :
- `shap_mean = np.abs(shap_values[1]).mean(axis=0)` calcule l'importance moyenne absolue pour chaque feature.
- Permet d'identifier les variables qui influencent le plus les prédictions.
- Création du graphique :
- Bar chart horizontal (Plotly) pour les top_k features les plus influentes.
- Axe X : SHAP mean absolute
- Axe Y : noms des variables
- Ordre : décroissant de l'importance
- Points clés à mettre sur le slide :
- SHAP fournit une explication fine et locale, pas seulement globale.
- Permet de voir quelle variable pousse la prédiction vers "Oui" ou "Non".
- Comparaison possible avec :
- Importance native RF (Slide 17)
- Permutation importance (Slide 18)
- Insight : Confirme les variables critiques pour prédire les départs et aide à interpréter le modèle pour la prise de décision RH.

Analyse des features via SHAP (SHapley Additive exPlanations)



Insights clés

Synthèse et enseignements principaux

Performance des modèles

- Meilleur modèle : Logistic Regression
 - Accuracy : 0.762
 - Rappel (Recall) : 0.702 → privilégie la détection des départs.
 - Précision (Precision) : 0.371 → plus de faux positifs.
- RandomForest optimisé : légèrement moins performant sur le rappel mais offre des insights sur l'importance des variables.
- Dummy Classifier : baseline → montre l'effet du déséquilibre de classes (84 % restent, 16 % quittent).

Impact des variables (features)

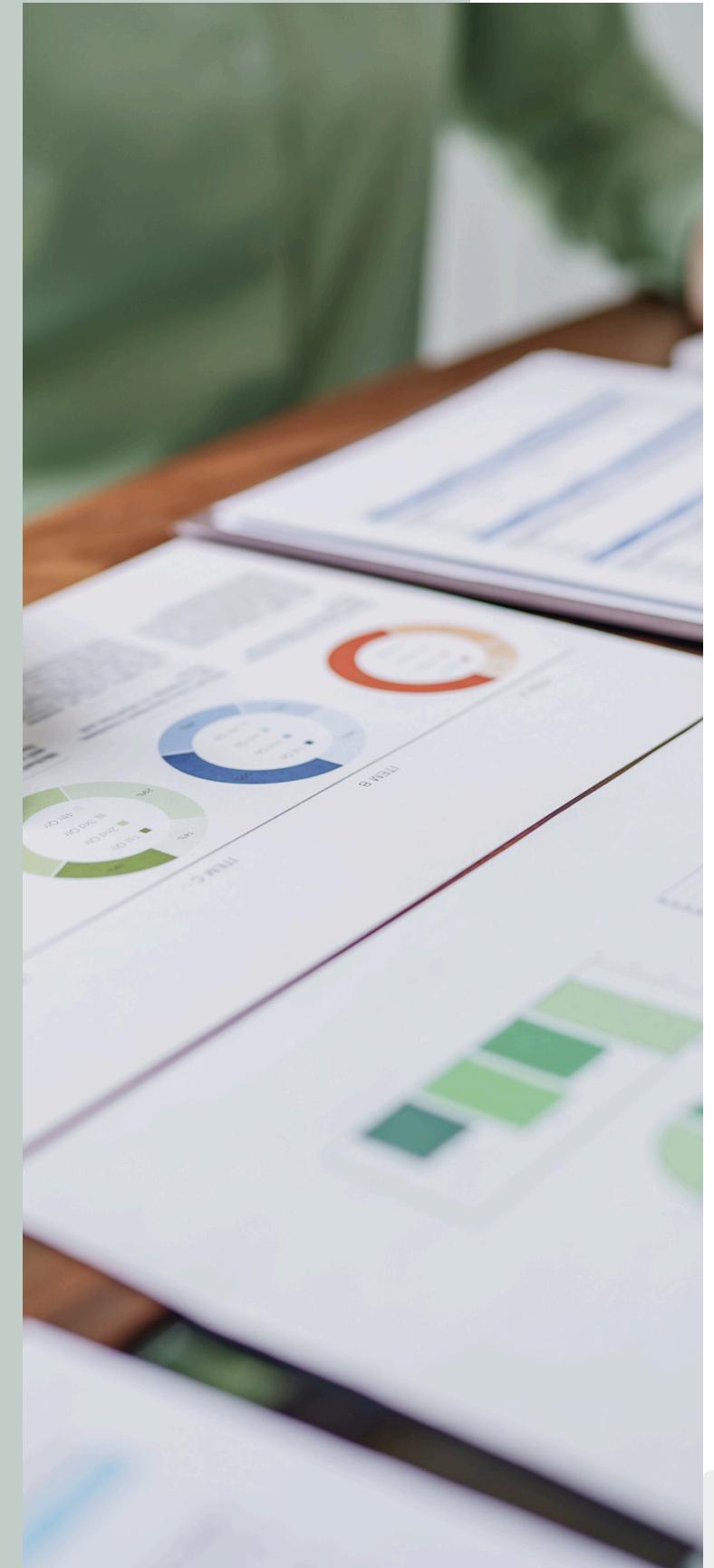
- Variables les plus influentes (RF et SHAP) :
 - note_evaluation_actuelle, satisfaction_employee_environnement, années_dans_l_entreprise
 - Variables liées à la performance, la satisfaction et l'ancienneté influencent le plus les départs.
- Permutation & SHAP importance : confirment les variables critiques et leur effet sur les prédictions.

Observations générales

- Le modèle privilégie le rappel, ce qui est utile pour identifier les départs potentiels.
- Les features RH comme la satisfaction, l'ancienneté et la performance sont clés pour la prédiction.
- Besoin possible d'améliorer les données ou d'ajouter de nouvelles features pour améliorer la précision globale.

Recommandation pour les RH

- Utiliser le modèle Logistic Regression pour détecter les départs à risque.
- Analyser les variables critiques pour guider les actions RH



Conclusion

Résumé des résultats :

- Modèles testés : Dummy Classifier, Logistic Regression, Random Forest optimisé
- Meilleur modèle : Logistic Regression
 - Accuracy : 0.762
 - F1-score : 0.485
 - Précision : 0.371
 - Rappel : 0.702 → priorité donnée à la détection des départs

Insights principaux :

- Les départs sont rares (16 % des cas), d'où l'importance de privilégier le rappel pour les identifier.
- Les variables les plus influentes pour prédire les départs :
 - note_evaluation_actuelle
 - satisfaction_employee_environnement
 - années_dans_l_entreprise
 - note_evaluation_precedente, niveau_hierarchique_poste, revenu_mensuel, etc.

Limitations et recommandations :

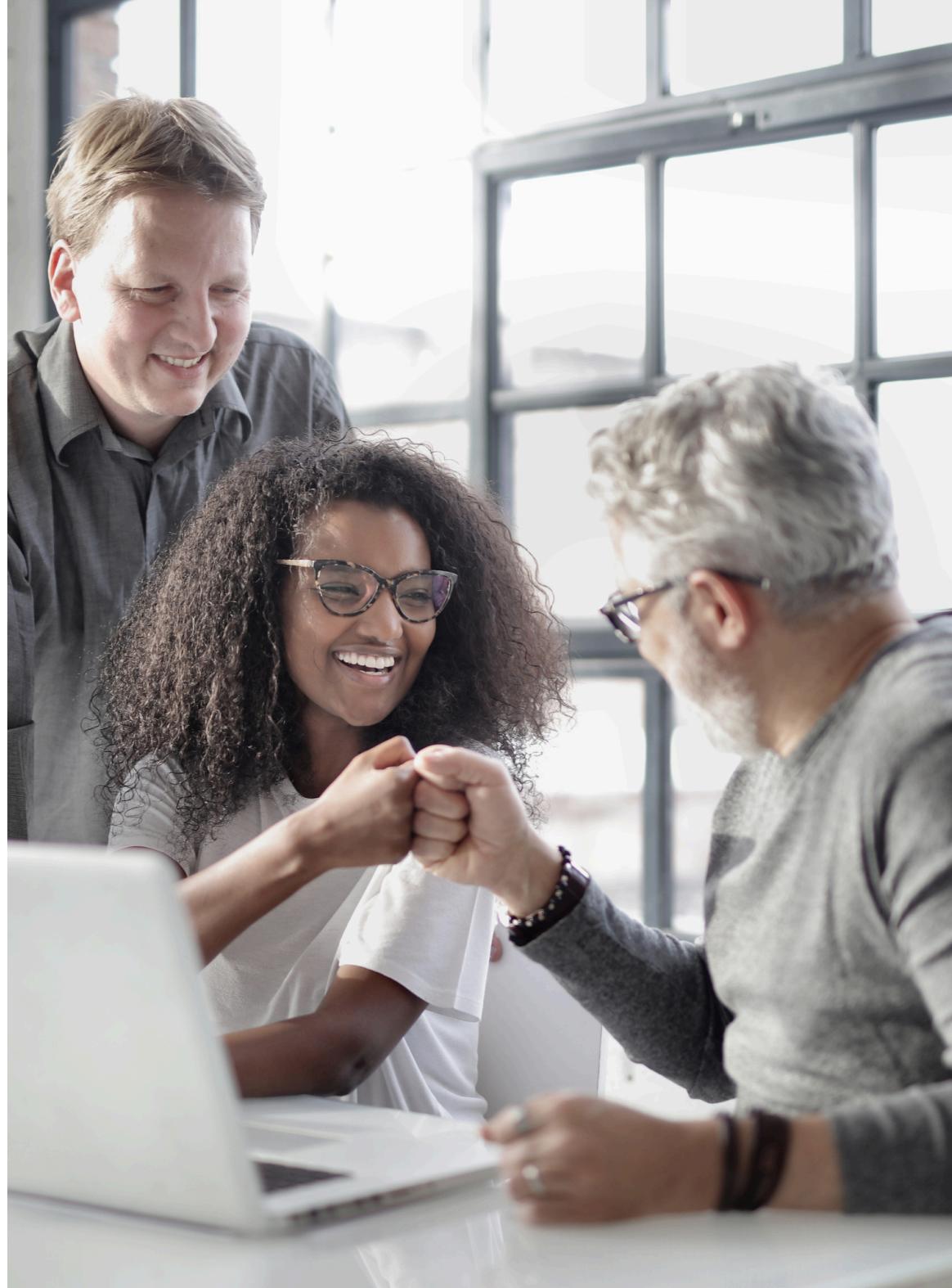
- Performance moyenne : amélioration possible avec davantage de features ou ajustement des hyperparamètres
- Possibilité d'explorer des modèles plus avancés ou du feature engineering pour mieux capturer les départs

Conclusion finale :

- Logistic Regression reste simple et efficace pour ce jeu de données
- Focus sur le rappel permet de mieux détecter les départs, ce qui est clé pour les actions RH



Améliorations futures



Amélioration des données :

- Collecter davantage de données pour réduire l'impact du déséquilibre des classes
- Ajouter des variables potentiellement prédictives : feedbacks qualitatifs, parcours de carrière, données de management, etc.
- Enrichir les variables existantes via feature engineering : interactions, ratios, tendances historiques

Optimisation des modèles :

- Tester des modèles plus avancés : Gradient Boosting, XGBoost, LightGBM
- Ajuster les hyperparamètres avec plus de finesse ou via Bayesian optimization
- Explorer des techniques de rééchantillonnage pour traiter le déséquilibre de classes (SMOTE, ADASYN)

Analyse et interprétabilité :

- Approfondir l'analyse SHAP pour mieux comprendre les facteurs clés
- Développer des dashboards interactifs pour suivre les prédictions et les insights RH

Déploiement et suivi :

- Mettre en place un pipeline de prédiction en production
- Suivi continu de la performance du modèle avec des données récentes
- Ajustement dynamique selon les changements dans l'entreprise



Merci pour
votre
attention !



<https://www.linkedin.com/in/huang-nicolas/>