

NOTE MÉTHODOLOGIQUE

NOTE MÉTHODOLOGIQUE

1) Dataset retenu (1 page max)

1.1 Pourquoi ce dataset

On utilise ce dataset car il relie directement des facteurs de **sommeil, stress et bien-être** à des variables liées à la **réussite scolaire**. Il est adapté à une approche “prévention” : repérer les profils plus vulnérables pour aider plus tôt.

1.2 Quelles données / quelles variables et pourquoi

Exemple de variables (selon votre fichier final.csv) :

- **Sleep_Duration** : la durée de sommeil est un indicateur central (fatigue, concentration).
- **Stress_Level_re** : le stress influence l'attention, la motivation, le sommeil.
- **WellBeing_Score** : reflète l'état global (utile comme cible ou comme feature selon le choix).
- **Physical_Activity** : peut améliorer sommeil et stress (facteur protecteur).
- **Academic_Performance** : représente le résultat scolaire (peut être la cible ou une variable d'intérêt).
- **Gender** : variable catégorielle parfois corrélée à des différences de sommeil/stress (à manier avec prudence).

1.3 Prétraitement (pourquoi et quoi)

Avant de modéliser, on crée un fichier propre **final.csv** :

- suppression des doublons,
- conversion des colonnes numériques,
- gestion des valeurs manquantes,
- vérification de cohérence (ex : sommeil dans des bornes réalistes).

But : que tous les tests soient reproductibles et comparables.

1.4 Cible (y) : ce qu'on prédit

Deux possibilités (tu en choisis une dans ton rapport) :

Option A: classification binaire “à risque / non à risque”

Exemple : “à risque” = les étudiants avec un bien-être très bas (ex : les 25% plus faibles) ou performance faible.

Option B : régression (score) puis seuil

On prédit un score (bien-être ou performance), puis on transforme en “à risque”.

Pour une démarche prévention, la classification binaire est souvent plus simple à expliquer.

Code (dataset + choix features)

```
import pandas as pd
```

```
df = pd.read_csv("final.csv", sep=";")
```

```
features = ["Gender", "Sleep_Duration", "Physical_Activity", "Stress_Level_re",  
"WellBeing_Score"]
```

```
# cible exemple (a adapter)
```

```
# df["At_Risk"] = (df["WellBeing_Score"] <=  
df["WellBeing_Score"].quantile(0.25)).astype(int)
```

```
target = "At_Risk"
```

2) Concepts de l’algorithme récent : CatBoost (2 pages max)

2.1 Fonctionnement (simple)

CatBoost est un modèle de **Gradient Boosting** :

- il crée un premier arbre de décision,
- puis il ajoute des arbres successifs qui corrigent les erreurs des précédents,
- au final, la prédiction est une “somme” de nombreux petits arbres.

C'est puissant car ça capte des relations **non linéaires** (ex : “stress haut + sommeil bas” → risque élevé).

2.2 Pourquoi CatBoost est “récent” et intéressant

CatBoost apporte deux points importants :

1. **Gestion des variables catégorielles** (comme Gender) avec des techniques dédiées.

2. **Ordered boosting** : une stratégie conçue pour réduire certaines formes de fuite d'information (“prediction shift”), ce qui rend l'apprentissage plus robuste.

Donc CatBoost est souvent très performant sur les données tabulaires, sans pipeline complexe.

Référence : Prokhorenkova et al., 2017.

3) Modélisation (2 pages max) — baseline + métriques + optimisation

3.1 Pourquoi une baseline

Une baseline est un **modèle simple** utilisé comme référence.

Si CatBoost ne fait pas mieux que la baseline, alors CatBoost n'est pas justifié (on préfère le modèle le plus simple).

Baseline recommandée :

- **Régression logistique** (classification) : simple, rapide, interprétable.

3.2 Méthode de validation

- séparation **train/test** (ex : 80/20)
- éventuellement **cross-validation** (si vous avez le temps) pour fiabiliser l'estimation

3.3 Métriques (avec explication claire)

On utilise plusieurs métriques car aucune ne résume tout.

- **Recall (sensibilité)** : “Parmi les vrais à risque, combien j'en détecte ?”
Important en prévention : on veut éviter de rater les étudiants à risque.
- **Precision** : “Parmi ceux que je dis ‘à risque’, combien le sont vraiment ?”
Important pour éviter trop de fausses alertes.
- **F1-score** : compromis entre recall et precision.
- **Matrice de confusion** : tableau (TP/FP/TN/FN) facile à expliquer.
- **ROC-AUC** (optionnel) : capacité globale à séparer les classes.

3.4 Optimisation (tuning)

On optimise CatBoost (et éventuellement le seuil de décision) :

- profondeur des arbres,
- learning rate,

- nb d'arbres,
- régularisation,
- pondération des classes si déséquilibre.

Code (baseline + CatBoost + métriques)

```

from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.linear_model import LogisticRegression


X = df[features].copy()

y = df[target].copy()

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

cat_cols = ["Gender"]

num_cols = [c for c in features if c not in cat_cols]

preprocess = ColumnTransformer(
    transformers=[
        ("cat", OneHotEncoder(handle_unknown="ignore"), cat_cols),
        ("num", "passthrough", num_cols),
    ]
)

```

```
baseline = Pipeline(steps=[  
    ("preprocess", preprocess),  
    ("model", LogisticRegression(max_iter=2000))  
])  
  
baseline.fit(X_train, y_train)  
pred_base = baseline.predict(X_test)  
proba_base = baseline.predict_proba(X_test)[:, 1]  
  
print("Baseline - matrice de confusion")  
print(confusion_matrix(y_test, pred_base))  
print(classification_report(y_test, pred_base))  
  
print("Baseline - ROC AUC (si possible)")  
print(roc_auc_score(y_test, proba_base))  
CatBoost (si vous l'utilisez côté ML) :  
from catboost import CatBoostClassifier  
  
# CatBoost peut gérer les catégories directement  
# Ici on convertit Gender en 'category' si besoin  
X_train_cb = X_train.copy()  
X_test_cb = X_test.copy()  
X_train_cb["Gender"] = X_train_cb["Gender"].astype("category")  
X_test_cb["Gender"] = X_test_cb["Gender"].astype("category")  
  
cat_features = ["Gender"]  
  
cb = CatBoostClassifier()
```

```

iterations=500,
depth=6,
learning_rate=0.05,
loss_function="Logloss",
eval_metric="Recall",
verbose=False,
random_seed=42

)

cb.fit(X_train_cb, y_train, cat_features=cat_features)
pred_cb = cb.predict(X_test_cb)
proba_cb = cb.predict_proba(X_test_cb)[:, 1]

print("CatBoost - matrice de confusion")
print(confusion_matrix(y_test, pred_cb))
print(classification_report(y_test, pred_cb))
print("CatBoost - ROC AUC")
print(roc_auc_score(y_test, proba_cb))

```

4) Synthèse des résultats (2 pages max) — comparaison + conclusion

4.1 Comment présenter les résultats

Tu fais un tableau comparatif (à remplir avec vos vrais scores) :

- Baseline (LogReg) : recall, precision, F1, AUC
- CatBoost : recall, precision, F1, AUC

Tu ajoutes une matrice de confusion pour chacun.

4.2 Comment conclure (phrases prêtées)

Cas 1 : CatBoost meilleur en recall (souvent recherché en prévention)

“CatBoost améliore le recall par rapport à la baseline, ce qui réduit le risque de rater des étudiants réellement en difficulté. Cela rend le modèle plus adapté à un contexte de prévention.”

Cas 2 : CatBoost augmente recall mais baisse precision

“Le modèle détecte plus d’étudiants à risque (recall ↑), au prix de davantage de fausses alertes (precision ↓). Ce compromis est acceptable si l’objectif est l’accompagnement préventif.”

Cas 3 : CatBoost n’apporte pas mieux

“Le nouvel algorithme n’améliore pas significativement la performance par rapport à la baseline. On privilégie alors le modèle plus simple, plus robuste et plus facile à interpréter.”

5) Feature importance globale et locale — pourquoi et comment

Pourquoi on le fait

Parce qu’un modèle “boîte noire” n’est pas acceptable dans le réel :

- les équipes veulent savoir **quels facteurs comptent**,
- et pourquoi un étudiant a été classé à risque.

5.1 Importance globale (sur tous les étudiants)

Objectif : “quelles variables influencent le modèle le plus ?”

On peut obtenir :

- l’importance interne de CatBoost (gain),
- ou mieux : SHAP global (plus explicable).

5.2 Importance locale (pour une prédiction)

Objectif : “pour cet étudiant, qu’est-ce qui a fait monter/descendre le risque ?”

Méthode recommandée : **SHAP**, qui explique les contributions des features pour une prédiction.

Code (SHAP global + local)

```
import shap
```

```
import numpy as np
```

```
explainer = shap.TreeExplainer(cb)
```

```
shap_values = explainer.shap_values(X_test_cb)

# Global: importance moyenne
shap.summary_plot(shap_values, X_test_cb)

# Local: expliquer 1 prediction
i = 0 # index d'un etudiant dans X_test_cb
shap.force_plot(explainer.expected_value, shap_values[i], X_test_cb.iloc[i],
matplotlib=True)
```

6) Limites et améliorations possibles

Limites (phrases claires)

- Dataset limité (peu de variables) : certaines causes réelles de décrochage ne sont pas présentes.
- Les données peuvent être auto-déclarées : risque de biais.
- Corrélation ≠ causalité : le modèle repère des associations, pas des causes.
- Si la cible “à risque” est construite par seuil, le résultat dépend du seuil choisi.

Améliorations

- Ajouter des variables (absences, charge de travail, historique).
- Validation plus robuste : cross-validation, calibration.
- Optimiser le seuil : maximiser recall en gardant une precision minimale.
- Explicabilité : SHAP systématique + règles simples pour aider la décision.