Rapport du Projet – Dashboard Analytique

1. Introduction

Dans le cadre de notre projet, nous avons développé un Dashboard Analytique interactif permettant de charger un fichier CSV, d'explorer ses données via des filtres dynamiques et de générer des statistiques affichées sous forme de textes et de graphiques. Ce projet vise à fournir un outil simple et efficace pour analyser rapidement un dataset, quel que soit son domaine (sessions, utilisateurs, performances, etc.).

Le système repose sur un backend Flask (Python) pour le traitement des données et un frontend HTML/CSS/JS assurant l'interface de visualisation. De plus, nous avons utilisé l'IA générative (Cursor) pour accélérer la conception du code, l'organisation du projet et la rédaction de documents.

2. Apport de chacun des membres de l'équipe

Serenic — Backend, Architecture & IA

- Création du backend Flask (routes API, endpoints, communication JSON).
- Développement du système de chargement CSV et des services analytiques.
- Conception des filtres dynamiques via pandas.
- Mise en place de l'architecture logicielle (organisation des fichiers, modules...).
- Intégration de l'IA générative pour corriger du code, optimiser des fonctions et ajouter des fonctionnalités.
- Rédaction majeure du README et du rapport final.
- Gestion du dépôt GitHub, commits et versioning.

Hakem — Frontend, UX & Tests

- Conception de l'interface utilisateur (dashboard.html).
- Création des scripts JavaScript pour :
  - l'envoi des requêtes API,
  - la mise à jour automatique des statistiques,
  - l'affichage des résultats.
- Mise en place du design général (CSS, structure HTML).
- Réalisation des tests manuels et fonctionnels du dashboard.
- Intégration des graphiques (ex : Chart.js si utilisé).
- Vérification du bon fonctionnement de toutes les interactions utilisateur.

3. Choix techniques

3.1 Backend : Flask (Python)

Nous avons choisi Flask car :

- c’est un framework léger, idéal pour les APIs REST,
- il permet une organisation flexible en blueprints,
- il s’intègre parfaitement avec pandas pour lire et manipuler le CSV.

Flask offre également une excellente compatibilité avec une interface web simple en JavaScript.

3.2 Traitement de données : pandas

pandas a été indispensable pour :

- charger le fichier CSV,
- nettoyer les données si nécessaire,
- appliquer des filtres dynamiques,
- calculer les statistiques (moyennes, max, min, counts…),
- préparer les données envoyées au frontend.

3.3 Frontend : HTML, CSS, JavaScript

Le frontend reste volontairement simple afin de garantir :

- une bonne lisibilité,
- une rapidité d’affichage,
- un faible niveau de complexité pour les interactions,
- une communication fluide via fetch() avec l’API.

3.4 Structure modulaire

L'arborescence du projet (backend, services, routes, utils) permet :

- une bonne lisibilité,
- une maintenance simplifiée,
- la possibilité d’ajouter facilement d’autres modules plus tard (ML, prédictions…).

4. Usage de l’IA générative

L’IA générative Cursor a joué un rôle important dans le projet. Nous l’avons utilisée pour :

4.1. Génération de code

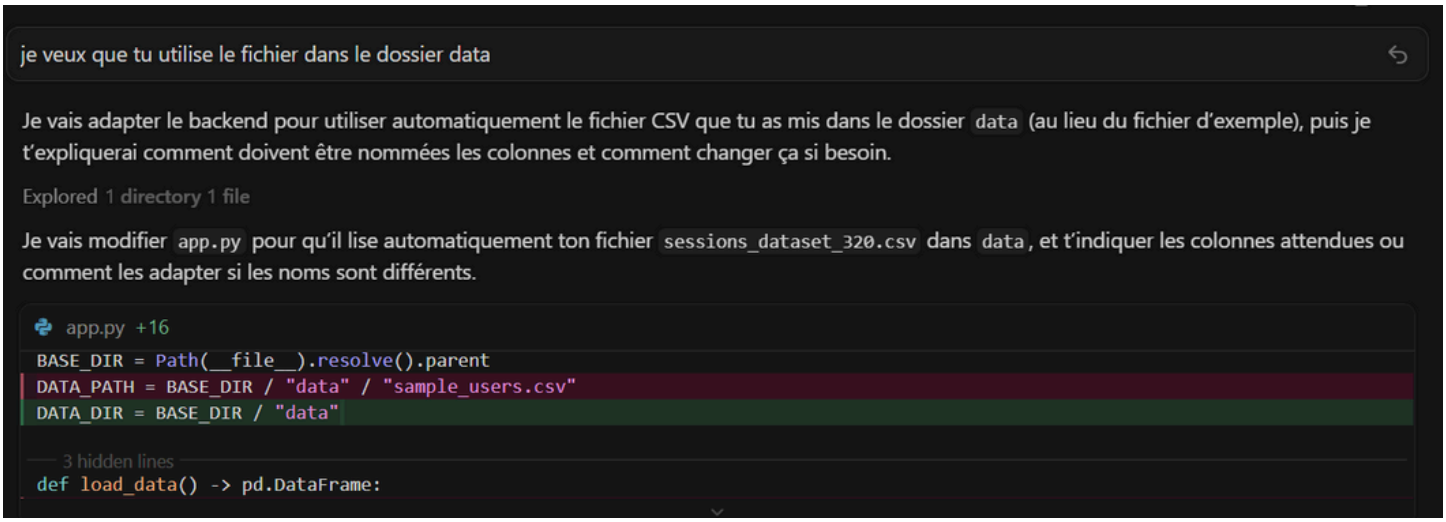
- Correction d’erreurs Flask
- Optimisation des fonctions pandas
- Génération de routes API
- Amélioration de l’architecture

4.2. Aide à la conception

- Propositions d'arborescence du projet
- Explication de bonnes pratiques backend
- Recommandations pour séparer services, routes et modules

4.3. Rédaction du rapport

- Une grande partie du rapport et du README a été rédigée avec l'aide de ChatGPT.
- demande d'architecture Flask,
- génération d'une fonction de filtrage,
- correction d'une erreur dans l'API,
- génération d'un texte de rapport.



5. Architecture du projet

Voici l'architecture améliorée et professionnelle utilisée :

```
Dashboard-Analytique/
├── app.py
├── backend/
│   ├── routes/
│   │   ├── api/
│   │   │   └── stats_api.py
│   │   └── dashboard.py
│   ├── services/
│   │   ├── data_service.py
│   │   ├── analytics_service.py
│   └── utils/
│       ├── filters.py
│       └── helpers.py
├── frontend/
│   ├── templates/
│   │   └── dashboard.html
│   └── static/
```

```
│       ├── css/
│       ├── js/
│       └── img/
└── data/
    └── sessions_dataset_320.csv
```

Points importants :

- séparation Backend / Frontend,
- modularisation en services,
- routes API claires,
- dossier dédié aux données,
- extensibilité du projet dans le futur.

6. Résultats obtenus

6.1 Fonctionnalités finales

Le Dashboard permet :

- d'importer un fichier CSV,
- d'extraire automatiquement ses colonnes,
- de créer et appliquer plusieurs filtres (catégories, valeurs, seuils...),
- de calculer des statistiques clés :
  - moyenne, médiane, min, max
  - nombre total d'enregistrements
  - distribution des valeurs
- d'afficher les résultats dynamiquement sur le dashboard.

L'interface est fluide et les données s'actualisent instantanément.

7. Limites du projet

7.1 Limites techniques

- Non adapté aux très gros datasets (plusieurs centaines de milliers de lignes)
- Le filtrage peut devenir lent selon la complexité
- Pas de base de données (tout est en mémoire)
- Pas de système d'authentification

7.2 Limites fonctionnelles

- Interface simple (design à améliorer)
- Peu de types de graphiques pour l'instant
- Une seule source de données possible à la fois
- Pas de sauvegarde des filtres ni des analyses

8. Conclusion

Ce projet nous a permis :

- de développer une application complète en équipe,
- d'apprendre à structurer un backend Flask professionnel,
- de manipuler efficacement des données réelles via pandas,
- d'utiliser l'IA générative comme un outil d'aide avancé,
- d'améliorer notre organisation et notre gestion de projet.

Le résultat final est un Dashboard fonctionnel, propre, extensible et utile, pouvant facilement évoluer vers :

- du machine learning,
- des prédictions,
- un système d'utilisateurs,
- une interface plus avancée.