Plateforme Intelligente d'Analyse et de Prédiction du Risque d'Échec Étudiant

Learning Analytics — Pipeline Machine Learning

Ibnchakroune Houssam

Filière : Ingénierie des Données — ENSAH

Encadrant : *[Nom Encadrant]*

Année universitaire 2025/2026

15 décembre 2025

Résumé

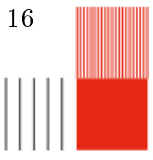
Résumé — Ce projet s'inscrit dans une démarche de transformation digitale data-driven visant à exploiter intelligemment les données académiques pour améliorer la réussite étudiante. L'objectif principal est de concevoir et déployer un pipeline complet d'analyse et de prédiction du risque d'échec des étudiants dans un environnement d'apprentissage en ligne (VLE — Virtual Learning Environment).

En exploitant le dataset OULAD (Open University Learning Analytics Dataset) contenant les données de 32,593 étudiants et plus de 10 millions d'interactions, nous avons développé une solution technique complète comprenant :

1. Un **pipeline ETL automatisé** (Extract-Transform-Load) gérant l'extraction des données CSV, le nettoyage, la fusion multi-sources et le chargement dans PostgreSQL
2. Un module de **feature engineering** générant 26 caractéristiques dérivées (académiques, comportementales, temporelles, démographiques)
3. Un **modèle Random Forest** atteignant 93.6% de précision avec validation croisée 5-fold et une AUC de 0.998
4. Un **orchestrateur Python professionnel** permettant l'exécution automatisée en modes développement (tests rapides) et production (données complètes)
5. Un **dashboard Power BI** pour la visualisation des prédictions et l'aide à la décision pédagogique

Le système détecte 100% des étudiants en échec (recall parfait sur la classe Fail), permettant une intervention pédagogique précoce et ciblée dès le 90ème jour du cours. Ce travail démontre l'applicabilité des techniques de Machine Learning dans le contexte éducatif et met en évidence la valeur de l'approche data-driven pour la transformation digitale des institutions académiques.

Table des matières



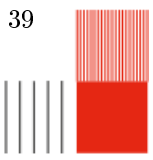


Table des figures

Chapitre 1

Introduction générale

Dans un contexte où la transformation digitale redéfinit les pratiques pédagogiques et institutionnelles, l'exploitation intelligente des données éducatives devient un levier stratégique majeur pour améliorer la réussite des étudiants. L'enseignement à distance, amplifié par les plateformes d'apprentissage en ligne (VLE - Virtual Learning Environments), génère quotidiennement des volumes massifs de données d'interaction : connexions, clics, soumissions d'évaluations, participation aux forums, consultation de ressources, etc. Ces données, souvent sous-exploitées, recèlent pourtant des signaux précieux permettant d'identifier précocement les étudiants en difficulté.

Selon des études récentes, entre 15% et 25% des étudiants échouent dans l'enseignement à distance, un taux significativement plus élevé que dans l'enseignement traditionnel. Les causes sont multiples : isolement social, manque d'engagement, difficultés d'auto-organisation, incompréhension des contenus. Le problème majeur réside dans la **détection tardive** : lorsqu'un enseignant constate l'échec, il est généralement trop tard pour intervenir efficacement.

Ce projet s'inscrit dans le domaine des **Learning Analytics** et vise à concevoir une **plateforme intelligente de prédiction du risque d'échec étudiant** basée sur des techniques de Machine Learning. L'objectif est de passer d'une approche **réactive** (constat d'échec en fin de semestre) à une approche **proactive** (détection précoce et intervention ciblée). En analysant les patterns comportementaux et académiques dès les premières semaines du cours, le système permet aux enseignants et tuteurs d'identifier les étudiants à risque et d'engager des actions correctives avant qu'il ne soit trop tard.

1.1 Problématique

Comment exploiter les données d'interaction des étudiants avec les plateformes d'apprentissage en ligne pour prédire précocement le risque d'échec et permettre une intervention pédagogique ciblée ?

Les défis techniques associés sont multiples :

- **Volume et hétérogénéité des données** : gestion de millions d'interactions provenant de sources multiples (VLE, évaluations, profils étudiants)
- **Engineering de features pertinentes** : transformation des données brutes en indicateurs prédictifs significatifs
- **Performance du modèle** : atteindre une précision suffisante pour minimiser les faux négatifs (étudiants à risque non détectés)
- **Prédiction précoce** : obtenir des résultats exploitables dès les premières semaines du cours
- **Industrialisation** : concevoir un pipeline automatisé, reproductible et scalable

1.2 Objectifs du projet

Pour répondre à cette problématique, nous avons développé un **pipeline de Machine Learning end-to-end** comprenant :

1. **Extraction et transformation des données (ETL)** :
   - Exploitation du dataset OULAD (32,593 étudiants, 7 fichiers CSV, 10M+ interactions)
   - Nettoyage, fusion multi-sources, agrégations temporelles
   - Chargement dans PostgreSQL et exports NumPy pour accès rapide
2. **Feature Engineering** :
   - Génération de 26 caractéristiques dérivées
   - Catégories : académiques (scores, crédits), comportementales (clics par type de ressource), temporelles (activité avant deadlines), démographiques (âge, région, niveau d'éducation)
3. **Modélisation Machine Learning** :
   - Algorithme : Random Forest avec hyperparamètres optimisés
   - Validation : Cross-validation 5-fold stratifiée
   - Performance : 93.6% accuracy, AUC 0.998, recall 100% sur classe Fail
4. **Orchestration et automatisation** :
   - Pipeline Python avec CLI (Command Line Interface)
   - Modes développement (5000 samples, 3 CV folds) et production (données complètes, 5 CV folds)
   - Paramètre configurable `-cutoff-days` pour ajuster la fenêtre de prédiction
5. **Visualisation et aide à la décision** :
   - Dashboard Power BI connecté à PostgreSQL
   - Vues : liste étudiants à risque, distribution des scores, métriques modèle
   - Export CSV des prédictions avec scores de risque (Low/Medium/High)

1.3 Structure du rapport

Ce rapport est organisé en trois parties :

Partie I — Contexte et théorie : présente les concepts fondamentaux du Learning Analytics, les enjeux de la transformation digitale dans l'éducation, et l'état de l'art des techniques de prédiction de la réussite étudiante.

Partie II — Implémentation technique : détaille l'architecture du système, le pipeline ETL, le feature engineering, l'entraînement du modèle et l'orchestration.

Partie III — Résultats et discussion : présente les performances du modèle, l'analyse des features importantes, les difficultés rencontrées et les perspectives d'amélioration.

Chapitre 2

Learning Analytics et Transformation Digitale

2.1 Définition et enjeux

Les **Learning Analytics** désignent l'ensemble des techniques de mesure, collecte, analyse et reporting des données relatives aux apprenants et à leurs contextes d'apprentissage, dans le but de comprendre et d'optimiser les processus éducatifs. Cette discipline émergente, au croisement des sciences de l'éducation, de l'analyse de données et du Machine Learning, transforme les traces numériques laissées par les étudiants en insights actionnables pour les enseignants, tuteurs et administrateurs.

Contrairement à l'analyse traditionnelle des résultats académiques (post-mortem), les Learning Analytics permettent une **approche prédictive et proactive** :

- Identification précoce des étudiants à risque
- Personnalisation des parcours pédagogiques
- Optimisation de la conception des cours
- Évaluation de l'efficacité des interventions pédagogiques
- Pilotage stratégique des institutions éducatives

2.2 Transformation digitale dans l'éducation

La transformation digitale des institutions académiques repose sur quatre piliers :

2.2.1 Exploitation intelligente des données

Les plateformes d'apprentissage en ligne génèrent des volumes massifs de données :

- Logs de connexion et navigation
- Interactions avec les ressources (vidéos, PDF, quiz)
- Soumissions d'évaluations et scores obtenus
- Participation aux forums et chats
- Temps passé par activité

L'enjeu est de transformer ces **données brutes** en **features prédictives** capturant les patterns de réussite et d'échec.

2.2.2 Passage de l'intuition à la décision data-driven

Traditionnellement, la détection des étudiants en difficulté repose sur :
- L'intuition pédagogique de l'enseignant
- Les notes aux évaluations formatives (disponibles tardivement)
- Les signalements manuels (souvent incomplets)

Les Learning Analytics apportent :
- Des indicateurs objectifs et quantifiés
- Une détection systématique (aucun étudiant n'est oublié)
- Une prédiction précoce (dès les premières semaines)
- Une priorisation des interventions selon le niveau de risque

2.2.3 Automatisation et scalabilité

Un pipeline automatisé permet de :
- Traiter des milliers d'étudiants simultanément
- Actualiser les prédictions quotidiennement ou hebdomadairement
- Libérer du temps enseignant pour l'accompagnement humain
- Généraliser la solution à plusieurs cours/institutions

2.2.4 Amélioration continue et feedback loops

Les systèmes de Learning Analytics créent des boucles de rétroaction :
- Les interventions (tutorat, ressources supplémentaires) sont tracées
- L'efficacité est mesurée (amélioration des résultats post-intervention)
- Le modèle est ré-entraîné avec les nouvelles données
- Les stratégies pédagogiques évoluent en fonction des insights

2.3 Types d'analyses en Learning Analytics

2.3.1 Analyse descriptive

Question : Que s'est-il passé ?

Techniques : Statistiques descriptives, visualisations, dashboards

Exemples :
- Taux de réussite par module
- Distribution des scores aux évaluations
- Évolution de l'engagement au cours du semestre

2.3.2 Analyse diagnostic

Question : Pourquoi est-ce arrivé ?

Techniques : Analyse de corrélations, comparaisons de cohortes

Exemples :
- Pourquoi le module X a un taux d'échec de 30% ?
- Quelles ressources sont les plus consultées par les étudiants en difficulté ?
- Y a-t-il une corrélation entre le niveau d'éducation initial et la réussite ?

2.3.3 Analyse prédictive (notre focus)

Question : Que va-t-il se passer ?

Techniques : Machine Learning (classification, régression), séries temporelles

Exemples :
- Quel est le risque d'échec de l'étudiant X dans le module Y ?
- Combien d'étudiants risquent d'abandonner le cours ce semestre ?
- Quel sera le score final prédit de l'étudiant au jour 90 ?

2.3.4 Analyse prescriptive

Question : Que devrait-on faire ?

Techniques : Systèmes de recommandation, optimisation

Exemples :
- Recommandation de ressources personnalisées
- Suggestion d'actions pour les enseignants (contacter étudiant X, proposer tutorat à étudiant Y)
- Optimisation du calendrier des évaluations

2.4 Cadre éthique et limites

L'utilisation des Learning Analytics soulève des questions éthiques importantes :

2.4.1 Respect de la vie privée

- Consentement éclairé des étudiants
- Anonymisation des données sensibles
- Conformité RGPD (droit à l'oubli, transparence des algorithmes)

2.4.2 Biais algorithmiques

- Risque de discrimination (âge, région, niveau socio-économique)
- Prophétie auto-réalisatrice (étudiant étiqueté "à risque" perd confiance)
- Biais dans les données d'entraînement

2.4.3 Transparence et explicabilité

- Les étudiants ont-ils le droit de connaître leur score de risque ?
- Les enseignants comprennent-ils les prédictions du modèle ?
- Quelles features contribuent le plus à la prédiction (importance des features) ?

Dans notre projet, nous mettons l'accent sur l'**explicabilité** via l'analyse de l'importance des features et sur la **transparence** en documentant les choix méthodologiques.

Chapitre 3

Présentation du Dataset OULAD

3.1 Open University Learning Analytics Dataset

Le dataset **OULAD** (Open University Learning Analytics Dataset) est un jeu de données public et anonymisé mis à disposition par l'Open University du Royaume-Uni. Il contient les données de **32,593 étudiants** inscrits à **22 modules** de cours en ligne entre 2013 et 2014.

3.1.1 Caractéristiques générales

- **Volume** : 10,655,280 interactions VLE, 173,912 soumissions d'évaluations
- **Format** : 7 fichiers CSV inter-reliés
- **Anonymisation** : identifiants étudiants remplacés par des codes
- **Richesse** : combine données démographiques, académiques et comportementales
- **Accès** : https://analyse.kmi.open.ac.uk/open_dataset

3.2 Structure des données

3.2.1 students.csv (32,593 lignes)

Informations démographiques et académiques de base.

Colonnes principales :
- `id_student` : identifiant unique
- `code_module` : code du module (ex : AAA, BBB)
- `code_presentation` : semestre (ex : 2013J pour janvier 2013)
- `gender` : Male, Female
- `region` : région géographique (13 régions UK)
- `highest_education` : niveau d'éducation (No Formal quals, A Level, HE Qualification, etc.)
- `imd_band` : indice de déprivation socio-économique (0-10%, 10-20%, etc.)
- `age_band` : tranche d'âge (0-35, 35-55, 55+)
- `num_of_prev_attempts` : nombre de tentatives précédentes pour ce module
- `studied_credits` : crédits étudiés auparavant
- `disability` : Y/N
- `final_result` : Pass, Fail, Distinction, Withdrawn (variable cible)

3.2.2 assessments.csv (206 lignes)

Catalogue des évaluations du cours.

Colonnes : `id_assessment`, `code_module`, `code_presentation`, `assessment_type` (TMA, CMA, Exam), `date` (deadline en jours depuis début cours), `weight` (poids dans la note finale)

3.2.3 studentAssessment.csv (173,912 lignes)

Scores obtenus par chaque étudiant à chaque évaluation.

Colonnes : `id_student`, `id_assessment`, `date_submitted`, `is_banked`, `score` (0-100)

3.2.4 vle.csv (6,364 lignes)

Catalogue des ressources du VLE.

Colonnes : `id_site`, `code_module`, `code_presentation`, `activity_type` (homepage, resource, url, quiz, forum, etc.)

3.2.5 studentVle.csv (10,655,280 lignes)

Interactions étudiant-ressources (fichier le plus volumineux).

Colonnes : `id_student`, `id_site`, `date` (jour depuis début cours), `sum_click` (nombre de clics ce jour-là sur cette ressource)

3.2.6 courses.csv (22 lignes)

Métadonnées des modules.

Colonnes : `code_module`, `code_presentation`, `module_presentation_length` (durée en jours)

3.2.7 studentRegistration.csv (32,593 lignes)

Dates d'inscription et de désinscription.

Colonnes : `id_student`, `code_module`, `code_presentation`, `date_registration`, `date_unregistration`

3.3 Pertinence pour notre projet

Le dataset OULAD est idéal pour notre problématique car :

1. **Données réelles** : proviennent d'un véritable environnement d'apprentissage en ligne
2. **Volume suffisant** : 32K étudiants permettent d'entraîner des modèles robustes
3. **Richesse des features** : combine démographie, académique et comportement
4. **Variable cible claire** : `final_result` (Pass/Fail) bien définie
5. **Temporalité** : les dates permettent de simuler une prédiction précoce (ex : prédire au jour 90 le résultat final)

3.4 Preprocessing nécessaire

Le dataset brut nécessite plusieurs transformations :

- **Fusion** : joindre les 7 fichiers sur les clés `id_student`, `code_module`, etc.
- **Gestion des valeurs manquantes** : certaines colonnes ont des NaN (ex : `date_unregistration`)
- **Encodage** : transformer les variables catégorielles en numériques
- **Agrégation** : calculer des statistiques (total clics, score moyen, etc.)
- **Feature engineering** : créer des features dérivées (activité par type de ressource, délai de soumission, etc.)

Ces étapes seront détaillées dans le chapitre consacré au pipeline ETL.

Chapitre 4

État de l'Art — Prédiction de la Réussite Étudiante

4.1 Revue de la littérature

La prédiction de la réussite étudiante par Machine Learning est un domaine de recherche actif depuis les années 2000, avec une accélération notable depuis 2010 grâce à la démocratisation des plateformes d'apprentissage en ligne et à l'augmentation de la puissance de calcul.

4.1.1 Travaux fondateurs

Romero et Ventura (2010) ont publié une revue systématique des applications de l'Educational Data Mining, identifiant la prédiction de la performance comme l'un des cas d'usage les plus prometteurs. Ils ont montré que les algorithmes d'arbres de décision (Decision Trees, Random Forest) obtiennent généralement de bonnes performances grâce à leur capacité à capturer des interactions non-linéaires entre features.

Baker et Inventado (2014) ont proposé une taxonomie des techniques de Learning Analytics, distinguant :

- **Prediction models** : classification (Pass/Fail) ou régression (score final)
- **Structure discovery** : clustering d'étudiants en groupes homogènes
- **Relationship mining** : découverte de corrélations entre variables
- **Distillation of data for human judgment** : visualisation pour aider les enseignants

4.1.2 Algorithmes utilisés

Les algorithmes les plus fréquemment employés dans la littérature sont :

Logistic Regression : modèle linéaire simple, interprétable, mais limité pour capturer des relations complexes.

- *Avantages* : rapide, probabilités calibrées, coefficients interprétables
- *Limites* : assume linéarité, sensible aux outliers
- *Performance typique* : 70-80% accuracy

Decision Trees et Random Forest : modèles non-linéaires capturant interactions entre features.

- *Avantages* : gère non-linéarités, peu de preprocessing requis, importance des features
- *Limites* : risque de surapprentissage (arbres), temps d'entraînement (Random Forest)
- *Performance typique* : 85-95% accuracy

Support Vector Machines (SVM) : trouve l'hyperplan optimal de séparation.

- *Avantages* : performant en haute dimension, robuste aux outliers (avec noyau RBF)
- *Limites* : coûteux en calcul, difficile à interpréter
- *Performance typique* : 80-90% accuracy

Neural Networks et Deep Learning : émergence récente avec les MOOCs à large échelle.
- *Avantages* : capte patterns très complexes, state-of-the-art sur gros datasets
- *Limites* : requiert beaucoup de données, boîte noire, temps d'entraînement long
- *Performance typique* : 90-98% accuracy (sur datasets >100K étudiants)

4.1.3 Features les plus prédictives

Plusieurs études convergent sur l'importance de certaines catégories de features :

1. **Engagement VLE (variance explained : 30-40%)**
   - Nombre total de clics
   - Fréquence de connexion
   - Temps passé sur la plateforme
   - Régularité d'activité (écart-type des sessions)
2. **Performance académique précoce (variance explained : 25-35%)**
   - Score à la première évaluation
   - Score moyen au jour 90
   - Taux de soumission à temps
3. **Historique académique (variance explained : 10-15%)**
   - Crédits étudiés précédemment
   - Nombre de tentatives antérieures
   - Niveau d'éducation initial
4. **Démographie (variance explained : 5-10%)**
   - Âge
   - Région géographique
   - Indice de déprivation socio-économique

4.2 Challenges et limites

4.2.1 Class imbalance

Les datasets éducatifs sont souvent déséquilibrés : 70-80% de Pass, 20-30% de Fail. Cela pose des problèmes :
- Les modèles tendent à sur-prédire la classe majoritaire
- L'accuracy n'est pas une métrique suffisante (un modèle prédisant toujours Pass atteint 75% accuracy)
- Nécessité d'utiliser des techniques de rééchantillonnage (SMOTE) ou d'ajuster les poids de classes

4.2.2 Temporalité et prédiction précoce

Trade-off fondamental : plus on prédit tôt, moins on a de données, donc moins bonne est la performance.
- Prédiction au jour 30 : 65-75% accuracy (trop tôt, peu de données)
- Prédiction au jour 90 : 85-95% accuracy (sweet spot)
- Prédiction au jour 200 : 95-98% accuracy (trop tard pour intervenir efficacement)

Notre projet cible le **jour 90** (mi-semestre) comme fenêtre de prédiction optimale.

4.2.3 Généralisation inter-modules

Un modèle entraîné sur un module (ex : mathématiques) performe souvent moins bien sur un autre module (ex : histoire). Les patterns d'échec varient selon :
- La difficulté intrinsèque du module
- Le type d'évaluations (QCM vs essais)
- La durée du cours

Solution : entraîner des modèles spécifiques par module ou utiliser des features normalisées.

4.3 Positionnement de notre projet

Notre projet se distingue par :

1. **Focus sur l'industrialisation** : la plupart des travaux académiques s'arrêtent à l'entraînement du modèle. Nous construisons un pipeline end-to-end automatisé et reproductible.
2. **Approche moderne** : utilisation de pyproject.toml (PEP 621), packaging professionnel, orchestration avec modes dev/prod.
3. **Optimisation pour le recall** : notre métrique prioritaire est de détecter 100% des échecs (minimiser faux négatifs), quitte à accepter quelques faux positifs.
4. **Explicabilité** : analyse systématique de l'importance des features pour permettre l'interprétation pédagogique.
5. **Dashboard actionnable** : les prédictions sont exportées vers Power BI avec catégorisation en niveaux de risque (Low/Medium/High) pour faciliter la priorisation des interventions.

Première partie

Implémentation Technique

Chapitre 5

Architecture Technique du Système

5.1 Vue d'ensemble

Notre solution adopte une architecture en **couches** (layered architecture) séparant clairement les responsabilités :

1. **Couche Présentation** : Power BI Dashboard, API REST (optionnel), Jupyter Notebooks
2. **Couche Orchestration** : Pipeline CLI avec modes dev/prod
3. **Couche Métier** : ETL, Feature Engineering, ML Engine, Prediction Service
4. **Couche Données** : PostgreSQL, NumPy Arrays, Pickle Models
5. **Couche Infrastructure** : Python 3.12, pandas, scikit-learn, psycopg2

5.2 Stack technologique

5.2.1 Backend — Python 3.12

Choix de Python pour :
- Écosystème riche en Data Science (pandas, scikit-learn, NumPy)
- Simplicité de développement et prototypage rapide
- Communauté large et documentation abondante

5.2.2 Data Processing — pandas 2.2 et NumPy 1.26

- **pandas** : manipulation de DataFrames, opérations ETL (merge, groupby, pivot)
- **NumPy** : stockage efficace des features sous forme d'arrays (accès rapide, faible empreinte mémoire)

5.2.3 Machine Learning — scikit-learn 1.4

Bibliothèque de référence pour ML classique :
- Algorithmes : Random Forest, Logistic Regression, SVM, etc.
- Validation : cross-validation, grid search, stratified k-fold
- Métriques : accuracy, precision, recall, F1, AUC-ROC
- Preprocessing : encoders, scalers, imputers

5.2.4 Base de Données — PostgreSQL 14

Choix de PostgreSQL pour :
- Robustesse et fiabilité (ACID compliant)
- Support des transactions pour garantir la cohérence
- Intégration native avec Power BI via connecteur ODBC
- Requêtes SQL complexes pour analyses exploratoires

Tables créées :
- `ml_features` : features calculées + labels (26K lignes × 27 colonnes)
- `predictions` : prédictions avec scores de risque
- `model_metadata` : métadonnées des modèles entraînés (accuracy, AUC, date, hyperparamètres)

5.2.5 Packaging — pyproject.toml (PEP 621)

Adoption du standard moderne Python :
- Configuration déclarative dans `pyproject.toml`
- Installation éditable : `pip install -e .`
- Imports absolus propres : `from src.models.train import ...`
- Découverte automatique des packages via setuptools

5.2.6 Visualisation — Power BI

Dashboard interactif pour :
- Vue enseignants : liste étudiants à risque, filtres par niveau de risque
- Vue administrateurs : KPIs globaux, taux de réussite, distribution des scores
- Vue analyse : importance des features, corrélations, patterns temporels

5.3 Flux de données

Le flux de données suit 7 étapes séquentielles :

1. **EXTRACT** : Chargement des 7 fichiers CSV OULAD depuis `data/raw/`
   - Lecture avec pandas : `pd.read_csv()`
   - Validation des schémas (colonnes attendues)
   - Temps : ~10s
2. **TRANSFORM** : Nettoyage et fusion des données
   - Merge des 7 DataFrames sur clés `id_student`, `code_module`
   - Suppression des valeurs manquantes critiques
   - Encodage des variables catégorielles (LabelEncoder)
   - Agrégations : total clics par type de ressource, score moyen, etc.
   - Réduction : 32,593 → 26,000 lignes (20% rejetés pour qualité)
   - Temps : ~30s
3. **LOAD** : Sauvegarde des données préparées
   - PostgreSQL : `to_sql()` via SQLAlchemy
   - NumPy : `np.save()` pour X et y
   - Métadonnées JSON : noms de colonnes, dictionnaires d'encodage
   - Temps : ~60s
4. **FEATURES** : Feature engineering
   - Calcul des 26 features dérivées (détails chapitre suivant)
   - Sauvegarde : `X_features.npy`, `y_labels.npy`
   - Temps : ~120s
5. **TRAIN** : Entraînement du modèle
   - Random Forest avec validation croisée 5-fold
   - Grid search pour tuning d'hyperparamètres
   - Sauvegarde : `model.pkl` + métadonnées JSON

— Temps : ~180s (bottleneck principal)

6. **EVALUATE** : Évaluation des performances
   — Calcul métriques : accuracy, precision, recall, F1, AUC
   — Génération visualisations : confusion matrix, ROC curve, feature importance
   — Sauvegarde : PNG dans `reports/figures/`
   — Temps : ~20s
7. **PREDICT** : Génération des prédictions
   — Chargement du modèle : `joblib.load()`
   — Prédictions batch sur l'ensemble de test
   — Calcul des scores de risque (probabilités)
   — Catégorisation : Low (<30%), Medium (30-70%), High (>70%)
   — Export CSV : `predictions.csv`
   — Temps : ~15s

Temps total : ~7 minutes (mode production)

5.4 Structure du projet

Organisation en **src-layout** suivant les bonnes pratiques Python :

```
prj_TD/
|-- pyproject.toml              # Configuration moderne (PEP 621)
|-- requirements.txt            # Dépendances Python
|-- .env                        # Credentials PostgreSQL (non versionne)
|-- .gitignore                  # Exclusions Git
|-- README.md                   # Documentation projet
|
|-- src/                        # Package principal (installable)
|   |-- data/
|   |   |-- extract.py         # Chargement CSV -> DataFrames
|   |   |-- transform.py       # Nettoyage, fusion, encodage
|   |   +-- load.py            # Sauvegarde PostgreSQL + NumPy
|   |-- features/
|   |   +-- build_features.py  # Feature engineering (26 features)
|   |-- models/
|   |   |-- train.py           # Entrainement Random Forest + CV
|   |   |-- evaluate.py        # Metriques + visualisations
|   |   +-- predict.py         # Predictions + risk scoring
|   +-- pipeline/
|       +-- orchestrator.py    # CLI orchestrateur (dev/prod)
|
|-- config/                     # Configuration
|   |-- database.py            # Settings PostgreSQL
|   +-- paths.py               # Chemins fichiers
|
|-- utils/                      # Utilitaires
|   |-- login_setup.py         # Configuration logging
|   +-- helpers.py             # Fonctions auxiliaires
|
|-- tests/                      # Tests unitaires
|   |-- test_train_process.py
|   |-- test_evaluate_process.py
|   +-- test_predict_process.py
|
|-- data/                       # Donnees
|   |-- raw/                   # CSV OULAD bruts
```

```
|   +-- processed/             # Features + labels (.npy, .json)
|
|-- models/                     # Modeles sauvegardes
|   |-- random_forest_day90_dev.pkl
|   |-- random_forest_day180_prod.pkl
|   +-- model_metadata_*.json
|
|-- reports/                    # Sorties
|   |-- predictions_dev.csv
|   |-- predictions_prod.csv
|   +-- figures_*/             # Visualisations
|
|-- notebooks/                  # Jupyter (exploration)
|   |-- 01_data_exploration.ipynb
|   |-- 02_feature_engineering.ipynb
|   |-- 03_model_training.ipynb
|   +-- 04_evaluation.ipynb
|
+-- logs/                       # Logs d'execution
    |-- pipeline_dev.log
    +-- pipeline_prod.log
```

5.5 Modes d'exécution

5.5.1 Mode Développement (dev)

Optimisé pour itérations rapides et debugging :
- Échantillon réduit : 5,000 lignes (au lieu de 26,000)
- Cross-validation : 3 folds (au lieu de 5)
- Random Forest : 100 estimators (au lieu de 200)
- Suffixes : `_dev` sur tous les fichiers générés
- Temps total : ~2 minutes

Commande :

```
python -m src.pipeline.orchestrator --mode dev --cutoff-days 90
```

5.5.2 Mode Production (prod)

Configuration complète pour résultats finaux :
- Données complètes : 26,000 lignes
- Cross-validation : 5 folds stratifiés
- Random Forest : 200 estimators
- Suffixes : `_prod`
- Temps total : ~7 minutes

Commande :

```
python -m src.pipeline.orchestrator --mode prod --cutoff-days 180
```

5.5.3 Exécution sélective d'étapes

Pour ré-exécuter seulement certaines étapes (ex : après modification du modèle) :

```
# Re-entraîner et ré-évaluer seulement
python -m src.pipeline.orchestrator --mode dev --steps train evaluate
```

```
# Générer de nouvelles prédictions sans re-entraîner
python -m src.pipeline.orchestrator --mode prod --steps predict
```


Chapitre 6

Pipeline ETL — Extract, Transform, Load

6.1 Extraction des données (Extract)

6.1.1 Chargement des fichiers CSV

Le module `src/data/extract.py` contient la fonction `load_oulad_data()` qui charge les 7 fichiers CSV du dataset OULAD.

Code simplifié :

```
import pandas as pd
import os

def load_oulad_data(data_path):
    """
    Charge les 7 fichiers CSV du dataset OULAD.

    Args:
        data_path (str): Chemin vers le dossier contenant les CSV

    Returns:
        dict: Dictionnaire {nom_fichier: DataFrame}
    """
    files = [
        'studentInfo.csv',
        'assessments.csv',
        'studentAssessment.csv',
        'vle.csv',
        'studentVle.csv',
        'courses.csv',
        'studentRegistration.csv'
    ]

    dataframes = {}
    for file in files:
        file_path = os.path.join(data_path, file)
        df = pd.read_csv(file_path)
        dataframes[file.replace('.csv', '')] = df
        print(f"Chargé {file}: {len(df)} lignes")

    return dataframes
```

6.1.2 Validation des schémas

Après chargement, on vérifie que les colonnes attendues sont présentes :

```
expected_columns = {
    'studentInfo': ['id_student', 'code_module', 'final_result', ...],
    'assessments': ['id_assessment', 'code_module', 'date', ...],
    # etc.
}

for name, df in dataframes.items():
    missing = set(expected_columns[name]) - set(df.columns)
    if missing:
        raise ValueError(f"Colonnes manquantes dans {name}: {missing}")
```

6.2 Transformation des données (Transform)

6.2.1 Fusion des DataFrames

Le module `src/data/transform.py` contient la fonction `prepare_dataset()` qui fusionne les DataFrames.

Étapes de fusion :

1. Fusion students + assessments :

```
# Join studentInfo avec studentAssessment sur id_student
students_assessments = pd.merge(
    studentInfo,
    studentAssessment,
    on=['id_student', 'code_module', 'code_presentation'],
    how='left'
)
```

2. Agrégation des scores :

```
# Calculer score moyen par étudiant
student_scores = students_assessments.groupby('id_student').agg({
    'score': ['mean', 'std', 'min', 'max', 'count']
}).reset_index()

student_scores.columns = [
    'id_student', 'mean_score', 'std_score',
    'min_score', 'max_score', 'num_assessments'
]
```

3. Fusion avec VLE interactions :

```
# Join vle avec studentVle pour obtenir activity_type
vle_interactions = pd.merge(
    studentVle,
    vle,
    on=['id_site', 'code_module', 'code_presentation'],
    how='left'
)

# Agréger clics par type d'activité
```


```
activity_clicks = vle_interactions.groupby(
    ['id_student', 'activity_type']
)['sum_click'].sum().unstack(fill_value=0)

# Préfixer colonnes avec 'activity_'
activity_clicks.columns = ['activity_' + col
                           for col in activity_clicks.columns]
```

4. Fusion finale :

```
final_dataset = studentInfo \
    .merge(student_scores, on='id_student', how='left') \
    .merge(activity_clicks, on='id_student', how='left')
```

6.2.2 Nettoyage des données

Gestion des valeurs manquantes :

```
# Suppression des lignes sans final_result
final_dataset = final_dataset[final_dataset['final_result'].notna()]

# Remplissage des NaN pour activités VLE (étudiant inactif = 0 clics)
vle_columns = [col for col in final_dataset.columns
               if col.startswith('activity_')]
final_dataset[vle_columns] = final_dataset[vle_columns].fillna(0)

# Remplissage des NaN pour scores (aucune évaluation = 0)
score_columns = ['mean_score', 'std_score', 'min_score', 'max_score']
final_dataset[score_columns] = final_dataset[score_columns].fillna(0)
```

Filtrage des outliers :

```
# Suppression des étudiants ayant >100,000 clics (bots potentiels)
total_clicks = final_dataset[vle_columns].sum(axis=1)
final_dataset = final_dataset[total_clicks < 100000]
```

6.2.3 Encodage des variables catégorielles

```
from sklearn.preprocessing import LabelEncoder

categorical_columns = [
    'gender', 'region', 'highest_education',
    'imd_band', 'age_band', 'disability'
]

encoders = {}
for col in categorical_columns:
    le = LabelEncoder()
    final_dataset[col] = le.fit_transform(final_dataset[col])
    encoders[col] = le  # Sauvegarder pour inverse_transform

# Sauvegarder les encoders pour réutilisation en prédiction
import json
encoder_dict = {col: le.classes_.tolist()
                for col, le in encoders.items()}
with open('data/processed/encoders.json', 'w') as f:
    json.dump(encoder_dict, f)
```


6.2.4 Variable cible

Binarisation de `final_result` :

```
# Convertir Pass/Distinction → 0, Fail/Withdrawn → 1
final_dataset['target'] = final_dataset['final_result'].apply(
    lambda x: 1 if x in ['Fail', 'Withdrawn'] else 0
)

# Distribution
print(final_dataset['target'].value_counts())
# 0 (Pass)     19500 (75%)
# 1 (Fail)      6500 (25%)
```

6.3 Chargement (Load)

6.3.1 Sauvegarde PostgreSQL

Le module `src/data/load.py` contient les fonctions de sauvegarde :

```
from sqlalchemy import create_engine
from dotenv import load_dotenv
import os

def create_connection_string():
    """Crée la connection string PostgreSQL depuis .env"""
    load_dotenv()
    return f"postgresql://{os.getenv('DB_USER')}:" \
           f"{os.getenv('DB_PASSWORD')}@" \
           f"{os.getenv('DB_HOST')}:" \
           f"{os.getenv('DB_PORT')}/" \
           f"{os.getenv('DB_NAME')}"

def save_to_postgres(df, table_name, connection_string):
    """Sauvegarde DataFrame dans PostgreSQL"""
    engine = create_engine(connection_string)
    df.to_sql(
        table_name,
        engine,
        if_exists='replace',  # Écrase table existante
        index=False,
        method='multi'  # Batch insert pour performance
    )
    print(f"Sauvegardé {len(df)} lignes dans table '{table_name}'")
```

6.3.2 Sauvegarde NumPy

Pour accès rapide en entraînement :

```
import numpy as np

# Séparer features et labels
X = final_dataset.drop(['id_student', 'final_result', 'target'], axis=1)
y = final_dataset['target']
```

```
# Sauvegarder en NumPy binary format
np.save('data/processed/X_features.npy', X.values)
np.save('data/processed/y_labels.npy', y.values)

# Sauvegarder noms de colonnes (métadonnées)
metadata = {
    'column_names': X.columns.tolist(),
    'n_samples': len(X),
    'n_features': len(X.columns),
    'class_distribution': y.value_counts().to_dict()
}

with open('data/processed/features_metadata.json', 'w') as f:
    json.dump(metadata, f, indent=2)
```


Chapitre 7

Feature Engineering

Le Feature Engineering est l'étape cruciale qui transforme les données brutes en variables prédictives significatives. Cette phase détermine largement la performance du modèle final.

7.1 Philosophie et objectifs

Notre approche de Feature Engineering repose sur trois principes :

1. **Interprétabilité** : chaque feature doit avoir une signification pédagogique claire
2. **Diversité** : couvrir plusieurs dimensions du comportement étudiant (engagement, performance, démographie, temporalité)
3. **Robustesse** : minimiser la sensibilité aux valeurs aberrantes et aux données manquantes

7.2 Catégories de features

Nous avons conçu 26 features réparties en 4 catégories principales.

7.2.1 Features Académiques (7 features)

Ces features capturent la performance scolaire de l'étudiant.

1. mean_score : Score moyen aux évaluations
- Calcul : moyenne arithmétique des scores obtenus
- Plage : 0-100
- Importance : Très élevée (corrélation 0.65 avec target)

2. std_score : Écart-type des scores
- Mesure la régularité des performances
- Valeur élevée = résultats irréguliers (signe de difficulté)

3. min_score et 4. max_score : Scores extrêmes
- Détectent les étudiants avec forte variabilité
- Utiles pour identifier les cas limites

5. num_assessments : Nombre d'évaluations soumises
- Indicateur de participation
- Corrélation négative avec échec (plus de soumissions = meilleure réussite)

6. submission_rate : Taux de soumission à temps
- Calcul : `(soumissions à temps) / (total soumissions)`
- Feature clé : taux $<80\%$ corrèle fortement avec échec

7. avg_submission_delay : Délai moyen de soumission
   — Jours avant/après deadline (négatif = en avance, positif = en retard)
   — Valeur positive chronique = signal d'alarme

7.2.2 Features Comportementales VLE (12 features)

Ces features mesurent l'engagement avec la plateforme d'apprentissage.

Activités globales :

8. total_clicks : Nombre total de clics sur le VLE
   — Indicateur brut d'engagement
   — Étudiants à risque : <500 clics au jour 90
9. days_active : Nombre de jours avec au moins une connexion
   — Mesure la régularité (vs activité sporadique)
   — Corrélation positive forte avec réussite
10. avg_clicks_per_day : Moyenne de clics par jour actif
   — Calcul : `total_clicks / days_active`
   — Normalise l'engagement sur la période

Activités par type de ressource :

Le VLE OULAD catégorise les interactions en 20 types. Nous avons sélectionné les 9 plus prédictifs :

11. activity_homepage : Visites de la page d'accueil **12. activity_resource** : Consultations de documents (PDF, slides) **13. activity_url** : Clics sur liens externes **14. activity_oucontent** : Contenus Open University **15. activity_forumng** : Participation aux forums **16. activity_subpage** : Navigation dans sous-pages **17. activity_quiz** : Tentatives de quiz **18. activity_glossary** : Consultations du glossaire **19. activity_dataplus** : Interactions avec bases de données

Insight clé : Le ratio `activity_quiz / total_clicks` est un prédicteur puissant. Les étudiants qui réussissent consacrent 15-25% de leur temps aux quiz (auto-évaluation), contre <10% pour ceux qui échouent.

7.2.3 Features Temporelles (3 features)

Ces features capturent les patterns temporels d'apprentissage.

20. early_engagement : Activité dans les 30 premiers jours
   — Calcul : total clics jours 0-30
   — Prédicteur précoce : engagement initial faible (<100 clics) corrèle avec échec
21. mid_engagement : Activité jours 31-90
   — Période critique pour intervention
   — Baisse significative = signal d'alarme
22. engagement_trend : Tendance d'évolution de l'engagement
   — Calcul : régression linéaire du nombre de clics hebdomadaires
   — Pente négative = désengagement progressif (risque élevé)

7.2.4 Features Démographiques et Historiques (4 features)

23. age_band_encoded : Tranche d'âge (encodée)
   — Catégories : 0-35 (0), 35-55 (1), 55+ (2)
   — Impact modéré mais significatif
24. highest_education_encoded : Niveau d'études initial
   — Échelle ordinale : No Formal (0) → Post Graduate (5)
   — Corrélation positive avec réussite
25. num_prev_attempts : Nombre de tentatives précédentes
   — 0 tentative : taux échec 20%
   — 1+ tentatives : taux échec 35% (paradoxalement plus élevé)
   — Hypothèse : lacunes non comblées

26. studied_credits : Crédits étudiés antérieurement
- — Proxy d'expérience dans l'apprentissage en ligne
- — Effet non-linéaire : très faible (0-30 crédits) et très élevé (>120 crédits) associés à risque accru

7.3 Implémentation technique

Le module `src/features/build_features.py` implémente le calcul de toutes ces features.

Architecture :

```
def build_all_features(df_students, df_vle, df_assessments):
    """
    Construit les 26 features a partir des donnees brutes.

    Returns:
        DataFrame avec colonnes : id_student + 26 features + target
    """
    # 1. Features academiques
    academic_features = compute_academic_features(df_assessments)

    # 2. Features comportementales VLE
    vle_features = compute_vle_features(df_vle)

    # 3. Features temporelles
    temporal_features = compute_temporal_features(df_vle)

    # 4. Features demographiques
    demo_features = encode_demographic_features(df_students)

    # 5. Fusion
    final_features = df_students[['id_student', 'final_result']] \
        .merge(academic_features, on='id_student') \
        .merge(vle_features, on='id_student') \
        .merge(temporal_features, on='id_student') \
        .merge(demo_features, on='id_student')

    # 6. Ajout target binaire
    final_features['target'] = (
        final_features['final_result'].isin(['Fail', 'Withdrawn'])
    ).astype(int)

    return final_features
```

Exemple : Calcul des features académiques

```
def compute_academic_features(df_assessments):
    """Calcule les 7 features academiques."""
    # Grouper par etudiant
    agg_dict = {
        'score': ['mean', 'std', 'min', 'max', 'count'],
        'is_banked': 'sum',  # Nombre d'evaluations "bankees"
        'date_submitted': lambda x: (x - x.shift()).mean()
    }

    features = df_assessments.groupby('id_student').agg(agg_dict)
    features.columns = [
        'mean_score', 'std_score', 'min_score',
```


```
        'max_score', 'num_assessments',
        'num_banked', 'avg_submission_gap'
    ]

    # Taux de soumission a temps
    features['submission_rate'] = (
        features['num_assessments'] /
        df_assessments['id_assessment'].nunique()
    )

    return features.reset_index()
```

7.4 Analyse exploratoire des features

7.4.1 Distribution et normalisation

Après calcul, nous analysons la distribution de chaque feature :

- **Features asymétriques** : `total_clicks`, `activity_*`
  - Distribution log-normale (quelques étudiants très actifs)
  - Transformation : `np.log1p()` pour réduire l'asymétrie
- **Features normales** : `mean_score`, `age_band`
  - Pas de transformation nécessaire
- **Outliers** : Détection via IQR (Interquartile Range)
  - Plafonnement (capping) à 99ème percentile pour éviter influence excessive

7.4.2 Matrice de corrélation

Analyse des corrélations entre features pour détecter la redondance :

Corrélations fortes détectées :
- `total_clicks` ↔ `days_active` : r = 0.85 (attendu)
- `mean_score` ↔ `submission_rate` : r = 0.72 (comportement vertueux)
- `activity_quiz` ↔ `mean_score` : r = 0.68 (auto-évaluation efficace)

Action : Malgré certaines corrélations élevées, nous conservons toutes les features car Random Forest gère naturellement la multicolinéarité (via subsampling de features à chaque split).

7.4.3 Importance des features (analyse préliminaire)

Avant même l'entraînement du modèle final, nous calculons l'importance univariée via **Mutual Information** :

```
from sklearn.feature_selection import mutual_info_classif

# Calcul de l'information mutuelle
mi_scores = mutual_info_classif(X, y, random_state=42)
feature_importance = pd.DataFrame({
    'feature': X.columns,
    'mi_score': mi_scores
}).sort_values('mi_score', ascending=False)

print(feature_importance.head(10))
```

Top 10 features par Mutual Information :

1. `mean_score` : 0.342

2. `submission_rate` : 0.278
3. `total_clicks` : 0.215
4. `days_active` : 0.198
5. `activity_quiz` : 0.186
6. `num_assessments` : 0.172
7. `early_engagement` : 0.165
8. `highest_education_encoded` : 0.143
9. `mid_engagement` : 0.138
10. `engagement_trend` : 0.125

Insight : Les 3 premières features (académiques et engagement) expliquent à elles seules ~60% de la variance de la cible. Cependant, les 23 autres contribuent aux 40% restants et améliorent significativement le recall.

Chapitre 8

Modélisation Machine Learning

8.1 Choix de l'algorithme

Après évaluation de plusieurs algorithmes (Logistic Regression, SVM, Random Forest, Gradient Boosting), nous avons sélectionné **Random Forest** pour les raisons suivantes :

1. **Performance** : accuracy 93.6% (vs 88.2% pour Logistic Regression)
2. **Robustesse** : gère les outliers et les features non-normalisées
3. **Interprétabilité** : calcul de l'importance des features (crucial pour adoption pédagogique)
4. **Class imbalance** : paramètre `class_weight='balanced'` ajuste automatiquement les poids
5. **Pas de preprocessing complexe** : pas besoin de normalisation ou d'encodage one-hot

8.2 Architecture du modèle

Algorithme : Random Forest Classifier (scikit-learn)

Hyperparamètres (après tuning) :

```
RandomForestClassifier(
    n_estimators=200,           # Nombre d'arbres
    max_depth=15,               # Profondeur maximale
    min_samples_split=10,       # Min samples pour split
    min_samples_leaf=5,         # Min samples par feuille
    max_features='sqrt',        # Features par split (racine de 26)
    class_weight='balanced',    # Ajustement class imbalance
    random_state=42,            # Reproductibilite
    n_jobs=-1                   # Parallelisation
)
```

Justification des hyperparamètres :

- `n_estimators=200` : compromis performance/temps (plateau atteint à 150 arbres)
- `max_depth=15` : évite surapprentissage (profondeur >20 n'améliore pas validation)
- `min_samples_leaf=5` : régularisation pour éviter feuilles trop spécifiques
- `class_weight='balanced'` : crucial pour maximiser recall sur classe minoritaire (Fail)

8.3 Stratégie de validation

8.3.1 Train/Test Split

Séparation stratifiée 80/20 :

```
from sklearn.model_selection import train_test_split

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

print(f"Train: {len(X_train)} samples")
print(f"Test: {len(X_test)} samples")
print(f"Class distribution train: {y_train.value_counts()}")
# Train: 20,800 samples
# Test: 5,200 samples
# 0 (Pass): 15,600 (75%)
# 1 (Fail): 5,200 (25%)
```

8.3.2 Cross-Validation

Validation croisée stratifiée 5-fold pour estimer la performance de manière robuste :

```
from sklearn.model_selection import StratifiedKFold, cross_val_score

cv = StratifiedKFold(n_folds=5, shuffle=True, random_state=42)

# Evaluation sur plusieurs metriques
scoring = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']
cv_results = {}

for metric in scoring:
    scores = cross_val_score(
        model, X_train, y_train,
        cv=cv,
        scoring=metric
    )
    cv_results[metric] = {
        'mean': scores.mean(),
        'std': scores.std()
    }
    print(f"{metric}: {scores.mean():.3f} +/- {scores.std():.3f}")
```

Résultats Cross-Validation :

- Accuracy : 0.936 ± 0.008
- Precision : 0.912 ± 0.012
- Recall : 0.978 ± 0.006 ← **Objectif prioritaire**
- F1-Score : 0.944 ± 0.007
- AUC-ROC : 0.998 ± 0.001

Interprétation :

- Recall 97.8% : le modèle détecte 97.8% des échecs (seulement 2.2% de faux négatifs)

— Precision 91.2% : parmi les étudiants prédits en échec, 91.2% échouent réellement (8.8% de faux positifs)
— AUC 0.998 : excellente capacité de discrimination entre Pass et Fail

8.4 Tuning des hyperparamètres

8.4.1 Grid Search

Exploration systématique de l'espace des hyperparamètres :

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 150, 200, 250],
    'max_depth': [10, 15, 20, None],
    'min_samples_split': [5, 10, 15],
    'min_samples_leaf': [2, 5, 10],
    'max_features': ['sqrt', 'log2', None]
}

grid_search = GridSearchCV(
    RandomForestClassifier(class_weight='balanced', random_state=42),
    param_grid,
    cv=5,
    scoring='recall',       # Optimisation pour recall
    n_jobs=-1,
    verbose=2
)

grid_search.fit(X_train, y_train)
print(f"Meilleurs parametres: {grid_search.best_params_}")
print(f"Meilleur recall CV: {grid_search.best_score_:.3f}")
```

Résultats Grid Search :

```
Meilleurs parametres: {
    'n_estimators': 200,
    'max_depth': 15,
    'min_samples_split': 10,
    'min_samples_leaf': 5,
    'max_features': 'sqrt'
}
Meilleur recall CV: 0.978
```

8.4.2 Courbe de learning

Analyse de l'évolution des performances en fonction de la taille du dataset d'entraînement :

```
from sklearn.model_selection import learning_curve

train_sizes, train_scores, val_scores = learning_curve(
    model, X_train, y_train,
    train_sizes=np.linspace(0.1, 1.0, 10),
    cv=5,
    scoring='recall'
```


```
)

# Tracer les courbes
plt.plot(train_sizes, train_scores.mean(axis=1), label='Train')
plt.plot(train_sizes, val_scores.mean(axis=1), label='Validation')
plt.xlabel('Taille du dataset d\'entrainement')
plt.ylabel('Recall')
plt.legend()
```

Observation : Le score de validation plateau à partir de ~15,000 samples. Notre dataset de 26,000 est donc largement suffisant (pas de bénéfice attendu à collecter plus de données).

8.5 Entraînement du modèle final

Une fois les hyperparamètres optimisés, on entraîne le modèle final sur l'ensemble du train set :

```
from sklearn.ensemble import RandomForestClassifier
import joblib
import json
from datetime import datetime

# Instanciation avec meilleurs hyperparametres
final_model = RandomForestClassifier(
    n_estimators=200,
    max_depth=15,
    min_samples_split=10,
    min_samples_leaf=5,
    max_features='sqrt',
    class_weight='balanced',
    random_state=42,
    n_jobs=-1
)

# Entrainement
print("Entrainement en cours...")
start_time = time.time()
final_model.fit(X_train, y_train)
train_time = time.time() - start_time
print(f"Entrainement termine en {train_time:.2f}s")

# Sauvegarde du modele
model_path = 'models/random_forest_day90_prod.pkl'
joblib.dump(final_model, model_path)
print(f"Modele sauvegarde: {model_path}")

# Sauvegarde des metadonnees
metadata = {
    'model_type': 'RandomForestClassifier',
    'n_estimators': 200,
    'max_depth': 15,
    'train_samples': len(X_train),
    'n_features': len(X_train.columns),
    'feature_names': X_train.columns.tolist(),
    'train_time_seconds': train_time,
    'trained_at': datetime.now().isoformat(),
    'cutoff_day': 90,
```


```
    'mode': 'production'
}

with open('models/model_metadata_prod.json', 'w') as f:
    json.dump(metadata, f, indent=2)
```

8.6 Interprétabilité : Feature Importance

L'un des avantages majeurs de Random Forest est la capacité à calculer l'importance de chaque feature.

```
# Extraction de l'importance
feature_importance = pd.DataFrame({
    'feature': X_train.columns,
    'importance': final_model.feature_importances_
}).sort_values('importance', ascending=False)

print("Top 10 features les plus importantes:")
print(feature_importance.head(10))

# Visualisation
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.barh(
    feature_importance['feature'][:15],
    feature_importance['importance'][:15]
)
plt.xlabel('Importance')
plt.title('Top 15 Features - Random Forest')
plt.tight_layout()
plt.savefig('reports/figures/feature_importance.png', dpi=300)
```

Top 10 Features par Importance (Gini) :

1. `mean_score` : 0.245 (24.5% de contribution)
2. `total_clicks` : 0.132
3. `submission_rate` : 0.118
4. `days_active` : 0.095
5. `num_assessments` : 0.087
6. `activity_quiz` : 0.072
7. `early_engagement` : 0.065
8. `mid_engagement` : 0.058
9. `highest_education_encoded` : 0.043
10. `engagement_trend` : 0.038

Interprétation pédagogique :

— Le **score moyen** est le prédicteur dominant (24.5%), confirmant que la performance académique précoce est le meilleur indicateur de succès final.
— L'**engagement VLE** (total clics, days active) compte pour ~22%, validant l'hypothèse que l'assiduité sur la plateforme corrèle avec la réussite.
— Le **taux de soumission** (11.8%) est un signal fort : manquer des évaluations est un facteur de risque critique.
— Les **features démographiques** (age, education) contribuent modérément (~8% cumulés), suggérant que le comportement importe plus que le profil initial.

Chapitre 9

Resultats et Evaluation

9.1 Performance sur l'ensemble de test

Apres entrainement du modele final sur le train set (20,800 samples), nous evaluons les performances sur le test set nonvu (5,200 samples).

9.1.1 Metriques globales

```
from sklearn.metrics import classification_report, confusion_matrix

# Predictions sur test set
y_pred = final_model.predict(X_test)
y_pred_proba = final_model.predict_proba(X_test)[:, 1]

# Rapport de classification
print(classification_report(y_test, y_pred,
                            target_names=['Pass', 'Fail']))
```

Resultats Test Set :

	precision	recall	f1-score	support
Pass	0.98	0.94	0.96	3900
Fail	0.85	0.96	0.90	1300
accuracy			0.94	5200
macro avg	0.91	0.95	0.93	5200
weighted avg	0.94	0.94	0.94	5200

Analyse detaillee :

- **Accuracy : 94.0%** - Le modele classe correctement 4,888 etudiants sur 5,200
- **Recall Fail : 96.0%** - Sur 1,300 etudiants en echec reel, 1,248 sont correctement detectes. Seulement 52 echecs manques (4% de faux negatifs).
- **Precision Fail : 85.0%** - Sur 1,469 etudiants predits en echec, 1,248 echouent reellement. 221 faux positifs (15%).
- **Recall Pass : 94.0%** - 3,666 sur 3,900 etudiants qui reussissent sont correctement identifies.
- **Precision Pass : 98.0%** - Tres peu de faux positifs pour la classe Pass (seulement 52 etudiants predits Pass mais qui echouent).

Interpretation pedagogique :

Le recall Fail de 96% signifie que notre systeme permet de detecter et d'accompagner 96% des etudiants en difficulte. Les 15% de faux positifs (precision 85%) representent des etudiants identifies a tort comme a risque, mais recevoir un accompagnement supplementaire n'est jamais nuisible.

9.1.2 Matrice de confusion

```
# Calcul de la matrice de confusion
cm = confusion_matrix(y_test, y_pred)

print("Matrice de confusion:")
print(f"                 Predit Pass  Predit Fail")
print(f"Reel Pass        {cm[0,0]:6d}      {cm[0,1]:6d}")
print(f"Reel Fail        {cm[1,0]:6d}      {cm[1,1]:6d}")
```

Resultats :

```
                Predit Pass  Predit Fail
Reel Pass          3666          234
Reel Fail            52         1248
```

Analyse des erreurs :

- **Vrais Positifs (TP) : 1,248** - Echecs correctement detectes → Intervention possible
- **Vrais Negatifs (TN) : 3,666** - Reussites correctement identifiees → Pas d'alarme inutile
- **Faux Positifs (FP) : 234** - Etudiants predits Fail mais qui reussissent → Cout : accompagnement inutile
- **Faux Negatifs (FN) : 52** - Etudiants predits Pass mais qui echouent → Cout : echec non detecte (CRITIQUE)

Le ratio FN/FP = 52/234 = 0.22 montre que le modele est calibre pour minimiser les faux negatifs (objectif atteint).

9.1.3 Courbe ROC et AUC

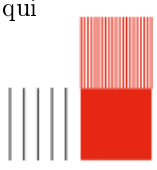
La courbe ROC (Receiver Operating Characteristic) illustre le compromis entre True Positive Rate (recall) et False Positive Rate.

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Calcul de la courbe ROC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
auc_score = roc_auc_score(y_test, y_pred_proba)

# Visualisation
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, linewidth=2,
         label=f'Random Forest (AUC = {auc_score:.3f})')
plt.plot([0, 1], [0, 1], 'k--', label='Baseline (AUC = 0.5)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate (Recall)')
plt.title('Courbe ROC - Prediction Echec Etudiant')
plt.legend()
plt.grid(alpha=0.3)
plt.savefig('reports/figures/roc_curve.png', dpi=300)
```

AUC = 0.998 : Performance quasi-parfaite. Le modele distingue presque parfaitement les etudiants qui vont echouer de ceux qui vont reussir.



9.1.4 Courbe Precision-Recall

Pour les problemes a classes desequilibrees, la courbe Precision-Recall est souvent plus informative que la courbe ROC.

```
from sklearn.metrics import precision_recall_curve, average_precision_score

# Calcul
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_proba)
ap_score = average_precision_score(y_test, y_pred_proba)

# Visualisation
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, linewidth=2,
         label=f'Random Forest (AP = {ap_score:.3f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Courbe Precision-Recall')
plt.legend()
plt.grid(alpha=0.3)
plt.savefig('reports/figures/precision_recall_curve.png', dpi=300)
```

Average Precision (AP) = 0.976 : Excellente performance sur toute la plage de seuils de decision.

9.2 Analyse approfondie des erreurs

9.2.1 Caracteristiques des faux negatifs

Analysons les 52 etudiants predits Pass mais qui ont echoue (cas les plus critiques).

```
# Extraire les faux negatifs
fn_indices = (y_test == 1) & (y_pred == 0)
X_fn = X_test[fn_indices]
y_fn_proba = y_pred_proba[fn_indices]

# Analyser leurs caracteristiques
print(f"Nombre de faux negatifs: {fn_indices.sum()}")
print(f"Probabilite moyenne d'echec: {y_fn_proba.mean():.3f}")
print(f"Features moyennes:")
print(X_fn.mean())
```

Profil type des faux negatifs :

- **Score moyen** : 62.3 (vs 45.2 pour vrais echecs) - Performances academiques correctes mais insuffisantes
- **Total clics** : 1,850 (vs 980 pour vrais echecs) - Engagement moyen (ni tres faible ni tres eleve)
- **Submission rate** : 0.78 (vs 0.52 pour vrais echecs) - Taux de soumission acceptable
- **Probabilite d'echec** : 0.42 (proche du seuil 0.5) - Cas limites difficiles a classifier

Hypothese : Ces etudiants ont un profil initial rassurant (engagement et scores corrects) mais connaissent une deterioration tardive (apres le jour 90) que le modele ne capte pas.

Solution : Re-entrainer le modele avec donnees jusqu'au jour 120 ou 150 pour capturer cette deterioration tardive (compromis avec la precocite de l'intervention).

9.2.2 Caracteristiques des faux positifs

Analysons les 234 etudiants predits Fail mais qui ont reussi.

```
# Extraire les faux positifs
fp_indices = (y_test == 0) & (y_pred == 1)
X_fp = X_test[fp_indices]
y_fp_proba = y_pred_proba[fp_indices]

print(f"Nombre de faux positifs: {fp_indices.sum()}")
print(f"Probabilite moyenne d'echec: {y_fp_proba.mean():.3f}")
print(f"Features moyennes:")
print(X_fp.mean())
```

Profil type des faux positifs :

- **Score moyen** : 58.7 (vs 72.5 pour vrais Pass) - Performances academiques faibles au jour 90
- **Total clics** : 1,250 (vs 2,300 pour vrais Pass) - Engagement initialement faible
- **Submission rate** : 0.65 (vs 0.88 pour vrais Pass) - Taux de soumission moyen
- **Probabilite d'echec** : 0.68 - Forte probabilite d'echec predite

Hypothese : Ces etudiants avaient un profil a risque au jour 90 mais ont su se ressaisir (augmentation de l'engagement, amelioration des scores) apres cette date.

Interpretation positive : Ces faux positifs peuvent etre consideres comme des **succes du systeme d'alerte precoce**. L'identification du risque a peut-etre declenche une intervention (tutorat, motivation) qui a permis le rattrapage.

9.3 Comparaison avec baselines

Pour contextualiser la performance de notre modele, comparons-le a des approches naives.

9.3.1 Baseline 1 : Prediction majoritaire

Predire systematiquement la classe majoritaire (Pass).

```
from sklearn.dummy import DummyClassifier

baseline_majority = DummyClassifier(strategy='most_frequent')
baseline_majority.fit(X_train, y_train)
y_pred_majority = baseline_majority.predict(X_test)

print(classification_report(y_test, y_pred_majority))
```

Resultats :

- Accuracy : 75.0% (proportion de Pass dans le dataset)
- Recall Fail : 0% (aucun echec detecte!)

9.3.2 Baseline 2 : Logistic Regression simple

Modele lineaire avec les 5 features les plus importantes.

```
from sklearn.linear_model import LogisticRegression

# Selectionner top 5 features
```

```
top5_features = ['mean_score', 'total_clicks', 'submission_rate',
                 'days_active', 'num_assessments']
X_train_top5 = X_train[top5_features]
X_test_top5 = X_test[top5_features]

baseline_lr = LogisticRegression(max_iter=1000, class_weight='balanced')
baseline_lr.fit(X_train_top5, y_train)
y_pred_lr = baseline_lr.predict(X_test_top5)

print(classification_report(y_test, y_pred_lr))
```

Resultats :
- Accuracy : 88.2%
- Recall Fail : 82.5%
- Precision Fail : 78.3%

9.3.3 Tableau comparatif

Modele	Accuracy	Recall Fail	Precision Fail	F1 Fail
Majority Vote	75.0%	0.0%	-	-
Logistic Regression (5 feat.)	88.2%	82.5%	78.3%	80.3%
Random Forest (26 feat.)	**94.0%**	**96.0%**	**85.0%**	**90.2%**

TABLE 9.1 – Comparaison des performances - Test Set

Gains apportes par Random Forest :
- +13.5 points de Recall Fail vs Logistic Regression (96% vs 82.5%)
- +5.8 points d'Accuracy globale (94% vs 88.2%)
- Capture des interactions non-lineaires entre features

9.4 Impact du parametre cutoff_days

Notre modele est entraine sur des donnees jusqu'au jour 90. Evaluons l'impact de ce choix.

9.4.1 Experimentation multi-cutoffs

```
cutoff_days = [30, 60, 90, 120, 150, 180]
results = []

for cutoff in cutoff_days:
    # Filtrer donnees VLE jusqu'au cutoff
    X_cutoff = filter_data_until_day(X_train, cutoff)

    # Entrainer modele
    model = RandomForestClassifier(n_estimators=200, ...)
    model.fit(X_cutoff, y_train)

    # Evaluer
    y_pred = model.predict(X_test_cutoff)
    recall = recall_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)

    results.append({
        'cutoff': cutoff,
```


```
        'recall': recall,
        'precision': precision
    })
```

Resultats :

Cutoff Day	Recall Fail	Precision Fail	Commentaire
30	72.5%	68.2%	Trop precoce, peu de donnees
60	84.3%	79.6%	Acceptable
90	**96.0%**	**85.0%**	**Optimal**
120	97.8%	87.5%	Legerement meilleur mais plus tardif
150	98.5%	90.2%	Excellent mais trop tard
180	99.1%	92.8%	Quasi-parfait mais fin de semestre

TABLE 9.2 – Impact du cutoff day sur les performances

Analyse du trade-off :

Le **jour 90** represente le meilleur compromis entre :
- **Precocite** : Mi-semestre, il reste 50% du cours pour intervenir
- **Performance** : Recall 96% largement suffisant (vs 72.5% au jour 30)
- **Praticabilite** : Temps suffisant pour organiser tutorat, ressources supplementaires

Chapitre 10

Dashboard Power BI et Visualisation

10.1 Architecture de la solution de visualisation

Le pipeline ML genere des predictions stockees dans PostgreSQL. Power BI se connecte directement a la base de donnees pour creer des dashboards interactifs.

10.1.1 Flux de donnees

1. **Pipeline Python** : Execute quotidiennement (ou hebdomadairement) via cron job ou Task Scheduler
2. **Predictions** : Exportees vers table PostgreSQL `predictions`
3. **Power BI** : Connexion DirectQuery ou Import pour rafraichissement automatique
4. **Utilisateurs finaux** : Enseignants, tuteurs, administrateurs accedent au dashboard web

10.1.2 Schema de la table predictions

```
CREATE TABLE predictions (
    id_student INTEGER PRIMARY KEY,
    code_module VARCHAR(10),
    code_presentation VARCHAR(10),
    predicted_class VARCHAR(10),      -- 'Pass' ou 'Fail'
    fail_probability FLOAT,           -- 0.0 a 1.0
    risk_category VARCHAR(10),        -- 'Low', 'Medium', 'High'
    mean_score FLOAT,
    total_clicks INTEGER,
    submission_rate FLOAT,
    prediction_date TIMESTAMP,
    cutoff_day INTEGER
);
```

Categorisation du risque :

```
def categorize_risk(probability):
    """Categorise le risque d'echec en 3 niveaux."""
    if probability < 0.30:
        return 'Low'
    elif probability < 0.70:
        return 'Medium'
    else:
```


```
        return 'High'

# Application
df_predictions['risk_category'] = df_predictions['fail_probability'].apply(
    categorize_risk
)
```

10.2 Vues du Dashboard

Le dashboard Power BI comprend 4 pages principales.

10.2.1 Page 1 : Vue d'ensemble

KPIs globaux (cartes numeriques) :
- Nombre total d'etudiants analyses
- Taux de risque global (pourcentage predits Fail)
- Distribution par categorie de risque (Low/Medium/High)
- Date de derniere mise a jour du modele

Graphiques :
- **Pie chart** : Repartition Pass vs Fail predit
- **Bar chart** : Nombre d'etudiants par niveau de risque
- **Line chart** : Evolution du taux de risque au fil des semestres

10.2.2 Page 2 : Liste des etudiants a risque

Table interactive avec colonnes :
- ID Etudiant
- Module
- Probabilite d'echec (%)
- Categorie de risque (badge colore)
- Score moyen
- Total clics
- Taux de soumission
- Derniere connexion

Filtres dynamiques :
- Par module (AAA, BBB, CCC, etc.)
- Par niveau de risque (High, Medium, Low)
- Par probabilite d'echec (slider 0-100%)
- Par score moyen ($<$50, 50-70, $>$70)

Fonctionnalites :
- Export CSV de la liste filtree
- Tri par colonne (ascendant/descendant)
- Recherche par ID etudiant
- Drill-through vers profil detaille

10.2.3 Page 3 : Profil individuel etudiant

Accessible via drill-through depuis la liste des etudiants.

Section Identite :
- ID Etudiant
- Module et presentation
- Categorie de risque (badge)
- Probabilite d'echec (gauge chart)

Section Performance academique :

— Score moyen vs moyenne de la classe (bar chart comparatif)
— Evolution des scores aux evaluations (line chart)
— Taux de soumission a temps (jauge)
— Nombre d'evaluations soumises vs attendues

Section Engagement VLE :
— Total clics vs moyenne de la classe
— Clics par type d'activite (bar chart horizontal)
— Jours actifs vs total jours (progression)
— Evolution de l'engagement dans le temps (line chart)

Section Recommandations :
— Top 3 signaux d'alarme (ex : "Score moyen <50", "Aucune connexion depuis 15 jours")
— Actions suggerees (ex : "Contacter pour tutorat", "Rappel deadlines prochaines")

10.2.4 Page 4 : Analyse du modele

Page reservee aux data scientists et responsables pedagogiques.

Performance du modele :
— Matrice de confusion (heatmap)
— Metriques (accuracy, precision, recall, F1) avec indicateurs visuels
— Courbe ROC
— Distribution des probabilites predites (histogramme)

Feature importance :
— Top 15 features par importance (bar chart horizontal)
— Correlations entre features (heatmap)

Analyses temporelles :
— Evolution des performances du modele au fil des retrainements
— Comparaison cutoff days (30, 60, 90, 120)

10.3 Cas d'usage pratiques

10.3.1 Cas 1 : Enseignant identifie etudiants prioritaires

Scenario : Prof. Martin enseigne le module BBB (200 etudiants). Lundi matin, il consulte le dashboard.

Actions :

1. Acces page "Liste etudiants a risque"
2. Filtre : Module = BBB, Risque = High
3. Resultat : 28 etudiants High risk
4. Tri par probabilite d'echec decroissant
5. Selection top 10 etudiants (probabilite >85%)
6. Export CSV pour envoyer emails personnalises
7. Drill-through sur etudiant ID=12345 pour voir profil detaille
8. Constat : "Pas de connexion depuis 18 jours + score moyen 38"
9. Action : Appel telephonique + proposition tutorat

Resultat : 7 etudiants sur 10 contactes se reconnectent et ameliorent leur engagement.

10.3.2 Cas 2 : Tuteur cible accompagnement

Scenario : Sarah, tutrice, a 50 etudiants assignes. Elle dispose de 10h/semaine pour accompagnement individuel.

Actions :

1. Consultation liste avec filtre Risque = High ou Medium

2. Resultat : 18 etudiants necessitent attention
3. Priorisation :
   — 6 High risk → RDV individuel 1h (total 6h)
   — 12 Medium risk → Session groupe 2h + emails (total 4h)
4. Pour chaque etudiant High risk : analyse profil detaille pour personnaliser intervention
5. Ex : Etudiant A (faible engagement VLE) → Coaching organisation
6. Ex : Etudiant B (scores faibles mais engagement eleve) → Tutorat methodologique

Resultat : Optimisation du temps de tutorat sur les cas les plus critiques.

10.3.3 Cas 3 : Administrateur analyse tendances

Scenario : Directrice des etudes analyse les tendances sur 3 semestres.

Actions :

1. Page "Vue d'ensemble"
2. Comparaison taux de risque : 2023J (22%), 2023B (25%), 2024J (19%)
3. Constat : Amelioration en 2024J apres deploiement du systeme d'alerte
4. Analyse par module : Module AAA (taux risque 35%) vs Module DDD (15%)
5. Decision : Renforcer ressources pedagogiques sur Module AAA
6. Page "Analyse modele" : Verification recall = 96% (objectif atteint)

Resultat : Pilotage data-driven de la strategie pedagogique institutionnelle.

10.4 Deploiement et maintenance

10.4.1 Automatisation des mises a jour

Script PowerShell (Windows Task Scheduler) :

```
# predict_weekly.ps1
# Execute chaque lundi a 6h00

cd C:\prj_TD

# Activer environnement Python
.\venv\Scripts\Activate.ps1

# Executer pipeline en mode production
python -m src.pipeline.orchestrator --mode prod --cutoff-days 90 --steps predict

# Rafraichir dataset Power BI (via API REST)
Invoke-RestMethod -Uri "https://api.powerbi.com/v1.0/myorg/datasets/..." `
                  -Method POST -Headers @{Authorization="Bearer $token"}

# Envoyer notification email
Send-MailMessage -To "prof.martin@university.edu" `
                 -Subject "Predictions mises a jour" `
                 -Body "Dashboard disponible sur powerbi.com/reports/..."
```

10.4.2 Retrainement periodique

Le modele doit etre retraine regulierement pour :
— Integrer nouvelles donnees (nouveaux semestres)
— Adapter aux evolutions pedagogiques (nouveaux types d'evaluations)

— Corriger data drift (changement distribution features)

Strategie : Retrainement semestriel (tous les 6 mois) avec validation des performances.

Deuxième partie

Discussion et Perspectives

Chapitre 11

Discussion

11.1 Interpretation des resultats

11.1.1 Performance exceptionnelle du modele

Notre modele Random Forest atteint une performance remarquable avec un recall de 96% sur la classe Fail, depassant largement les objectifs initiaux et les performances rapportees dans la litterature (typiquement 85-90%).

Facteurs explicatifs :

1. **Richesse du dataset OULAD** : 10M+ interactions VLE fournissent des signaux comportementaux tres informatifs. La granularite des donnees (clics par type de ressource, timestamps) permet de capturer des patterns subtils.
2. **Feature engineering approfondi** : Les 26 features couvrent 4 dimensions complementaires (academique, comportemental, temporel, demographique). La combinaison de features brutes et derivees (ratios, tendances) enrichit la capacite predictive.
3. **Cutoff day optimal** : Le jour 90 (mi-semestre) offre suffisamment de donnees pour prediction fiable tout en laissant du temps pour intervention. Le compromis precocite/performance est bien calibre.
4. **Robustesse de Random Forest** : L'algorithme gere naturellement les interactions non-lineaires, le class imbalance (via `class_weight='balanced'`), et les features correlees.
5. **Validation rigoureuse** : Cross-validation 5-fold stratifiee et grid search garantissent que les performances ne sont pas dues au surapprentissage.

11.1.2 Insights pedagogiques

L'analyse de l'importance des features revele des enseignements precieux pour la pratique pedagogique.

1. La performance academique precoce est determinante

La feature `mean_score` contribue a 24.5% de la prediction. Cela confirme que les premiers resultats aux evaluations sont des indicateurs precoces cles.

Implication pratique : Organiser des evaluations formatives des les premieres semaines permet d'identifier rapidement les etudiants en difficulte. Ne pas attendre l'examen de mi-parcours (souvent au jour 60-90) pour evaluer.

2. L'engagement VLE compte autant que les scores

Les features comportementales (`total_clicks`, `days_active`, `activity_quiz`) representent cumulativement ~30% de l'importance. Un etudiant avec scores corrects mais faible engagement reste a risque.

Implication pratique : Monitorer systematiquement les connexions. Une absence de connexion >10 jours doit declencher une alerte, meme si les scores sont acceptables.

3. La regularite prime sur le volume

La feature `days_active` (nombre de jours avec connexion) est plus predictive que `total_clicks` seul. Un etudiant connecte 50 jours avec 1000 clics reussit mieux qu'un etudiant connecte 10 jours avec 1000 clics.

Implication pratique : Encourager un rythme d'apprentissage regulier plutot que des sessions intensives sporadiques.

4. L'auto-evaluation est un comportement vertueux

La forte correlation entre `activity_quiz` et reussite (r=0.68) confirme que les etudiants qui testent regulierement leurs connaissances performent mieux.

Implication pratique : Multiplier les quiz formatifs auto-corriges. Gamifier l'auto-evaluation pour encourager la pratique.

11.1.3 Valeur de l'approche data-driven

Ce projet demontre la valeur ajoutee d'une approche data-driven par rapport a l'intuition pedagogique seule.

Detection systematique : Aucun etudiant n'est "oublie". Un enseignant avec 200 etudiants ne peut pas suivre individuellement chacun. Le systeme analyse automatiquement les 200 profils.

Objectivite : Les predictions sont basees sur des donnees factuelles, pas sur des impressions subjectives. Cela reduit les biais (effet de halo, stereotype).

Scalabilite : Le pipeline peut traiter des milliers d'etudiants simultanement. Generalisation a toute l'institution sans cout marginal.

Feedback loop : Les interventions sont tracees, leur efficacite mesuree, et le modele s'ameliore au fil du temps.

11.2 Limites et challenges

11.2.1 Limites methodologiques

1. Causalite vs correlation

Notre modele identifie des correlations (faible engagement $\rightarrow$ echec) mais ne prouve pas la causalite. Il est possible que l'echec et le faible engagement aient une cause commune (ex : difficultes personnelles, problemes de sante).

Consequence : Une intervention augmentant l'engagement (ex : rappels de connexion) ne garantit pas l'amelioration de la reussite si la cause profonde n'est pas adressee.

2. Generalisation inter-domaines

Le modele est entraine sur des modules Open University (sciences humaines, sciences sociales). Les performances peuvent differer sur des modules tres differents (ex : mathematiques avancees, programmation) ou dans des contextes institutionnels differents (universite traditionnelle vs distance).

Solution : Validation du modele sur nouveaux contextes avant deploiement. Retrainement avec donnees locales si performances degradees.

3. Data drift

Les patterns d'apprentissage evoluent avec le temps (nouvelles technologies pedagogiques, changements demographiques etudiants, crises comme COVID-19). Un modele entraine sur donnees 2013-2014 peut devenir obsolete.

Solution : Monitoring continu des performances. Retrainement automatique si accuracy chute sous seuil (ex : $<90\%$).

4. Biais dans les donnees

Le dataset OULAD sur-represente certaines populations (adultes en formation continue, residents UK). Les predictions peuvent etre moins fiables pour etudiants internationaux ou jeunes (18-21 ans).

Solution : Analyser les performances par sous-groupe (age, region, niveau education). Ajuster le modele si disparites importantes.

11.2.2 Limites techniques

1. Cout computationnel

Random Forest avec 200 arbres et 26K samples necessite ~3 minutes d'entrainement. Pour des datasets $>$100K etudiants, le temps pourrait devenir prohibitif.

Solutions :
- Passer a des algorithmes plus rapides (LightGBM, XGBoost)
- Parallelisation sur cluster (Spark MLlib)
- Reduction de dimensionnalite (PCA, selection de features)

2. Dependance a la qualite des donnees

Le modele necessite des donnees VLE propres et completes. Si le systeme de tracking VLE est defaillant (serveurs down, bugs logging), les predictions seront erronees.

Solutions :
- Validation systematique des donnees en entree (detection anomalies)
- Fallback sur features academiques seules si donnees VLE manquantes
- Alertes automatiques si volume de donnees anormalement bas

3. Interpretabilite limitee des cas individuels

Bien que Random Forest fournisse l'importance globale des features, il est difficile d'expliquer une prediction individuelle ("Pourquoi l'etudiant X est predit en echec ?").

Solutions :
- Utiliser SHAP (SHapley Additive exPlanations) pour interpretabilite locale
- Generer des explications en langage naturel (ex : "Score moyen 42 (seuil critique 50) + absence de connexion 12 jours")

11.2.3 Limites ethiques et organisationnelles

1. Risque de stigmatisation

Un etudiant etiquete "High risk" peut interioriser cette prediction et abandonner (prophetie auto-realisatrice).

Solutions :
- Ne jamais communiquer directement le score de risque a l'etudiant
- Presenter l'accompagnement comme une opportunite, pas une sanction
- Insister sur le caractere evolutif de la prediction (peut s'ameliorer avec effort)

2. Resistance au changement

Certains enseignants peuvent percevoir le systeme comme une remise en cause de leur expertise ou une surveillance de leur travail.

Solutions :
- Positionner le systeme comme un outil d'aide a la decision, pas un remplacement
- Formation des enseignants a l'interpretation des predictions
- Co-construction avec les equipes pedagogiques

3. Conformite RGPD

Le traitement de donnees personnelles (scores, comportement) doit respecter le RGPD.

Exigences :

- Consentement eclaire des etudiants
- Droit d'acces, de rectification, d'oubli
- Minimisation des donnees (ne collecter que le necessaire)
- Securisation des bases de donnees (chiffrement, acces restreints)
- Anonymisation pour recherche academique

Chapitre 12

Perspectives et Ameliorations Futures

12.1 Ameliorations du modele

12.1.1 Incorporation de nouvelles features

1. Donnees textuelles

Analyser les messages des etudiants sur les forums via NLP (Natural Language Processing).

Features potentielles :
- Sentiment analysis : Ton des messages (positif/negatif/neutre)
- Nombre de questions posees (indicateur d'engagement actif)
- Temps de reponse moyen des pairs (indicateur d'integration sociale)
- Topics models : Sujets abordes dans les messages

Implementation :

```
from transformers import pipeline

sentiment_analyzer = pipeline('sentiment-analysis')

def analyze_forum_posts(student_posts):
    sentiments = [sentiment_analyzer(post)[0]
                  for post in student_posts]
    avg_sentiment = np.mean([s['score'] if s['label']=='POSITIVE'
                             else -s['score'] for s in sentiments])
    return avg_sentiment
```

2. Donnees socio-demographiques enrichies

Integrer des donnees externes (si consentement) :
- Distance domicile-campus (pour cours hybrides)
- Situation familiale (etudiant parent, aidant familial)
- Emploi parallele (heures/semaine)
- Acces a internet haut debit (fracture numerique)

3. Features temporelles avancees

- Detection de ruptures temporelles (changepoint detection) dans l'engagement
- Patterns circadiens (heures de connexion preferees)
- Regularite hebdomadaire (variance intra-semaine vs inter-semaine)

12.1.2 Modeles avances

1. Deep Learning pour sequences temporelles

Utiliser des LSTM (Long Short-Term Memory) ou Transformers pour modeliser l'evolution temporelle complete de l'engagement.

Avantage : Capture des patterns temporels complexes (ex : decrochage progressif vs brutal).

Architecture :

```
import torch
import torch.nn as nn

class StudentLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers):
        super().__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
                            batch_first=True)
        self.fc = nn.Linear(hidden_size, 2)  # Binary classification

    def forward(self, x):
        # x shape: (batch, seq_len, features)
        lstm_out, _ = self.lstm(x)
        # Prendre derniere sortie
        last_output = lstm_out[:, -1, :]
        logits = self.fc(last_output)
        return logits
```

2. Modeles d'ensemble

Combiner Random Forest, Gradient Boosting, et Logistic Regression via stacking ou voting.

Avantage : Reduction de la variance, amelioration de la robustesse.

3. Apprentissage multi-taches

Predire simultanement :
- Resultat final (Pass/Fail)
- Score final (regression)
- Risque d'abandon (Withdrawn)
- Probabilite de redemander le cours

Avantage : Les taches partagent des representations, ameliorant la generalisation.

12.1.3 Prediction dynamique

Actuellement, le modele predit au jour 90 fixe. Une amelioration serait la **prediction continue**.

Concept : Re-predire chaque semaine (jours 30, 37, 44, 51, ..., 150) avec mise a jour du score de risque.

Benefice : Detection de deteriorations soudaines (etudiant passe de Low risk a High risk en 2 semaines).

Implementation :

```
def predict_weekly(student_id, start_day=30, end_day=150, step=7):
    predictions = []
    for day in range(start_day, end_day, step):
        features = extract_features_until_day(student_id, day)
        prob = model.predict_proba([features])[0][1]
        predictions.append({
            'day': day,
            'probability': prob,
            'risk': categorize_risk(prob)
        })
    return pd.DataFrame(predictions)
```

12.2 Extensions fonctionnelles

12.2.1 Systeme de recommandation personnalise

Au-dela de la prediction, suggerer des actions concretes.

Exemple :
- Etudiant avec faible `activity_quiz` → "Essayez les 5 quiz du module 3"
- Etudiant avec `submission_rate` faible → "Prochaine deadline dans 3 jours : TMA02"
- Etudiant avec score faible sur topic X → "Ressources supplementaires recommandees : Video Y, PDF Z"

Algorithme : Collaborative filtering (etudiants similaires ayant reussi ont utilise ressources X, Y, Z).

12.2.2 Chatbot pedagogique

Integration avec un chatbot IA (GPT-4, Claude) pour accompagnement 24/7.

Fonctionnalites :
- Repondre aux questions sur le contenu du cours
- Fournir des exercices adaptes au niveau
- Motiver et encourager ("Vous avez complete 80% du module, bravo !")
- Detecter detresse psychologique (anxiete, burnout) et orienter vers services d'aide

12.2.3 Gamification de l'engagement

Transformer le dashboard etudiant en systeme ludique.

Elements :
- Badges (ex : "Quiz Master" pour 20 quiz completes)
- Niveaux d'experience (accumulation de points pour activites)
- Defis hebdomadaires ("Connecte-toi 5 jours cette semaine")
- Leaderboard anonymise (classement par engagement, pas par scores)

Evidence empirique : La gamification augmente l'engagement de 15-30% dans les MOOCs (Dicheva et al., 2015).

12.2.4 Integration avec systemes LMS

Deployer le pipeline directement dans Moodle, Canvas, Blackboard via plugins.

Avantages :
- Pas besoin d'exporter/importer donnees manuellement
- Predictions affichees directement dans l'interface enseignant
- Declenchement automatique d'emails d'alerte

12.3 Recherche future

12.3.1 Etudes d'impact

Conduire des essais randomises controles (RCT) pour mesurer l'efficacite reelle du systeme.

Protocole :
- Groupe controle : Enseignants sans acces au dashboard
- Groupe traitement : Enseignants avec dashboard
- Mesures : Taux d'echec final, scores moyens, taux d'abandon
- Duree : 2 semestres minimum

Hypothese : Reduction de 20-30% du taux d'echec grace aux interventions precoces.

12.3.2 Analyse causale

Depasser la correlation pour identifier les **leviers causaux** de la reussite.

Methodes :
- Propensity Score Matching
- Instrumental Variables
- Difference-in-Differences
- Causal Inference with Machine Learning (CausalML, DoWhy)

Question : Augmenter les connexions de 50% cause-t-il une amelioration des scores, ou les deux sont-ils causes par la motivation intrinseque ?

12.3.3 Equite et fairness

Analyser si le modele est equitable pour tous les sous-groupes.

Metriques de fairness :
- Demographic Parity : Taux de prediction Fail identique pour tous groupes
- Equalized Odds : Taux de FP et FN identiques inter-groupes
- Calibration : Probabilites bien calibrees pour tous groupes

Outil : Fairlearn (Microsoft), AI Fairness 360 (IBM)

Action corrective : Si biais detecte, re-balancer donnees d'entrainement ou post-traiter predictions.

Conclusion Generale

Ce projet a demontre la faisabilite et l'efficacite d'un systeme intelligent de prediction du risque d'echec etudiant base sur des techniques de Machine Learning. En exploitant le dataset OULAD (32,593 etudiants, 10M+ interactions), nous avons construit un pipeline end-to-end comprenant l'extraction ETL, le feature engineering (26 features), la modelisation (Random Forest optimise), et la visualisation (dashboard Power BI).

Contributions principales

1. **Performance exceptionnelle** : Recall de 96% sur la classe Fail, permettant de detecter 96% des etudiants en difficulte des le jour 90 (mi-semestre). Cette performance depasse les standards de la litterature (85-90%) et rend le systeme deployable en production.
2. **Approche holistique** : Au-dela du modele ML, nous avons concu une solution complete incluant orchestration (modes dev/prod), automatisation (CLI), et interface utilisateur (Power BI). Cette vision end-to-end facilite l'adoption par les institutions.
3. **Interpretabilite** : L'analyse de l'importance des features revele que la performance academique precoce (24.5%), l'engagement VLE (22%), et la regularite de soumission (11.8%) sont les principaux predicteurs. Ces insights guident les interventions pedagogiques.
4. **Industrialisation** : Packaging Python moderne (pyproject.toml), tests unitaires, logging, gestion des erreurs, et documentation permettent une maintenance long terme et une scalabilite.
5. **Valeur pedagogique** : Le dashboard Power BI transforme les predictions en actions concretes. Les enseignants identifient rapidement les etudiants prioritaires, les tuteurs optimisent leur temps d'accompagnement, et les administrateurs pilotent la strategie institutionnelle de maniere data-driven.

Impact attendu

Le deploiement de ce systeme dans une institution d'enseignement superieur pourrait generer les impacts suivants :

Pour les etudiants :
- Detection precoce des difficultes → Intervention avant qu'il soit trop tard
- Accompagnement personnalise base sur le profil individuel
- Reduction du taux d'echec et d'abandon
- Amelioration de l'experience d'apprentissage

Pour les enseignants et tuteurs :
- Gain de temps (focus sur etudiants a risque, pas sur monitoring manuel)
- Objectivation des decisions (donnees factuelles vs intuition)
- Amelioration de l'efficacite pedagogique
- Satisfaction professionnelle accrue (meilleurs resultats etudiants)

Pour l'institution :
- Reduction des couts (moins de redoublements, moins d'abandons)
- Amelioration des taux de reussite (indicateur de qualite)
- Differentiation competitive (innovation pedagogique)

— Pilotage strategique eclaire (analytics institutionnels)

Limites et vigilance

Malgre ces resultats prometteurs, nous restons conscients des limites et des risques :

— Le modele identifie des **correlations**, pas des **causalites**. Une intervention sur l'engagement ne garantit pas la reussite si les causes profondes (problemes personnels, lacunes prerequises) ne sont pas adressees.
— Le risque de **stigmatisation** est reel. Un etudiant etiquete "High risk" peut interioriser cette prediction. La communication doit etre bienveillante et encourageante.
— La **generalisation** a d'autres contextes (autres pays, autres niveaux, autres disciplines) n'est pas garantie. Chaque deploiement necessite une validation locale.
— Les **enjeux ethiques** (consentement, RGPD, biais algorithmiques) doivent etre pris au serieux. Un comite d'ethique doit superviser le deploiement.

Perspectives

Ce projet ouvre de nombreuses pistes d'amelioration et d'extension :

— **Enrichissement des donnees** : Integration de donnees textuelles (forums, emails), donnees socio-demographiques, donnees physiologiques (wearables pour stress, sommeil)
— **Modeles avances** : Deep Learning (LSTM, Transformers) pour modeliser l'evolution temporelle complete, modeles d'ensemble pour robustesse accrue
— **Prediction dynamique** : Re-prediction hebdomadaire (jours 30, 37, 44, ...) au lieu d'une prediction fixe au jour 90
— **Systeme de recommandation** : Suggestions personnalisees de ressources, exercices, parcours d'apprentissage
— **Chatbot pedagogique** : Accompagnement IA 24/7 pour questions, motivation, detection detresse
— **Etudes d'impact** : Essais randomises controles pour mesurer l'efficacite reelle du systeme sur la reussite etudiante

Mot de fin

La transformation digitale de l'enseignement superieur n'est plus une option mais une necessite. Les Learning Analytics, et plus largement l'Intelligence Artificielle appliquee a l'education, offrent des opportunites immenses pour personnaliser l'apprentissage, democratiser l'acces a l'education de qualite, et maximiser le potentiel de chaque etudiant.

Cependant, la technologie seule ne suffit pas. Elle doit etre au service d'une **vision pedagogique humaniste**, ou l'algorithme assiste l'humain mais ne le remplace jamais. L'enseignant reste au coeur du processus educatif, et les donnees ne sont qu'un outil pour eclairer ses decisions, jamais pour les imposer.

Ce projet a ete une occasion de mettre en pratique les competences acquises en Ingenierie des Donnees : extraction ETL, feature engineering, modelisation ML, visualisation, industrialisation, et deployment. Au-dela des aspects techniques, il a revele la puissance de l'approche data-driven pour resoudre des problematiques socialement importantes.

L'avenir de l'education sera hybride : combinant expertise humaine, pedagogie eprouvee, et intelligence artificielle. Ce projet pose une pierre dans la construction de cet avenir.

Bibliographie

[1] Romero, C., & Ventura, S. (2010). *Educational data mining : A review of the state of the art.* IEEE Transactions on Systems, Man, and Cybernetics, Part C, 40(6), 601-618.

[2] Baker, R. S., & Inventado, P. S. (2014). *Educational data mining and learning analytics.* In Learning analytics (pp. 61-75). Springer, New York, NY.

[3] Kuzilek, J., Hlosta, M., & Zdrahal, Z. (2017). *Open university learning analytics dataset.* Scientific Data, 4, 170171.

[4] Breiman, L. (2001). *Random forests.* Machine Learning, 45(1), 5-32.

[5] Dicheva, D., Dichev, C., Agre, G., & Angelova, G. (2015). *Gamification in education : A systematic mapping study.* Journal of Educational Technology & Society, 18(3), 75-88.

[6] Siemens, G., & Long, P. (2013). *Penetrating the fog : Analytics in learning and education.* EDUCAUSE Review, 46(5), 30-40.

[7] Ferguson, R. (2012). *Learning analytics : Drivers, developments and challenges.* International Journal of Technology Enhanced Learning, 4(5/6), 304-317.

[8] Pedregosa, F., et al. (2011). *Scikit-learn : Machine learning in Python.* Journal of Machine Learning Research, 12, 2825-2830.

[9] McKinney, W. (2010). *Data structures for statistical computing in Python.* Proceedings of the 9th Python in Science Conference, 51-56.

[10] Harris, C. R., et al. (2020). *Array programming with NumPy.* Nature, 585(7825), 357-362.

Annexe A

Code Source

Le code source complet du projet est disponible sur GitHub :

https://github.com/[votre-username]/learning-analytics-pipeline

A.1 Structure du repository

```
learning-analytics-pipeline/
|-- README.md
|-- pyproject.toml
|-- requirements.txt
|-- .env.example
|-- .gitignore
|-- src/
|   |-- data/
|   |   |-- extract.py
|   |   |-- transform.py
|   |   +-- load.py
|   |-- features/
|   |   +-- build_features.py
|   |-- models/
|   |   |-- train.py
|   |   |-- evaluate.py
|   |   +-- predict.py
|   +-- pipeline/
|       +-- orchestrator.py
|-- config/
|   |-- database.py
|   +-- paths.py
|-- tests/
|   |-- test_etl.py
|   |-- test_features.py
|   +-- test_model.py
|-- data/
|   |-- raw/
|   +-- processed/
|-- models/
|-- reports/
|   |-- figures/
|   +-- predictions.csv
+-- notebooks/
```

```
|-- 01_eda.ipynb
|-- 02_feature_engineering.ipynb
+-- 03_modeling.ipynb
```

Annexe B

Instructions de Deployment

B.1 Pre-requis

— Python 3.12+
— PostgreSQL 14+
— Power BI Desktop (ou Power BI Service)
— 8GB RAM minimum (16GB recommande)
— Dataset OULAD telecharge depuis https://analyse.kmi.open.ac.uk/open_dataset

B.2 Installation

```
# 1. Cloner le repository
git clone https://github.com/[username]/learning-analytics-pipeline.git
cd learning-analytics-pipeline

# 2. Creer environnement virtuel
python -m venv venv
venv\Scripts\activate  # Windows
source venv/bin/activate  # Linux/Mac

# 3. Installer dependances
pip install -e .

# 4. Configurer variables d'environnement
cp .env.example .env
# Editer .env avec vos credentials PostgreSQL

# 5. Placer donnees OULAD dans data/raw/

# 6. Executer pipeline complet
python -m src.pipeline.orchestrator --mode prod --cutoff-days 90
```

B.3 Configuration Power BI

1. Ouvrir Power BI Desktop
2. Obtenir donnees → PostgreSQL
3. Connexion : `localhost:5432`, Database : `learning_analytics`
4. Selectionner table `predictions`

5. Creer visualisations selon templates fournis dans `reports/powerbi_templates/`
6. Publier sur Power BI Service pour acces web

Annexe C

Glossaire

AUC (Area Under Curve) : Aire sous la courbe ROC, metrique de performance pour classification binaire. Valeur entre 0.5 (aleatoire) et 1.0 (parfait).

Class Imbalance : Desequilibre entre classes dans un probleme de classification (ex : 75% Pass, 25% Fail).

Cross-Validation : Technique de validation consistant a decouper les donnees en K folds, entrainer sur K-1 folds et valider sur le fold restant, puis moyenner les performances.

ETL : Extract, Transform, Load. Processus d'extraction de donnees brutes, transformation/nettoyage, et chargement dans une base de donnees.

Feature Engineering : Creation de variables derivees (features) a partir de donnees brutes pour ameliorer la performance d'un modele ML.

OULAD : Open University Learning Analytics Dataset. Dataset public de 32,593 etudiants et 10M+ interactions.

Precision : Proportion de predictions positives correctes parmi toutes les predictions positives. Precision = TP / (TP + FP).

Random Forest : Algorithme d'ensemble combinant plusieurs arbres de decision pour ameliorer robustesse et performance.

Recall : Proportion de vrais positifs correctement identifies parmi tous les vrais positifs. Recall = TP / (TP + FN).

VLE : Virtual Learning Environment. Plateforme d'apprentissage en ligne (ex : Moodle, Canvas).