# Veille Technologique - Projet AutoML-Gr4-1

## Table des Matières

1. [Introduction](#introduction)
   * [1.1. Contexte du Projet](#X588776d890705d2aad81917249297cfe25c5562)
   * [1.2. Objectifs de la Veille Technologique](#X5d2ad83d7e62efa93a38874ffdffb988778007d)
   * [1.3. Méthodologie de la Veille](#Xe821eba3433182670822902061c3964f966e9c0)
2. [Analyse des Outils et Technologies](#analyse-des-outils-et-technologies)
   * [2.1. Frameworks de Développement Web](#Xe8c9b6b552e59c61d68271afb3af8316eae8b17)
     + [2.1.1. Django](#Xe6a3a09cbddf27e9b9438cb666ffdd7a3ae1914)
     + [2.1.2. Flask](#X533acea10557793f321ea2463c8b8d0cd8edd1e)
     + [2.1.3. FastAPI](#X14eed159919f4c97a1e952d93cc9819a04e2e9d)
     + [2.1.4. Tableau Comparatif et Choix](#X284098dea285029f94bd3a65e101471032f05e7)
   * [2.2. Bibliothèques de Machine Learning](#X9d0954b34825861f94d51b6ff215c6bf2350095)
     + [2.2.1. Scikit-learn](#X704e217faf9542051a9e3f4a41913c08a2cc600)
     + [2.2.2. TensorFlow](#X25a46aa0e0ed9f9996567ba736ff34e7d5467fb)
     + [2.2.3. PyTorch](#X8f01a852ad24120ba4963c95c58d9bab2d136ef)
     + [2.2.4. Tableau Comparatif et Choix](#Xac79d38aa07d02f948c270af3b55fabf6ac0f3b)
   * [2.3. Outils de MLOps](#X64f616563b6e788f8912f0d398ae32b35bffa30)
     + [2.3.1. MLflow](#X5fb4d4137dea669020dfc746af725ea2a3dde96)
     + [2.3.2. Kubeflow](#X03d3dc68340de2532620deeff6f5785fffee3bd)
     + [2.3.3. DVC (Data Version Control)](#X1ee028bb304f437ef0becd7d2251e171d4af97a)
     + [2.3.4. Tableau Comparatif et Choix](#Xbeb6beb427afbb115d5636c623ec2f4bbdae3de)
   * [2.4. Solutions de Conteneurisation](#X4596c5fd8eefb15cbeb97e545fc68a6777a1def)
     + [2.4.1. Docker](#X46a1d3f4cf03b2866ac9d3a6678444b72e81860)
     + [2.4.2. Podman](#Xd110a3abbaf12e0c26e271094c695ae92f73856)
     + [2.4.3. Tableau Comparatif et Choix](#Xe82e2e58ae3b1b50cb53a01a5e53b53b01b502f)
   * [2.5. Systèmes de Monitoring](#Xc2b71723aeb03f0ea65b4db40a1bbb746bd3bb4)
     + [2.5.1. Prometheus et Grafana](#X6bd00d7f1af0e3e16ee7b7fdbf247a22dc5d670)
     + [2.5.2. Datadog](#X62c9dce3cf79591713caac0f95b337013c5bd3a)
     + [2.5.3. ELK Stack (Elasticsearch, Logstash, Kibana)](#X04822b7a547dc4a767e6ecb0fbcc3df4893d9a9)
     + [2.5.4. Tableau Comparatif et Choix](#X75b5e36bbc119780a3e29ad7dbf3afef95616aa)
3. [Architecture Technique Cible](#architecture-technique-cible)
   * [3.1. Schéma d’Architecture Global](#Xe296dc97253d1099474d877341c776a9a717970)
   * [3.2. Justification des Choix d’Architecture](#X676014fdc9080779316d8a309372996a4ffb3ea)
   * [3.3. Flux de Données](#X8b23d9e78214ca86756832b024dd9c3eafdd532)
4. [Feuille de Route (Roadmap)](#feuille-de-route-roadmap)
   * [4.1. Phase 1 : Développement Initial et PoC (Preuve de Concept)](#Xef4c158e1030fdf345b8dae986a6d007d216c7e)
   * [4.2. Phase 2 : Intégration MLOps et API](#Xc4bf08e292fde8df547eb028383a7b0f4c0de26)
   * [4.3. Phase 3 : Monitoring et Déploiement Continu](#Xb53bb316333156fc1d268bc96604be5d2dcea25)
   * [4.4. Phase 4 : Améliorations et Évolutions Futures](#X37b4707f9c0a37d2307c879ec2b574c447ca4ab)
5. [Analyse des Risques et Stratégies d’Atténuation](#Xaac222f9a71a683d8b424d5cc07c2bcdceb9324)
   * [5.1. Risques Techniques](#Xa112be6f57c35b456645f7783dba066957c6f91)
   * [5.2. Risques Organisationnels](#Xaec8a6aef0b89e5a96d8cc4a232ea5ad7ac984b)
   * [5.3. Risques liés à la Donnée](#Xae741793e662ae6e7e4a37711564f3bef8c8b69)
6. [Indicateurs Clés de Performance (KPIs)](#indicateurs-clés-de-performance-kpis)
   * [6.1. KPIs Produit](#X669d41fb4ffec0cfffc52b271fbaea0487fb6e7)
   * [6.2. KPIs Techniques](#Xc1aed921ee650579b0026e580a05f7cd7fa4a8f)
   * [6.3. KPIs de Performance du Modèle](#X8abb041218d8067ec67f389fe4257c328378fc2)
7. [Conclusion](#conclusion)
8. [Bibliographie et Ressources](#bibliographie-et-ressources)
9. [Glossaire](#glossaire)
10. [Annexes](#annexes)

## 1. Introduction

### 1.1. Contexte du Projet

Le projet AutoML-Gr4-1, baptisé “CSV Analyzer”, vise à développer une application web permettant de simplifier et d’automatiser le processus de création de modèles de Machine Learning à partir de fichiers CSV. Dans un contexte où la data science se démocratise, il existe un besoin croissant d’outils “low-code” ou “no-code” qui permettent à des utilisateurs non-experts (analystes métier, étudiants, etc.) de tirer de la valeur de leurs données sans avoir à écrire de code complexe.

L’application doit permettre de téléverser un jeu de données, de le visualiser, de le nettoyer, d’entraîner un modèle de classification ou de régression, d’évaluer ses performances et, en fin, de l’exposer via une API pour des prédictions futures. Ce projet s’inscrit donc au carrefour du développement web, du Machine Learning et des pratiques MLOps (Machine Learning Operations).

### 1.2. Objectifs de la Veille Technologique

Cette veille technologique a pour objectif principal de **justifier les choix technologiques** qui seront faits pour la réalisation du projet. Elle vise à :

* **Identifier et analyser** les outils, frameworks et plateformes pertinents pour chaque composant de l’application (backend, frontend, ML, MLOps, monitoring).
* **Comparer** les différentes options sur la base de critères objectifs (performance, maturité, communauté, facilité d’utilisation, coût, etc.).
* **Sélectionner la stack technique** la plus adaptée aux besoins spécifiques du projet “CSV Analyzer”.
* **Définir une architecture technique cible** cohérente et évolutive.
* **Anticiper les risques** techniques et proposer des stratégies pour les atténuer.

Cette démarche structurée est essentielle pour garantir la pérennité, la maintenabilité et la scalabilité de la solution développée.

### 1.3. Méthodologie de la Veille

La méthodologie de veille adoptée pour ce rapport repose sur un processus itératif en plusieurs étapes :

1. **Définition des besoins :** Lister les grandes fonctionnalités de l’application pour identifier les catégories d’outils à étudier.
2. **Sourcing de l’information :** Consulter une variété de sources pour collecter des informations fiables et à jour :
   * **Documentation officielle** des outils.
   * **Articles de blogs techniques** (ex: Medium, Towards Data Science).
   * **Forums et communautés** (ex: Stack Overflow, Reddit).
   * **Rapports d’analystes** (ex: Gartner, Forrester).
   * **Comparatifs et benchmarks** publiés.
3. **Analyse et Synthèse :** Pour chaque catégorie d’outils, créer des tableaux comparatifs basés sur des critères prédéfinis. Rédiger une synthèse des avantages et inconvénients de chaque option.
4. **Prise de Décision :** Sur la base de l’analyse, formuler une recommandation argumentée pour le choix de chaque technologie.
5. **Formalisation :** Rédiger le présent rapport pour documenter l’ensemble de la démarche et des décisions prises.

Cette veille se veut continue et sera mise à jour si de nouvelles technologies ou de nouvelles versions majeures des outils étudiés apparaissent pendant la durée du projet.

## 2. Analyse des Outils et Technologies

Cette section constitue le cœur de la veille technologique. Nous allons passer en revue les principales options pour chaque brique de notre application.

### 2.1. Frameworks de Développement Web

Le framework web est le socle de notre application. Il gérera les requêtes HTTP, le routage, l’interaction avec la base de données et le rendu des pages.

#### 2.1.1. Django

* **Description :** Django est un framework web Python de haut niveau qui encourage un développement rapide et une conception propre et pragmatique. Il suit le motif de conception Modèle-Vue-Template (MVT).
* **Avantages :**
  + **“Batteries included” :** Fournit une grande partie des fonctionnalités nécessaires “out-of-the-box” (ORM, admin, authentification, etc.), ce qui accélère le développement.
  + **Sécurité :** Intègre des protections contre les failles de sécurité courantes (XSS, CSRF, SQL injection).
  + **Maturité et Stabilité :** Très mature, avec une large communauté et une excellente documentation.
  + **Scalabilité :** Utilisé par des sites à très fort trafic comme Instagram et Pinterest.
* **Inconvénients :**
  + **Monolithique :** Peut être perçu comme trop lourd ou rigide pour de très petites applications (microservices).
  + **Courbe d’apprentissage :** Sa nature “tout compris” peut être intimidante pour les débutants.

#### 2.1.2. Flask

* **Description :** Flask est un micro-framework Python. Il est beaucoup plus minimaliste que Django et laisse une grande liberté au développeur pour choisir ses outils et bibliothèques.
* **Avantages :**
  + **Léger et Flexible :** Idéal pour les petites applications, les API ou les microservices.
  + **Facile à apprendre :** Sa simplicité permet une prise en main très rapide.
  + **Extensible :** Un vaste écosystème d’extensions permet d’ajouter des fonctionnalités au besoin.
* **Inconvénients :**
  + **Nécessite plus de configuration :** Le développeur doit assembler et configurer lui-même de nombreux composants (ORM, formulaires, etc.), ce qui peut être plus long.
  + **Moins de garde-fous :** La flexibilité peut conduire à des architectures moins structurées si le développeur n’est pas rigoureux.

#### 2.1.3. FastAPI

* **Description :** FastAPI est un framework web moderne, haute performance, basé sur les standards Python 3.6+ (type hints) et conçu pour créer des API.
* **Avantages :**
  + **Performance :** L’un des frameworks Python les plus rapides, grâce à Starlette (pour la partie web) et Pydantic (pour la validation des données).
  + **Documentation automatique :** Génère automatiquement une documentation interactive (Swagger UI / OpenAPI) à partir du code.
  + **Facilité d’utilisation :** Très intuitif, surtout pour ceux qui sont familiers avec les “type hints” de Python.
* **Inconvénients :**
  + **Focalisé sur les API :** Moins adapté pour les applications web traditionnelles avec rendu de templates côté serveur (bien que cela soit possible).
  + **Plus jeune :** Bien que très populaire, il est plus jeune que Django et Flask et son écosystème est moins mature.

#### 2.1.4. Tableau Comparatif et Choix

| Critère | Django | Flask | FastAPI |
| --- | --- | --- | --- |
| **Philosophie** | Batteries included | Micro-framework | API-first, performance |
| **Courbe d’apprentissage** | Moyenne | Faible | Faible à Moyenne |
| **Performance** | Bonne | Bonne | Excellente |
| **Développement initial** | Très rapide | Rapide | Très rapide (pour les API) |
| **Sécurité intégrée** | Élevée | Moyenne (via extensions) | Élevée |
| **Communauté** | Très grande et mature | Grande | En forte croissance |
| **Cas d’usage idéal** | Applications web complètes | Microservices, petites apps | API REST haute performance |

**Choix pour le projet “CSV Analyzer” :** **Django**

**Justification :** Le projet “CSV Analyzer” est une application web complète qui nécessite une gestion des utilisateurs, une interface d’administration, des interactions avec une base de données et des formulaires complexes. L’approche “batteries included” de Django permettra d’accélérer considérablement le développement en fournissant des composants robustes et sécurisés dès le départ. La structure claire (MVT) imposée par le framework sera également un atout pour la maintenabilité du projet.

### 2.2. Bibliothèques de Machine Learning

Le cœur de notre application est sa capacité à entraîner des modèles de Machine Learning.

#### 2.2.1. Scikit-learn

* **Description :** C’est la bibliothèque de référence en Machine Learning classique en Python. Elle fournit une large gamme d’algorithmes de classification, régression, clustering, et des outils de prétraitement des données, de sélection de modèles et d’évaluation.
* **Avantages :**
  + **API unifiée et simple :** L’interface .fit(), .predict(), .transform() est cohérente à travers tous les estimateurs.
  + **Excellente documentation :** Considérée comme l’une des meilleures documentations dans l’écosystème Python.
  + **Robuste et fiable :** Très bien testée et optimisée pour la performance (repose sur NumPy, SciPy et Cython).
* **Inconvénients :**
  + **Pas de support GPU :** N’est pas conçu pour tirer parti des GPU.
  + **Limité au ML classique :** Ne couvre pas le Deep Learning.

#### 2.2.2. TensorFlow

* **Description :** Développée par Google, TensorFlow est une plateforme open-source de bout en bout pour le Machine Learning, particulièrement puissante pour le Deep Learning.
* **Avantages :**
  + **Scalabilité :** Conçue pour le déploiement à grande échelle et sur des clusters (support GPU et TPU).
  + **Écosystème complet (TFX) :** Offre des outils pour l’ensemble du cycle de vie MLOps.
  + **Flexibilité :** L’API Keras, intégrée à TensorFlow, rend la construction de réseaux de neurones très accessible.
* **Inconvénients :**
  + **Complexité :** Peut être complexe à maîtriser, surtout pour des tâches de ML classique.
  + **Verbosité :** Le code peut être plus verbeux que celui de Scikit-learn pour des modèles simples.

#### 2.2.3. PyTorch

* **Description :** Développée par le laboratoire de recherche en IA de Facebook, PyTorch est une autre bibliothèque majeure de Deep Learning, très populaire dans le monde de la recherche.
* **Avantages :**
  + **Approche “Pythonique” :** Très intuitive et facile à déboguer.
  + **Graphe de calcul dynamique :** Offre une grande flexibilité pour la conception de modèles complexes.
  + **Forte communauté de recherche :** Souvent, les derniers modèles de recherche sont d’abord implémentés en PyTorch.
* **Inconvénients :**
  + **Moins d’outils de déploiement :** Historiquement, l’écosystème pour la mise en production était moins mature que celui de TensorFlow, bien que cela change rapidement avec TorchServe.
  + **Surdimensionné pour le ML classique.**

#### 2.2.4. Tableau Comparatif et Choix

| Critère | Scikit-learn | TensorFlow | PyTorch |
| --- | --- | --- | --- |
| **Domaine principal** | Machine Learning Classique | Deep Learning, Production | Deep Learning, Recherche |
| **Facilité d’utilisation** | Très élevée | Moyenne (élevée avec Keras) | Élevée |
| **Support GPU** | Non | Oui | Oui |
| **Déploiement** | Simple (pickle) | Écosystème complet (TFX) | En amélioration (TorchServe) |
| **Communauté** | Très grande | Très grande | Très grande |

**Choix pour le projet “CSV Analyzer” :** **Scikit-learn**

**Justification :** Le besoin du projet est de fournir des modèles de classification et de régression standards sur des données tabulaires (CSV). Scikit-learn est l’outil parfait pour cela. Il est simple à utiliser, robuste, et son API s’intégrera très facilement dans notre application Django. Utiliser TensorFlow ou PyTorch serait surdimensionné et inutilement complexe pour les besoins actuels du projet.

#### 2.2.5. Modèles d’Intelligence Artificielle Utilisés

Dans le cadre du projet CSV Analyzer, nous avons sélectionné et implémenté plusieurs modèles d’intelligence artificielle pour répondre à différents besoins. Voici une analyse comparative de ces modèles et les raisons de leur sélection.

##### 2.2.5.1. Modèles de Classification

* **DecisionTreeClassifier :**
  + **Description :** Un modèle d’arbre de décision qui prédit la classe d’une cible en apprenant des règles de décision simples déduites des caractéristiques des données.
  + **Avantages :**
    - **Interprétabilité :** Les règles de décision sont faciles à comprendre et à expliquer aux utilisateurs non-techniques.
    - **Efficacité sur les données tabulaires :** Particulièrement adapté aux données structurées comme les CSV.
    - **Pas de prétraitement complexe :** Fonctionne bien avec des données mixtes (numériques et catégorielles) sans normalisation.
  + **Cas d’utilisation dans notre projet :** Utilisé pour le modèle de nettoyage automatique des données (SimpleCleanModel) qui suggère des actions de prétraitement en fonction des caractéristiques des colonnes.
* **RandomForestClassifier :**
  + **Description :** Un ensemble d’arbres de décision qui améliore la précision et réduit le risque de surapprentissage.
  + **Avantages :**
    - **Performance supérieure :** Généralement plus précis que les arbres de décision simples.
    - **Robustesse :** Moins sensible au bruit et au surapprentissage.
    - **Importance des caractéristiques :** Fournit une mesure de l’importance relative des variables.
  + **Cas d’utilisation dans notre projet :** Proposé comme option pour les tâches de classification générale sur les données utilisateur.

##### 2.2.5.2. Modèles de Régression

* **LinearRegression :**
  + **Description :** Modèle qui suppose une relation linéaire entre les variables d’entrée et la variable cible.
  + **Avantages :**
    - **Simplicité :** Facile à comprendre et à interpréter.
    - **Rapidité :** Entraînement et prédiction très rapides.
    - **Baseline solide :** Fournit une référence de performance pour des modèles plus complexes.
  + **Cas d’utilisation dans notre projet :** Proposé comme option par défaut pour les tâches de régression simples.
* **GradientBoostingRegressor :**
  + **Description :** Un ensemble de modèles faibles (généralement des arbres de décision) entraînés séquentiellement pour corriger les erreurs des modèles précédents.
  + **Avantages :**
    - **Haute précision :** Souvent parmi les modèles les plus performants pour les données tabulaires.
    - **Gestion des valeurs manquantes :** Robuste face aux données incomplètes.
    - **Flexibilité :** Capture efficacement les relations non linéaires.
  + **Cas d’utilisation dans notre projet :** Proposé comme option avancée pour les tâches de régression complexes.

##### 2.2.5.3. Tableau Comparatif et Justification des Choix

| Modèle | Type | Complexité | Interprétabilité | Performance | Vitesse d’entraînement | Cas d’utilisation principal |
| --- | --- | --- | --- | --- | --- | --- |
| **DecisionTreeClassifier** | Classification | Faible | Très élevée | Moyenne | Très rapide | Nettoyage automatique des données |
| **RandomForestClassifier** | Classification | Moyenne | Élevée | Élevée | Rapide | Classification générale |
| **LinearRegression** | Régression | Très faible | Très élevée | Moyenne (relations linéaires) | Très rapide | Régression simple |
| **GradientBoostingRegressor** | Régression | Élevée | Moyenne | Très élevée | Moyenne | Régression complexe |

**Justification de nos choix :**

1. **Priorité à l’interprétabilité :** Dans un outil comme CSV Analyzer, destiné à des utilisateurs qui ne sont pas nécessairement des experts en ML, il est crucial que les modèles soient compréhensibles. C’est pourquoi nous avons privilégié des modèles comme les arbres de décision et la régression linéaire.
2. **Équilibre performance/complexité :** Nous avons sélectionné des modèles offrant un bon équilibre entre performance et complexité computationnelle, adaptés à une application web où le temps de réponse est important.
3. **Adaptation aux données tabulaires :** Tous les modèles choisis sont particulièrement efficaces sur les données tabulaires structurées (CSV), qui constituent le cœur de notre application.
4. **Intégration avec scikit-learn :** L’ensemble des modèles s’intègre parfaitement dans l’écosystème scikit-learn, ce qui garantit une API cohérente et simplifie le développement.
5. **Évolutivité :** Notre architecture permet d’ajouter facilement de nouveaux modèles à l’avenir, en fonction des besoins des utilisateurs et de l’évolution du projet.

Cette sélection de modèles nous permet de couvrir un large éventail de cas d’utilisation tout en maintenant une expérience utilisateur fluide et compréhensible.

### 2.3. Outils de MLOps

Pour gérer le cycle de vie de nos modèles (suivi des expériences, versioning, déploiement), nous avons besoin d’un outil MLOps.

#### 2.3.1. MLflow

* **Description :** MLflow est une plateforme open-source pour gérer le cycle de vie du ML, incluant le suivi des expériences, le packaging du code, le versioning des modèles et le déploiement.
* **Avantages :**
  + **Agnostique :** Fonctionne avec n’importe quelle bibliothèque de ML (Scikit-learn, TensorFlow, etc.) et n’importe quel langage.
  + **Composants modulaires :** On peut utiliser uniquement les composants dont on a besoin (ex: juste le Tracking).
  + **Facile à intégrer :** Quelques lignes de code suffisent pour commencer à suivre des expériences.
* **Inconvénients :**
  + **Moins complet que Kubeflow :** Ne gère pas l’orchestration de pipelines complexes de manière native.

#### 2.3.2. Kubeflow

* **Description :** Dédié à rendre les déploiements de workflows de ML sur Kubernetes simples, portables et scalables. C’est une solution très complète et puissante.
* **Avantages :**
  + **Scalabilité et Portabilité :** Tire toute la puissance de Kubernetes.
  + **Pipelines de ML :** Outil très puissant pour orchestrer des workflows de ML complexes.
* **Inconvénients :**
  + **Complexité :** Nécessite une bonne connaissance de Kubernetes. La mise en place et la maintenance sont lourdes.
  + **Surdimensionné pour les petits projets.**

#### 2.3.3. DVC (Data Version Control)

* **Description :** DVC est un outil de versioning pour les données et les modèles de ML, qui s’intègre avec Git.
* **Avantages :**
  + **Basé sur Git :** Utilise les concepts de Git (branches, tags) pour les données, ce qui est très intuitif pour les développeurs.
  + **Reproductibilité :** Permet de lier précisément une version du code, des données et un modèle.
* **Inconvénients :**
  + **Périmètre plus restreint :** Se concentre sur le versioning et les pipelines, mais n’a pas d’interface de suivi des expériences aussi riche que MLflow.

#### 2.3.4. Tableau Comparatif et Choix

| Critère | MLflow | Kubeflow | DVC |
| --- | --- | --- | --- |
| **Périmètre** | Suivi, Packaging, Registre | Orchestration de pipelines sur K8s | Versioning de données/modèles |
| **Complexité** | Faible | Très élevée | Faible à Moyenne |
| **Dépendances** | Python | Kubernetes | Git, Stockage Cloud |
| **UI de suivi** | Oui, très complète | Oui (via Kubeflow Pipelines) | Non (via extensions) |

**Choix pour le projet “CSV Analyzer” :** **MLflow**

**Justification :** MLflow offre le meilleur compromis entre simplicité et fonctionnalités pour notre projet. Son composant “Tracking” est exactement ce dont nous avons besoin pour enregistrer les paramètres et les performances de chaque modèle entraîné. Le “Model Registry” sera également très utile pour gérer les versions des modèles. Sa faible complexité d’installation et d’utilisation en fait un choix idéal pour démarrer.

### 2.4. Solutions de Conteneurisation

La conteneurisation est cruciale pour garantir que notre application fonctionne de manière cohérente dans tous les environnements (développement, test, production).

#### 2.4.1. Docker

* **Description :** Docker est la plateforme de conteneurisation la plus populaire. Elle permet d’empaqueter une application et ses dépendances dans un conteneur isolé.
* **Avantages :**
  + **Standard de l’industrie :** Immense communauté, documentation abondante et intégration avec tous les outils de l’écosystème (CI/CD, cloud, etc.).
  + **Docker Compose :** Outil très pratique pour orchestrer des applications multi-conteneurs en développement.
* **Inconvénients :**
  + **Daemon Docker :** Nécessite un daemon tournant avec des privilèges root, ce qui peut poser des problèmes de sécurité dans certains contextes.

#### 2.4.2. Podman

* **Description :** Podman est une alternative à Docker qui se veut “daemonless” (sans daemon). Il est développé par Red Hat.
* **Avantages :**
  + **Sécurité :** Ne nécessite pas de privilèges root pour exécuter des conteneurs.
  + **Compatibilité :** Les commandes de Podman sont des alias des commandes Docker (podman run vs docker run), ce qui facilite la transition.
* **Inconvénients :**
  + **Moins mature :** Plus jeune que Docker, son écosystème est moins développé (par exemple, l’équivalent de Docker Compose est encore en développement).
  + **Moins répandu sur Windows et macOS.**

#### 2.4.3. Tableau Comparatif et Choix

| Critère | Docker | Podman |
| --- | --- | --- |
| **Popularité** | Très élevée | En croissance |
| **Architecture** | Client-Serveur (Daemon) | Daemonless |
| **Sécurité (rootless)** | Possible mais complexe | Natif |
| **Écosystème** | Très mature | En développement |
| **Docker Compose** | Oui | Alternative (podman-compose) |

**Choix pour le projet “CSV Analyzer” :** **Docker**

**Justification :** Docker est le standard de l’industrie. Sa maturité, son écosystème (en particulier Docker Compose) et la documentation disponible en font le choix le plus pragmatique et le plus efficace pour notre projet. Les problèmes de sécurité liés au daemon ne sont pas un obstacle majeur pour notre cas d’usage.

### 2.5. Systèmes de Monitoring

Une fois l’application déployée, il sera essentiel de la surveiller pour détecter les problèmes de performance et les erreurs.

#### 2.5.1. Prometheus et Grafana

* **Description :** C’est la stack open-source de référence pour le monitoring. Prometheus est une base de données de séries temporelles qui collecte les métriques. Grafana est un outil de visualisation qui permet de créer des tableaux de bord à partir des données de Prometheus.
* **Avantages :**
  + **Puissant et flexible :** Modèle de données très efficace et langage de requêtes (PromQL) puissant.
  + **Standard de l’écosystème Cloud Native :** Intégration parfaite avec Kubernetes et Docker.
  + **Open-source et grande communauté.**
* **Inconvénients :**
  + **Configuration :** La mise en place et la configuration de Prometheus et des alertes (via Alertmanager) peuvent être complexes.

#### 2.5.2. Datadog

* **Description :** Datadog est une plateforme de monitoring SaaS (Software as a Service) très complète.
* **Avantages :**
  + **Facilité d’utilisation :** Très simple à mettre en place (il suffit d’installer un agent).
  + **Solution tout-en-un :** Combine le monitoring de métriques, de logs et de traces (APM) dans une seule interface.
* **Inconvénients :**
  + **Coût :** Le prix est basé sur le nombre d’hôtes ou de conteneurs, ce qui peut devenir très cher à grande échelle.
  + **Propriétaire :** On est dépendant d’un fournisseur tiers.

#### 2.5.3. ELK Stack (Elasticsearch, Logstash, Kibana)

* **Description :** La stack ELK est principalement axée sur la gestion et l’analyse de logs.
* **Avantages :**
  + **Très puissant pour les logs :** Elasticsearch est un moteur de recherche très performant pour analyser de grands volumes de logs.
* **Inconvénients :**
  + **Moins adapté pour les métriques :** Le monitoring basé sur les métriques (séries temporelles) n’est pas son point fort principal, bien que cela soit possible.
  + **Complexe et gourmand en ressources.**

#### 2.5.4. Tableau Comparatif et Choix

| Critère | Prometheus + Grafana | Datadog | ELK Stack |
| --- | --- | --- | --- |
| **Type** | Open-source | SaaS (Commercial) | Open-source |
| **Focus principal** | Métriques | Métriques, Logs, Traces | Logs |
| **Coût** | Gratuit (coût d’hébergement) | Élevé | Gratuit (coût d’hébergement) |
| **Complexité** | Moyenne à Élevée | Faible | Élevée |

**Choix pour le projet “CSV Analyzer” :** **Prometheus et Grafana**

**Justification :** La stack Prometheus + Grafana est le choix le plus cohérent avec le reste de notre stack technique open-source. Elle est extrêmement puissante et nous donnera une flexibilité totale pour créer les tableaux de bord dont nous avons besoin. Bien que la configuration puisse être un peu complexe, l’investissement en vaut la peine pour la maîtrise complète de notre chaîne de monitoring. L’intégration avec Docker est excellente.

## 3. Architecture Technique Cible

Sur la base des choix technologiques effectués, nous pouvons maintenant définir l’architecture cible de notre application.

### 3.1. Schéma d’Architecture Global

graph TD  
 subgraph "Utilisateur"  
 U[Navigateur Web]  
 end  
  
 subgraph "Infrastructure (Docker Compose)"  
 LB(Load Balancer / Reverse Proxy - ex: Nginx)  
   
 subgraph "Service Web"  
 App[Application Django]  
 end  
  
 subgraph "Service MLOps"  
 MLF[Serveur MLflow]  
 end  
  
 subgraph "Service Monitoring"  
 PROM[Prometheus]  
 GRAF[Grafana]  
 end  
  
 subgraph "Stockage"  
 DB[(Base de données - PostgreSQL)]  
 FS[/ (Volume Docker pour Fichiers & Modèles)]  
 end  
 end  
  
 U -- HTTPS --> LB  
 LB -- HTTP --> App  
 App -- "CRUD, Auth" --> DB  
 App -- "Lecture/Écriture" --> FS  
 App -- "Log Métriques/Modèles" --> MLF  
 MLF -- "Lecture/Écriture" --> FS  
 PROM -- "Scrape Métriques" --> App  
 GRAF -- "Query Métriques" --> PROM  
 GRAF -- "Affiche Dashboards" --> U

### 3.2. Justification des Choix d’Architecture

* **Monolithique Modulaire (Django) :** L’application principale est un monolithe, ce qui est plus simple à développer et à déployer pour une petite équipe et un projet de cette taille. La modularité est assurée par les “apps” Django, qui séparent les préoccupations (ex: users, csv\_processor).
* **Conteneurisation (Docker) :** Tous les services (Django, MLflow, Prometheus, Grafana, PostgreSQL) sont conteneurisés. Cela garantit la reproductibilité de l’environnement et simplifie grandement le déploiement et la gestion des dépendances. Docker Compose orchestre ces services en développement.
* **Séparation des Services :** Bien que l’application soit monolithique, les services de support (MLOps, Monitoring) sont des entités séparées. Cela respecte le principe de la séparation des préoccupations et permet de faire évoluer ou de remplacer chaque brique indépendamment.
* **Stockage Persistant :** Les données critiques (base de données, fichiers CSV téléversés, modèles ML) sont stockées dans des volumes Docker pour garantir leur persistance même si les conteneurs sont redémarrés.

### 3.3. Flux de Données

1. **Flux Utilisateur :** L’utilisateur interagit avec l’application Django via son navigateur. Les requêtes passent par un reverse proxy (non représenté dans le docker-compose.yml initial mais une bonne pratique pour la production) qui les transmet au conteneur Django.
2. **Flux de Données CSV :** Lorsqu’un fichier est téléversé, Django le sauvegarde sur un volume persistant et enregistre ses métadonnées dans la base de données PostgreSQL.
3. **Flux d’Entraînement ML :**
   * L’utilisateur déclenche un entraînement depuis l’interface Django.
   * L’application Django lance une exécution MLflow.
   * Elle utilise Scikit-learn pour entraîner le modèle.
   * Les paramètres, les métriques et le modèle sérialisé (.pkl) sont envoyés au serveur MLflow, qui les stocke (le modèle lui-même est un artefact stocké sur le volume persistant).
4. **Flux de Monitoring :**
   * L’application Django expose un endpoint /metrics (via une bibliothèque comme django-prometheus).
   * Le serveur Prometheus est configuré pour “scraper” (collecter) régulièrement les données de cet endpoint.
   * Grafana est connecté à Prometheus comme source de données.
   * L’utilisateur peut consulter les tableaux de bord Grafana pour visualiser l’état de santé de l’application.

## 4. Feuille de Route (Roadmap)

La réalisation du projet sera découpée en plusieurs phases logiques.

### 4.1. Phase 1 : Développement Initial et PoC (Preuve de Concept)

* **Objectif :** Valider les fonctionnalités de base de l’application.
* **Tâches :**
  + Mettre en place le projet Django.
  + Développer les modèles de données (User, CSVFile).
  + Implémenter la gestion des utilisateurs (inscription, connexion, déconnexion).
  + Développer les fonctionnalités de téléversement, visualisation et suppression de lignes de CSV.
* **Livrable :** Une application web fonctionnelle mais sans la partie Machine Learning.

### 4.2. Phase 2 : Intégration MLOps et API

* **Objectif :** Intégrer la brique Machine Learning et le suivi des expériences.
* **Tâches :**
  + Intégrer Scikit-learn pour l’entraînement des modèles.
  + Mettre en place le serveur MLflow et intégrer le suivi (logging) dans le code d’entraînement.
  + Développer l’API de prédiction.
  + Créer l’interface de test du modèle.
* **Livrable :** L’application complète avec toutes ses fonctionnalités métier.

### 4.3. Phase 3 : Monitoring et Déploiement Continu

* **Objectif :** Mettre en place l’infrastructure de production.
* **Tâches :**
  + Conteneuriser l’ensemble de l’application avec Docker et Docker Compose.
  + Mettre en place la stack de monitoring (Prometheus, Grafana).
  + Instrumenter l’application Django pour exposer les métriques.
  + Créer une chaîne d’intégration continue (CI) avec GitHub Actions pour automatiser les tests.
* **Livrable :** Une application prête à être déployée, avec une chaîne de CI/CD fonctionnelle.

### 4.4. Phase 4 : Améliorations et Évolutions Futures

* **Objectif :** Améliorer la robustesse et ajouter de nouvelles fonctionnalités.
* **Tâches :**
  + Augmenter la couverture de tests.
  + Mettre en place le déploiement continu (CD).
  + Ajouter des fonctionnalités avancées (ex: plus d’options de prétraitement, plus de types de modèles).
  + Améliorer l’interface utilisateur.
* **Livrable :** Une version 2.0 de l’application.

## 5. Analyse des Risques et Stratégies d’Atténuation

| Risque | Probabilité | Impact | Stratégie d’Atténuation |
| --- | --- | --- | --- |
| **Dérive du modèle (Model Drift)** | Élevée | Élevé | Mettre en place un monitoring des performances du modèle en production (via MLflow) et des alertes en cas de dégradation. Prévoir un processus de ré-entraînement régulier. |
| **Qualité des données d’entrée** | Élevée | Élevé | Implémenter une validation robuste des données téléversées. Fournir des outils de nettoyage et de visualisation à l’utilisateur. |
| **Complexité de la stack technique** | Moyenne | Élevé | Adopter une approche itérative (commencer simple). Documenter rigoureusement l’architecture et les processus. S’appuyer sur des technologies matures et bien documentées. |
| **Sécurité de l’application** | Moyenne | Très élevé | Utiliser les mécanismes de sécurité intégrés de Django. Suivre les bonnes pratiques de l’OWASP. Effectuer des revues de code régulières. |
| **Montée en charge (Scalabilité)** | Faible | Moyenne | Concevoir une architecture découplée et conteneurisée dès le départ. Utiliser des outils standards (Docker, Nginx) qui permettent une mise à l’échelle horizontale si nécessaire. |

## 6. Indicateurs Clés de Performance (KPIs)

Pour mesurer le succès du projet, nous suivrons plusieurs types de KPIs.

### 6.1. KPIs Produit

* **Nombre d’utilisateurs actifs :** Mesure l’adoption de la plateforme.
* **Nombre de modèles entraînés par jour :** Indique l’utilisation de la fonctionnalité principale.
* **Taux de rétention :** Pourcentage d’utilisateurs qui reviennent après leur première visite.

### 6.2. KPIs Techniques

* **Temps de réponse moyen de l’API :** Mesure la performance de l’application.
* **Taux d’erreur (5xx) :** Indique la stabilité de la plateforme.
* **Couverture de tests :** Pourcentage du code couvert par les tests automatisés. Objectif > 80%.

### 6.3. KPIs de Performance du Modèle

* **Accuracy / F1-score (Classification) :** Mesure la pertinence des modèles générés.
* **R² / MSE (Régression) :** Mesure la précision des modèles de régression.
* **Temps d’entraînement moyen :** Indique l’efficacité du processus d’entraînement.

## 7. Conclusion

Cette veille technologique a permis de définir une stack technique cohérente, moderne et adaptée aux besoins du projet “CSV Analyzer”. En choisissant **Django, Scikit-learn, MLflow, Docker et Prometheus/Grafana**, nous nous appuyons sur des technologies open-source, matures et reconnues par la communauté.

L’architecture cible proposée est modulaire et évolutive, et la feuille de route définit un chemin clair pour la réalisation du projet. Les risques ont été identifiés et des stratégies d’atténuation ont été proposées.

Ce travail de recherche et de planification constitue une base solide pour démarrer le développement de l’application sur des fondations saines et robustes, en maximisant les chances de succès du projet.

## 8. Bibliographie et Ressources

* Documentation officielle de Django, Flask, FastAPI, Scikit-learn, MLflow, Docker.
* Articles de blog de “Towards Data Science”, “Medium”, “Real Python”.
* Discussions sur “Stack Overflow” et “Reddit” (r/learnpython, r/datascience).

## 9. Glossaire

* **API (Application Programming Interface) :** Interface de programmation applicative, permettant à deux applications de communiquer entre elles.
* **CI/CD (Continuous Integration/Continuous Deployment) :** Intégration continue / Déploiement continu. Pratiques visant à automatiser les tests et la mise en production.
* **MLOps (Machine Learning Operations) :** Ensemble de pratiques visant à déployer et maintenir des modèles de Machine Learning en production de manière fiable et efficace.
* **ORM (Object-Relational Mapping) :** Technique de programmation qui permet de convertir les tables d’une base de données en objets manipulables dans un langage de programmation.

## 10. Annexes