1. **Introduction :** Objectif du projet
2. **Technologies utilisées :** Python, Flask, JS, Chart.js
3. **Structure du site :** Accueil, SQL, HTTP, Historique, Graphiques
4. **Fonctionnement du modèle :** SVC / MultinomialNB
5. **Détection d’attaques :** Types, exemple de détection
6. **Interactivité :** Historique dynamique, analyse en temps réel
7. **Améliorations possibles**
8. **Conclusion**

**1. 🧠 Ajout d’une explication du résultat**

* Exemple : pour chaque attaque détectée, afficher la raison (ex : "Caractère suspect ' OR 1=1 détecté").

**2. 🔍 Page de recherche ou filtre de l'historique**

* Permet de filtrer les requêtes par type (danger/safe) ou mot-clé.

**3. 📂 Upload de fichiers logs**

* Permettre à l'utilisateur d’envoyer un fichier .txt ou .log contenant des requêtes.
* Le serveur les analyse ligne par ligne et affiche les résultats.

**4. 📈 Page d’administration avec statistiques plus poussées**

* Nombre total de requêtes analysées % de requêtes dangereuses
* Graphique évolution horaire ou journalière (fictif ou réel)

**5. 🧪 Ajout de tests automatiques**

* Petits tests dans un fichier Python (test\_predict.py) pour vérifier si les prédictions sont correctes.

**6. 🔐 Ajout d’un système de connexion (authentification simple)**

* Pour que seuls certains utilisateurs puissent accéder aux pages sensibles comme /analystic.
* Flask avec session suffit.

Au début de mon projet, j’ai séparé mes données en trois ensembles distincts : un pour l’entraînement, un pour la validation, et un pour le test.  
Je faisais donc moi-même la séparation (manuelle) des données avant d’entraîner les modèles.

Cependant, cette méthode avait un inconvénient :  
elle ne permettait pas d’utiliser toute la puissance de la validation croisée, car une partie des données (par exemple l’ensemble de validation) n’était jamais utilisée pour entraîner le modèle, et inversement.  
De plus, les résultats pouvaient varier selon la façon dont je séparais les données, ce qui rendait l’évaluation moins fiable.

Après discussion avec mon professeur, j’ai changé d’approche :

* J’ai d’abord fusionné tous les jeux de données en un seul ensemble.
* Ensuite, j’ai mis de côté 20 % des données pour constituer un “test set” final, qui n’est utilisé qu’à la toute fin pour évaluer la performance réelle du modèle.
* Les 80 % restants sont utilisés pour **l’entraînement et la validation croisée** grâce à la fonction GridSearchCV de scikit-learn.
* Cette validation croisée consiste à diviser plusieurs fois les données internes en différentes parties : à chaque fois, une partie sert à entraîner le modèle, l’autre à le valider, et cela permet d’évaluer la robustesse du modèle de façon plus objective.

**Grâce à cette nouvelle méthode :**

* Toutes les données (sauf le test) servent à la fois à l’entraînement et à la validation, ce qui améliore l’utilisation des données et l’estimation des performances.
* GridSearchCV optimise automatiquement les hyperparamètres tout en utilisant la validation croisée.
* Le résultat sur le “test set” correspond à une performance réelle, indépendante de tout l’apprentissage.

Ce changement m’a permis d’obtenir une évaluation plus fiable, et de respecter les bonnes pratiques en machine learning.