# Conception et Développement d’une Application d’Intelligence Artificielle - CSV Analyzer

## Table des Matières

1. [Introduction et Cadrage du Projet](#introduction-et-cadrage-du-projet)
   * [1.1. Vision du Produit](#Xac487c56a76b89c5b0531b6e984c562e951c2e6)
   * [1.2. Objectifs Métier (Business Goals)](#Xcecff31327ebc5ae3fc7544850d52732a0707ae)
   * [1.3. Public Cible (Target Audience)](#X3c87ed1a5274611237010298879393e1cd01af5)
   * [1.4. Périmètre Fonctionnel](#X3fbd3087e66efb65b8679d55286524175372a91)
2. [Analyse des Besoins et Spécifications](#analyse-des-besoins-et-spécifications)
   * [2.1. Besoins des Utilisateurs (User Needs)](#Xc5c8aca452b995f0be82a177c0969753de8b57a)
   * [2.2. User Stories](#Xf50c1e96d0f01b3715d4ea339581e1b04f737b9)
   * [2.3. Spécifications Fonctionnelles Détaillées](#X64da2c1e517b5f6b923edf30c882a5eb82976ea)
   * [2.4. Spécifications Non-Fonctionnelles](#Xb7ed46abbc2c3cea5c50f4f74515024f4922c6d)
3. [Conception de la Solution](#conception-de-la-solution)
   * [3.1. Modélisation des Données](#X1a045e0447e5ade0c23170b1b336a66270f2306)
     + [3.1.1. Schéma Conceptuel de Données (SCD)](#X9d79554d82848bfab4569eb4f3fe0810de7bf57)
     + [3.1.2. Dictionnaire des Données](#Xa11ecc8057014aa36be935fd7c277b1c4faac3c)
   * [3.2. Architecture Logicielle](#X6ae4ac9a199a9ed232b2220741ef4e0de84aa73)
     + [3.2.1. Vue d’Ensemble de l’Architecture (Style MVC/MVT)](#Xd150196d6eb45b0a6961232323f829398108f47)
     + [3.2.2. Description des Composants](#X81159fc2b5e87e8172a4ae816aa6faab9284b80)
   * [3.3. Conception de l’Interface Utilisateur (UI) et de l’Expérience Utilisateur (UX)](#Xc1b8c67303f3fdd00bc1bcce883cccb48d87856)
     + [3.3.1. Parcours Utilisateur (User Journey Mapping)](#X0c75d50723ff3dd6392ea416a28c27173c6cd78)
     + [3.3.2. Wireframes et Maquettes (Mockups)](#X35b07f7ca70ab8435ce01372a1fc447a57b01bb)
     + [3.3.3. Charte Graphique](#Xfeca01684e11470a51e4b1364d7f9bc2e1def42)
4. [Développement de l’Application](#développement-de-lapplication)
   * [4.1. Environnement de Développement](#X00c5303d4e77e34cd4452eea5281172250a3a53)
   * [4.2. Implémentation du Backend](#X538c9c8f89bce98c180aa5df7963391e6c08808)
     + [4.2.1. Gestion des Utilisateurs et Authentification](#Xcdbe47921de58a98fa8f9db39eefcc7c1099382)
     + [4.2.2. Logique Métier du Traitement CSV](#Xfc10d74f163ad2d89124f8fe4196c61b2b03989)
     + [4.2.3. Intégration du Module de Machine Learning](#Xafa33341e30d4dcb2e624ee82e19d2dccc32aa4)
     + [4.2.4. Développement de l’API REST](#X38caefef37eab23930620432c4961553cf220fb)
   * [4.3. Implémentation du Frontend](#X3f65d1002f4f245faa086eec54e888d00c78e12)
     + [4.3.1. Structure des Templates](#Xe1ef78dce4521ef7355ee5ff915381a07025b01)
     + [4.3.2. Intégration avec le Backend](#X4a0b8c8d4c20ee0e1654d2e120b528469f9552d)
   * [4.4. Sécurité](#Xb464f829ac95b970821385185da27218265f7a8)
     + [4.4.1. Mesures de Sécurité Implémentées](#Xdcaf598e006c946eac8c47de7e82d6f73e9cc7b)
5. [Tests et Validation](#tests-et-validation)
   * [5.1. Stratégie de Test](#X69ef32d3b94c814f1a9f1163020392853c844b5)
   * [5.2. Tests Unitaires](#Xe558e6b647c405fd87f4c4d419da0ce72721b42)
   * [5.3. Tests d’Intégration](#X5759bd599137782b29144643d6de9e717e6c18b)
   * [5.4. Tests de Performance](#X642c87c23c47e960b001853871b119c2cb6739f)
   * [5.5. Couverture de Code](#Xed7ddf8df7c43f72611bd340f87107e8223a398)
6. [Déploiement et Intégration Continue (CI/CD)](#déploiement-et-intégration-continue-cicd)
   * [6.1. Conteneurisation avec Docker](#Xb13fe065fa081a7a55b5772500da471c8f00828)
   * [6.2. Orchestration Locale avec Docker Compose](#X5b4edb82bed4eaef6d7bc3a08a8447ecbf177ff)
   * [6.3. Pipeline d’Intégration Continue (GitHub Actions)](#Xe9d9d10836e13a90178c6c76230da702439d602)
7. [Documentation Technique](#documentation-technique)
   * [7.1. Documentation du Code (Docstrings)](#X9751ccb99957197a49630386196b3195a712ffd)
   * [7.2. Documentation de l’API (Swagger/OpenAPI)](#X447f97f7a5cca1199cb386bdcdaeeff247fd23b)
   * [7.3. Guide d’Installation et de Déploiement](#X10704a9be9a043028ee9900d56a42d6dc603e38)
8. [Gestion de Projet et Coordination](#gestion-de-projet-et-coordination)
   * [8.1. Méthodologie (Agile/Scrum)](#X2371879b2fb26ae5dc042a9dfa4c1cc22a92c7e)
   * [8.2. Outils de Suivi (Jira, Trello, GitHub Projects)](#Xbac96e605b7deef630736406b1d7d8951660864)
   * [8.3. Planification et Suivi des Tâches](#X5591529e27c1fbbd3173344ff7618fec7570307)
9. [Conclusion et Perspectives](#conclusion-et-perspectives)
   * [9.1. Bilan du Projet](#X2b3449be2f8acac8d79af00996b7252a664449a)
   * [9.2. Leçons Apprises](#Xf8a8ba3f4b1a7e6c9c9eb9eb910df64993e10d2)
   * [9.3. Perspectives d’Évolution](#X4c982d7cff8ddca454cc7715bdc29a5c0dbfe68)
10. [Annexes](#annexes)
11. [Prétraitement Avancé des Données](#prétraitement-avancé-des-données)
    * [11.1. Gestion des Valeurs Manquantes](#Xdb9154be87dfb29130b9e0f094ebb1ef7329da0)
    * [11.2. Encodage des Variables Catégorielles](#X6a6aba50f3b6d4312c0edb494c60fc08c750157)
    * [11.3. Normalisation et Standardisation](#Xd552e09766eb489d0380685152a5c37551fd842)
    * [11.4. Réduction de Dimensionalité](#X91b9d9a0682a069f46d00407afe58615d70430f)
12. [Stratégies de Déploiement de Modèles](#stratégies-de-déploiement-de-modèles)
    * [12.1. Déploiement en tant qu’API REST (Microservice)](#X3b0ad851885e8b41215f2a0c82a8d449466b914)
    * [12.2. Déploiement en Batch](#X1ecf5c7844f5ebfa701238a930834a3f9fa11a3)
    * [12.3. Déploiement Edge/Embarqué](#Xad968f06bf9a3b65983e8d7e442d6b5b14444c8)
    * [12.4. Serverless Functions](#Xe1c5a877b3231f1db936937479a8a515ae5ce2d)
13. [Intégration MLOps](#intégration-mlops)
    * [13.1. Versionnement des Données et des Modèles](#X0e6ba2b1f76bf71cd986a0d3900b49456c34fcb)
    * [13.2. Pipelines d’Entraînement Automatisées](#Xf9d1b9621918b7ecd5ef8fc6ad27f1472a0a7a0)
    * [13.3. Monitoring de Modèles en Production](#X555b1b85b2efe106538772464d340701e066150)
14. [Tests Avancés et Qualité du Code](#tests-avancés-et-qualité-du-code)
    * [14.1. Tests de Régression](#Xc1018496dd90888c16475b9f9e9b84e9657a93e)
    * [14.2. Tests de Sécurité (SAST/DAST)](#Xff37bb462bdb5f82ab9f8ee2d991050b09ea632)
    * [14.3. Tests de Résilience](#X851bebffa2884568c0ccb7d29cc7aeafe7ad213)
    * [14.4. Revue de Code et Pair Programming](#Xd5a12c2f42eeda8cbf09582054b93a19901f4d7)
15. [Feuille de Route et Améliorations Futures](#X453fae13d2b2982be937721d64a208a659a36a7)
    * [15.1. Fonctionnalités Prioritaires](#Xa6e6f1df256c399fd2683a9ca62f8d436d8a4d6)
    * [15.2. Évolutions Techniques](#X8d80ec14561193cff19b0a527b42aa268545b01)
    * [15.3. Recherche et Développement](#Xab5cc55fb39dd839a076307cd67db4d93153661)

## 1. Introduction et Cadrage du Projet

### 1.1. Vision du Produit

La vision du projet “CSV Analyzer” est de devenir une plateforme web de référence, simple et intuitive, permettant à des utilisateurs ayant peu ou pas de compétences en programmation de tirer parti de la puissance du Machine Learning pour leurs propres jeux de données. L’application doit transformer le processus complexe de la modélisation prédictive en une expérience fluide, guidée et visuelle, depuis l’importation des données jusqu’à l’obtention de prédictions concrètes via une API.

### 1.2. Objectifs Métier (Business Goals)

* **Démocratiser l’IA :** Rendre le Machine Learning accessible à un public plus large (étudiants, analystes métier, chercheurs).
* **Accélérer l’Expérimentation :** Permettre aux utilisateurs de tester rapidement des hypothèses sur leurs données sans investissement initial en infrastructure ou en compétences de codage.
* **Fournir une Solution de Bout en Bout :** Offrir un service complet, de l’analyse exploratoire des données à l’exposition d’un modèle fonctionnel.
* **Créer une Base pour de Futures Évolutions :** Construire une architecture solide pouvant être étendue avec des fonctionnalités plus avancées (Deep Learning, traitement d’autres formats de fichiers, etc.).

### 1.3. Public Cible (Target Audience)

* **Analystes Métier :** Souhaitent enrichir leurs analyses Excel/BI avec des modèles prédictifs sans dépendre d’une équipe de data scientists.
* **Étudiants et Enseignants :** Cherchent un outil pratique pour apprendre et enseigner les concepts du Machine Learning de manière appliquée.
* **Chefs de Produit / Marketeurs :** Veulent analyser des données clients (ex: segmentation, prédiction de churn) de manière autonome.
* **Développeurs non-spécialistes en IA :** Ont besoin d’intégrer rapidement une fonctionnalité de prédiction dans leur propre application.

### 1.4. Périmètre Fonctionnel

Le périmètre de la version initiale (MVP - Minimum Viable Product) inclut :

* Gestion des comptes utilisateurs (inscription, connexion).
* Téléversement de fichiers CSV.
* Visualisation et analyse exploratoire simple du jeu de données.
* Nettoyage basique des données (gestion des valeurs manquantes).
* Entraînement de modèles de classification et de régression.
* Visualisation des performances du modèle.
* Exposition du meilleur modèle via une API REST sécurisée.
* Interface pour tester l’API.

## 2. Analyse des Besoins et Spécifications

### 2.1. Besoins des Utilisateurs (User Needs)

* “En tant qu’analyste, je veux pouvoir charger facilement mes fichiers CSV sans me soucier du formatage.”
* “Je veux comprendre rapidement la structure de mes données : quelles sont les colonnes, leur type, y a-t-il des valeurs manquantes ?”
* “Je veux pouvoir choisir une colonne cible et laisser l’outil me proposer le meilleur modèle possible.”
* “Je veux comprendre si le modèle est performant avec des métriques simples et des graphiques clairs.”
* “Une fois que j’ai un bon modèle, je veux pouvoir l’utiliser facilement pour faire de nouvelles prédictions, idéalement dans mon tableur ou une autre application.”

### 2.2. User Stories

| ID | En tant que… | Je veux… | Afin de… |
| --- | --- | --- | --- |
| **U-01** | Nouvel utilisateur | Créer un compte et me connecter | Sécuriser l’accès à mes données et modèles. |
| **U-02** | Utilisateur connecté | Téléverser un fichier CSV depuis mon ordinateur | Analyser mes propres données. |
| **U-03** | Utilisateur | Visualiser les premières lignes et les informations de base de mon CSV | Vérifier que le fichier a été correctement importé. |
| **U-04** | Utilisateur | Lancer un processus d’entraînement automatique | Obtenir un modèle prédictif sans avoir à choisir l’algorithme. |
| **U-05** | Utilisateur | Voir les résultats de l’entraînement, y compris les métriques de performance | Évaluer la qualité du modèle généré. |
| **U-06** | Utilisateur | Accéder à une interface pour tester mon modèle avec de nouvelles données | Valider le comportement du modèle en temps réel. |
| **U-07** | Utilisateur avancé | Obtenir les informations nécessaires pour appeler mon modèle via une API | Intégrer la prédiction dans mes propres outils. |

### 2.3. Spécifications Fonctionnelles Détaillées

* **SF-01 : Gestion des utilisateurs**
  + Le système doit permettre la création de compte via un formulaire (email, mot de passe).
  + Le système doit permettre la connexion et la déconnexion.
  + Les mots de passe doivent être stockés de manière sécurisée (hachage).
* **SF-02 : Gestion des CSV**
  + L’utilisateur doit pouvoir téléverser des fichiers .csv d’une taille maximale de 50 Mo.
  + Le système doit détecter l’encodage (UTF-8, ISO-8859-1) et le séparateur (virgule, point-virgule).
  + L’utilisateur peut visualiser la liste de ses fichiers et les supprimer.
* **SF-03 : Entraînement de Modèle**
  + L’utilisateur sélectionne un fichier et une colonne cible.
  + Le système détermine automatiquement s’il s’agit d’un problème de classification ou de régression.
  + Le système entraîne plusieurs modèles Scikit-learn en parallèle.
  + Le système utilise MLflow pour logger les paramètres et les métriques de chaque exécution.
* **SF-04 : API de Prédiction**
  + Pour chaque modèle entraîné, une API REST est disponible.
  + L’endpoint de l’API doit être sécurisé (par exemple, par une clé d’API unique par utilisateur).
  + L’API accepte en entrée les données d’une nouvelle observation (au format JSON) et retourne la prédiction.

### 2.4. Spécifications Non-Fonctionnelles

* **SNF-01 : Performance**
  + Le temps de réponse pour les pages web doit être inférieur à 2 secondes.
  + Le temps d’entraînement pour un fichier de 10 Mo ne doit pas dépasser 5 minutes.
* **SNF-02 : Sécurité**
  + L’application doit être protégée contre les failles du Top 10 OWASP (XSS, CSRF, Injections SQL).
  + Les données des utilisateurs doivent être isolées les unes des autres.
* **SNF-03 : Maintenabilité**
  + Le code doit suivre les conventions de style PEP 8.
  + La couverture de tests unitaires doit être supérieure à 80%.
* **SNF-04 : Disponibilité**
  + L’application doit viser une disponibilité de 99.9% en production.

## 3. Conception de la Solution

### 3.1. Modélisation des Données

#### 3.1.1. Schéma Conceptuel de Données (SCD)

erDiagram  
 USER {  
 int id PK  
 string username  
 string password\_hash  
 string email  
 datetime created\_at  
 }  
  
 CSV\_FILE {  
 int id PK  
 int user\_id FK  
 string file\_path  
 string file\_name  
 datetime uploaded\_at  
 }  
  
 ML\_MODEL {  
 int id PK  
 int csv\_file\_id FK  
 string mlflow\_run\_id  
 string model\_name  
 string model\_type "Classification ou Regression"  
 float performance\_metric  
 datetime created\_at  
 }  
  
 USER ||--o{ CSV\_FILE : owns  
 CSV\_FILE ||--o{ ML\_MODEL : has

#### 3.1.2. Dictionnaire des Données

* **USER :** Stocke les informations des utilisateurs enregistrés.
* **CSV\_FILE :** Référence les fichiers CSV téléversés par chaque utilisateur.
  + file\_path : Chemin vers le fichier sur le système de fichiers du serveur.
* **ML\_MODEL :** Stocke les métadonnées des modèles entraînés.
  + mlflow\_run\_id : Identifiant unique de l’exécution dans MLflow, permettant de retrouver tous les détails de l’entraînement.

### 3.2. Architecture Logicielle

#### 3.2.1. Vue d’Ensemble de l’Architecture (Style MVT)

L’application est basée sur le framework Django, qui implémente une variante du patron d’architecture Modèle-Vue-Contrôleur (MVC) appelée Modèle-Vue-Template (MVT).

* **Modèle (Model) :** Définit la structure des données (les classes User, CSVFile, MLModel dans models.py). C’est la seule interface avec la base de données.
* **Vue (View) :** Contient la logique métier. Elle reçoit les requêtes HTTP, interagit avec les modèles, exécute les traitements (ex: lancer un entraînement) et passe les données au template.
* **Template :** La couche de présentation. Ce sont les fichiers HTML qui contiennent la structure de la page et des balises spéciales pour afficher les données passées par la vue.

#### 3.2.2. Description des Composants

Le projet est structuré en plusieurs “apps” Django pour une meilleure séparation des préoccupations :

* **csv\_analyzer (Projet principal) :** Contient la configuration globale du projet (settings.py, urls.py).
* **users (App) :** Gère tout ce qui est lié aux utilisateurs (modèles, vues pour l’inscription/connexion, formulaires).
* **csv\_processor (App) :** Le cœur de l’application. Gère le téléversement, l’analyse, l’entraînement des modèles et l’API.
* **static / templates (Dossiers) :** Contiennent respectivement les fichiers statiques (CSS, JS) et les templates HTML globaux.

### 3.3. Conception de l’Interface Utilisateur (UI) et de l’Expérience Utilisateur (UX)

#### 3.3.1. Parcours Utilisateur (User Journey Mapping)

1. **Découverte :** L’utilisateur arrive sur la page d’accueil qui présente l’application.
2. **Inscription :** Il crée un compte.
3. **Premier Upload :** Il est guidé vers la page d’upload pour téléverser son premier fichier.
4. **Analyse :** Il visualise son jeu de données et lance l’entraînement.
5. **Résultats :** Il consulte la page de résultats avec les performances du modèle.
6. **Test :** Il utilise l’interface de test pour faire une prédiction.
7. **Intégration :** Il récupère l’URL de l’API et sa clé pour l’utiliser ailleurs.

#### 3.3.2. Wireframes et Maquettes (Mockups)

*(Cette section contiendrait des schémas basse-fidélité (wireframes) et haute-fidélité (mockups) pour chaque écran principal de l’application : page de connexion, tableau de bord, page d’upload, page de visualisation des données, page des résultats du modèle.)*

**Exemple de Wireframe pour la page d’Upload :**

+----------------------------------------------------+  
| [Logo] CSV Analyzer [Profil Utilisateur] |  
+----------------------------------------------------+  
| |  
| +-----------------------------------+ |  
| | | |  
| | Glissez-déposez votre fichier | |  
| | ou | |  
| | [ Parcourir les fichiers ] | |  
| | | |  
| +-----------------------------------+ |  
| |  
| [Liste des fichiers déjà uploadés] |  
| - fichier1.csv [Supprimer] |  
| - données\_clients.csv [Supprimer] |  
| |  
+----------------------------------------------------+

#### 3.3.3. Charte Graphique

*(Cette section décrirait les choix de couleurs, typographies, icônes et principes de design pour assurer une cohérence visuelle et une expérience utilisateur agréable.)*

## 4. Développement de l’Application

### 4.1. Environnement de Développement

* **Langage :** Python 3.9+
* **Framework :** Django 4.x
* **Base de Données :** PostgreSQL (en développement, SQLite peut être utilisé)
* **Gestionnaire de Paquets :** pip
* **Conteneurisation :** Docker
* **IDE :** VS Code, PyCharm

### 4.2. Implémentation du Backend

#### 4.2.1. Gestion des Utilisateurs et Authentification

Utilisation du système d’authentification intégré de Django, étendu si nécessaire avec un modèle UserProfile pour des informations additionnelles.

# users/views.py  
from django.contrib.auth.forms import UserCreationForm  
from django.urls import reverse\_lazy  
from django.views import generic  
  
class SignUpView(generic.CreateView):  
 form\_class = UserCreationForm  
 success\_url = reverse\_lazy("login")  
 template\_name = "registration/register.html"

#### 4.2.2. Logique Métier du Traitement CSV

Le traitement des CSV est géré dans l’application csv\_processor. Cela inclut la lecture du fichier, la détection du type de données, la gestion des valeurs manquantes et la préparation des données pour l’entraînement.

# csv\_processor/utils.py  
import pandas as pd  
  
def load\_and\_preprocess\_csv(file\_path):  
 df = pd.read\_csv(file\_path)  
 # Logique de nettoyage et de prétraitement  
 return df

#### 4.2.3. Intégration du Module de Machine Learning

Le module de ML utilise scikit-learn pour l’entraînement et MLflow pour le suivi des expérimentations. Les modèles sont sérialisés avec joblib.

# csv\_processor/train\_model.py  
from sklearn.ensemble import RandomForestClassifier  
import mlflow  
import joblib  
  
def train\_and\_log\_model(X, y, model\_name="RandomForest"):  
 with mlflow.start\_run():  
 model = RandomForestClassifier()  
 model.fit(X, y)  
 mlflow.sklearn.log\_model(model, model\_name)  
 # Sauvegarde locale pour l'API  
 joblib.dump(model, f"media/models/{model\_name}.pkl")  
 return model

#### 4.2.4. Développement de l’API REST

Utilisation de Django REST Framework pour créer des endpoints RESTful pour l’upload, l’entraînement et la prédiction.

# csv\_processor/views.py  
from rest\_framework.views import APIView  
from rest\_framework.response import Response  
from rest\_framework import status  
  
class CSVUploadView(APIView):  
 def post(self, request, format=None):  
 # Logique d'upload et de traitement  
 return Response({"status": "success"}, status=status.HTTP\_201\_CREATED)

### 4.3. Implémentation du Frontend

#### 4.3.1. Structure des Templates

Les templates Django sont organisés pour la réutilisation, avec un base.html et des blocs spécifiques pour chaque page.

<!-- templates/base.html -->  
<!DOCTYPE html>  
<html>  
<head>  
 <title>{% block title %}CSV Analyzer{% endblock %}</title>  
 <!-- CSS, JS -->  
</head>  
<body>  
 <nav><!-- Navigation Bar --></nav>  
 <main>  
 {% block content %}{% endblock %}  
 </main>  
</body>  
</html>

#### 4.3.2. Intégration avec le Backend

Les formulaires Django sont utilisés pour l’interaction avec le backend, et des requêtes AJAX peuvent être utilisées pour des interactions asynchrones (ex: affichage de la progression de l’entraînement).

### 4.4. Sécurité

#### 4.4.1. Mesures de Sécurité Implémentées

* **Protection CSRF :** Django intègre une protection contre les attaques Cross-Site Request Forgery.
* **Protection XSS :** Les templates Django échappent automatiquement le contenu pour prévenir les attaques Cross-Site Scripting.
* **Hachage des Mots de Passe :** Utilisation d’algorithmes de hachage robustes pour les mots de passe.
* **Permissions :** Contrôle d’accès basé sur les rôles pour les fonctionnalités critiques.

## 5. Tests et Validation

### 5.1. Stratégie de Test

Une stratégie de test complète est essentielle pour garantir la qualité et la fiabilité de l’application. Nous adoptons une approche multi-niveaux :

* **Tests Unitaires :** Vérifient le bon fonctionnement des plus petites unités de code (fonctions, méthodes).
* **Tests d’Intégration :** S’assurent que les différents modules interagissent correctement entre eux.
* **Tests Fonctionnels :** Valident les fonctionnalités de l’application du point de vue de l’utilisateur.
* **Tests de Performance :** Évaluent la réactivité et la stabilité de l’application sous charge.

### 5.2. Tests Unitaires

Utilisation de pytest et du module unittest de Python pour les tests unitaires. Les tests couvrent les modèles, les vues, les formulaires et les utilitaires.

# csv\_processor/tests/test\_models.py  
from django.test import TestCase  
from csv\_processor.models import CSVFile  
  
class CSVFileModelTest(TestCase):  
 def test\_csv\_file\_creation(self):  
 csv\_file = CSVFile.objects.create(file\_name="test.csv", file\_path="/path/to/test.csv")  
 self.assertEqual(csv\_file.file\_name, "test.csv")

### 5.3. Tests d’Intégration

Les tests d’intégration vérifient les interactions entre les composants, par exemple, l’upload d’un fichier CSV et le déclenchement de l’entraînement du modèle.

# csv\_processor/tests/test\_views.py  
from django.test import TestCase, Client  
from django.urls import reverse  
  
class CSVUploadViewTest(TestCase):  
 def setUp(self):  
 self.client = Client()  
 self.upload\_url = reverse('csv\_upload') # Assurez-vous que 'csv\_upload' est défini dans urls.py  
  
 def test\_csv\_upload\_success(self):  
 with open('test\_data.csv', 'rb') as fp:  
 response = self.client.post(self.upload\_url, {'file': fp})  
 self.assertEqual(response.status\_code, 201) # HTTP 201 Created

### 5.4. Tests de Performance

Utilisation d’outils comme Locust ou JMeter pour simuler un grand nombre d’utilisateurs et mesurer les temps de réponse et la capacité du système.

### 5.5. Couverture de Code

Utilisation de coverage.py pour mesurer la proportion du code source exécutée par les tests. L’objectif est de maintenir une couverture élevée (ex: > 80%) pour s’assurer que la majorité du code est testée.

## 6. Déploiement et Intégration Continue (CI/CD)

### 6.1. Conteneurisation avec Docker

Docker est utilisé pour empaqueter l’application et ses dépendances dans des conteneurs isolés, garantissant un environnement d’exécution cohérent de développement à production.

# Dockerfile  
FROM python:3.9-slim-buster  
WORKDIR /app  
COPY requirements.txt .  
RUN pip install --no-cache-dir -r requirements.txt  
COPY . .  
EXPOSE 8000  
CMD ["gunicorn", "csv\_analyzer.wsgi:application", "--bind", "0.0.0.0:8000"]

### 6.2. Orchestration Locale avec Docker Compose

docker-compose est utilisé pour définir et exécuter des applications multi-conteneurs (application Django, base de données PostgreSQL, MLflow).

# docker-compose.yml  
version: '3.8'  
services:  
 web:  
 build: .  
 ports:  
 - "8000:8000"  
 depends\_on:  
 - db  
 db:  
 image: postgres:13  
 environment:  
 POSTGRES\_DB: mydatabase  
 POSTGRES\_USER: user  
 POSTGRES\_PASSWORD: password

### 6.3. Pipeline d’Intégration Continue (GitHub Actions)

Un workflow GitHub Actions est configuré pour automatiser les tests et le build de l’image Docker à chaque push sur le dépôt.

# .github/workflows/ci.yml  
name: CI/CD Pipeline  
on:  
 push:  
 branches: [ main ]  
 pull\_request:  
 branches: [ main ]  
jobs:  
 build:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v3  
 - name: Set up Python  
 uses: actions/setup-python@v3  
 with:  
 python-version: '3.9'  
 - name: Install dependencies  
 run: pip install -r requirements.txt  
 - name: Run tests  
 run: pytest  
 - name: Build Docker image  
 run: docker build . --tag my-app:latest

## 7. Documentation Technique

### 7.1. Documentation du Code (Docstrings)

Chaque fonction, classe et méthode est documentée à l’aide de docstrings conformes à la PEP 257, facilitant la compréhension et la maintenance du code.

# Exemple de docstring  
def calculate\_accuracy(predictions, true\_labels):  
 """  
 Calcule la précision d'un modèle de classification.  
  
 Args:  
 predictions (list): Liste des prédictions du modèle.  
 true\_labels (list): Liste des vraies étiquettes.  
  
 Returns:  
 float: La précision calculée.  
 """  
 # ... implémentation

### 7.2. Documentation de l’API (Swagger/OpenAPI)

Utilisation de drf-yasg ou drf-spectacular pour générer automatiquement une documentation interactive de l’API au format OpenAPI (anciennement Swagger), permettant aux développeurs tiers de comprendre et d’interagir facilement avec l’API.

### 7.3. Guide d’Installation et de Déploiement

Un fichier README.md détaillé fournit les instructions étape par étape pour configurer l’environnement de développement, installer les dépendances, exécuter les tests et déployer l’application localement avec Docker Compose.

## 8. Gestion de Projet et Coordination

### 8.1. Méthodologie (Agile/Scrum)

Le projet est géré selon la méthodologie Agile, avec des sprints courts (ex: 2 semaines), des réunions quotidiennes (Daily Scrum), des revues de sprint et des rétrospectives. Cette approche favorise la flexibilité, l’adaptation aux changements et la livraison incrémentale de valeur.

### 8.2. Outils de Suivi (Jira, Trello, GitHub Projects)

Utilisation de GitHub Projects pour le suivi des tâches, la gestion du backlog, l’attribution des issues et la visualisation de l’avancement du projet via des tableaux Kanban.

### 8.3. Planification et Suivi des Tâches

Les tâches sont décomposées en User Stories, estimées et assignées aux membres de l’équipe. L’avancement est suivi quotidiennement et ajusté lors des Daily Scrums.

## 9. Conclusion et Perspectives

### 9.1. Bilan du Projet

Le projet “CSV Analyzer” a permis de développer une application web fonctionnelle, capable de gérer le cycle de vie complet de l’analyse de données CSV avec des modèles de Machine Learning. Nous avons mis en œuvre une architecture robuste, des pratiques de développement modernes (tests, CI/CD) et une attention particulière à l’expérience utilisateur.

### 9.2. Leçons Apprises

* L’importance d’une bonne gestion des dépendances et des environnements (Docker).
* La valeur de l’intégration continue pour détecter rapidement les régressions.
* La complexité de la gestion des données et des modèles de ML en production.
* La nécessité d’une communication claire et régulière au sein de l’équipe.

### 9.3. Perspectives d’Évolution

* **Amélioration de l’UI/UX :** Tableaux de bord plus interactifs, visualisations de données avancées.
* **Support de Modèles Avancés :** Intégration de modèles de Deep Learning (TensorFlow, PyTorch).
* **AutoML plus Poussé :** Optimisation des hyperparamètres, sélection automatique de fonctionnalités.
* **Déploiement Cloud :** Intégration avec des plateformes cloud (AWS SageMaker, Azure ML) pour un déploiement et un scaling facilités.
* **Gestion des Versions de Données :** Utilisation d’outils comme DVC (Data Version Control).

## 10. Annexes

*(Cette section contiendrait des informations supplémentaires telles que des diagrammes d’architecture détaillés, des extraits de code plus longs, des captures d’écran de l’application, ou des références à des ressources externes.)*

## 11. Prétraitement Avancé des Données

### 11.1. Gestion des Valeurs Manquantes

* **Imputation Statistique :** Remplacement par la moyenne, la médiane ou le mode.
* **Imputation par Modèle :** Utilisation de modèles prédictifs (ex: IterativeImputer de Scikit-learn) pour estimer les valeurs manquantes.
* **Suppression :** Suppression des lignes ou colonnes avec trop de valeurs manquantes (à utiliser avec prudence).

### 11.2. Encodage des Variables Catégorielles

* **One-Hot Encoding :** Pour les variables nominales (sans ordre).
* **Label Encoding / Ordinal Encoding :** Pour les variables ordinales (avec ordre).
* **Target Encoding :** Encodage basé sur la variable cible (peut introduire un surapprentissage).

### 11.3. Normalisation et Standardisation

* **Standardisation (Z-score) :** Rend les données avec une moyenne de 0 et un écart-type de 1. Utile pour les algorithmes sensibles à l’échelle (SVM, K-Means).
* **Normalisation (Min-Max Scaling) :** Met les données dans une plage fixe (ex: [0, 1]).

### 11.4. Réduction de Dimensionalité

* **Analyse en Composantes Principales (PCA) :** Transforme les données en un nouvel ensemble de variables non corrélées, tout en conservant la majeure partie de la variance.
* **Sélection de Caractéristiques :** Utilisation de méthodes statistiques (tests Chi-2, ANOVA) ou basées sur des modèles (Feature Importance des forêts aléatoires) pour sélectionner les variables les plus pertinentes.

## 12. Stratégies de Déploiement de Modèles

### 12.1. Déploiement en tant qu’API REST (Microservice)

* **Avantages :** Indépendance, scalabilité, réutilisation.
* **Défis :** Latence réseau, gestion des versions d’API.

### 12.2. Déploiement en Batch

* Pour les prédictions non urgentes sur de grands volumes de données.
* Utilisation de jobs planifiés (ex: Cron, Airflow) pour traiter les données et stocker les résultats.

### 12.3. Déploiement Edge/Embarqué

* Pour les applications nécessitant des prédictions en temps réel avec une faible latence, directement sur l’appareil (ex: mobile, IoT).
* Conversion des modèles en formats optimisés (ex: ONNX, TensorFlow Lite).

### 12.4. Serverless Functions

* Déploiement de modèles en tant que fonctions sans serveur (AWS Lambda, Azure Functions) pour une scalabilité automatique et un paiement à l’usage.

## 13. Intégration MLOps

### 13.1. Versionnement des Données et des Modèles

* **DVC (Data Version Control) :** Pour versionner les jeux de données et les pipelines de prétraitement.
* **MLflow Model Registry :** Pour gérer le cycle de vie des modèles (staging, production, archivé).

### 13.2. Pipelines d’Entraînement Automatisées

* Utilisation d’outils comme **Kubeflow Pipelines** ou **Airflow** pour orchestrer les étapes d’entraînement, de validation et de déploiement des modèles.

### 13.3. Monitoring de Modèles en Production

* **Détection de Dérive (Drift Detection) :** Surveillance des changements dans la distribution des données d’entrée (Data Drift) ou dans la relation entre les entrées et les sorties (Concept Drift).
* **Explicabilité des Modèles (XAI) :** Utilisation de techniques comme SHAP ou LIME pour comprendre pourquoi un modèle a fait une prédiction donnée, essentiel pour le débogage et la conformité.

## 14. Tests Avancés et Qualité du Code

### 14.1. Tests de Régression

* S’assurer que les nouvelles modifications n’introduisent pas de bugs dans les fonctionnalités existantes.
* Automatisation des tests de régression dans la pipeline CI/CD.

### 14.2. Tests de Sécurité (SAST/DAST)

* **SAST (Static Application Security Testing) :** Analyse du code source pour détecter les vulnérabilités (ex: Bandit pour Python).
* **DAST (Dynamic Application Security Testing) :** Teste l’application en cours d’exécution pour trouver des vulnérabilités (ex: OWASP ZAP).

### 14.3. Tests de Résilience

* **Chaos Engineering :** Introduction délibérée de pannes dans le système pour tester sa robustesse et sa capacité à récupérer (ex: Chaos Monkey).

### 14.4. Revue de Code et Pair Programming

* Pratiques essentielles pour améliorer la qualité du code, partager les connaissances et détecter les erreurs tôt dans le cycle de développement.

## 15. Feuille de Route et Améliorations Futures

### 15.1. Fonctionnalités Prioritaires

* **Visualisation Avancée des Données :** Intégration de bibliothèques comme Plotly ou Bokeh pour des graphiques interactifs.
* **Support Multi-Modèle :** Permettre à l’utilisateur de choisir parmi différents algorithmes de ML.
* **Interface de Nettoyage de Données :** Outils visuels pour la gestion des valeurs manquantes, l’encodage, etc.

### 15.2. Évolutions Techniques

* **Migration vers un Cloud Provider :** Déploiement sur AWS, Azure ou GCP pour une scalabilité et une disponibilité accrues.
* **Architecture Microservices :** Découper l’application en services plus petits et indépendants.
* **Base de Données NoSQL :** Exploration de bases de données NoSQL pour certains types de données (ex: logs, métriques).

### 15.3. Recherche et Développement

* **AutoML Avancé :** Intégration de frameworks AutoML plus sophistiqués (ex: Auto-Sklearn, H2O.ai).
* **Traitement du Langage Naturel (NLP) :** Extension pour l’analyse de données textuelles.
* **Vision par Ordinateur :** Potentielle intégration pour l’analyse d’images (si pertinent pour les cas d’usage CSV).