

Master Économiste d'Entreprise



Mémoire de recherche

---

Développement d'une application web  
de traitement et visualisation de  
données tabulaires avec Dash

---

Présentée par :

LYNA BOUDAMOUS

Promotion 2025

Université de Tours

Tuteur universitaire : Mr LURET

Responsable de formation : Mme SCHOLLER

# Table des matières

<b>Introduction</b>	<b>5</b>
<b>1 Contexte, problématique et objectifs</b>	<b>6</b>
1.1 Contexte et problématique . . . . .	6
1.2 Objectifs du projet . . . . .	6
<b>2 Organisation du projet</b>	<b>8</b>
2.1 Difficultés rencontrées . . . . .	8
2.2 Choix technologiques . . . . .	9
2.3 Architecture logicielle . . . . .	9
2.4 Typologie des utilisateurs ciblés . . . . .	10
<b>3 Analyse détaillée des données et méthodologie</b>	<b>11</b>
3.1 Module callbacks.py : Gestion dynamique de l'interface et des interactions utilisateur . . . . .	11
3.1.1 Objectifs et rôle architectural . . . . .	11
3.1.2 Principaux composants fonctionnels . . . . .	12
3.2 Module data-manager.py . . . . .	13
3.2.1 Fonctionnalités principales . . . . .	13
3.2.2 Contribution dans l'architecture globale . . . . .	14
3.3 Module layouts.py : Construction de l'interface utilisateur graphique avec Dash . . . . .	14
3.3.1 Organisation générale . . . . .	14
3.3.2 Structure fonctionnelle du layout . . . . .	14
3.4 Module app.py : point d'entrée principal de l'application Dash . . . . .	16
3.4.1 Structure fonctionnelle de app . . . . .	16
<b>4 Documentation utilisateur</b>	<b>18</b>
4.1 Présentation générale . . . . .	18
4.2 Interface utilisateur . . . . .	18
4.2.1 Chargement et fusion des données . . . . .	18
4.2.2 Gestion des bases chargées . . . . .	19
4.2.3 Recherche et filtres avancé . . . . .	19
4.2.4 Gestion des valeurs manquantes . . . . .	19
4.2.5 Statistiques . . . . .	19
4.2.6 Visualisations . . . . .	20
4.2.7 Export des données . . . . .	20

<b>5 Exemple pratique : Exploitation des données culturelles et socio-économiques départementales dans l'application Dash</b>	<b>21</b>
5.1 Introduction . . . . .	21
5.2 Méthodologie . . . . .	21
5.2.1 Prérequis techniques . . . . .	21
5.2.2 Initialisation de l'application . . . . .	22
5.3 Étapes d'analyse interactive . . . . .	22
5.3.1 Chargement et préparation des jeux de données . . . . .	22
5.3.2 Gestion des bases chargées . . . . .	23
5.3.3 Recherche au sein des données fusionnées . . . . .	23
5.3.4 Application de filtres avancés . . . . .	24
5.3.5 Gestion des valeurs manquantes . . . . .	25
5.3.6 Calculs statistiques . . . . .	25
5.3.7 Visualisation graphique . . . . .	26
5.3.8 Export des données . . . . .	27
<b>6 Conclusion</b>	<b>28</b>
<b>Bibliographie</b>	<b>29</b>
Annexes . . . . .	30
Annexe : Quiz sur l'application « Analyseur de Données » . . . . .	30
.1 Schéma fonctionnel de l'application . . . . .	33
.2 Base de données fusionnée — Export PDF . . . . .	34
.3 Aperçu du code source des modules . . . . .	35
.3.1 Fichier <code>app.py</code> . . . . .	35
.3.2 Fichier <code>data_manager.py</code> . . . . .	36
.3.3 Fichier <code>layouts.py</code> . . . . .	39
.3.4 Fichier <code>callbacks.py</code> . . . . .	42

# Table des figures

5.1	Confirmation de la réussite du chargement avec le message d'état « 2 base(s) chargée(s) » . . . . .	23
5.2	Fusion de deux bases importées par la colonne « Département » avec affichage du message de confirmation . . . . .	23
5.3	Suppression de la base fusionnée avec affichage du message de confirmation « Fusion supprimée » . . . . .	23
5.4	Champ de recherche textuelle permettant de filtrer les données par mot-clé, ici « Ain » . . . . .	24
5.5	Affichage de la variable correspondant au mot-clé « Ain » avec surlignage dans la table . . . . .	24
5.6	Remplissage des champs de filtre : sélection de la colonne « Ciné-100k-hab » et de la condition « > 4 » . . . . .	24
5.7	Affichage des départements répondant au filtre « Ciné-100k-hab > 4 » dans la table . . . . .	25
5.8	Liste dynamique des colonnes avec valeurs manquantes, ici la variable « Salaire-moyen(euro-net-ans) » . . . . .	25
5.9	Remplissage des valeurs manquantes par la moyenne, confirmé par un message d'état . . . . .	25
5.10	Récapitulatif statistique synthétique pour la colonne numérique « Salaire-moyen-net-ans » généré par le bouton « Calculer » . . . . .	26
5.11	Choix des variables : Axe X « Département », Axe Y « Salaire net par an », et coloration par « Salaire net par ans » . . . . .	26
5.12	Graphique résultant (diagramme en barres) généré selon les paramètres sélectionnés . . . . .	27
1	Extrait du fichier PDF exporté contenant la base fusionnée . . . . .	34
2	Structure du fichier app.py : initialisation de l'application Dash . . . . .	35
3	Partie 1 : gestion des données dans data_manager.py . . . . .	36
4	Partie 2 : traitement des données dans data_manager.py . . . . .	37
5	Partie 3 : traitement avancé dans data_manager.py . . . . .	38
6	Partie 1 : structure de l'interface dans layouts.py . . . . .	39
7	Partie 2 : composants visuels dans layouts.py . . . . .	40
8	Partie 3 : finalisation de l'interface dans layouts.py . . . . .	41
9	Callback 1 dans callbacks.py . . . . .	42
10	Callback 2 dans callbacks.py . . . . .	43
11	Callback 3, partie 1 dans callbacks.py . . . . .	44
12	Callback 4, partie 2 dans callbacks.py . . . . .	45
13	Callback 5, partie 3 dans callbacks.py . . . . .	46
14	Callback 5 dans callbacks.py . . . . .	47

15	Callback 6, partie 1 dans callbacks.py . . . . .	48
16	Callback 7, partie 2 dans callbacks.py . . . . .	49

# Introduction

On entend souvent parler de Big Data, Open Data, Social Data, Scientific Data, entre autres. L'importance accordée aux données ne cesse de croître, tant leur analyse est devenue essentielle pour en extraire une valeur exploitable. Le concept d'Open Data a été introduit pour la première fois en 1995 par le groupe américain GCDIS (Global Change Data and Information System), qui encourageait les parties prenantes à partager leurs données dans une logique de mutualisation [Data et System, 1995]. Ce mouvement des données ouvertes, bien que relativement récent, connaît aujourd'hui un essor rapide, soutenu par les initiatives des gouvernements et des institutions publiques en faveur de l'accessibilité des données. Dans un monde de plus en plus orienté par les données, la capacité à manipuler, analyser et visualiser efficacement les données tabulaires représente un enjeu stratégique dans de nombreux domaines : finance, marketing, logistique, santé, recherche scientifique, etc. Chaque jour, entreprises et organisations collectent d'importants volumes de données sous forme tabulaire (CSV, Excel, bases relationnelles), dont l'exploitation pertinente est devenue une nécessité. Ces données, de par leur structure claire, facilitent leur traitement, comparaison et visualisation. Elles servent de socle à de nombreuses applications, notamment en apprentissage automatique. La montée en puissance des approches comme le deep learning et le machine learning traditionnel accentue l'importance de leur bonne maîtrise.

# Chapitre 1

## Contexte, problématique et objectifs

### 1.1 Contexte et problématique

L'analyse de données structurées s'inscrit dans une histoire longue, remontant à l'Antiquité, où des civilisations telles que les Sumériens utilisaient déjà des tablettes pour le suivi des stocks et des transactions commerciale<sup>1</sup>. L'évolution majeure intervient cependant en 1970, lorsque Edgar F. Codd formalise le modèle relationnel : ce paradigme organise l'information sous forme de tables, chaque enregistrement étant défini par un ensemble d'attributs<sup>2</sup>. Cette innovation marque un tournant décisif, facilitant non seulement l'accès et la manipulation des données, mais aussi leur structuration logique et l'automatisation des traitements. L'essor de l'informatique personnelle dans les décennies 1980 et 1990 démocratise à son tour l'emploi de formats tabulaires, grâce à la généralisation des tableurs comme Excel ou Lotus 1-2-3, ainsi qu'à l'apparition des fichiers CSV désormais standards pour l'échange et le stockage de jeux de données volumineux (McKinney, 2018). Toutefois, il s'avère que ces outils, bien que polyvalents et largement adoptés, rencontrent rapidement d'importantes limites sur le plan de la performance, de la flexibilité et de la personnalisation, en particulier dès lors qu'il s'agit de manipuler des jeux de données hétérogènes ou de répondre à des besoins analytiques avancés. Dans ce contexte, la problématique centrale de ce mémoire peut être formulée ainsi : comment permettre à un utilisateur dénué de compétences techniques spécifiques d'analyser et de visualiser de manière rapide, fiable et flexible des données tabulaires issues de sources variées, tout en s'affranchissant des contraintes des outils traditionnels ?

### 1.2 Objectifs du projet

Ce mémoire vise à concevoir et implémenter une application web interactive, développée en Python, destinée à l'exploration, l'analyse et la visualisation de données tabulaires provenant de fichiers CSV ou Excel. L'objectif fondamental est de mettre à disposition des utilisateurs, indépendamment de leur maîtrise technique, un environnement intuitif permettant de charger différents jeux de données, d'opérer des opérations de filtrage, de nettoyage, d'analyse statistique, et de générer des visualisations adaptées, ainsi que d'exporter les résultats. L'application s'appuiera exclusivement sur des technologies open

---

1. [1] Gelernter, H. (1988). *Data Tables : History and Principles*. Computing Reviews, 29(4), 45-52.  
2. [2] Codd, E. F. (1970). *A Relational Model of Data for Large Shared Data Banks*. Communications of the ACM, 13(6), 377–387.

source et éprouvées : la bibliothèque pandas pour le traitement et la manipulation des données, Dash et Plotly pour la construction de l'interface web interactive et la visualisation graphique, ainsi que ReportLab pour la génération de rapports PDF. Elle intégrera, entre autres, la gestion simultanée de plusieurs fichiers, l'affichage interactif des données, l'application de filtres avancés par colonne, le calcul de statistiques descriptives (moyenne, écart-type, médiane...), la production de visualisations variées selon le type de données analysées, l'export au format CSV ou PDF, mais aussi la gestion flexible des valeurs manquantes (par suppression, imputation ou remplacement) et la possibilité de fusionner ou concaténer plusieurs bases hétérogènes. En délimitant strictement son périmètre autour de l'analyse exploratoire et descriptive de données tabulaires, le projet exclut de fait l'implémentation de modèles d'apprentissage automatique ou de traitements distribués à grande échelle, afin de se concentrer sur l'accessibilité, la performance et la robustesse de l'outil pour des utilisateurs non experts.

# Chapitre 2

## Organisation du projet

Le projet a été réalisé sur une période de deux mois, structurée en trois grandes phases distinctes. La première phase, étalée sur le premier mois, a porté sur l’analyse préalable, la conception et la préparation technique. Cette étape a inclus une étude bibliographique approfondie des bibliothèques principales telles que Dash, pandas, Plotly et ReportLab, ainsi qu’une comparaison critique entre les technologies potentielles pour l’interface utilisateur, notamment Tkinter et Dash. Le choix s’est finalement porté sur Dash en raison de sa richesse graphique et de sa compatibilité avec les environnements web. Par ailleurs, un cahier des charges fonctionnel a été rédigé afin de définir précisément les fonctionnalités attendues, et une architecture logicielle modulaire a été conçue, organisant le code autour de modules distincts : app.py, callbacks.py, layouts.py et data-manager.py. La seconde phase, déroulée durant le deuxième mois, a consisté au développement effectif des fonctionnalités essentielles, incluant l’importation multi-fichiers des formats CSV et Excel, ainsi que les opérations de filtrage, de nettoyage des données et de calculs statistiques. Parallèlement, des composants dynamiques de visualisation graphique ont été créés, avec la génération automatique de graphiques adaptés selon le type de données. Le système d’export a également été implémenté pour permettre la sauvegarde des résultats au format CSV ou PDF. Le développement de l’interface utilisateur s’est appuyé sur Dash Bootstrap Components afin d’assurer une présentation ergonomique et responsive. Enfin, l’intégration des callbacks a permis d’orchestrer l’interaction entre l’interface et la logique métier, avec un ensemble de tests fonctionnels réalisés sur plusieurs jeux de données variés, couvrant différents formats, encodages et cas de valeurs manquantes. Cette phase a été clôturée par des ajustements ergonomiques visant à améliorer l’expérience utilisateur, notamment par la gestion des messages d’erreur et l’adaptation de l’affichage aux différentes tailles d’écran, ainsi que la rédaction finale du mémoire.

### 2.1 Difficultés rencontrées

Le développement du projet a été confronté à plusieurs défis techniques notables. L’un des premiers a concerné la gestion des fichiers d’entrée présentant des encodages variés, tels que UTF-8 ou ISO-8859-1, ce qui a nécessité la mise en place de mécanismes robustes pour garantir une lecture fiable et sans perte des données. Une autre difficulté majeure a porté sur l’harmonisation des valeurs manquantes provenant de sources disparates, afin d’assurer une cohérence dans leur traitement ultérieur. Par ailleurs, l’adaptation automatique et dynamique des visualisations graphiques en fonction du type qualitatif ou quantitatif des colonnes sélectionnées a requis une gestion fine des cas d’usage pour offrir une expérience

utilisateur fluide et pertinente. Enfin, le maintien de la cohérence de l'état global de l'application s'est avéré complexe, notamment lorsque plusieurs opérations de filtrage, concaténation ou tri étaient enchaînées, ce qui a conduit à une organisation rigoureuse des états internes gérés par les callbacks.

## 2.2 Choix technologiques

Dans le cadre de ce projet, deux principales options d'interface utilisateur ont été envisagées : Tkinter, offrant une interface locale traditionnellement utilisée pour les applications de bureau, et Dash, destiné à la création d'interfaces web interactives. Après une étude comparative, la décision s'est portée sur Dash, en raison de ses capacités fonctionnelles étendues, de son interactivité avancée ainsi que de ses possibilités graphiques optimales via la bibliothèque Plotly. En effet, Dash permet de concevoir des applications web réactives où la manipulation et la visualisation des données peuvent évoluer en temps réel, offrant ainsi une expérience utilisateur particulièrement fluide et adaptée à l'analyse exploratoire. L'environnement technologique retenu repose donc sur un ensemble cohérent de bibliothèques open source reconnues dans la communauté Python pour leur robustesse et leur modularité. La bibliothèque pandas assure le traitement et la manipulation efficace des données tabulaires. Plotly, et plus particulièrement son extension Dash, est utilisée pour la création de graphiques interactifs et la construction d'une interface web ergonomique, agrémentée par dash-bootstrap-components qui fournit un ensemble de styles et composants respectant les standards Bootstrap. Par ailleurs, la génération de rapports PDF est prise en charge par ReportLab, adaptée à la production de documents formalisés et exportables. Enfin, des modules standards tels que base64 et io ont été intégrés pour la gestion efficace des fichiers uploadés et des opérations d'export. Cette pile technologique garantit non seulement la modularité et la flexibilité du développement, mais également la facilité d'évolution et de maintenance de l'application, tout en s'assurant d'une compatibilité étendue avec les environnements web modernes.

## 2.3 Architecture logicielle

L'application est conçue selon une architecture modulaire clairement organisée autour de plusieurs fichiers principaux, permettant une séparation nette des responsabilités et facilitant la maintenance ainsi que l'évolution du code. Le fichier app.py constitue le point d'entrée de l'application. Il représente le cœur du système Dash, orchestrant l'ensemble des composants. Ce module initialise le serveur web, applique les styles graphiques en intégrant plusieurs thèmes Bootstrap, et assure l'interconnexion entre la gestion des données, les callbacks définissant la logique interactive, et l'interface utilisateur. Le fichier layouts.py est dédié à la définition et à la construction des éléments graphiques de l'interface. Il centralise la description complète du layout, assurant ainsi la dissociation entre la conception visuelle et la logique métier. Cette organisation garantit une meilleure clarté structurelle et une évolutivité facilitée du projet. Quant au module callbacks.py, il joue un rôle central en assurant le lien entre l'interface web et les traitements en arrière-plan. Il définit les différentes fonctions réactives qui répondent aux actions exécutées par l'utilisateur, telles que le chargement de fichiers, l'application de filtres, la sélection de colonnes pour l'analyse, ou encore les exportations au format PDF.

Enfin, le fichier `data-manager.py` constitue la couche de gestion centralisée des données tabulaires. Ce module prend en charge les opérations fondamentales : chargement, filtrage, nettoyage, calculs statistiques et export des données au format CSV. Il isole la logique métier de la présentation, conformément aux bonnes pratiques architecturales, et assure la cohérence des données manipulées dans l'ensemble de l'application. Cette organisation modulaire, inspirée du modèle MVC (Modèle-Vue-Contrôleur) adapté à Dash, favorise une répartition fonctionnelle claire et optimise la maintenabilité du code.

## 2.4 Typologie des utilisateurs ciblés

L'application développée dans ce projet s'adresse à un large éventail d'utilisateurs potentiels ayant des profils variés mais partageant un intérêt commun pour l'analyse et l'exploration de données tabulaires. Parmi ces utilisateurs, on distingue principalement les étudiants en data science ou en statistiques, qui bénéficient d'un outil simple pour manipuler des données sans avoir à recourir systématiquement à la programmation. De même, les analystes de données ne maîtrisant pas le langage Python constituent une cible privilégiée, car l'interface interactive développée permet d'effectuer des opérations complexes de filtrage, de nettoyage et de visualisation sans compétences techniques approfondies. Par ailleurs, cette application s'adresse aux enseignants et formateurs souhaitant intégrer dans leurs pratiques pédagogiques un outil accessible pour l'exploration guidée de jeux de données. Enfin, elle est également utile à toute personne disposant de fichiers de données au format CSV ou Excel et désireuse de les nettoyer, d'explorer rapidement leur contenu et de visualiser les informations essentielles à leur prise de décision ou à leur analyse métier. Ainsi, la conception fonctionnelle a été orientée pour maximiser l'accessibilité, l'intuitivité et la souplesse d'utilisation, afin de répondre efficacement aux besoins de ces différents profils d'utilisateurs, qu'ils soient novices ou utilisateurs occasionnels, dans un cadre académique, professionnel ou personnel.

# Chapitre 3

## Analyse détaillée des données et méthodologie

Cette section propose une analyse technique approfondie des principaux composants logiciels développés dans le cadre de ce projet. Elle vise à expliciter la structure et le fonctionnement du code contenu dans les fichiers majeurs de l'application : data-manager.py, app.py, callbacks.py et layouts.py.

L'objectif est de présenter la méthodologie adoptée pour gérer de manière modulaire et cohérente les traitements de données tabulaires, la construction de l'interface utilisateur, ainsi que l'orchestration des interactions dynamiques entre l'utilisateur et le système. Une attention particulière est portée à la manière dont chaque module contribue à la robustesse, à la flexibilité et à la maintenabilité générale de l'application, conformément aux bonnes pratiques du développement logiciel.

Cette analyse détaille notamment les choix architecturaux, les principales responsabilités des modules, les flux de données et les mécanismes d'interaction événementielle (callbacks). Elle illustre également les stratégies utilisées pour garantir la mise à jour en temps réel des données affichées, l'application efficace des filtres, le calcul des statistiques, ainsi que la génération des visualisations et des exportations.

### 3.1 Module callbacks.py : Gestion dynamique de l'interface et des interactions utilisateur

Le module callbacks.py constitue le noyau central de l'interactivité dans l'application Dash développée. Il regroupe l'ensemble des fonctions de callbacks, c'est-à-dire des fonctions réactives chargées de synchroniser en temps réel l'état des composants graphiques avec celui des données manipulées. Cette organisation assure une expérience utilisateur fluide, sans nécessiter de recharge complet de page, tout en intégrant une gestion robuste et complète des erreurs potentielles.

#### 3.1.1 Objectifs et rôle architectural

Le rôle fondamental de ce module est de faire le lien entre l'interface graphique et la logique métier portée par le gestionnaire de données (data-manager.py). À ce titre, il assure la cohérence globale de l'application en coordonnant des fonctions telles que l'import multi-fichiers, la gestion et le traitement des valeurs manquantes, le filtrage avancé,

la fusion et concaténation de bases, la réalisation de calculs statistiques, ainsi que la génération de visualisations graphiques interactives. La fonction centrale register-callbacks permet d'enregistrer facilement l'ensemble des comportements dynamiques au moment de l'initialisation, garantissant ainsi une architecture modulaire, claire et évolutive.

### 3.1.2 Principaux composants fonctionnels

#### 1. Importation robuste des fichiers

Une attention particulière est portée à la lecture sécurisée des fichiers CSV et Excel, qu'ils proviennent de sources diverses aux encodages variés (UTF-8, Latin1, etc.). Les fonctions utilitaires robust-read-csv et na-harmonize uniformisent les représentations des valeurs manquantes, évitant ainsi les plantages liés à une mauvaise interprétation des données. Cette homogénéisation est cruciale pour la stabilité et la qualité du traitement ultérieur.

#### 2. Gestion dynamique de la liste des jeux de données et options de fusion

Dès l'importation d'un ou plusieurs fichiers, la fonction update-bases-list met à jour l'affichage des bases chargées et identifie, si pertinent, les colonnes communes susceptibles de servir à une fusion. Cela répond aux besoins d'analyse multi-fichiers, permettant à l'utilisateur de préparer aisément la consolidation de bases hétérogènes.

#### 3. Callback centralisé de gestion des données

Au cœur du module réside la fonction manage-data, un callback unifié qui supervise toutes les interactions clés issues des actions utilisateur : importations, fusions, concaténations, suppressions, nettoyages des valeurs manquantes, application de filtres simples ou complexes, et recherches textuelles. L'utilisation de ctx.triggered-id permet d'identifier précisément la source de l'événement déclencheur pour exécuter le traitement adapté. Cette centralisation réduit les risques de conflit d'état, améliore la lisibilité, et favorise une maintenance facilitée. Après chaque action, l'interface est automatiquement rafraîchie pour refléter les modifications sur la table, les messages d'état, et les options disponibles.

#### 4. Calcul dynamique des statistiques descriptives

Le callback calculats-statistics génère, à la demande, un ensemble d'indicateurs statistiques standard tels que la moyenne, l'écart-type, la médiane, le minimum et le maximum, calculés sur une colonne numérique sélectionnée. Une validation stricte limite cet accès aux colonnes numériques, avec une rétroaction claire en cas d'invalidité.

#### 5. Génération adaptative de visualisations interactives

La fonction generate-visualization détecte automatiquement la nature qualitative ou quantitative des colonnes sélectionnées et choisit en conséquence un type de graphique approprié (nuage de points, boîte à moustaches, histogramme, camembert, etc.). Ce mécanisme permet à l'utilisateur d'explorer ses données selon différentes dimensions sans nécessiter de connaissances graphiques spécifiques.

#### 6. Export des données au format CSV et PDF

L'application propose une double modalité d'export : un export CSV natif, simple et performant, et un export PDF élaboré via la bibliothèque ReportLab. Ce dernier garantit un rendu professionnel avec gestion du formatage, titrage, ajustement automatique des

colonnes, traitement des longues valeurs, et prise en charge de polices universelles pour prévenir les erreurs d'encodage.

## 3.2 Module data-manager.py

Le module data-manager.py joue un rôle fondamental dans l'architecture logicielle de l'application, en assurant la gestion centralisée et cohérente des données tabulaires manipulées. Porté par la classe DataManager, ce composant encapsule l'ensemble des opérations essentielles au traitement des jeux de données : importation, homogénéisation des valeurs manquantes, application et combinaison de filtres logiques complexes, tri, calculs statistiques descriptifs, nettoyage par suppression ou imputation, ainsi que l'export au format CSV. Cette séparation claire des responsabilités, isolant la logique métier de la présentation graphique, est conforme aux bonnes pratiques d'ingénierie logicielle et garantit la robustesse, la maintenabilité et l'évolutivité de l'application. Les méthodes proposées offrent une interface simple et modulaire, facilitant l'interaction avec les données tout en assurant une réactivité optimale dans le cadre de l'interface Dash.

### 3.2.1 Fonctionnalités principales

#### 1. Importation et harmonisation des valeurs manquantes

Lors du chargement d'un jeu de données, la méthode load-data garantit que toutes les valeurs manquantes, quelle que soit leur représentation initiale (chaînes vides, « NA », « N/A », « NAN », etc.), sont uniformisées en pd.NA via la méthode harmonize-na. Cette normalisation homogénéise le traitement statistique et facilite les opérations ultérieures.

#### 2. Application de filtres multiples et combinés

La classe gère une liste de filtres combinables via des opérateurs logiques « et » ou « ou ». Chaque filtre est défini par la colonne ciblée, la condition (par exemple « > », « < », « contient »), la valeur de comparaison et l'opérateur logique. La méthode apply-all-filters applique séquentiellement ces conditions, produisant un sous-ensemble cohérent et dynamique des données initiales.

#### 3. Tri des données

Un tri dynamique est possible sur le jeu complet ou filtré selon une colonne spécifiée et un ordre croissant ou décroissant, selon les besoins de visualisation ou d'analyse.

#### 4. Traitement des valeurs manquantes

Plusieurs stratégies sont proposées pour la gestion des valeurs absentes : suppression des lignes ou colonnes concernées, suppression des doublons, remplacement par la moyenne ou la médiane (imputation statistique), ou par une valeur personnalisée. Ces opérations sont accessibles facilement par l'interface utilisateur et assurent un nettoyage flexible des jeux de données.

#### 5. Calcul des statistiques descriptives

Pour les colonnes numériques, le module calcule un ensemble complet d'indicateurs : nombre de valeurs non manquantes, moyenne, médiane, écart-type, minimum, maximum, somme, ainsi que le nombre de valeurs manquantes. Ces statistiques sont conservées en cache pour optimiser les consultations répétées.

## **6. Fourniture d'informations sur les colonnes**

La méthode get-column-info renvoie des métadonnées pour chaque colonne (type de données, nombre de valeurs nulles, nombre de valeurs uniques, indication de type numérique), informations utiles pour configurer dynamiquement les composants de l'interface.

## **7. Export des données**

La classe offre la possibilité d'exporter le jeu de données tel qu'il est actuellement affiché (filtré ou non) sous forme de chaîne CSV, facilitant la génération de fichiers à télécharger.

### **3.2.2 Contribution dans l'architecture globale**

En se positionnant clairement dans la couche « modèle » (Model) de l'application, le module data-manager.py assure l'isolation de la logique métier par rapport aux aspects visuels et interactifs développés avec Dash. Cette structuration favorise une meilleure maintenabilité, facilite la réutilisation du code, et permet de garantir la cohérence et la performance des opérations de gestion des données.

## **3.3 Module layouts.py : Construction de l'interface utilisateur graphique avec Dash**

Le module layouts.py joue un rôle central dans la présentation visuelle et ergonomique de l'application Dash. Il est responsable de la définition complète de la structure graphique (layout) qui sera affichée dans le navigateur, en organisant les différents composants interactifs et éléments de contenu selon une disposition claire, hiérarchisée et responsive.

### **3.3.1 Organisation générale**

Le layout principal est construit à l'aide de la bibliothèque Dash Bootstrap Components (dbc), qui facilite la création d'interfaces web basées sur le système de grille responsive Bootstrap. Cette approche garantit une présentation harmonieuse et adaptable aux différentes tailles d'écran, tout en offrant des composants graphiques esthétiques et cohérents. Le contenu est encapsulé dans un composant dbc.Container avec fluidité activée (fluid=True), ce qui procure un remplissage horizontal souple. Ce container contient une succession ordonnée de rows (dbc.Row), elles-mêmes composées d'une ou plusieurs columns (dbc.Col), permettant ainsi de structurer le contenu en sections verticales et supports horizontaux modulables.

### **3.3.2 Structure fonctionnelle du layout**

#### **1. En-tête et introduction**

Le haut de page regroupe un titre principal (avec une icône emoji pour symboliser l'analyse), une ligne horizontale de séparation, et une courte phrase d'introduction. Cette section donne un contexte immédiat à l'utilisateur sur l'objet et les capacités de l'outil.

## **2.Zone d'import et de fusion des fichiers**

Une carte (dbc.Card) contient le composant dcc.Upload permettant le chargement de plusieurs fichiers simultanément via glisser-déposer ou sélection. Ce bloc comprend également une zone de feedback (upload-status) pour informer l'utilisateur sur le statut des importations.

## **3.Options de fusion et de gestion des bases**

Des boutons d'action (fusion, concaténation, suppression) sont disposés horizontalement, accompagnés d'un menu déroulant (dcc.Dropdown) pour sélectionner une colonne commune, facilitant des opérations avancées sur plusieurs jeux de données. Un affichage dynamique liste les bases couramment chargées, et un autre bloc permet la gestion fine (suppression totale ou partielle).

## **4.Recherche et filtres avancés**

Un input texte combiné à un bouton offre une fonction de recherche globale sur les données. Parallèlement, un bloc de contrôle sous forme de carte propose un système de filtres multi-critères avec sélection de la colonne, condition, valeur, et opérateur logique « ET » ou « OU », accompagnés de boutons d'application et de réinitialisation.

## **5.Gestion des valeurs manquantes**

Ce bloc regroupe les boutons permettant de supprimer les lignes ou colonnes contenant des valeurs manquantes, supprimer les doublons, ou de les remplacer par la moyenne, la médiane ou une valeur personnalisée. Il inclut également un champ de saisie et un bouton de réinitialisation, apportant à l'utilisateur un contrôle complet sur le nettoyage des données.

## **6.Tri et calculs statistiques**

Une nouvelle carte propose la sélection d'une colonne pour le tri (avec choix de l'ordre croissant ou décroissant), et une sélection dédiée au calcul des statistiques descriptives sur une colonne numérique avec un bouton de déclenchement.

## **7.Affichage des statistiques et tableau des données**

Deux cartes distinctes présentent respectivement les résultats statistiques calculés et la table interactive des données, incorporée par une zone html.Div mise à jour dynamiquement via callbacks.

## **8.Visualisations graphiques**

Une carte permet de configurer la visualisation via trois dropdowns définissant les axes X, Y (optionnel) et la variable de couleur pour les graphiques. Un bouton lance la génération automatique d'un graphique adapté, rendu dans un composant dcc.Graph situé dans une rangée dédiée. Ce dispositif offre de la flexibilité tout en simplifiant l'exploitation des données visuelles.

## **9.Export des données**

Un groupe de boutons permet aux utilisateurs de télécharger les résultats au format CSV ou PDF, assurant une exportabilité simple et directe.

## **10.Composants de stockage invisible**

Plusieurs composants dcc.Store sont placés dans la partie inférieure du layout pour gérer côté client l'état des données, des bases importées et des filtres. Cette solution améliore la réactivité et l'efficacité en évitant de charger ces informations côté serveur à chaque interaction.

## **3.4 Module app.py : point d'entrée principal de l'application Dash**

Le fichier app.py constitue le point d'entrée principal de l'application web développée avec Dash. Il assure l'initialisation globale du système, la connexion des différents modules, ainsi que le lancement du serveur local pour le déploiement en environnement de développement.

### **3.4.1 Structure fonctionnelle de app**

#### **1.Initialisation et configuration**

La première étape consiste à définir une liste de feuilles de style externes (external-stylesheets), intégrant plusieurs thèmes Bootstrap. Ces thèmes multiples offrent la possibilité d'un changement dynamique ou d'une adaptation graphique selon les préférences utilisateur, garantissant ainsi une interface sobre, moderne et professionnelle. Parmi les thèmes inclus figurent Minty, Pulse, Sandstone, Litera, Darkly, Cyborg, et Solar, complétés par une feuille de style spécifique provenant d'un CDN externe pour uniformiser certains composants Dash Bootstrap.

La fonction load-figure-template est appelée afin d'associer un thème Plotly cohérent à l'un des thèmes Bootstrap sélectionnés. Cette synchronisation des styles permet d'harmoniser l'apparence graphique des visualisations avec celle de l'interface utilisateur, assurant une expérience visuelle homogène et agréable.

#### **2. Création de l'application Dash**

L'instance principale de l'application Dash est créée via la classe Dash, qui prend en paramètres le nom du module Python (name), les styles externes définis et une option suppress-callback-exceptions=True. Cette dernière est nécessaire pour gérer des layouts dynamiques, où des composants peuvent être ajoutés ou supprimés à la volée. Le titre de l'application est fixé à "Analyseur de Données", ce qui s'affiche dans l'onglet du navigateur.

#### **3. Gestion centralisée des données**

L'objet DataManager est instancié dans ce module, jouant le rôle de gestionnaire central des données importées, transformées et visualisées tout au long de la session utilisateur. Cette centralisation permet d'assurer l'intégrité et la cohérence des opérations sur les jeux de données, avec une séparation nette entre la logique métier et l'interface graphique. Dans certains cas, cette instance pourrait être remplacée par None pour désactiver cette gestion spécifique.

#### **4. Mise en place de l'interface utilisateur**

Le layout principal de l'application est créé par l'appel à la fonction `create-layout()` importée depuis le module `layouts.py`. Cette fonction définit la structure visuelle complète de l'application (menus, zones d'affichage, composants interactifs), garantissant ainsi une séparation claire entre la définition de l'interface et la logique fonctionnelle sous-jacente.

#### **5. Enregistrement des callbacks**

Les relations entre les composants de l'interface utilisateur et les traitements logiques déclenchés sont établies par l'appel à la fonction `register-callbacks()`, importée depuis `callbacks.py` et paramétrée avec l'instance `app` et le gestionnaire de données `data-manager`. Cette organisation respecte les principes de développement déclaratif et modulaire recommandés dans la documentation Dash, facilitant la maintenabilité et l'évolutivité du projet.

#### **6. Lancement du serveur**

Enfin, l'application est lancée en mode développement local via la méthode `app.run()`, configurée pour exécuter le serveur web sur le port 8051 avec le mode debug activé. Cette configuration favorise un cycle de développement interactif, avec un recharge automatique à la moindre modification du code, facilitant ainsi les phases de mise au point et d'évolution.

# Chapitre 4

## Documentation utilisateur

### 4.1 Présentation générale

L'Analyseur de Données est une application web interactive développée en Python avec Dash, destinée aux utilisateurs souhaitant charger, fusionner, filtrer, analyser et visualiser facilement leurs fichiers de données (CSV, Excel). Elle est adaptée pour un usage académique ou professionnel où la manipulation rapide et visuelle des données tabulaires est nécessaire.

### 4.2 Interface utilisateur

L'application se décompose en plusieurs modules fonctionnels :

#### 4.2.1 Chargement et fusion des données

Les utilisateurs peuvent importer un ou plusieurs fichiers au format CSV ou Excel en utilisant la zone "Charger et fusionner des fichiers". Une fois importés, les fichiers apparaissent dans la liste des bases disponibles. Si plusieurs bases ont été chargées, il est possible de fusionner ou concaténer les deux dernières en vous appuyant sur une colonne commune que vous sélectionnez.



**Figure 1 :** Interface de chargement et fusion des données

## 4.2.2 Gestion des bases chargées

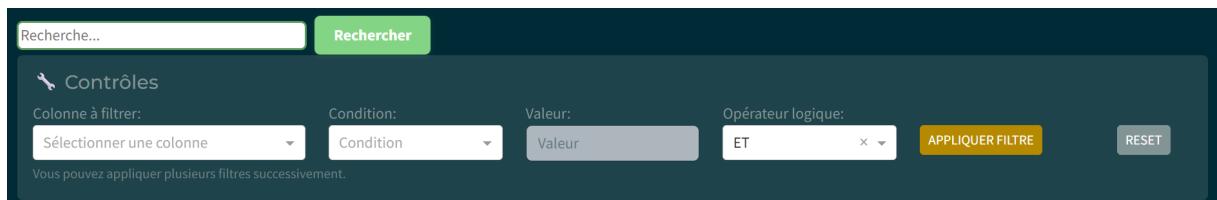
Supprimez toutes les bases ou sélectionnez-en une à supprimer via la zone « Gestion des bases chargées ».



**Figure 2 :** Interface gestion des bases chargées

## 4.2.3 Recherche et filtres avancé

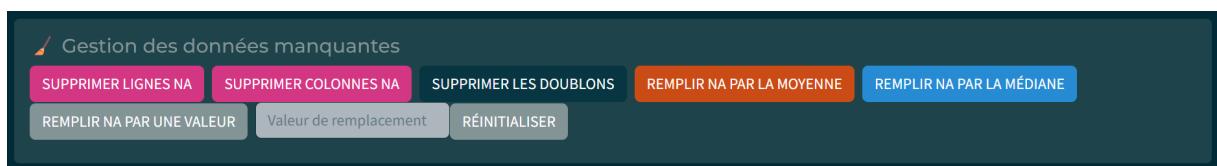
Un moteur de recherche global permet de retrouver rapidement une valeur. Des filtres avancés (colonnes, conditions, valeurs cibles, logique ET/OU) permettent une exploration fine et multi-critères des données.



**Figure 3 :** Interface de recherche et filtres avancés

## 4.2.4 Gestion des valeurs manquantes

L'utilisateur dispose d'outils dédiés pour gérer les données manquantes : suppression (lignes/colonnes), remplacement par la moyenne, médiane ou valeur spécifique, supprimer les doublons ou réinitialisation complète.



**Figure 4 :** Interface gestion des valeurs manquantes

## 4.2.5 Statistiques

La sélection d'une colonne numérique déclenche, sur demande, le calcul et l'affichage des indicateurs-clés : moyenne, médiane, écart-type, etc.

The screenshot shows a dark-themed user interface for descriptive statistics. At the top, there are three dropdown menus: 'Trier par:' (Column), 'Ordre:' (Ascending), and 'Statistiques pour:' (Numerical column). A blue 'CALCULER' button is located to the right of the third menu. Below these, a checkbox labeled 'Statistiques' is checked, and a message says 'Sélectionnez une colonne et cliquez sur "Calculer"' (Select a column and click on "Calculate").

**Figure 5 :** Interface statistiques descriptives

#### 4.2.6 Visualisations

Le module de visualisation adapte automatiquement le type de graphique généré selon la structure des données sélectionnées. Les graphiques sont interactifs (zoom, survol, légende dynamique via Plotly), favorisant l’exploration visuelle.

The screenshot shows a dark-themed user interface for graph visualization. It has three dropdown menus: 'Axe X' (Column X), 'Axe Y (optionnel)' (Column Y), and 'Colorer par (optionnel)' (Variable color). Below these is a green button labeled 'GÉNÉRER AUTOMATIQUEMENT UN GRAPHIQUE ADAPTÉ' (Generate automatically an adapted graph).

**Figure 6 :** Interface visualisation graphique

#### 4.2.7 Export des données

L’utilisateur peut exporter à tout moment le contenu affiché sous forme de fichier CSV ou PDF. L’export PDF privilégie la lisibilité avec une mise en forme adaptée aux 50 premières lignes.

The screenshot shows a dark-themed user interface for data export. It features a blue 'Export' icon and two buttons: 'TÉLÉCHARGER CSV' (Download CSV) in a yellow box and 'TÉLÉCHARGER PDF' (Download PDF) in a white box.

**Figure 7 :** Interface export des fichiers csv/pdf

# Chapitre 5

## Exemple pratique : Exploitation des données culturelles et socio-économiques départementales dans l'application Dash

### 5.1 Introduction

Dans le cadre de ce mémoire, cette section pratique vise à présenter de manière concrète les fonctionnalités de l'application « Analyseur de Données », développée sous Dash. L'objectif est d'illustrer l'intérêt et la faisabilité d'une approche interactive pour l'analyse croisée de données issues de différentes sources publiques, telles que l'INSEE et le Ministère de la Culture. Cette démonstration s'appuie sur des jeux de données publics, soigneusement préparés en amont pour garantir leur adéquation avec l'outil développé. Avant toute exploitation, un travail de sélection des indicateurs pertinents et d'harmonisation des intitulés de colonnes (tels que « Département », « Ciné-100k-hab », « Salaire-moyennet-an », etc.) a été mené, afin de favoriser la clarté et la cohérence des analyses. Les bases considérées sont :

- Base équipements culturels par département : recensement des cinémas, bibliothèques, musées et autres établissements culturels pour 100 000 habitants, ainsi que des indicateurs économiques comme le salaire net annuel moyen et les dépenses culturelles par habitant (100 lignes, 10 variables) ;
- Base indicateurs socio-économiques : population totale, salaire moyen mensuel net, taux de chômage par département (100 lignes, 4 variables).

L'exemple déroule, étape par étape, la manière dont un·e utilisateur·rice peut importer, explorer, fusionner et analyser ces données à partir d'une interface conçue avec Dash. Chaque étape est illustrée dans le mémoire par des captures d'écran, mettant en valeur les interactions et résultats obtenus.

### 5.2 Méthodologie

#### 5.2.1 Prérequis techniques

Pour utiliser l'Analyseur de Données, il convient de procéder comme suit :

- Disposer d'une installation Python 3.7 ou supérieure.
- Avoir installé les bibliothèques nécessaires : dash, dash-bootstrap-components, pandas, plotly, et reportlab.

Cette installation s'effectue via la commande :

```
pip install dash dash-bootstrap-components pandas plotly reportlab
```

- Le regroupement dans un même dossier de tous les scripts du projet (app.py, layouts.py, callbacks.py, data-manager.py) et des fichiers de données utilisés.

### 5.2.2 Initialisation de l'application

Dans le terminal, il convient d'exécuter

```
python app.py
```

```
Dash is running on http://127.0.0.1:8051/
* Serving Flask app 'app'
* Debug mode: on
```

**Figure 8** : Interface de l'application accessible à l'adresse `http://127.0.0.1:8051`

## 5.3 Étapes d'analyse interactive

### 5.3.1 Chargement et préparation des jeux de données

L'utilisateur importe les fichiers Equipement-culturels.xlsx et revenus.csv par l'intermédiaire du module d'upload accessible depuis l'interface (glisser-déposer ou sélection classique). L'application détecte automatiquement les en-têtes, harmonise les valeurs manquantes (NA, N/A, etc.) et affiche pour chaque base le nombre de lignes et de colonnes. La réussite du chargement est confirmée par des messages d'état, tels que « 2 base(s) chargée(s) ». Par exemple :

- La base « Equipement-culturels » affiche 100 lignes et 10 colonnes ;
- La base « revenus.csv » comporte 100 lignes et 5 colonnes.

Étant donné la présence de deux bases importées, l'interface propose plusieurs options : fusionner les bases sur une colonne commune, supprimer la dernière base chargée, ou concaténer les bases. L'utilisateur choisit ici de fusionner les deux bases par la colonne commune « Département ». Un message de confirmation « Base fusionnée par Département » est alors affiché.



FIGURE 5.1 – Confirmation de la réussite du chargement avec le message d'état « 2 base(s) chargée(s) »

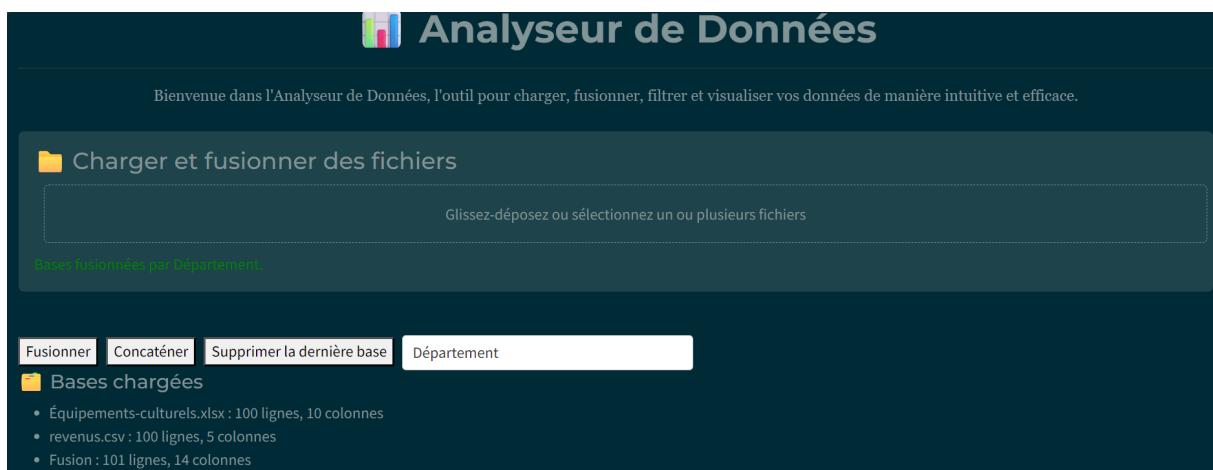


FIGURE 5.2 – Fusion de deux bases importées par la colonne « Département » avec affichage du message de confirmation

### 5.3.2 Gestion des bases chargées

L'utilisateur dispose des fonctionnalités permettant soit de supprimer toutes les bases chargées pour recommencer une analyse propre, soit de supprimer une base sélectionnée individuellement. Par exemple, en supprimant la dernière base, qui correspond ici à la base fusionnée, un message de confirmation précise que la base « Fusion » a été supprimée.



FIGURE 5.3 – Suppression de la base fusionnée avec affichage du message de confirmation « Fusion supprimée »

### 5.3.3 Recherche au sein des données fusionnées

L'interface offre un champ de recherche textuelle : en y saisissant « Ain » (nom d'un département), la variable apparaît en surlignage vert dans la table, facilitant ainsi la localisation rapide d'informations relatives au département concerné.

FIGURE 5.4 – Champ de recherche textuelle permettant de filtrer les données par mot-clé, ici « Ain »

Département	Ciné_100k_hab	Biblio_100k_hab	Spectacles_100k_hab	Etablissements_cult_100k_hab	Musées_100k_hab	Salaire_mc
filter data...						
Ain	3.2	36.9	0.8	0.9	2.6	27420
Charente	3.9	22.4	1.4	1.4	1.9	25620
charente-Maritime	4.4	35.2	1.5	0.9	3.2	25236
Corrèze	4	56.6	2	1.2	2.8	24732
Creuse	1.1	17.7	0.5	0.2	0.5	22932
Deux-Sèvres	3.6	38.2	2.1	1.3	2.3	26112
Dordogne	3.8	52.1	1.9	0.7000000000000001	4.7	23868
Gironde	2.9	18.4	2	0.6000000000000001	1.2	28560

FIGURE 5.5 – Affichage de la variable correspondant au mot-clé « Ain » avec surlignage dans la table

### 5.3.4 Application de filtres avancés

Dans la section dédiée aux contrôles, l'utilisateur peut appliquer des filtres conditionnels sur les colonnes souhaitées. Par exemple, en sélectionnant la colonne « Ciné-100k-hab » et la condition « > 4 », l'outil filtre automatiquement les départements répondant à ce critère, tels que les Alpes-Maritimes, Ardèche, ou Ariège. Les données filtrées sont ensuite affichées dans la table. Un bouton « Reset » permet de réinitialiser les données à l'état d'origine.

FIGURE 5.6 – Remplissage des champs de filtrage : sélection de la colonne « Ciné-100k-hab » et de la condition « > 4 »

**Données**

Export

♦Département	♦Ciné_100k_hab	♦Biblio_100k_hab	♦Spectacles_100k_hab	♦Etablissements_cult_100k_hab	♦Musées_100k_hab	♦Salaire_moyen_euro_net_ans
filter data...						
Alpes-Maritimes	26.7	86.7	4.2	7.7	22.6	28980
Ardèche	7.1	82.5	1.4	1.4	3.2	25524
Ariège	7	52.8	1.9	0.6000000000000001	3.8	24348
Aveyron	5.5	70.8	1.7000000000000002	0.7000000000000001	4.1	24252
Calvados	4.4	19.2	1.7000000000000002	1.8	3.8	25932
Cantal	5.3	97.9	2	1.3	4.7	23028
Charente-Maritime	4.4	35.2	1.5	0.9	3.2	25236
Corse-du-Sud	6.9	31.5	1.3	0.6000000000000001	6.9	25368
Côte-d'Or	13.1	186.7	5.7	6.6	18	27072
Gers	8.1	58.6	1	1	3.5	25572
Haute-Corse	5.5	10.4	1.6	2.2	4.4	24744
Haute-Loire	5.1	80.7	1.3	0.4	2.6	24216

FIGURE 5.7 – Affichage des départements répondant au filtre « Ciné-100k-hab > 4 » dans la table

### 5.3.5 Gestion des valeurs manquantes

Le tableau de bord fournit une liste dynamique indiquant les colonnes contenant des données manquantes ainsi que leur nombre. Par exemple, la variable « Salaire-moyen(euro-net-ans) » comporte certaines valeurs absentes. L’utilisateur peut alors choisir diverses opérations : suppression des lignes ou colonnes manquantes, suppression des doublons, remplissage des NA par la moyenne, la médiane, ou une valeur personnalisée. Ici, l’utilisateur choisit de remplir les valeurs manquantes par la moyenne, ce qui est confirmé par un message d’état.

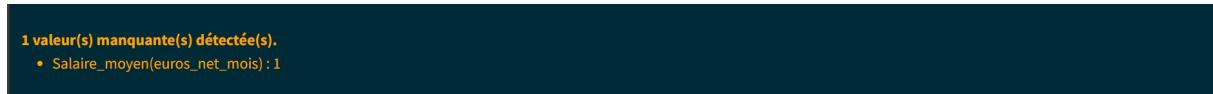


FIGURE 5.8 – Liste dynamique des colonnes avec valeurs manquantes, ici la variable « Salaire-moyen(euro-net-ans) »



FIGURE 5.9 – Remplissage des valeurs manquantes par la moyenne, confirmé par un message d’état

### 5.3.6 Calculs statistiques

Pour toute colonne numérique sélectionnée, un bouton « Calculer » génère un récapitulatif statistique comprenant le nombre de valeurs, la moyenne, la médiane, l’écart-type, les valeurs minimale et maximale et le nombre de valeurs manquantes. Par exemple, en

sélectionnant la colonne « Salaire-moyen-net-an », ce tableau statistique est affiché de manière synthétique et lisible.

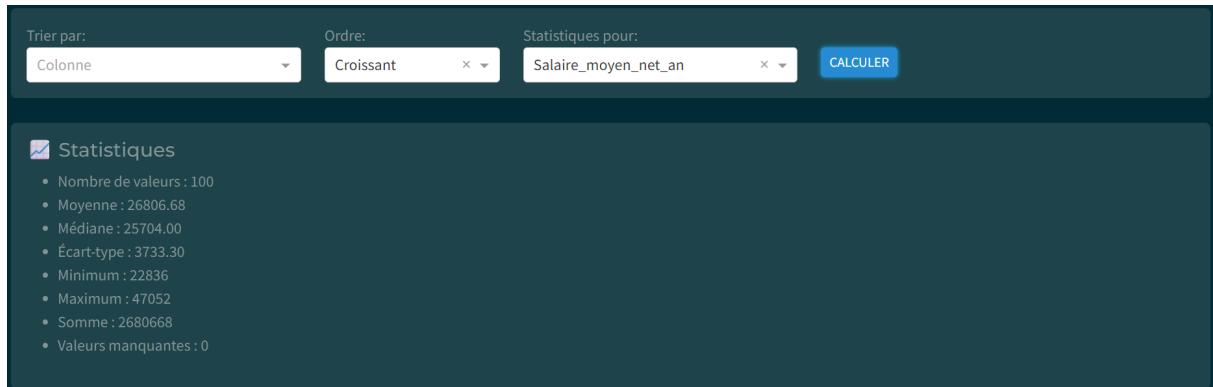


FIGURE 5.10 – Récapitulatif statistique synthétique pour la colonne numérique « Salaire-moyen-net-ans » généré par le bouton « Calculer »

### 5.3.7 Visualisation graphique

L'utilisateur peut choisir les variables pour les axes X et Y ainsi que la variable de coloration sur des listes

déroulantes. Par exemple, un graphique est généré avec :

- Axe X : « Département »
- Axe Y : « Salaire net par an »
- Colorer par : « Salaire net par an »

Le système génère automatiquement un graphique adapté aux données sélectionnées (diagramme en barres, boîte à moustaches ou nuage de points).

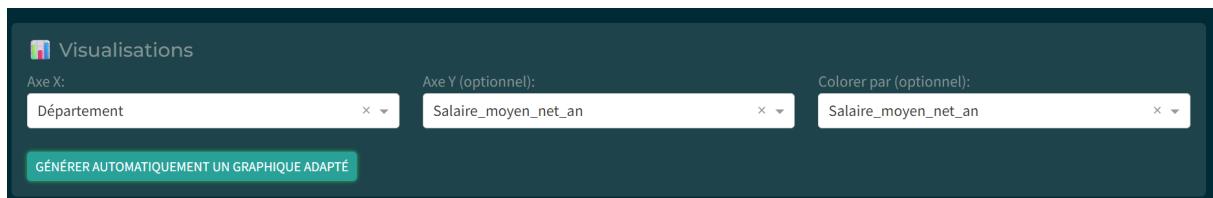


FIGURE 5.11 – Choix des variables : Axe X « Département », Axe Y « Salaire net par an », et coloration par « Salaire net par ans »

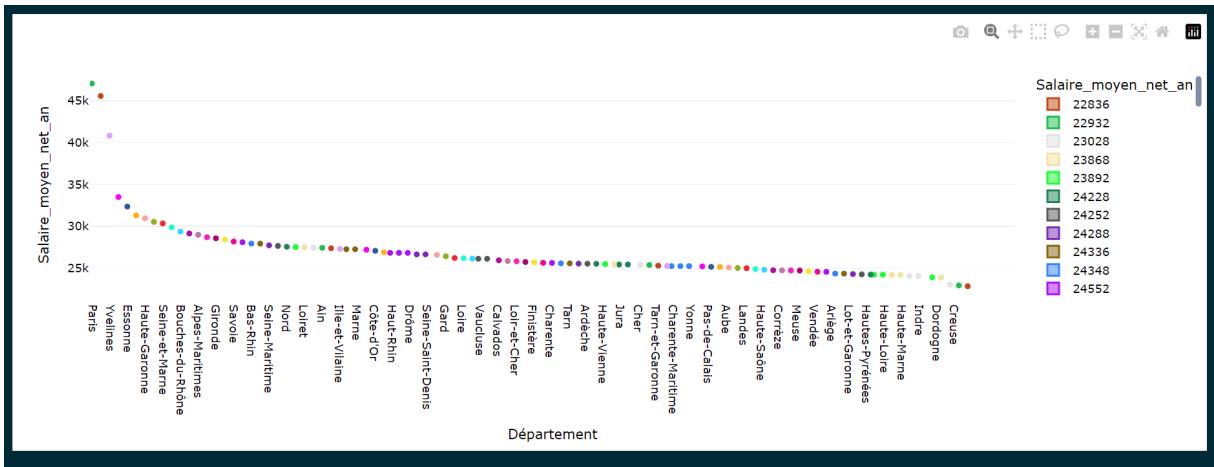


FIGURE 5.12 – Graphique résultant (diagramme en barres) généré selon les paramètres sélectionnés

### 5.3.8 Export des données

Enfin, l'application propose de télécharger les données sous forme de fichiers CSV, ainsi que la génération automatique d'un rapport PDF reprenant les données (limité à 50 lignes par souci de lisibilité). L'export peut être déclenché en un clic via les boutons dédiés.

# Chapitre 6

## Conclusion

Ce mémoire a abouti à la réalisation d'une application Python modulaire dotée d'une interface web interactive, conçue pour la gestion, l'analyse et la visualisation avancée de données tabulaires. Développée avec Dash, Pandas et Plotly, cette solution intègre des fonctionnalités clés telles que la fusion de fichiers, le filtrage dynamique, le traitement des données manquantes, l'analyse statistique et l'exportation des résultats.

Cette approche modulaire et professionnelle facilite la maintenance, la réutilisabilité du code ainsi que l'évolution future de l'application, tout en renforçant mes compétences en développement Python, structuration logicielle, manipulation de données et conception d'interfaces graphiques. Répondant aux exigences d'un mémoire de master axé sur le développement d'outils d'analyse de données, ce travail établit les fondations d'une plateforme évolutive dont plusieurs perspectives d'amélioration ont été identifiées, notamment :

- la mise en place d'une gestion multi-utilisateur sécurisée, avec authentification et profils personnalisables ;
- le renforcement de la gestion des erreurs et du diagnostic utilisateur,
- l'intégration directe de bases de données SQL et NoSQL, pour assurer une synchronisation et une mise à jour dynamique des données ;
- l'optimisation des performances pour le traitement de grands volumes de données,
- le développement d'analyses prédictives et d'algorithmes d'apprentissage automatique,

Ces axes consolidant l'application comme un outil adaptable à des contextes professionnels complexes, offrant un socle robuste pour de futurs développements technologiques et scientifiques.

# Bibliographie

## Livres

- Walkenbach, J. (2013). *Excel 2013 Bible*. Wiley Publishing.

## Articles scientifiques

- Gelernter, H. (1988). *Data Tables : History and Principles*. Computing Reviews, 29(4), 45–52.
- Codd, E. F. (1970). *A Relational Model of Data for Large Shared Data Banks*. Communications of the ACM, 13(6), 377–387.

## Sites web

- fpdf2 contributors. *fpdf2 – PDF creation library for Python*. <https://py-pdf.github.io/fpdf2/>
- Inconnu. *Tutoriel Python - Gestion des données*. <https://fr.python-3.com/?p=79>
- Various. *ResearchGate - Réseau scientifique*. <https://www.researchgate.net>

## Documentation logicielle

- Plotly Technologies Inc. *Dash : A Python Framework for Building Analytical Web Applications*. <https://dash.plotly.com/>
- Plotly Technologies Inc. *Plotly Graphing Libraries for Python*. <https://plotly.com/python/>
- ReportLab. *The ReportLab User Guide*. <https://www.reportlab.com/docs/reportlab-userguide.pdf>
- The Pandas Development Team. *pandas : powerful Python data analysis toolkit*. <https://pandas.pydata.org/>

## Outils d'IA

- OpenAI's ChatGPT. *Je les ai mobilisés pour retravailler la conclusion et concevoir le quiz*. (2025)
- GitHub Copilot. *Aide à rédiger une partie du code et des commentaires*. Utilisé pour la gestion des données manquantes. (2025)

## Annexes

### Quiz sur l'application « Analyseur de Données »

#### 1. Fonctionnalités générales et utilisation de l'application Analyseur de Données

1. Quelle est la principale fonctionnalité de l'application « Analyseur de Données » développée sous Dash ?
  - a) Collecter des données en temps réel
  - b) Analyser et visualiser des données issues de sources publiques de manière interactive
  - c) Gérer une base de données MySQL
  - d) Automatiser des tâches administratives
2. Quels types de données sont utilisés dans l'exemple pratique du mémoire ?
  - a) Données climatiques et environnementales
  - b) Données culturelles et socio-économiques départementales
  - c) Données médicales et hospitalières
  - d) Données financières et boursières
3. Quelle bibliothèque Python n'est pas mentionnée comme prérequis pour utiliser l'application ?
  - a) pandas
  - b) dash-bootstrap-components
  - c) Matplotlib
  - d) reportlab
4. Quelle commande permet d'installer les bibliothèques nécessaires pour l'application ?
  - a) install dash dash-bootstrap-components pandas plotly reportlab
  - b) pip install dash dash-bootstrap-components pandas plotly reportlab
  - c) python setup.py install dash dash-bootstrap-components pandas plotly reportlab
  - d) pip install matplotlib dash pandas
5. Quelle opération peut-on réaliser après avoir importé plusieurs bases de données dans l'interface ?
  - a) Fusionner les bases sur une colonne commune
  - b) Transformer les bases en bases de données relationnelles
  - c) Indexer les bases dans Elasticsearch
  - d) Créer un modèle prédictif
6. Que fait l'application lorsqu'on filtre la colonne « Ciné-100k-hab » avec la condition « > 4 » ?
  - a) Affiche les départements dont le nombre de cinémas pour 100 000 habitants est supérieur à 4
  - b) Supprime les valeurs inférieures à 4
  - c) Remplace les données manquantes par la valeur 4
  - d) Génère un graphique en camembert
7. Comment l'utilisateur peut-il gérer les valeurs manquantes dans les données ?

- a) En supprimant systématiquement toutes les lignes contenant des NAs
  - b) En remplissant les valeurs manquantes par la moyenne, la médiane ou une valeur personnalisée
  - c) En désactivant les colonnes contenant des valeurs manquantes
  - d) L'application ne propose aucune gestion des valeurs manquantes
8. Quel type de graphique Dash génère-t-il automatiquement selon les variables sélectionnées ?
- a) Diagramme en barres, boîte à moustaches ou nuage de points
  - b) Carte thermique uniquement
  - c) Graphique en secteurs exclusivement
  - d) Graphique en ligne uniquement
9. Quel format de rapport peut être exporté automatiquement par l'application ?
- a) DOCX
  - b) PDF
  - c) XLSX
  - d) HTML
10. Sur quelle adresse l'application Dash est-elle accessible par défaut lors de son lancement ?
- a) http://localhost:5000
  - b) http://127.0.0.1:8051
  - c) http://0.0.0.0:4000
  - d) http://192.168.1.1:8080

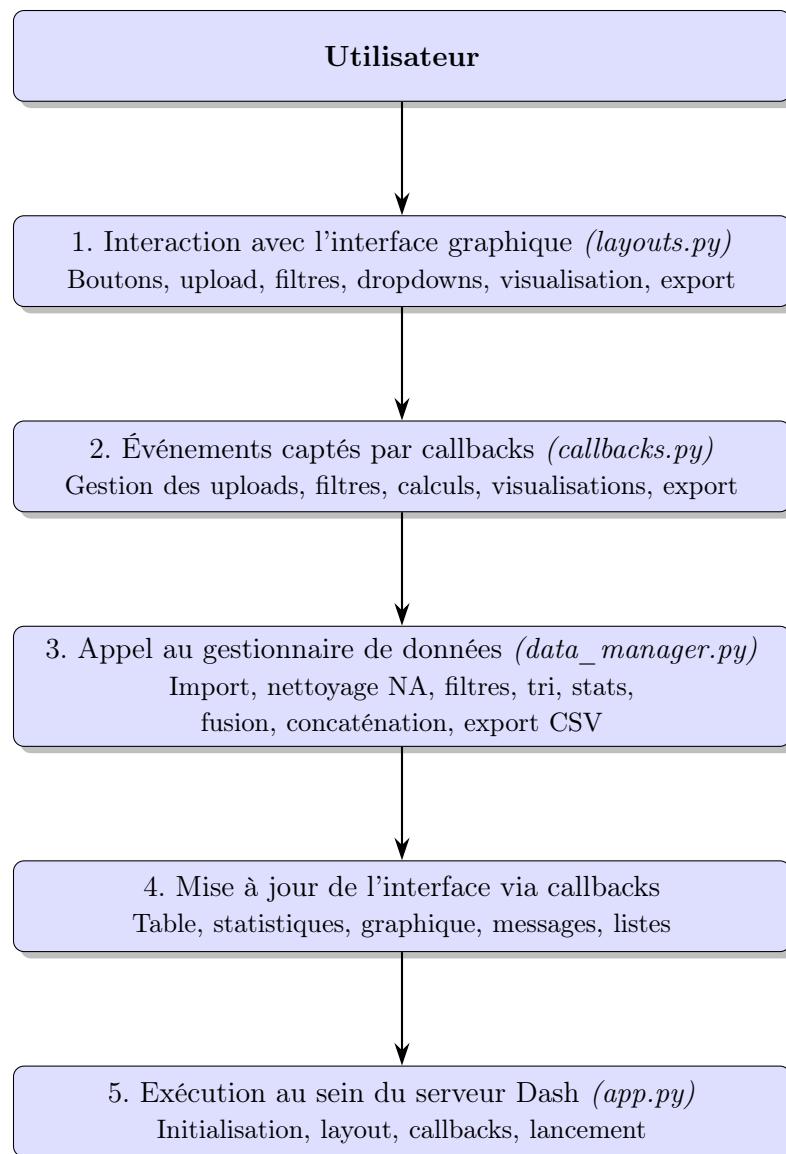
## 2. Gestion des erreurs et conseils en cas de problème d'importation

[resume] Que faire si l'importation de la base de données échoue dans l'application Dash ?

1. a) Vérifier que les fichiers sont au bon format (Excel, CSV) et que leurs en-têtes sont corrects  
 b) Redémarrer automatiquement l'ordinateur sans autre vérification  
 c) Supprimer l'application Dash et la réinstaller complètement  
 d) Ignorer l'erreur et continuer l'analyse
2. Quelle est une cause fréquente d'échec lors de l'importation de données dans une application Dash développée en Python ?  
 a) Absence des bibliothèques Python requises (ex : pandas, dash)  
 b) Trop de colonnes dans le fichier  
 c) Le poids du fichier est trop faible  
 d) L'ordinateur n'a pas assez de RAM au démarrage
3. En cas de problème d'importation, quelle commande peut-on utiliser pour s'assurer que toutes les dépendances nécessaires sont bien installées ?  
 a) pip install dash dash-bootstrap-components pandas plotly reportlab  
 b) python app.py  
 c) pip uninstall dash  
 d) git clone
4. Lors de l'import de plusieurs bases, que peut-on vérifier si la fusion ne fonctionne pas correctement ?

- a) Que la colonne commune (par exemple « Département ») existe dans toutes les bases et que le format des données est homogène
  - b) Que chaque base a le même nombre de colonnes
  - c) Que les fichiers portent exactement le même nom
  - d) Que les colonnes des bases sont triées alphabétiquement
5. Que permet de faire le message de confirmation affiché lors du chargement des bases dans l'application ?
- a) Valider la réussite du chargement ou signaler un problème à corriger
  - b) Envoyer un rapport aux développeurs automatiquement
  - c) Fermer l'application Dash
  - d) Modifier automatiquement les données non conformes

## .1 Schéma fonctionnel de l'application



## .2 Base de données fusionnée — Export PDF

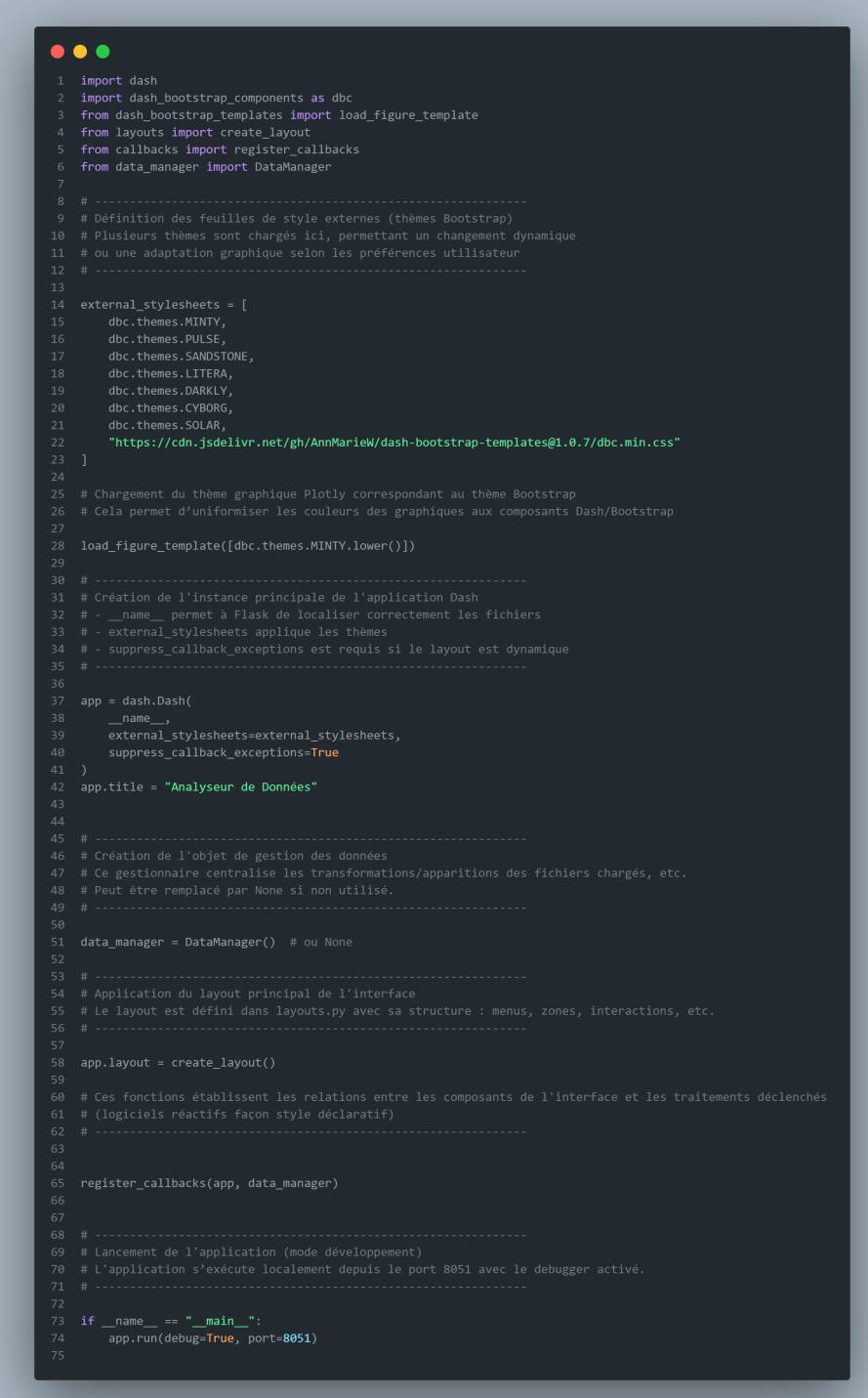
Base de Données

Département	Ciné_100k_hab	Biblio_100k_hab	Spectacles_100k_hab	Etablissements_cult_100k_hab	Musées_100k_hab	Salaire_moyen_ne_t_an	Participation_ur_baine	Dépenses_culturelles_parc_hab	Région	Population	Salaire_moyen(euros_net_mois)	Taux_chômage(po urcentages)
Ain	3.2	36.9	0.8	0.9	2.6	27420	67.0	19	Auvergne-Rhône-Alpes	659180	2285.0	6
Aisne	2.7	24.5	1.8	1.1	3.7	25236	53.2	18	Hauts-de-France	546527	2103.0	11,7
Allier	2.3	70.3	2.6	0.9	3.2	24552	58.3	23	Auvergne-Rhône-Alpes	347035	2046.0	8,9
Alpes-Maritimes	26.7	86.7	4.2	7.7	22.6	28980	95.9	19	Provence-Alpes-Côte d'Azur	168381	2415.0	8,7
Alpes-de-Haute-Provence	1.3	8.6	0.2	0.1	0.9	25428	61.9	34	Provence-Alpes-Côte d'Azur	1097496	2119.0	9,8
Ardennes	1.5	30.8	1.2	0.300000000004	3.0	25140	57.3	14	Grand Est	334688	2095.0	9,9
Ardèche	7.1	82.5	1.4	1.4	3.2	25524	63.3	18	Auvergne-Rhône-Alpes	280032	2234.96969697	9
Ariège	7.0	52.8	1.9	0.600000000001	3.8	24348	55.5	57	Occitanie	157210	2029.0	10,6
Aube	1.3	48.2	0.9	1.6	4.4	25068	61.3	24	Grand Est	317118	2089.0	10,7
Aude	3.2	67.3	1.1	0.5	3.2	23892	61.6	22	Occitanie	379094	1991.0	11,6
Aveyron	5.5	70.8	1.700000000002	0.700000000001	4.1	24252	47.7	21	Occitanie	289488	2021.0	6,3
Bas-Rhin	2.0	19.4	2.0	0.600000000001	2.5	27936	80.0	12	Grand Est	1141511	2328.0	7,2
Bouches-du-Rhône	2.9	6.9	1.6	0.9	2.2	29364	99.3	11	Provence-Alpes-Côte d'Azur	2048504	2447.0	10,1
Calvados	4.4	19.2	1.700000000002	1.8	3.8	25932	68.9	9	Normandie	708344	2161.0	7,7
Cantal	5.3	97.9	2.0	1.3	4.7	23028	35.3	6	Auvergne-Rhône-Alpes	150185	1919.0	4,9
Charente	3.9	22.4	1.4	1.4	1.9	25620	46.7	11	Nouvelle-Aquitaine	361539	2135.0	8,1
Charente-Maritime	4.4	35.2	1.5	0.9	3.2	25236	60.9	17	Nouvelle-Aquitaine	659968	2103.0	8,4

FIGURE 1 – Extrait du fichier PDF exporté contenant la base fusionnée

## .3 Aperçu du code source des modules

### .3.1 Fichier app.py



```
1 import dash
2 import dash_bootstrap_components as dbc
3 from dash_bootstrap_templates import load_figure_template
4 from layouts import create_layout
5 from callbacks import register_callbacks
6 from data_manager import DataManager
7
8 # -----
9 # Définition des feuilles de style externes (thèmes Bootstrap)
10 # Plusieurs thèmes sont chargés ici, permettant un changement dynamique
11 # ou une adaptation graphique selon les préférences utilisateur
12 # -----
13
14 external_stylesheets = [
15     dbc.themes.MINTY,
16     dbc.themes.PULSE,
17     dbc.themes.SANDSTONE,
18     dbc.themes.LITERA,
19     dbc.themes.DARKLY,
20     dbc.themes.CYBORG,
21     dbc.themes.SOLAR,
22     "https://cdn.jsdelivr.net/gh/AnnMarieW/dash-bootstrap-templates@1.0.7dbc.min.css"
23 ]
24
25 # Chargement du thème graphique Plotly correspondant au thème Bootstrap
26 # Cela permet d'uniformiser les couleurs des graphiques aux composants Dash/Bootstrap
27
28 load_figure_template([dbc.themes.MINTY.lower()])
29
30 # -----
31 # Création de l'instance principale de l'application Dash
32 # - __name__ permet à Flask de localiser correctement les fichiers
33 # - external_stylesheets applique les thèmes
34 # - suppress_callback_exceptions est requis si le layout est dynamique
35 #
36
37 app = dash.Dash(
38     __name__,
39     external_stylesheets=external_stylesheets,
40     suppress_callback_exceptions=True
41 )
42 app.title = "AnalysEUR de Données"
43
44
45 # -
46 # Création de l'objet de gestion des données
47 # Ce gestionnaire centralise les transformations/apparitions des fichiers chargés, etc.
48 # Peut être remplacé par None si non utilisé.
49 #
50
51 data_manager = DataManager() # ou None
52
53 # -
54 # Application du layout principal de l'interface
55 # Le layout est défini dans layouts.py avec sa structure : menus, zones, interactions, etc.
56 #
57
58 app.layout = create_layout()
59
60 # Ces fonctions établissent les relations entre les composants de l'interface et les traitements déclenchés
61 # (logiciels réactifs façon style déclaratif)
62 #
63
64
65 register_callbacks(app, data_manager)
66
67
68 #
69 # Lancement de l'application (mode développement)
70 # L'application s'exécute localement depuis le port 8051 avec le debugger activé.
71 #
72
73 if __name__ == "__main__":
74     app.run(debug=True, port=8051)
```

FIGURE 2 – Structure du fichier app.py : initialisation de l’application Dash

### 3.2 Fichier data\_manager.py

```

1 import pandas as pd
2 import numpy as np
3
4 class DataManager:
5     """
6         Classe utilitaire pour la gestion centralisée des données dans l'application.
7         Elle permet l'import, le nettoyage, le filtrage, le tri, l'analyse statistique,
8         ainsi que la gestion et l'export des données tabulaires en mémoire.
9     """
10
11     def __init__(self):
12         """
13             Initialise le gestionnaire de données :
14             - data : jeu de données importé (DataFrame)
15             - filtered_data : sous-ensemble filtré (DataFrame)
16             - filters : liste d'informations sur les filtres appliqués
17             - current_stats : dictionnaire des statistiques calculées
18         """
19         self.data = None
20         self.filtered_data = None
21         self.filters = []
22         self.current_stats = {}
23
24     def harmonize_na(self, df):
25         """
26             Standardise les valeurs manquantes dans tout le DataFrame :
27             Remplace "", "NA", "N/A", "NAN" (et équivalents) par pd.NA pour un traitement homogène.
28             @param df: DataFrame à harmoniser
29             @return: DataFrame harmonisé
30         """
31         # Application d'une règle sur chaque cellule pour uniformiser les NA
32         return df.applymap(lambda x: pd.NA if pd.isna(x) or str(x).strip().upper() in ["", "NA", "N/A", "NAN"] else x)
33
34     def load_data(self, df: pd.DataFrame):
35         """
36             Charge un nouveau DataFrame, harmonise les NA, réinitialise les filtres et statistiques.
37             @param df: DataFrame initial à charger
38         """
39         df = self.harmonize_na(df)
40         self.data = df.copy()
41         self.filtered_data = None
42         self.filters = []
43         self.current_stats = {}
44
45     def apply_filter(self, column, condition, value, logic_op='and'):
46         """
47             Applique un filtre sur les données selon une colonne, une condition et une valeur,
48             et combine éventuellement avec les filtres existants via 'and' ou 'or'.
49             @param column: colonne à filtrer
50             @param condition: opérateur ('>', '<', 'contains', etc.)
51             @param value: valeur de comparaison
52             @param logic_op: opérateur logique ('and', 'or')
53             @return: DataFrame filtré
54         """
55         if self.data is None:
56             return pd.DataFrame()
57         filter_info = {'column': column, 'condition': condition, 'value': value, 'logic_op': logic_op}
58         self.filters.append(filter_info)
59         return self.apply_all_filters()
60
61     def apply_all_filters(self):
62         """
63             Applique l'ensemble des filtres stockés sur le DataFrame d'origine.
64             @return: DataFrame filtré
65         """
66         if self.data is None or not self.filters:
67             self.filtered_data = None
68             return self.data
69         df = self.data
70         mask = pd.Series([True] * len(df))
71         # Application séquentielle de chaque filtre (ET/OU)
72         for f in self.filters:
73             col, cond, val, logic = f['column'], f['condition'], f['value'], f['logic_op']
74             if cond == 'contains':
75                 new_mask = df[col].astype(str).str.contains(str(val), na=False)
76             elif cond == '>':
77                 new_mask = df[col] > float(val)
78             elif cond == '<':
79                 new_mask = df[col] < float(val)
80             elif cond == '>=':
81                 new_mask = df[col] >= float(val)
82             elif cond == '<=':
83                 new_mask = df[col] <= float(val)
84             elif cond == '==':
85                 new_mask = df[col] == val
86             elif cond == '!=':
87                 new_mask = df[col] != val
88             else:
89                 new_mask = pd.Series([True] * len(df))
90             mask = mask & new_mask if logic == 'and' else mask | new_mask
91         # Mise à jour du sous-ensemble filtré
92         self.filtered_data = df[mask]
93         return self.filtered_data
94

```

FIGURE 3 – Partie 1 : gestion des données dans data\_manager.py



```

1 def reset_filters(self):
2     """
3     Supprime tous les filtres actifs et les calculs statistiques associés.
4     """
5     self.filtered_data = None
6     self.filters = []
7     self.current_stats = {}
8
9 def sort_data(self, column, ascending=True):
10    """
11    Trie le DataFrame (filtré ou original) par une colonne en ordre croissant/décroissant.
12    @param column: colonne cible pour le tri
13    @param ascending: booléen (True pour croissant)
14    @return: DataFrame trié (ou vide si colonne absente)
15    """
16    df = self.filtered_data if self.filtered_data is not None else self.data
17    if df is None or column not in df.columns:
18        return pd.DataFrame()
19    sorted_df = df.sort_values(by=column, ascending=ascending)
20    if self.filtered_data is not None:
21        self.filtered_data = sorted_df
22    return sorted_df
23
24 def calculate_statistics(self, column):
25    """
26    Calcule les statistiques descriptives principales pour une colonne numérique :
27    count, moyenne, médiane, std, min, max, somme, 1er/3e quartile, nombre de NA.
28    @param column: nom de la colonne analysée
29    @return: dict de statistiques ou erreur si non numérique
30    """
31    df = self.filtered_data if self.filtered_data is not None else self.data
32    if df is None or column not in df.columns:
33        return {}
34    col_data = df[column].dropna()
35    if not pd.api.types.is_numeric_dtype(col_data):
36        return {'error': 'Colonne non numérique'}
37    stats = {
38        'count': int(len(col_data)),
39        'mean': float(col_data.mean()),
40        'median': float(col_data.median()),
41        'std': float(col_data.std()),
42        'min': float(col_data.min()),
43        'max': float(col_data.max()),
44        'sum': float(col_data.sum()),
45        'q25': float(col_data.quantile(0.25)),
46        'q75': float(col_data.quantile(0.75)),
47        'missing': int(df[column].isnull().sum())
48    }
49    self.current_stats[column] = stats
50    return stats
51
52 def drop_na_rows(self):
53    """
54    Supprime toutes les lignes contenant au moins une valeur manquante (NA).
55    @return: DataFrame nettoyé
56    """
57    df = self.fi

```

FIGURE 4 – Partie 2 : traitement des données dans `data_manager.py`

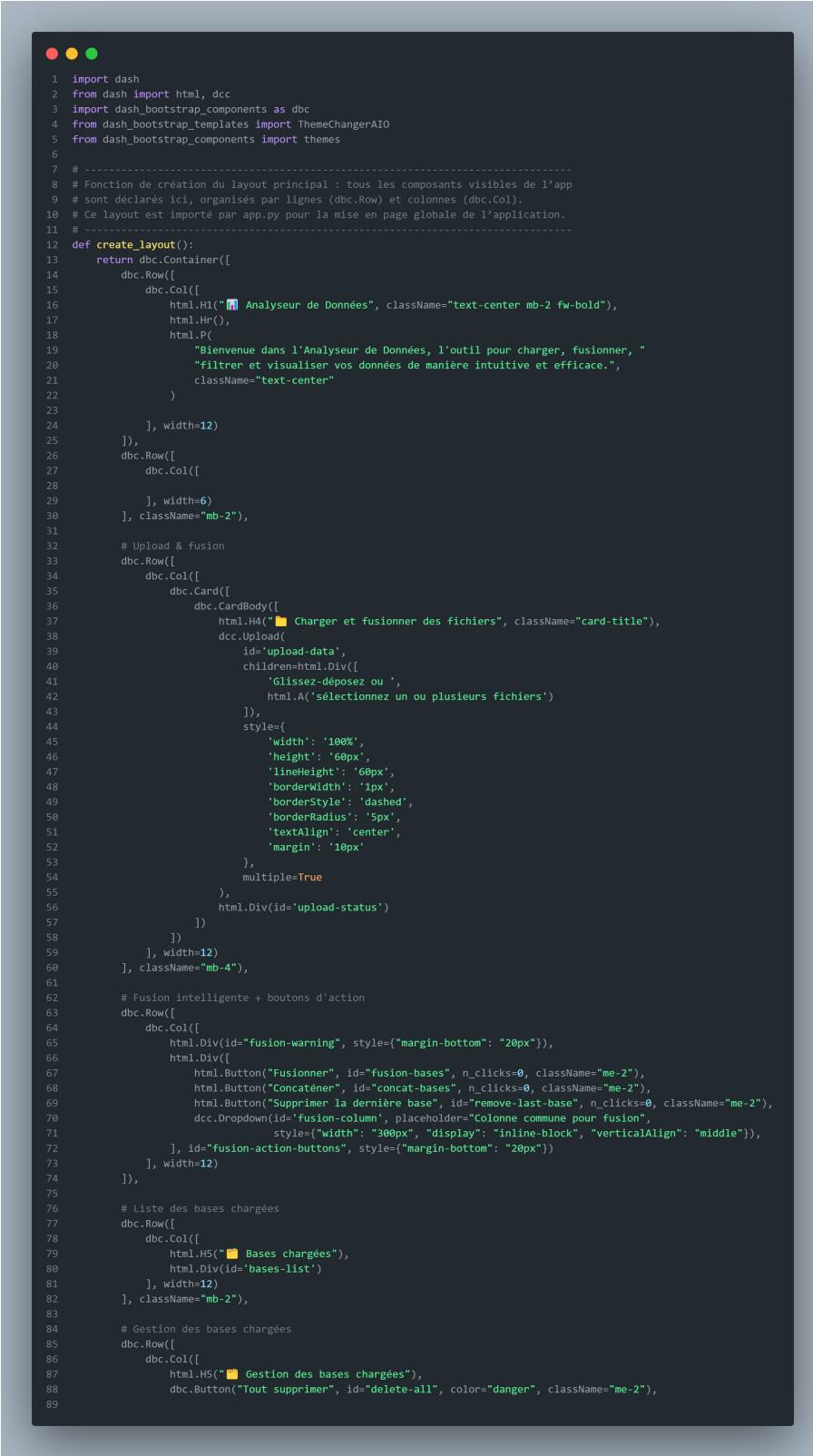
```

1  def drop_na_rows(self):
2      """
3          Supprime toutes les lignes contenant au moins une valeur manquante (NA).
4          @return: DataFrame nettoyé
5          """
6      df = self.filtered_data if self.filtered_data is not None else self.data
7      if df is not None:
8          df = df.dropna()
9          df = self.harmonize_na(df)
10         self.filtered_data = df
11     return df
12
13 def drop_na_cols(self):
14     """
15         Supprime toutes les colonnes contenant au moins une valeur manquante (NA).
16         @return: DataFrame nettoyé
17         """
18     df = self.filtered_data if self.filtered_data is not None else self.data
19     if df is not None:
20         df = df.dropna(axis=1)
21         df = self.harmonize_na(df)
22         self.filtered_data = df
23     return df
24
25 def fill_na_mean(self):
26     """
27         Remplace les NA des colonnes numériques par leur moyenne calculée.
28         @return: DataFrame mis à jour
29         """
30     df = self.filtered_data if self.filtered_data is not None else self.data
31     if df is not None:
32         df = df.apply(lambda col: col.fillna(col.mean()) if pd.api.types.is_numeric_dtype(col) else col)
33         df = self.harmonize_na(df)
34         self.filtered_data = df
35     return df
36
37 def fill_na_median(self):
38     """
39         Remplace les NA des colonnes numériques par leur médiane calculée.
40         @return: DataFrame mis à jour
41         """
42     df = self.filtered_data if self.filtered_data is not None else self.data
43     if df is not None:
44         df = df.apply(lambda col: col.fillna(col.median()) if pd.api.types.is_numeric_dtype(col) else col)
45         df = self.harmonize_na(df)
46         self.filtered_data = df
47     return df
48
49 def fill_na_value(self, value):
50     """
51         Remplace tous les NA du DataFrame par une valeur utilisateur.
52         @param value: valeur de remplacement
53         @return: DataFrame mis à jour
54         """
55     df = self.filtered_data if self.filtered_data is not None else self.data
56     if df is not None:
57         df = df.fillna(value)
58         df = self.harmonize_na(df)
59         self.filtered_data = df
60     return df
61
62 def get_current_data(self):
63     """
64         Retourne toujours les données filtrées si elles existent, sinon le jeu de données complet.
65         @return: DataFrame courant
66         """
67     return self.filtered_data if self.filtered_data is not None else self.data
68
69 def get_column_info(self):
70     """
71         Fournit un résumé des colonnes (type, nombre de valeurs null, nombre de valeurs uniques, numérique ou non)
72         @return: dict d'informations sur chaque colonne
73         """
74     if self.data is None:
75         return {}
76     info = {}
77     for col in self.data.columns:
78         info[col] = {
79             'dtype': str(self.data[col].dtype),
80             'null_count': int(self.data[col].isnull().sum()),
81             'unique_count': int(self.data[col].nunique()),
82             'is_numeric': pd.api.types.is_numeric_dtype(self.data[col])
83         }
84     return info
85
86 def export_to_csv_content(self):
87     """
88         Exporte le DataFrame courant (filtré ou non) en contenu CSV (string, séparé par des virgules).
89         @return: représentation CSV des données courantes
90         """
91     df = self.get_current_data()
92     if df is None:
93         return ""
94     return df.to_csv(index=False)
95

```

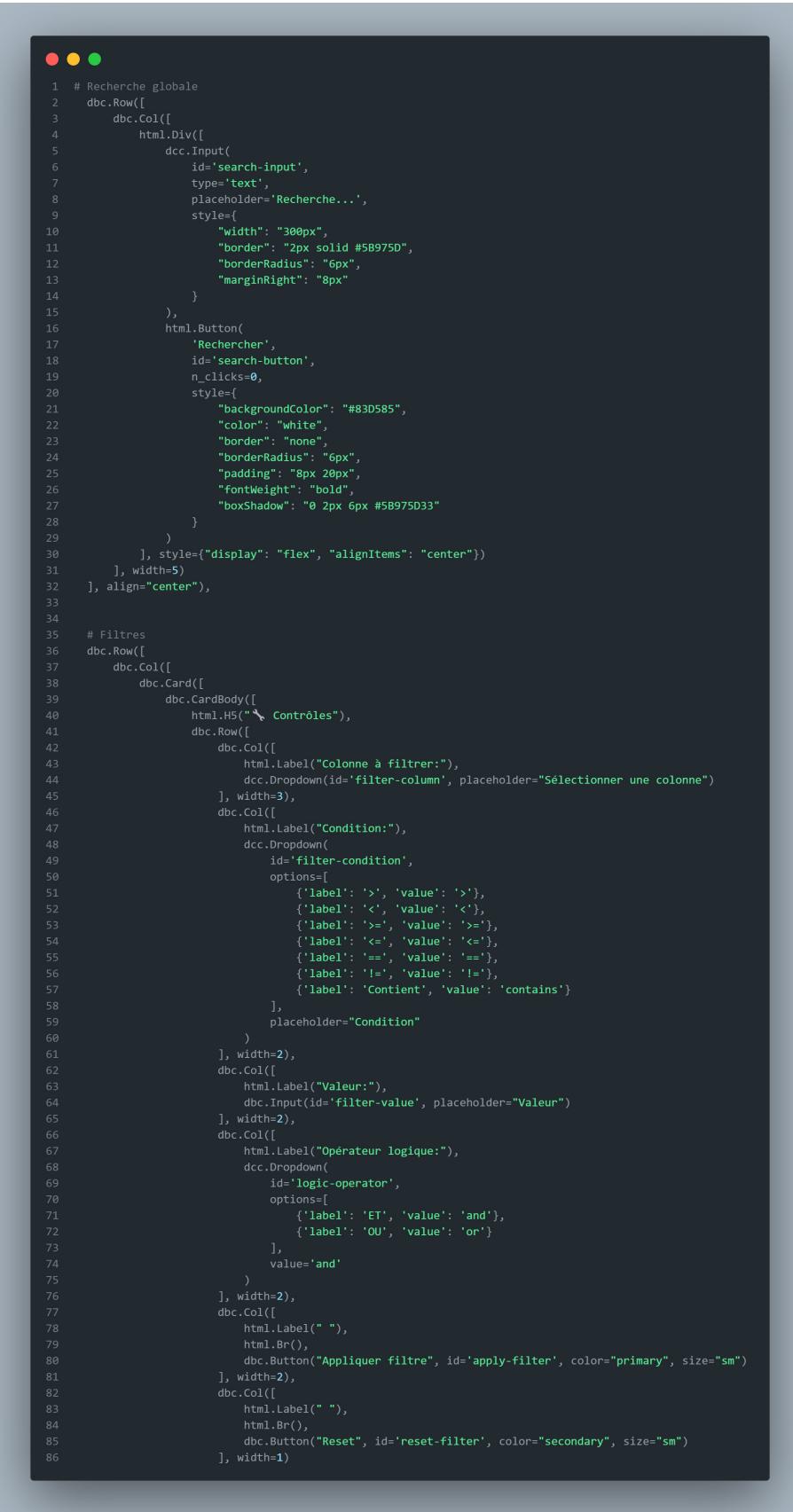
FIGURE 5 – Partie 3 : traitement avancé dans `data_manager.py`

### 3.3 Fichier layouts.py



```
1 import dash
2 from dash import html, dcc
3 import dash_bootstrap_components as dbc
4 from dash_bootstrap_templates import ThemeChangerAIO
5 from dash_bootstrap_components import themes
6
7 # ...
8 # Fonction de création du layout principal : tous les composants visibles de l'app
9 # sont déclarés ici, organisés par lignes (dbc.Row) et colonnes (dbc.Col).
10 # Ce layout est importé par app.py pour la mise en page globale de l'application.
11 # ...
12 def create_layout():
13     return dbc.Container([
14         dbc.Row([
15             dbc.Col([
16                 html.H1("Analyseur de Données", className="text-center mb-2 fw-bold"),
17                 html.Hr(),
18                 html.P(
19                     "Bienvenue dans l'Analyseur de Données, l'outil pour charger, fusionner, "
20                     "filtrer et visualiser vos données de manière intuitive et efficace.",
21                     className="text-center"
22                 )
23             ],
24             width=12
25         ),
26         dbc.Row([
27             dbc.Col([
28                 ], width=6)
29         ],
30         className="mb-2"),
31
32     # Upload & fusion
33     dbc.Row([
34         dbc.Col([
35             dbc.Card([
36                 dbc.CardBody([
37                     html.H4("Charger et fusionner des fichiers", className="card-title"),
38                     dcc.Upload(
39                         id='upload-data',
40                         children=html.Div([
41                             'Glissez-déposez ou ',
42                             html.A('sélectionnez un ou plusieurs fichiers')
43                         ]),
44                         style={
45                             'width': '100%',
46                             'height': '60px',
47                             'lineHeight': '60px',
48                             'borderWidth': '1px',
49                             'borderStyle': 'dashed',
50                             'borderRadius': '5px',
51                             'textAlign': 'center',
52                             'margin': '10px'
53                         },
54                         multiple=True
55                     ),
56                     html.Div(id='upload-status')
57                 ])
58             ],
59             width=12
60         ],
61         className="mb-4"),
62
63     # Fusion intelligente + boutons d'action
64     dbc.Row([
65         dbc.Col([
66             html.Div(id="fusion-warning", style={"margin-bottom": "20px"}),
67             html.Div([
68                 html.Button("Fusionner", id="fusion-bases", n_clicks=0, className="me-2"),
69                 html.Button("Concaténer", id="concat-bases", n_clicks=0, className="me-2"),
70                 html.Button("Supprimer la dernière base", id="remove-last-base", n_clicks=0, className="me-2"),
71                 dcc.Dropdown(id="fusion-column", placeholder="Colonne commune pour fusion",
72                             style={"width": "300px", "display": "inline-block", "verticalAlign": "middle"}),
73             ],
74             id="fusion-action-buttons",
75             style={"margin-bottom": "20px"}
76         ],
77         width=12
78     ],
79     ],
80     className="mb-2"),
81
82     # Liste des bases chargées
83     dbc.Row([
84         dbc.Col([
85             html.H5("Bases chargées"),
86             html.Div(id="bases-list")
87         ],
88         width=12
89     ],
90     className="mb-2"),
91
92     # Gestion des bases chargées
93     dbc.Row([
94         dbc.Col([
95             html.H5("Gestion des bases chargées"),
96             dbc.Button("Tout supprimer", id="delete-all", color="danger", className="me-2"),
97         ],
98     ],
99     className="mb-2"),
100 ])
```

FIGURE 6 – Partie 1 : structure de l'interface dans layouts.py



```

1 # Recherche globale
2     dbc.Row([
3         dbc.Col([
4             html.Div([
5                 dcc.Input(
6                     id='search-input',
7                     type='text',
8                     placeholder='Recherche...',
9                     style={
10                        "width": "300px",
11                        "border": "2px solid #5B975D",
12                        "borderRadius": "6px",
13                        "marginRight": "8px"
14                    }
15                ),
16                html.Button(
17                    'Rechercher',
18                    id='search-button',
19                    n_clicks=0,
20                    style={
21                        "backgroundColor": "#83D585",
22                        "color": "white",
23                        "border": "none",
24                        "borderRadius": "6px",
25                        "padding": "8px 20px",
26                        "fontWeight": "bold",
27                        "boxShadow": "0 2px 6px #5B975D33"
28                    }
29                )
30            ], style={"display": "flex", "alignItems": "center"})
31        ], width=5)
32    ], align="center"),
33
34
35 # Filtres
36     dbc.Row([
37         dbc.Col([
38             dbc.Card([
39                 dbc.CardBody([
40                     html.H5("Contrôles"),
41                     dbc.Row([
42                         dbc.Col([
43                             html.Label("Colonne à filtrer:"),  

44                             dcc.Dropdown(id='filter-column', placeholder="Sélectionner une colonne")
45                         ], width=3),
46                         dbc.Col([
47                             html.Label("Condition:"),  

48                             dcc.Dropdown(
49                                 id='filter-condition',
50                                 options=[
51                                     {'label': '>', 'value': '>'},
52                                     {'label': '<', 'value': '<'},
53                                     {'label': '>=', 'value': '>='},
54                                     {'label': '<=', 'value': '<='},
55                                     {'label': '==', 'value': '=='},
56                                     {'label': '!=', 'value': '!='},
57                                     {'label': 'Contient', 'value': 'contains'}
58                                 ],
59                                 placeholder="Condition"
60                             )
61                         ], width=2),
62                         dbc.Col([
63                             html.Label("Valeur:"),  

64                             dbc.Input(id='filter-value', placeholder="Valeur")
65                         ], width=2),
66                         dbc.Col([
67                             html.Label("Opérateur logique:"),  

68                             dcc.Dropdown(
69                                 id='logic-operator',
70                                 options=[
71                                     {'label': 'ET', 'value': 'and'},
72                                     {'label': 'OU', 'value': 'or'}
73                                 ],
74                                 value='and'
75                             )
76                         ], width=2),
77                         dbc.Col([
78                             html.Label(" "),  

79                             html.Br(),
80                             dbc.Button("Appliquer filtre", id='apply-filter', color="primary", size="sm")
81                         ], width=2),
82                         dbc.Col([
83                             html.Label(" "),  

84                             html.Br(),
85                             dbc.Button("Reset", id='reset-filter', color="secondary", size="sm")
86                         ], width=1)

```

FIGURE 7 – Partie 2 : composants visuels dans `layouts.py`

```

1         dbc.Button("Reset", id='reset-filter', color="secondary", size="sm")
2     ], width=1)
3   ],
4   html.Small("Vous pouvez appliquer plusieurs filtres successivement.", className="text-muted")
5   ])
6   ],
7   ], width=12)
8 ], className="mb-4"),
9
10 # Diagnostic NA dynamique
11 dbc.Row([
12   dbc.Col([
13     html.Div(id='na-diagnostic-container')
14   ], width=12)
15 ], className="mb-2"),
16
17
18 # Gestion NA
19 dbc.Row([
20   dbc.Col([
21     dbc.Card([
22       dbc.CardBody([
23         html.H5("✓ Gestion des données manquantes"),
24         dbc.Button("Supprimer lignes NA", id="drop-na-rows", color="danger", className="me-2"),
25         dbc.Button("Supprimer colonnes NA", id="drop-na-cols", color="danger", className="me-2"),
26         dbc.Button("Supprimer les doublons", id="drop-duplicates", color="dark", className="me-2"),
27         dbc.Button("Remplir NA par la moyenne", id="fill-na-mean", color="warning", className="me-2"),
28         dbc.Button("Remplir NA par la médiane", id="fill-na-median", color="info", className="me-2"),
29         dbc.Button("Remplir NA par une valeur", id="fill-na-value", color="secondary", className="me-2"),
30         dbc.Input(id="na-fill-value", type="text", placeholder="Valeur de remplacement",
31                   style={'width': '200px', 'display': 'inline-block'}),
32         dbc.Button("Réinitialiser", id="reset-na", color="secondary", className="me-2"),
33         html.Div(id="na-status", className="mt-2")
34       ])
35     ],
36   ],
37   ], width=12)
38 ], className="mb-2"),
39
40 # Tri & statistiques
41 dbc.Row([
42   dbc.Col([
43     dbc.Card([
44       dbc.CardBody([
45         dbc.Row([
46           dbc.Col([
47             html.Label("Trier par:"),  

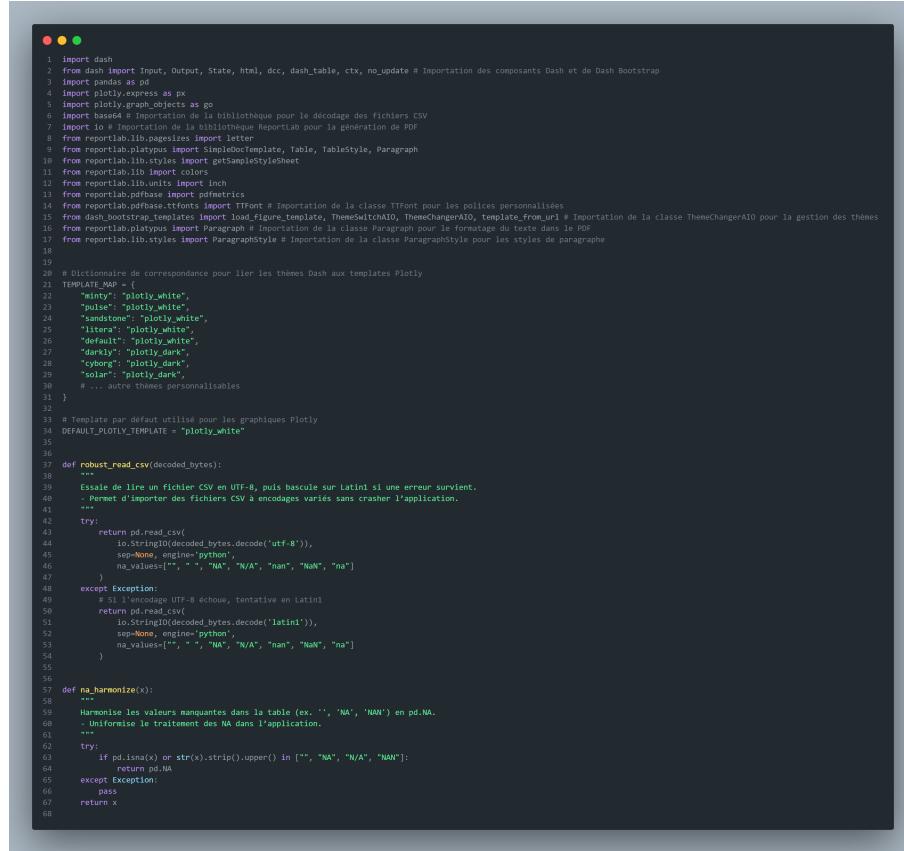
48             dcc.Dropdown(id='sort-column', placeholder="Colonne")
49           ], width=3),
50           dbc.Col([
51             html.Label("Ordre:"),  

52             dcc.Dropdown(
53               id='sort-order',
54               options=[
55                 {'label': 'Croissant', 'value': True},
56                 {'label': 'Décroissant', 'value': False}
57               ],
58               value=True
59             )
60           ],
61         ])
62       ])
63     ],
64   ],
65   ], width=12)
66 ], className="mb-2")

```

FIGURE 8 – Partie 3 : finalisation de l’interface dans `layouts.py`

### .3.4 Fichier callbacks.py



```
● ● ●
1 import dash
2 from dash import Input, Output, State, html, dcc, dash_table, ctx, no_update # Importation des composants Dash et de Dash Bootstrap
3 import pandas as pd
4 import plotly.express as px
5 import plotly.graph_objects as go
6 import base64 # Importation de la bibliothèque pour le décodage des fichiers CSV
7 import io # Importation de la bibliothèque ReportLab pour la génération de PDF
8 from reportlab.lib.pagesizes import letter
9 from reportlab.lib.platypus import SimpleTableTemplate, Table, TableStyle, Paragraph
10 from reportlab.lib.styles import getSampleStyleSheet
11 from reportlab.lib.colors import black
12 from reportlab.lib.units import inch
13 from reportlab.lib.libmetrics import Metrics
14 from reportlab.pdfbase.ttfonts import TTFont # Importation de la classe TTFont pour les polices personnalisées
15 from dash_bootstrap_templates import load_figure_template, ThemeSwitchAIO, template_from_url # Importation de la classe ThemeChangerAIO pour la gestion des thèmes
16 from reportlab.platypus import Paragraph # Importation de la classe Paragraph pour le formatage du texte dans le PDF
17 from reportlab.lib.styles import ParagraphStyle # Importation de la classe ParagraphStyle pour les styles de paragraphe
18
19
20 # Dictionnaire de correspondance pour lier les thèmes Dash aux templates Plotly
21 TEMPLATE_MAP = {
22     "mány": "plotly_white",
23     "dark": "plotly_white",
24     "sandstone": "plotly_white",
25     "literal": "plotly_white",
26     "default": "plotly_white",
27     "darkly": "plotly_dark",
28     "superhero": "plotly_dark",
29     "solar": "plotly_dark",
30     # ... autres thèmes personnalisables
31 }
32
33 # Template par défaut utilisé pour les graphiques Plotly
34 DEFAULT_PLOTLY_TEMPLATE = "plotly_white"
35
36
37 def robust_read_csv(decoded_bytes):
38     """
39         Essaie de lire un fichier CSV en UTF-8, puis bascule sur Latin1 si une erreur survient.
40         - Permet d'importer des fichiers CSV à encodages variés sans crasher l'application.
41     """
42     try:
43         return pd.read_csv(
44             io.StringIO(decoded_bytes.decode('utf-8')),
45             sep=None, engine='python',
46             na_values=['', '', "NA", "N/A", "nan", "NaN", "na"]
47         )
48     except Exception:
49         # Si l'encodage UTF-8 échoue, tentative en Latin1
50         return pd.read_csv(
51             io.StringIO(decoded_bytes.decode('latin1')),
52             sep=None, engine='python',
53             na_values=['', '', "NA", "N/A", "nan", "NaN", "na"]
54         )
55
56
57 def na_harmonize(x):
58     """
59         Harmonise les valeurs manquantes dans la table (ex. '', 'NA', 'NaN') en pd.NA.
60         - Uniformise le traitement des NA dans l'application.
61     """
62     try:
63         if pd.isna(x) or str(x).strip().upper() in ["", "NA", "N/A", "NaN"]:
64             return pd.NA
65     except Exception:
66         pass
67     return x
68
```

FIGURE 9 – Callback 1 dans callbacks.py

```

1 def register_callbacks(app, data_manager):
2     """
3     Enregistre tous les callbacks nécessaires au fonctionnement dynamique de l'interface Dash.
4     """
5
6     # --- Affichage des bases et options de fusion ---
7     @app.callback(
8         Output('bases-list', 'children'),
9         Output('fusion-column', 'options'),
10        Output('fusion-action-buttons', 'style'),
11        Input('uploaded-bases', 'data'),
12    )
13    def update_bases_list(uploaded_bases):
14        # Aucun fichier chargé
15        if not uploaded_bases:
16            return "Aucune base chargée", [], {"display": "none"}
17
18        # Afficher pour chaque base importée son nom, nombre de lignes et de colonnes
19        bases_infos = [
20            html.Li(
21                f"{{b['name']}} : "
22                f"{{pd.read_json(io.StringIO(b['data']), orient='split').shape[0]}} lignes, "
23                f"{{pd.read_json(io.StringIO(b['data']), orient='split').shape[1]}} colonnes"
24            )
25            for b in uploaded_bases
26        ]
27
28        # Si plus d'une base : proposer la fusion sur colonnes communes
29        if len(uploaded_bases) > 1:
30            dfs = [pd.read_json(io.StringIO(b['data']), orient='split') for b in uploaded_bases]
31            common_cols = list(set(dfs[0].columns) & set(dfs[1].columns))
32            return html.Ul(bases_infos), [{"label": c, "value": c} for c in common_cols], {"display": "block"}
33
34        # Sinon, cacher les options de fusion
35        return html.Ul(bases_infos), [], {"display": "none"}
36
37
38    # --- Affichage des bases à supprimer dans la dropdown ---
39    @app.callback(
40        Output('base-to-delete', 'options'),
41        Input('uploaded-bases', 'data'),
42    )
43    def update_base_delete_options(uploaded_bases):
44        if not uploaded_bases:
45            return []
46        return [{"label": b['name'], 'value': b['name']} for b in uploaded_bases]
47
48
49
50    # --- Callback central pour la gestion des import/export, NA, filtres, fusion, etc.
51    @app.callback(
52        [
53            Output('stored-data', 'data'),
54            Output('uploaded-bases', 'data'),
55            Output('upload-status', 'children'),
56            Output('na-status', 'children'),
57            Output('na-diagnostic-container', 'children'),
58            Output('data-table-container', 'children'),
59            Output('stats-column', 'options'),
60            Output('x-axis', 'options'),
61            Output('y-axis', 'options'),
62            Output('color-column', 'options'),
63            Output('filter-column', 'options')
64        ],
65        [
66            Input('upload-data', 'contents'),
67            Input('drop-na-rows', 'n_clicks'),
68            Input('drop-na-cols', 'n_clicks'),
69            Input('drop-duplicates', 'n_clicks'),
70            Input('fill-na-mean', 'n_clicks'),
71            Input('fill-na-median', 'n_clicks'),
72            Input('fill-na-value', 'n_clicks'),
73            Input('reset-na', 'n_clicks'),
74            Input('fusion-bases', 'n_clicks'),
75            Input('concat-bases', 'n_clicks'),
76            Input('remove-last-base', 'n_clicks'),
77            Input('search-input', 'value'),
78            Input('apply-filter', 'n_clicks'),
79            Input('reset-filter', 'n_clicks'),
80            Input('delete-all', 'n_clicks'),
81            Input('delete-selected', 'n_clicks'),
82        ],
83        [
84            State('upload-data', 'filename'),
85            State('uploaded-bases', 'data'),
86            State('stored-data', 'data'),
87            State('na-fill-value', 'value'),
88            State('fusion-column', 'value'),
89            State('filter-column', 'value'),
90            State('filter-condition', 'value'),
91            State('filter-value', 'value'),
92            State('logic-operator', 'value'),
93            State('base-to-delete', 'value'),
94        ]
95    )
96    def manage_data(
97        contents, drop_rows, drop_cols, drop_duplicates, fill_mean, fill_median,
98        fill_value, reset_na, fusion_bases, concat_bases,
99        remove_last_base,
100    )

```

FIGURE 10 – Callback 2 dans callbacks.py

```

1 def manage_data(
2     contents, drop_rows, drop_cols, drop_duplicates, fill_mean, fill_median,
3     fill_value, reset_na, fusion_bases, concat_bases,
4     remove_last_base, search_value, apply_filter_click,
5     reset_filter_click, delete_all_click, delete_selected_click,
6     filenames, uploaded_bases, stored_data, na_value,
7     fusion_col, filter_col, filter_cond, filter_val,
8     logic_operator, base_to_delete
9 ):
10    # Constantes pour les retours vides
11    NB_OUTPUTS = 11
12    EMPTY_RETURN = (
13        no_update, [], "", "", "",
14        html.Div("Aucune donnée chargée"),
15        [], [], [], [], []
16    )
17
18    triggered = ctx.triggered_id
19    uploaded_bases = uploaded_bases or []
20    msg_upload = ""
21    msg_na = ""
22
23    # Fonctions utilitaires pour alimenter les dropdowns
24    def cols_options(df):
25        return {'label': c, 'value': c} for c in df.columns
26
27    # Gestion suppression de toutes les bases
28    if triggered == 'delete-all':
29        return (
30            None, [], "Toutes les bases supprimées.", "", "", "",
31            html.Div("Aucune donnée chargée"),
32            [], [], [], [], []
33        )
34
35    # Suppression d'une base spécifique
36    elif triggered == 'delete-selected' and base_to_delete is not None:
37        uploaded_bases = [b for b in uploaded_bases if b['name'] != base_to_delete]
38        if uploaded_bases:
39            df = pd.read_json(io.StringIO(uploaded_bases[-1]['data']), orient='split')
40            return (
41                df.to_json(date_format='iso', orient='split'),
42                uploaded_bases,
43                f"Base '{base_to_delete}' supprimée.",
44                "", "", no_update,
45                [], [], [], [], []
46            )
47        else:
48            return (
49                None, [], f"Base '{base_to_delete}' supprimée.",
50                "", "", html.Div("Aucune donnée chargée"),
51                [], [], [], [], []
52            )
53
54
55    # Import de fichiers : gestion du décodage et harmonisation NA
56    if triggered == 'upload-data' and contents:
57        new_bases = []
58        if not isinstance(contents, list):
59            contents = [contents]
60            filenames = [filenames]
61        for content, filename in zip(contents, filenames):
62            content_type, content_string = content.split(',')
63            decoded = base64.b64decode(content_string)
64            try:
65                if filename.endswith('.csv'):
66                    df = robust_read_csv(decoded)
67                elif filename.endswith('.xls', '.xlsx'):
68                    df = pd.read_excel(
69                        io.BytesIO(decoded),
70                        na_values=["", " ", "NA", "N/A", "nan", "NaN", "na"]
71                    )
72                else:
73                    continue
74            except Exception as e:
75                return (
76                    no_update,
77                    uploaded_bases,
78                    html.Div(f"Erreur chargement {filename}: {str(e)}", style={'color': 'red'}),
79                    "", "", no_update, [], [], [], [], []
80                )
81            df = df.apply(lambda col: col.map(na_harmonize))
82            new_bases.append({'name': filename, 'data': df.to_json(date_format='iso', orient='split')})
83            uploaded_bases.extend(new_bases)
84        df = pd.read_json(io.StringIO(new_bases[-1]['data']), orient='split')
85        msg_upload = f'{len(new_bases)} base(s) chargée(s).'
86

```

FIGURE 11 – Callback 3, partie 1 dans callbacks.py

```

1 # Fusion de deux bases sur une colonne commune
2     elif triggered == "fusion-bases":
3         if uploaded_bases and len(uploaded_bases) > 1 and fusion_col:
4             df1 = pd.read_json(io.StringIO(uploaded_bases[-2]['data']), orient='split')
5             df2 = pd.read_json(io.StringIO(uploaded_bases[-1]['data']), orient='split')
6             merged = pd.merge(df1, df2, on=fusion_col, how='outer')
7             uploaded_bases.append({'name': "Fusion", 'data': merged.to_json(date_format='iso', orient='split')})
8             df = merged
9             msg_upload = f"Bases fusionnées par {fusion_col}."
10        else:
11            return (no_update, uploaded_bases, "Sélectionnez une colonne commune.", "", "", no_update, [], [], [], [], [])
12
13 # Concaténation simple de deux bases
14     elif triggered == "concat-bases":
15         if uploaded_bases and len(uploaded_bases) > 1:
16             df1 = pd.read_json(io.StringIO(uploaded_bases[-2]['data']), orient='split')
17             df2 = pd.read_json(io.StringIO(uploaded_bases[-1]['data']), orient='split')
18             concat = pd.concat([df1, df2], axis=0, ignore_index=True)
19             uploaded_bases.append({'name': "Concat", 'data': concat.to_json(date_format='iso', orient='split')})
20             df = concat
21             msg_upload = "Bases concaténées."
22        else:
23            return (no_update, uploaded_bases, "Impossible de concaténer.", "", "", no_update, [], [], [], [], [])
24
25 # Suppression du dernier import (undo)
26     elif triggered == "remove-last-base":
27         if uploaded_bases and len(uploaded_bases) > 1:
28             uploaded_bases.pop()
29             last_df = pd.read_json(io.StringIO(uploaded_bases[-1]['data']), orient='split')
30             df = last_df
31             msg_upload = "Dernière base supprimée."
32        else:
33            return (no_update, uploaded_bases, "Impossible de supprimer.", "", "", no_update, [], [], [], [], [])
34
35 # Récupération de l'état courant
36     elif stored_data:
37         df = pd.read_json(io.StringIO(stored_data), orient='split')
38     else:
39         return EMPTY_RETURN
40
41 # --- Gestion des valeurs manquantes ---
42     elif triggered == "drop-na-rows":
43         df = df.dropna()
44         msg_na = "Lignes avec NA supprimées."
45         if df.empty:
46             msg_na += " Attention : toutes les lignes ont été supprimées !"
47
48     elif triggered == "drop-na-cols":
49         df = df.dropna(axis=1)
50         msg_na = "Colonnes avec NA supprimées."
51         if df.empty:
52             msg_na += " Attention : toutes les colonnes ont été supprimées !"
53
54     elif triggered == "fill-na-mean":
55         df = df.apply(lambda col: col.fillna(col.mean()) if pd.api.types.is_numeric_dtype(col) else col)
56         msg_na = "NA remplacés par la moyenne pour les colonnes numériques."
57
58     elif triggered == "fill-na-median":
59         df = df.apply(lambda col: col.fillna(col.median()) if pd.api.types.is_numeric_dtype(col) else col)
60         msg_na = "NA remplacés par la médiane pour les colonnes numériques."
61
62     elif triggered == "fill-na-value":
63         if na_value is None or na_value == "":
64             msg_na = "Veuillez entrer une valeur de remplacement."
65         else:
66             try:
67                 val = float(na_value)
68             except Exception:
69                 val = na_value
70             df = df.fillna(val)
71             msg_na = f"NA remplacés par '{val}'."
72
73     elif triggered == "reset-na":
74         if not uploaded_bases:
75             msg_na = "Aucune base initiale pour réinitialiser."
76         else:
77             last_base = uploaded_bases[-1]
78             df = pd.read_json(io.StringIO(last_base['data']), orient='split')
79             msg_na = "Données réinitialisées."
80
81     elif triggered == "drop-duplicates":
82         # On convertit les colonnes de type "object" (texte) en minuscules
83         df_temp = df.copy()
84         for col in df_temp.select_dtypes(include='object').columns:
85             df_temp[col] = df_temp[col].str.lower()
86             mask = ~df_temp.duplicated()
87             df = df[mask]
88             msg_na = "Les doublons ont été supprimés (sans tenir compte de la casse)."
89             if df.empty:
90                 msg_na += " Attention : toutes les lignes étaient des doublons."
91
92

```

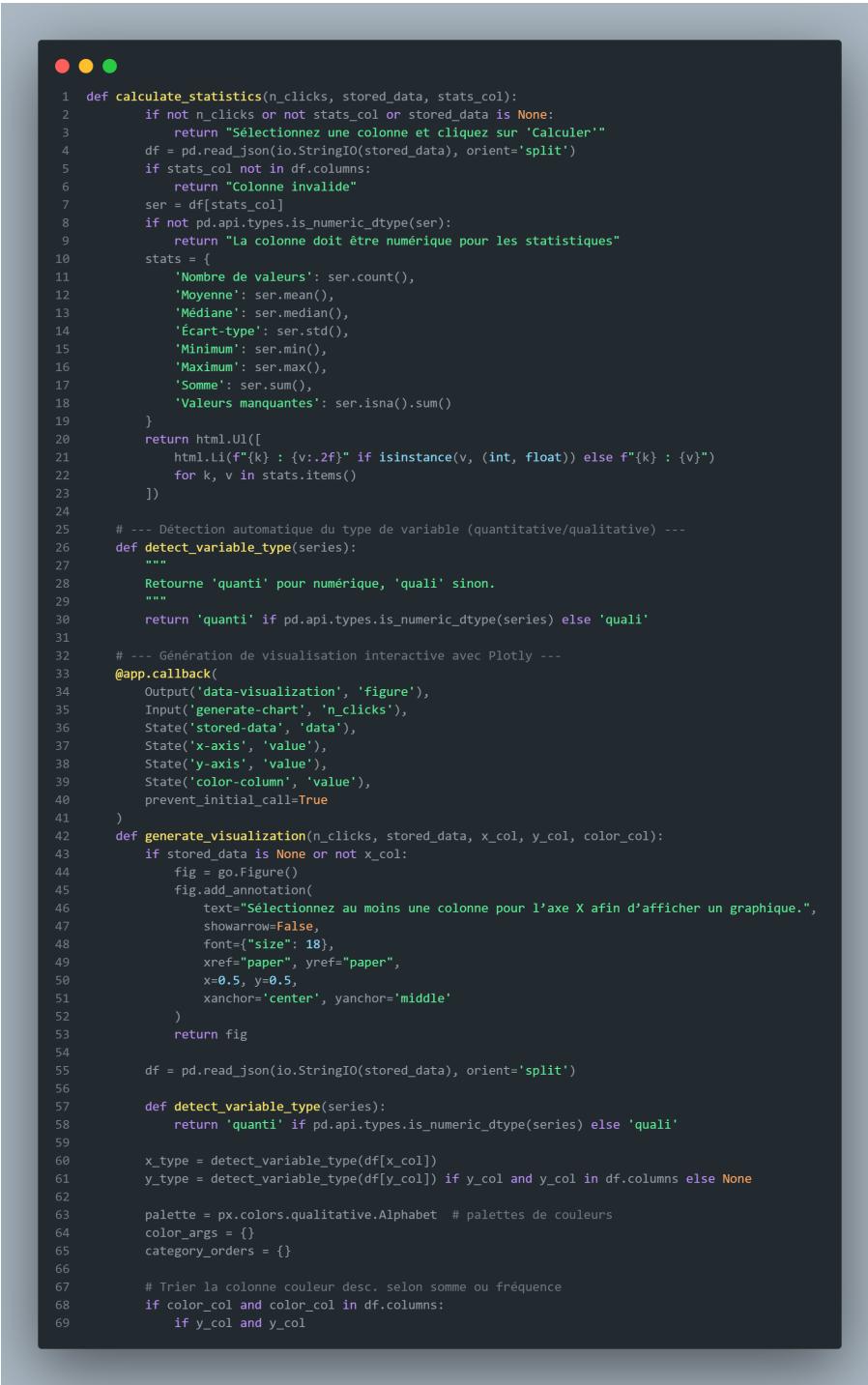
FIGURE 12 – Callback 4, partie 2 dans callbacks.py

```

1
2 # --- Recherche globale ---
3 if triggered == "search-input":
4     if search_value and search_value.strip() != "":
5         # Filtre sur les lignes où au moins une valeur contient la recherche
6         mask = df.apply(lambda row: row.astype(str).str.contains(search_value, case=False, na=False), axis=1).any(axis=1)
7         df = df[mask]
8         # Style pour surligner les résultats (fond vert)
9         style_data_conditional = []
10        for col in df.columns:
11            style_data_conditional.append({
12                'if': {
13                    'filter_query': '{{{col}}} contains "{}".format(col=col, search_val=search_value),
14                    'column_id': col
15                },
16                'backgroundColor': "#5B975D",
17                'color': 'black',
18            })
19        else:
20            # Efface la recherche : réinitialise les données
21            if uploaded_bases:
22                last_base = uploaded_bases[-1]
23                df = pd.read_json(io.StringIO(last_base['data']), orient='split')
24            else:
25                df = pd.DataFrame()
26            style_data_conditional = []
27        else:
28            style_data_conditional = []
29
30 # --- Filtres avancés (colonne/condition/valeur) ---
31 if triggered == 'apply-filter' and all([filter_col, filter_cond, filter_val]):
32     try:
33         if filter_cond == 'contains':
34             mask = df[filter_col].astype(str).str.contains(str(filter_val), na=False)
35         elif filter_cond == '>':
36             mask = df[filter_col].astype(float) > float(filter_val)
37         elif filter_cond == '<':
38             mask = df[filter_col].astype(float) < float(filter_val)
39         elif filter_cond == '>=':
40             mask = df[filter_col].astype(float) >= float(filter_val)
41         elif filter_cond == '<=':
42             mask = df[filter_col].astype(float) <= float(filter_val)
43         elif filter_cond == '=':
44             mask = df[filter_col] == filter_val
45         elif filter_cond == '!=':
46             mask = df[filter_col] != filter_val
47         else:
48             mask = pd.Series([True] * len(df))
49         df = df[mask]
50         msg_upload = "Filtre appliqué."
51     except Exception:
52         pass
53
54 elif triggered == 'reset-filter':
55     if uploaded_bases:
56         last_base = uploaded_bases[-1]
57         df = pd.read_json(io.StringIO(last_base['data']), orient='split')
58         msg_upload = "Filtres réinitialisés."
59     else:
60         msg_upload = "Aucune base pour réinitialiser les filtres."
61
62 # --- Diagnostic des valeurs manquantes (affichage dans l'interface) ---
63 na_counts = df.isna().sum()
64 total_na = int(na_counts.sum())
65 if total_na == 0:
66     na_diag = html.Div("Aucune donnée manquante détectée.", style={'color': 'green'})
67 else:
68     na_diag = html.Div([
69         html.Strong(f"total_na} valeur(s) manquante(s) détectée(s)."),
70         html.Ul([html.Li(f"{col} : {int(na)}") for col, na in na_counts.items() if na > 0]),
71     ], style={'color': 'orange'})
72
73
74 # --- Affichage du tableau interactif (Dash DataTable) ---
75 table = dash_table.DataTable(
76     data=df.to_dict('records'),
77     columns=[{"name": i, "id": i} for i in df.columns],
78     page_size=20,
79     style_table={'overflowX': 'auto'},
80     style_cell={'textAlign': 'left', 'padding': '5px'},
81     style_data={'color': 'black', 'backgroundColor': 'white'},
82     style_header={'backgroundColor': '#F5E8D8', 'color': 'white', 'fontWeight': 'bold'},
83     filter_action='native',
84     sort_action='native',
85     export_format='csv',
86     style_data_conditional=style_data_conditional,
87 )
88
89 # Génère les options pour tous les dropdowns reliés aux colonnes
90 options_cols = cols_options(df)
91
92 # Retourne l'état de l'application après chaque opération
93 return (
94     df.to_json(date_format='iso', orient='split'),
95     uploaded_bases,
96     html.Div(msg_upload, style={'color': 'green'}),
97     html.Div(msg_na, style={'color': 'blue'}),
98     na_diag,
99     table,
100    options_cols,
101    options_cols,
102    options_cols,
103    options_cols,
104    options_cols,
105 )
106

```

FIGURE 13 – Callback 5, partie 3 dans callbacks.py

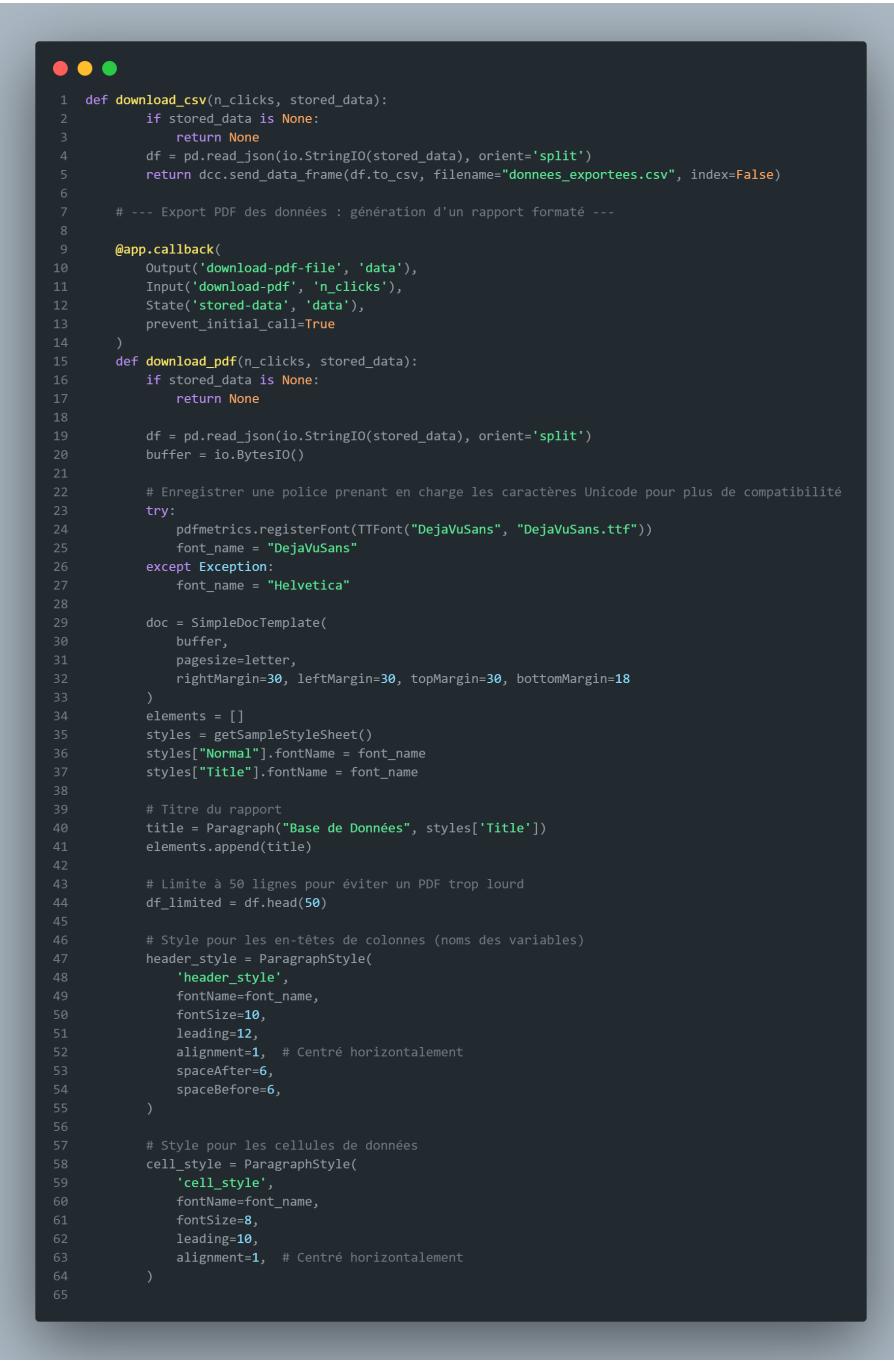


```

1  def calculate_statistics(n_clicks, stored_data, stats_col):
2      if not n_clicks or not stats_col or stored_data is None:
3          return "Sélectionnez une colonne et cliquez sur 'Calculer'"
4      df = pd.read_json(io.StringIO(stored_data), orient='split')
5      if stats_col not in df.columns:
6          return "Colonne invalide"
7      ser = df[stats_col]
8      if not pd.api.types.is_numeric_dtype(ser):
9          return "La colonne doit être numérique pour les statistiques"
10     stats = {
11         'Nombre de valeurs': ser.count(),
12         'Moyenne': ser.mean(),
13         'Médiane': ser.median(),
14         'Écart-type': ser.std(),
15         'Minimum': ser.min(),
16         'Maximum': ser.max(),
17         'Somme': ser.sum(),
18         'Valeurs manquantes': ser.isna().sum()
19     }
20     return html.Ul([
21         html.Li(f'{k} : {v:.2f}' if isinstance(v, (int, float)) else f'{k} : {v}')
22         for k, v in stats.items()
23     ])
24
25 # --- Détection automatique du type de variable (quantitative/qualitative) ---
26 def detect_variable_type(series):
27     """
28     Retourne 'quanti' pour numérique, 'quali' sinon.
29     """
30     return 'quanti' if pd.api.types.is_numeric_dtype(series) else 'quali'
31
32 # --- Génération de visualisation interactive avec Plotly ---
33 @app.callback(
34     Output('data-visualization', 'figure'),
35     Input('generate-chart', 'n_clicks'),
36     State('stored-data', 'data'),
37     State('x-axis', 'value'),
38     State('y-axis', 'value'),
39     State('color-column', 'value'),
40     prevent_initial_call=True
41 )
42 def generate_visualization(n_clicks, stored_data, x_col, y_col, color_col):
43     if stored_data is None or not x_col:
44         fig = go.Figure()
45         fig.add_annotation(
46             text="Sélectionnez au moins une colonne pour l'axe X afin d'afficher un graphique.",
47             showarrow=False,
48             font={"size": 18},
49             xref="paper", yref="paper",
50             x=0.5, y=0.5,
51             xanchor='center', yanchor='middle'
52         )
53     return fig
54
55 df = pd.read_json(io.StringIO(stored_data), orient='split')
56
57 def detect_variable_type(series):
58     return 'quanti' if pd.api.types.is_numeric_dtype(series) else 'quali'
59
60 x_type = detect_variable_type(df[x_col])
61 y_type = detect_variable_type(df[y_col]) if y_col and y_col in df.columns else None
62
63 palette = px.colors.qualitative.Alphabet # palettes de couleurs
64 color_args = {}
65 category_orders = {}
66
67 # Trier la colonne couleur desc. selon somme ou fréquence
68 if color_col and color_col in df.columns:
69     if y_col and y_col

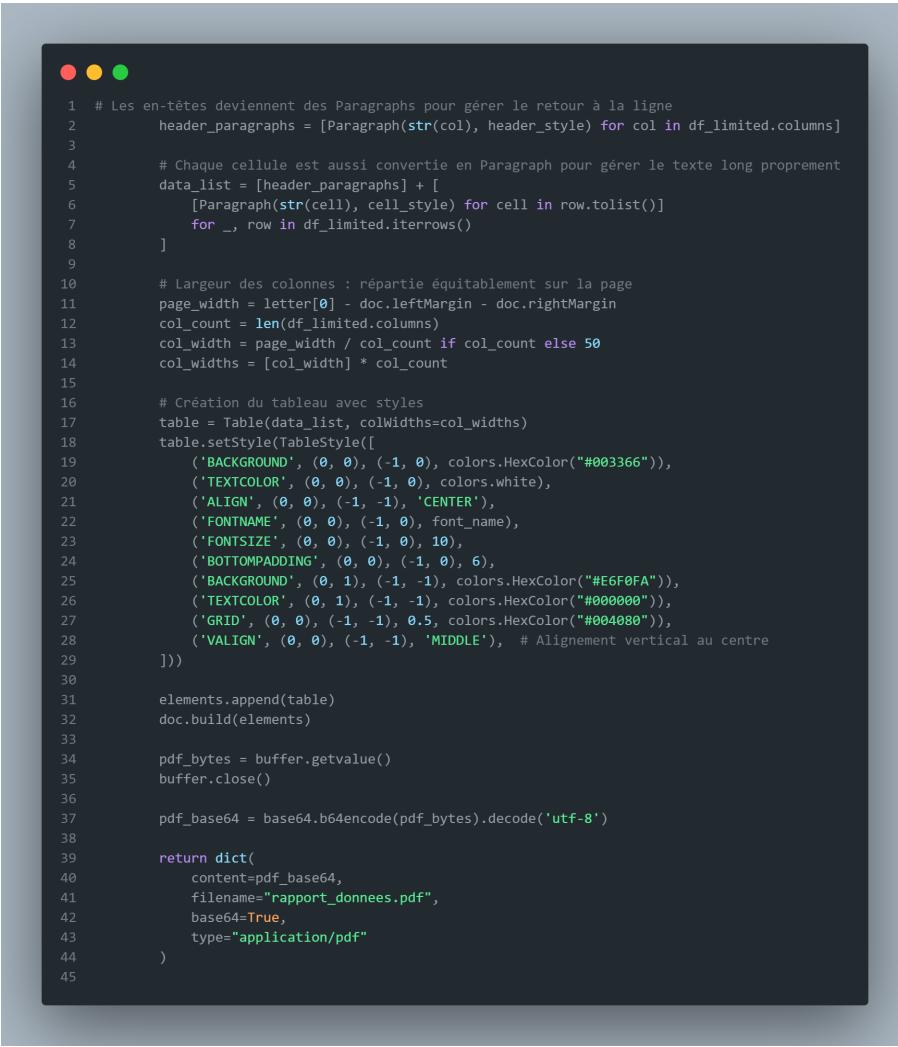
```

FIGURE 14 – Callback 5 dans callbacks.py



```
1 def download_csv(n_clicks, stored_data):
2     if stored_data is None:
3         return None
4     df = pd.read_json(io.StringIO(stored_data), orient='split')
5     return dcc.send_data_frame(df.to_csv, filename="donnees_exportees.csv", index=False)
6
7 # --- Export PDF des données : génération d'un rapport formaté ---
8
9 @app.callback(
10     Output('download-pdf-file', 'data'),
11     Input('download-pdf', 'n_clicks'),
12     State('stored-data', 'data'),
13     prevent_initial_call=True
14 )
15 def download_pdf(n_clicks, stored_data):
16     if stored_data is None:
17         return None
18
19     df = pd.read_json(io.StringIO(stored_data), orient='split')
20     buffer = io.BytesIO()
21
22     # Enregistrer une police prenant en charge les caractères Unicode pour plus de compatibilité
23     try:
24         pdfmetrics.registerFont(TTFont("DejaVuSans", "DejaVuSans.ttf"))
25         font_name = "DejaVuSans"
26     except Exception:
27         font_name = "Helvetica"
28
29     doc = SimpleDocTemplate(
30         buffer,
31         pagesize=letter,
32         rightMargin=30, leftMargin=30, topMargin=30, bottomMargin=18
33     )
34     elements = []
35     styles = getSampleStyleSheet()
36     styles["Normal"].fontName = font_name
37     styles["Title"].fontName = font_name
38
39     # Titre du rapport
40     title = Paragraph("Base de Données", styles['Title'])
41     elements.append(title)
42
43     # Limite à 50 lignes pour éviter un PDF trop lourd
44     df_limited = df.head(50)
45
46     # Style pour les en-têtes de colonnes (noms des variables)
47     header_style = ParagraphStyle(
48         'header_style',
49         fontName=font_name,
50         fontSize=10,
51         leading=12,
52         alignment=1, # Centré horizontalement
53         spaceAfter=6,
54         spaceBefore=6,
55     )
56
57     # Style pour les cellules de données
58     cell_style = ParagraphStyle(
59         'cell_style',
60         fontName=font_name,
61         fontSize=8,
62         leading=10,
63         alignment=1, # Centré horizontalement
64     )
65
```

FIGURE 15 – Callback 6, partie 1 dans callbacks.py



```
1 # Les en-têtes deviennent des Paragraphs pour gérer le retour à la ligne
2 header_paragraphs = [Paragraph(str(col), header_style) for col in df_limited.columns]
3
4 # Chaque cellule est aussi convertie en Paragraph pour gérer le texte long proprement
5 data_list = [header_paragraphs] + [
6     [Paragraph(str(cell), cell_style) for cell in row.tolist()]
7     for _, row in df_limited.iterrows()
8 ]
9
10 # Largeur des colonnes : répartie équitablement sur la page
11 page_width = letter[0] - doc.leftMargin - doc.rightMargin
12 col_count = len(df_limited.columns)
13 col_width = page_width / col_count if col_count else 50
14 col_widths = [col_width] * col_count
15
16 # Création du tableau avec styles
17 table = Table(data_list, colWidths=col_widths)
18 table.setStyle(TableStyle([
19     ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor("#003366")),
20     ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
21     ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
22     ('FONTNAME', (0, 0), (-1, 0), font_name),
23     ('FONTSIZE', (0, 0), (-1, 0), 10),
24     ('BOTTOMPADDING', (0, 0), (-1, 0), 6),
25     ('BACKGROUND', (0, 1), (-1, -1), colors.HexColor("#E6F0FA")),
26     ('TEXTCOLOR', (0, 1), (-1, -1), colors.HexColor("#000000")),
27     ('GRID', (0, 0), (-1, -1), 0.5, colors.HexColor("#004080")),
28     ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'), # Alignement vertical au centre
29 ])
30 )
31 elements.append(table)
32 doc.build(elements)
33
34 pdf_bytes = buffer.getvalue()
35 buffer.close()
36
37 pdf_base64 = base64.b64encode(pdf_bytes).decode('utf-8')
38
39 return dict(
40     content=pdf_base64,
41     filename="rapport_donnees.pdf",
42     base64=True,
43     type="application/pdf"
44 )
```

FIGURE 16 – Callback 7, partie 2 dans callbacks.py