# Projet Streamlit

Fillot Romain, Dupin Théo, Valade Lou

25 juin 2024

**Résumé**

Dans le contexte actuel de la transformation numérique et de l'explosion des volumes de données, la capacité à analyser, nettoyer et visualiser efficacement des ensembles de données est devenue essentielle pour les entreprises, les chercheurs et les professionnels de divers domaines. Cependant, de nombreuses personnes ne possèdent pas les compétences techniques nécessaires pour utiliser des outils d'analyse de données complexes. C'est dans ce cadre que ce projet prend forme.

Ce projet vise à développer une application web interactive à l'aide de Streamlit pour analyser, nettoyer et visualiser n'importe quel fichier CSV. L'application est conçue pour être intuitive et accessible, permettant ainsi aux utilisateurs de divers niveaux de compétence en informatique et en analyse de données de tirer parti de ses fonctionnalités.

## 1 Introduction

Nous avons conçu une solution qui permet aux utilisateurs de gérer efficacement leurs datasets. L'application offre plusieurs fonctionnalités, notamment le nettoyage, la normalisation, le clustering, la prédiction et la visualisation des résultats. Pour mener à bien ce projet, nous avons utilisé Streamlit, une bibliothèque open-source qui permet de créer des applications web interactives en Python. Elle permet donc aux développeurs de créer des interfaces utilisateur graphiques (GUI) pour leurs scripts Python, ce qui facilite la visualisation et l'exploration des données. Dans un premier temps, nous allons explorer l'architecture du projet, puis du choix de la conception. Ensuite nous discuterons des problèmes rencontrés et des solutions apportées pour les résoudre. Enfin nous finirons avec les axes d'amélioration et le pourcentage d'utilisation de l'IA générative. Etant 3 à piloter ce projet, nous avons utilisé git/github afin de pouvoir travailler efficacement en équipe. Chaque fonctionnalité a été développée sur une branche spécifique puis merge avec l'utilisation de Pull Request dans la branche master (main).

## 2 Architecture du projet

Une partie de l'architecture nous a été conseillée afin de diviser la partie visuelle et la partie logique de notre application. Nous avons donc deux dossiers : front et back. Le dossier front possède un main.py qui correspond à la racine du projet. C'est ce programme que nous lançons pour exécuter notre application. Pour améliorer l'expérience utilisateur et faciliter la navigation dans l'application, nous avons décidé de diviser l'application en plusieurs pages à l'aide de Streamlit Multi-Page. Chaque page est dédiée à une fonctionnalité spécifique de l'application, ce qui permet aux utilisateurs de trouver facilement ce dont ils ont besoin. Pour avoir une application multi-page, streamlit recommande de mettre nos différentes pages dans un dossier "pages". Nous retrouvons dans ce dossier toutes nos pages qui sont accessibles depuis notre menu. Ce dossier "pages" se trouve dans le dossier front. Enfin, notre dossier back possède un fichier par fonctionnalité et possède toute la logique de notre application :

— *exploration.py* pour la partie exploration
— *clean_data.py* pour nettoyer le jeu de données avec des options
— *standardization_back.py* pour la normalisation
— *clustering_back.py* pour la partie cluster et visualisation
— *prediction_back.py* pour la partie prédiction avec les résultats (précision, matrice de confusion)

Chacun de ces fichiers possèdent des fonctions qui sont appelées dans les pages de notre front.

# 3 Choix de conception

## 3.1 Les bibliothèques

Pour assurer la continuité entre les différentes pages de l'application, nous avons utilisé la fonctionnalité **session_state** de Streamlit. Cette fonctionnalité permet de stocker des données temporaires entre les sessions de l'application, ce qui signifie que les utilisateurs peuvent passer d'une page à l'autre sans perdre leurs données. Cela permet également aux utilisateurs de revenir à une page précédente sans avoir à recommencer tout le processus de nettoyage ou de normalisation du dataset.

Ensuite, nous avons importé plusieurs bibliothèques comme matplotlib.pyplot, pandas, numpy ou encore sklearn.preprocessing. Vous trouverez juste en dessous une explication pour chacune de ces bibliothèques.

**matplotlib.pyplot** : utilisée pour créer des visualisations de données de haute qualité. Elle offre une grande variété de types de graphiques, tels que les histogrammes, les diagrammes à barres et les nuages de points, ainsi que des options de personnalisation pour les couleurs, les étiquettes et les légendes.

**pandas** : utilisée pour la manipulation et l'analyse des données. Elle permet de lire et d'écrire des fichiers de données dans différents formats, tels que CSV, Excel et SQL. Elle offre également des fonctionnalités de nettoyage et de filtrage des données, ainsi que des opérations de groupe et de fusion.

**numpy** : utilisée pour les calculs numériques. Elle offre des fonctions pour les opérations mathématiques, telles que les opérations sur les matrices, les fonctions trigonométriques et les statistiques.

**sklearn.preprocessing** : utilisée pour la préparation des données en vue de l'apprentissage automatique. Elle offre des fonctionnalités de normalisation, de standardisation et de transformation des données.

Comme déjà cité plus haut, notre application possède un menu pour que l'utilisateur navigue simplement. Nous avons décidé de diviser le processus en plusieurs étapes, chacune d'entre elles étant accessible uniquement lorsque la précédente est terminée. Par exemple dès qu'un utilisateur arrive sur notre application, s'il veut pouvoir accéder à l'étape "clean data", il va devoir importer un CSV. Sans importation de CSV, il ne peut accéder au menu. Nous avons fait ce choix pour que notre application s'adapte à ceux qui ont peu de connaissances en informatique. Cela permet de les guider et de réduire les potentielles erreurs.

## 3.2 Pré-traitement et nettoyage des données

### 3.2.1 Gestion des valeurs manquantes

Une fois le jeux de données importé, l'utilisateur à le choix de supprimer ou non les lignes ou colonnes où se trouve des valeurs manquantes.

S'il fait le choix de conserver les lignes/colonnes où se trouvent des données manquantes, ce dernier a alors le choix entre plusieurs méthodes pour remplacer ces valeurs :

— Median
— Mean
— k-Nearest Neighbors
— Linear Regression

Il a également à cette étape, la possibilité de remplacer les valeurs non-numérique avec Mode.

### 3.2.2 Normalisation

L'accès à la fonctionnalité de normalisation est bloquée tant que l'utilisateur n'a pas importé et nettoyé son jeux de données. Cela permet de forcer l'utilisateur à suivre une à une les étapes de l'application comme nous l'avons dit plus haut.

Une fois cela fait, il a alors accès aux trois méthodes de standardisation des données les plus connues :

— Min-Max
— Z-score
— Robust

Enfin, après avoir choisi sa méthode de normalisation, il va pouvoir choisir entre clustering et prediction à l'aide d'un button.

## 3.3 Clustering ou prédiction

### 3.3.1 Clustering

Pour la partie clustering, l'utilisateur peut choisir entre deux méthodes : la méthode K-means et la méthode DBSCAN. Elles permettent toutes les deux de diviser un jeu/ensemble de données en différents "paquets", de manière à ce que les données de chaque sous-ensemble partagent des caractéristiques communes, mais elles ne fonctionnent pas du tout de la même manière.

**K-means** : Cette méthode basée sur les centroïdes, est une méthode qui permet de diviser les points en k groupes, souvent appelés clusters, de façon à minimiser une certaine fonction. On considère la distance d'un point à la moyenne des points de son cluster ; la fonction à minimiser est la somme des carrés de ces distances. Dans notre application streamlit, nous avons fait en sorte que l'utilisateur puisse choisir différents paramètres pour cet algorithme de manière à lui fournir une analyse de données totalement flexible et adaptée à n'importe quel dataset. Il peut d'abord sélectionner les colonnes qu'il souhaite utiliser pour le clustering. Il doit en choisir minimum 2 car nous appliquons PCA (Principal Component Analysis) sur cet algorithme, qui est une technique utilisée principalement pour réduire la dimensionnalité des données tout en préservant autant que possible la variance présente dans les données d'origine. La méthode PCA va donc projeter les données sur un nouvel espace de dimensions réduites. Ces nouvelles dimensions sont appelées "composantes principales" et sont des combinaisons linéaires des variables d'origines. De plus, les composantes principales sont orthogonales entre elles, ce qui signifie qu'elles sont non corrélées. Cela permet donc de mieux comprendre les relations entre les variables et de détecter les tendances et les patterns dans les données. Par défaut, notre application initialise le nombre de composantes principales à 2 pour réduire l'ensemble des données dans 2 dimensions, mais l'utilisateur peut également en choisir 3, auquel cas, il devra sélectionner minimum 3 colonnes pour le clustering. Il pourra visualiser un graphique en 2D ou 3D suivant le nombre de composantes principales qu'il aura choisi. Il peut également modifier 4 autres paramètres pour analyser son jeu de données. Le nombre de clusters (k) peut varier de 2 à 10. Il peut aussi choisir la méthode d'initialisation qu'il souhaite entre "k-means++" et "random" qui sont couramment utilisées avec l'algorithme K-means. La méthode k-means++ se déroule de la manière suivante : le premier centre de cluster est choisi parmi les points de données ; ensuite, pour chaque point de donnée $x$, la distance minimale $D(x)$ est calculée entre $x$ et le centre de cluster le plus proche déjà choisi ; puis, un nouveau centre de cluster est choisi parmi les points de données avec une probabilité proportionnelle à $D(x)^2$ ; enfin, les étapes 2 et 3 sont répétées jusqu'à ce que k centres de clusters aient été choisis. La méthode "random" est beaucoup plus simple puisqu'elle consiste à choisir des centres de clusters de manière aléatoire parmi les points de données. Elle peut être utilisée pour des jeux de données simples ou lorsque la rapidité d'initialisation est cruciale, mais elle comporte un risque plus élevé de mauvaise performance. Par défaut, c'est la méthode k-means++ qui est choisie dans notre application. Ensuite, l'utilisateur peut modifier le nombre d'itérations maximum pour K-means. Il peut choisir entre 100 et 1000 itérations au maximum. Cela permet de s'assurer que l'algorithme a suffisamment de temps pour converger, particulièrement pour des ensembles de données complexes. Cela équilibre le compromis entre performance et précision, évitant des calculs excessifs et garantissant des clusters stables, adaptés aux spécificités du dataset de l'utilisateur. Par défaut, notre application l'initialise à 300. L'utilisateur peut également choisir jusqu'à 10 initialisations différentes. Cela permet d'augmenter les chances d'obtenir un meilleur clustering en évitant des solutions sous-optimales dues à des points de départs aléatoires. Plus d'initialisations améliorent la stabilité et la qualité des clusters, mais avec un compromis sur le temps de calcul. Nous avons choisi de mettre ce paramètre en int plus qu'en "auto" car nous voulons laisser pleinement le choix à l'utilisateur de faire ce qu'il et d'adapter le paramètre par rapport à ses jeux de données et à ses envies. Le fait de spécifier un entier pour ce paramètre permet de contrôler combien de fois l'algortihme K-means sera exécuté avec différentes initialisations. L'algorithme conservera le résultat avec la meilleure (c'est-à-dire la plus basse) somme des distances au carré des points aux centres les plus proches. Dans notre aplication, le nombre d'initialisations différentes par défaut est initialisé à 10. Enfin, nous affichons des statistiques sur ces clusters pour fournir une analyse un petit peu plus détaillée sur son jeu de données. Pour cet algorithme, nous lui fournissons le centroïde de chaque cluster ainsi que le nombre de points de données dans chaque cluster.

**DBSCAN** : Il s'agit d'un algorithme fondé sur la densité dans la mesure qui s'appuie sur la densité estimée des clusters pour effectuer le partitionnement. Cet algorithme utilise principalement 2 paramètres qui sont les suivants : la distance $\epsilon$ et le nombre minimum de points devant se trouver dans un rayon $\epsilon$ pour que ces points soient considérés

comme un cluster. Pour cet algorithme, nous reprenons le même principe que la méthode K-means en appliquant PCA. L'utilisateur a donc le choix entre 2 ou 3 composantes principales avec les mêmes contraintes évoquées dans la partie au-dessus. Il pourra donc visualiser par la suite un graphique en 2D ou en 3D suivant le nombre de composantes principales qu'il a sélectionné. Pour le paramètre "distance maximale entre deux points", l'utilisateur peut choisir de 0.10 à 1 sur notre application. Par défaut, nous l'avons initialisé à 0.5. Ce paramètre permet de définir la distance maximale entre deux points pour qu'ils soient considérés comme voisins dans un cluster. En l'ajustant, l'utilisateur peut contrôler la granularité des clusters : un petit $\epsilon$ va pouvoir détecter des clusters très denses et fins, alors qu'un grand $\epsilon$ va pouvoir identifier des clusters plus larges et moins denses. Ce paramètre permet également de détecter ce que l'on appelle "points de bruit". Les points de bruit sont des points qui n'ont pas assez de voisins dans la distance $\epsilon$ pour faire partie d'un cluster formé par l'algorithme DBSCAN, ils sont considérés comme des anomalies ou valeurs aberrantes. Le nombre de points de bruit décrit donc la qualité d'un jeu de données. Dans notre application, nous l'affichons dans le tableau décrivant le nombre de points de chaque cluster. Ensuite, nous laissons la possibilité à l'utilisateur de choisir le nombre minimum de points pour former un cluster (min-samples). Par défaut, nous l'initialisons à 5, mais l'utilisateur peut le choisir de 2 jusqu'à 10. Grâce à ce paramètre, les points qui ne satisfont pas la densité minimale requise par "min-samples" sont considérés comme des points de bruit. Ainsi, augmenter "min-samples" peut réduire le nombre de points de bruit détectés, car seuls les groupes plus denses seront formés en clusters. Enfin, nous affichons des statistiques pour donner une analyse assez détaillée de l'algorithme DBSCAN. Parmi elles, il y a : la densité de chaque cluster, le nombre de clusters estimés par DBSCAN, l'homogénéité des données qui évalue à quel point chaque cluster contient uniquement des points qui appartiennent à une seule classe, et le coefficient de silhouette qui évalue à la fois la compacité des clusters et leur séparation les uns par rapport aux autres. Plus un coefficient de silhouette est proche de 1, plus cela indique que les points au sein d'un même cluster sont très similaires les uns aux autres en termes de distance, et en même temps, ils sont bien séparés des points des autres clusters.

### 3.3.2 Prédiction

Dans cette partie, l'utilisateur a le choix d'exécuter sur son jeu de données des algorithmes de classification ou de régression.

Pour chaque type de prédiction, l'utilisateur a le choix entre 3 algorithmes issus de la bibliothèque sklearn :

Classification
- SGDClassifier : algorithme de classification linéaire basé sur la méthode de la descente de gradient stochastique
- KNN : algorithme de classification supervisée qui prédit la classe d'une nouvelle observation en fonction de ses voisins les plus proches dans un espace
- RandomForest : algorithme qui se base sur l'assemblage d'arbres de décision

Régression
- RandomForestRegressor : algorithme de régression basé sur la méthode des forêts aléatoires
- Ridge : un algorithme de régression linéaire régularisée utilisé pour prédire une valeur
- Lasso :un algorithme de régression linéaire régularisée utilisé pour prédire une valeur

Voici les paramètres utilisés pour chaque algorithme de classification :

**SGDClassifier**
- **loss** : la fonction de perte à minimiser. Plusieurs options disponibles.
- **penalty** : le type de régularisation à appliquer.
- **alpha** : le coefficient de régularisation.
- **max_iter** : le nombre maximum d'itérations à effectuer.
- **tol** : la tolérance pour la convergence.

**KNN (KNeighborsClassifier)**
- **n_neighbors** : le nombre de voisins les plus proches à considérer pour la prédiction.
- **weights** : la méthode de pondération des voisins.
- **algorithm** : l'algorithme utilisé pour rechercher les voisins les plus proches.
- **leaf_size** : le nombre de feuilles dans les arbres utilisés pour la recherche des voisins.

**RandomForest**
- **n_estimators** : le nombre d'arbres décisionnels dans la forêt.
- **max_depth** : la profondeur maximale de chaque arbre décisionnel.
- **min_samples_split** : le nombre minimal d'échantillons requis pour diviser un nœud.
- **min_samples_leaf** : le nombre minimal d'échantillons requis pour qu'un nœud soit considéré comme une feuille.
- **max_features** : le nombre maximal de caractéristiques à considérer pour chaque partition de l'arbre décisionnel.
- **max_leaf_nodes** : le nombre maximal de feuilles dans chaque arbre décisionnel.

Voici les paramètres utilisés pour chaque algorithme de régression :

**RandomForestRegressor**
- **max_depth** : la profondeur maximale de chaque arbre décisionnel dans la forêt aléatoire.
- **criterion** : la fonction de coût utilisée pour évaluer la qualité de chaque partition de l'arbre décisionnel.
- **n_estimators** : le nombre d'arbres décisionnels dans la forêt aléatoire.

**Ridge et Lasso**
- **alpha** : le coefficient de régularisation qui contrôle la pénalité appliquée aux coefficients de régression.
- **max_iter** : le nombre maximal d'itérations à effectuer pour optimiser les coefficients de régression.
- **tol** : la tolérance pour la convergence de l'optimisation
- **fit_intercept** : un booléen indiquant si l'ordonnée à l'origine doit être ajustée ou non.

Chacun de ces algorithmes sont donc paramétrable et permettent à l'utilisateur de sélectionner la variable cible, ainsi que le pourcentage pour le jeu de test. Les valeurs de tous les paramètres ne sont pas modifiables par l'utilisateur. L'idée est de mettre à disposition une customisation des principaux paramètres uniquement, dans le but de simplifier au maximum l'utilisation de l'outil.

Pour évaluer la performance des algorithmes sur le jeux de données, certaines métriques sont calculés par l'application tel que le score F1 ou la matrice de confusion dans le cas d'un algorithme de classification. Pour la partie régression, nous affichons le coefficient de détermination et un graphe montrant la prédiction par rapport aux valeurs réelles.

Dans l'ensemble, notre conception pour l'application Streamlit vise à offrir une expérience utilisateur fluide et intuitive tout en fournissant des fonctionnalités puissantes pour la gestion de dataset. En divisant l'application en plusieurs pages, en utilisant la fonctionnalité 'session_state' et en ayant fait le choix d'un menu qui s'affiche progressivement, nous avons pu créer une application cohérente et facile à utiliser pour les utilisateurs.

# 4 Problèmes rencontrés et Solutions apportées

## 4.1 Architecture

Streamlit est une solution facile à prendre en main. Cependant nous avons eu des difficultés en ce qui concerne la mise en place de l'architecture du projet. En effet, nous n'arrivions pas à importer les fichiers du dossier /back du projet dans les fichiers du dossier /front du projet.

Pour résoudre ce problème, nous avons du utiliser le module sys ainsi que la bibliothèque pathlib qui nous on tous deux permis de modifier la manière dont les fichiers sont importés dans le fichier /front/main.py.

Bien que cela est résolu le problème en lui-même, cela nous force à redémarrer le projet lors de chaque changement dans les fonctions côté back pour que ces dernières soient appliquées au projet.

## 4.2 Prédiction

Nous avons rencontré des difficultés pour afficher correctement les métriques, telles que la matrice de confusion. Bien que ce ne soit pas très compliqué à résoudre, cela nous a fait perdre du temps en raison de notre faible connaissance de Streamlit. Heureusement, nous avons découvert un site de discussion dédié à Streamlit :

$$< https://discuss.streamlit.io >$$

où les utilisateurs peuvent publier leurs problèmes et recevoir de l'aide de la part d'autres membres de la communauté. Grâce à ce site, nous avons pu résoudre le problème rapidement. Si nous aurions su plus tôt que cette ressource existait, nous aurions certainement pu gagner du temps.

# 5 Utilisation des outils d'IA génératives

## 5.1 Lou

Pour ma part, j'ai participer au développement de la normalisation, ainsi qu'à la prédiction des données.
Je pense avoir utilisé GitHub Copilot à 85% pour ma contribution au projet.

## 5.2 Romain

Pour ma part, je me suis occupé de la partie session_state avec le menu et de l'exploration des données. J'ai aussi participé à la prédiction des donnés en rajoutant la classification multiclasse. J'ai utilisé ChatGPT et Le chat (Mistral IA) à hauteur de 80%.

## 5.3 Théo

Et pour ma part, je me suis occupé de la partie nettoyage des données ainsi que de la partie clustering (K-means et DBSCAN). Je pense avoir utilisé environ 80% d'IA pour réaliser ce projet (ChatGPT, Mistral AI et Github Copilot). Je me suis aussi aidé de la documentation de scikit learn notamment pour la partie clustering pour avoir un ordre d'idée sur les paramètres à utiliser pour les fonctions concernant les algortihmes K-means et DBSCAN.

# Références

https ://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans
https ://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN
https ://penseeartificielle.fr/clustering-avec-lalgorithme-dbscan/
https ://streamlit.io/