

MASTER ÉCONOMISTE D'ENTREPRISE



Mémoire de master 1

Datazen : application de traitement de données

VINCENT Alexis

Université de Tours

Promotion 2025

M. Frederic Luret

1 juin 2025 - 31 juillet 2025

1 Remerciements

Je tiens à remercier mon tuteur, M. Frédéric Luret, pour son soutien et ses conseils tout au long de ce mémoire. Sa disponibilité et son expertise ont été précieuses pour la réalisation de ce travail.

2 Résumé

Ce mémoire présente le développement d’une application modulaire entièrement codée en Python. Son objectif principal est de permettre aussi bien aux débutants qu’aux professionnels de manipuler des données de manière simple, rapide et intuitive, sans nécessiter de connaissances techniques en programmation.

Les logiciels de traitement de données comme Excel peuvent devenir complexes à utiliser dès qu’on atteint un certain niveau de fonctionnalités. De plus, ils rencontrent des problèmes de performance avec des fichiers volumineux. L’application développée vise à répondre à ces deux limites majeures, en proposant une interface accessible et fluide.

Conçue avec Dash (framework Python), elle intègre les fonctionnalités de base suivantes :

- importation de fichiers CSV et Excel,
- filtrage, tri et modification des données,
- fusion de fichiers,
- visualisation sous forme de tableaux interactifs, graphiques dynamiques et indicateurs,
- export des résultats au format CSV ou Excel.

L’ensemble de ces outils permet une gestion simple et efficace des données, dans un environnement web.

Table des matières

1	Remerciements	2
2	Résumé	3
3	Contexte	6
4	Problématique	6
5	Objectifs	7
6	Méthodologie	7
6.1	Spécifications fonctionnelles	7
6.2	fonctionnalités de base	8
6.2.1	Importation de fichiers	8
6.2.2	Filtrage et tri des données	8
6.2.3	Modification des données et sauvegarde	9
6.2.4	Fusion de fichiers	9
6.2.5	Exportation de données	10
6.2.6	Visualisation des données	10
6.3	Technologies utilisées	10
6.3.1	Python	10
6.3.2	Dash	11
6.3.3	Pandas	12
6.3.4	Plotly	12
6.3.5	Le cache	13
6.4	Environnement de développement	14
6.5	Methodologie de développement	14
6.6	Planification du projet	15
6.7	Type d'interface	15
6.7.1	Thème de l'interface	16
7	Architecture de l'application	17
7.1	Préambule	17
7.2	Organisation du code	17
7.3	app.py	18

8 Composants	21
8.1 menudata	21
8.1.1 dcc.Store	23
8.1.2 Importation de fichiers	24
8.1.3 Affichage des popups	24
8.1.4 Logique d'importation	28
8.1.5 Fonctions d'importation	32
8.1.6 Résumé et comparaison de l'importation des fichiers	34
8.1.7 Stockage des fichiers dans l'UI	34
8.2 table_container	35
8.2.1 Sauvegarde	36
8.2.2 Les filtres et tris	36
8.2.3 Exemple de filtre	38
8.2.4 Historique et application des filtres et tris	41
8.2.5 Application des filtres et tris	42
8.2.6 Conclusion filtre et tri	45
8.3 Fusion de données	45
8.3.1 Exportation des données	45
8.4 Visualization_container	45
8.5 Tableaux et graphiques	47
8.5.1 Graphique avancé	48
8.6 Conclusion	50
9 Discussion et optimisation	50
10 Bilan personnel	51
11 Conclusion	51
12 Annexes	52
12.1 Guide utilisateur	52
12.1.1 Cas données : automobile	52
12.1.2 Valeurs aberrantes	62
12.1.3 Graphiques avancés	66
12.1.4 Fusion de données	68
12.2 Guide d'installation de Datazen	72
12.2.1 Installation rapide	72
12.3 Lancement de l'application	73
13 Bibliographie	73

3 Contexte

Aujourd'hui, de nombreux logiciels de traitement et de visualisation de données sont utilisés à travers le monde. L'apprentissage des notions de traitement et de visualisation de données est devenu un enjeu majeur pour divers acteurs, notamment les entreprises. Ces dernières ont pour but de maximiser leurs bénéfices chaque année. Pour cela, elles doivent se fixer des objectifs ou des quotas à respecter afin d'assurer une croissance continue. Cela est relativement simple pour les petites entreprises. Cependant, dès qu'une entreprise se compose de plusieurs outils internes et externes, ainsi que d'une clientèle importante, la gestion des données devient plus complexe. Avec l'ère du numérique, les entreprises, notamment les plus grandes, ont pris l'habitude de digitaliser toutes sortes de données, aussi bien côté client que dans leurs structures internes. Selon les estimations, le volume mondial de données générées est passé de 2 zettaoctets en 2010 à 64 zettaoctets en 2020, et devrait atteindre 180 zettaoctets d'ici 2025, soit une croissance annuelle moyenne de près de 40 %. Pour se représenter cette quantité, il faudrait environ 640 millions de disques SSD de 100 To pour stocker les données générées en 2020. Pourtant, seulement 2 % de ces données sont effectivement sauvegardées.

Conserver ces données est important, mais il faut surtout savoir les utiliser efficacement. C'est là qu'interviennent des logiciels de traitement de données, comme le plus connu Excel, mais aussi Power BI dans un cadre plus avancé. Ces outils permettent, avec un peu de formation, d'interagir avec les données et d'effectuer des opérations variées, allant de l'édition simple à des calculs plus poussés, comme des statistiques ou la création rapide de graphiques.

Cependant, deux problèmes majeurs apparaissent avec ce type de logiciels. Le premier est qu'ils nécessitent une solide formation et une certaine facilité de prise en main pour être utilisés à leur plein potentiel. Le second concerne des problèmes de flexibilité et de performance : par exemple, dans Excel, dès qu'un fichier devient trop volumineux, la fluidité et la rapidité de lecture peuvent être impactées. De plus, l'automatisation de certaines tâches via VBA demande encore plus de connaissances que l'utilisation des fonctions et raccourcis déjà présents dans l'application.

Il serait donc intéressant de disposer d'une alternative, certes moins puissante que Excel en termes de fonctionnalités, mais capable d'effectuer des traitements simples de données et de les visualiser de manière tout aussi simple.

4 Problématique

Comment créer un logiciel de traitement et de visualisation de données simple, efficace et accessible à tous, sans nécessiter de formation poussée ni de connaissances techniques avancées et entièrement en Python ?

5 Objectifs

Ce mémoire a pour objectif de concevoir et développer une application modulaire en Python, destinée à la gestion, au traitement et à la visualisation de données tabulaires. L'application visera à offrir une alternative simple et accessible, adaptée tant aux débutants qu'aux professionnels, sans chercher à rivaliser avec des outils complexes et puissants comme Excel ou Power BI.

L'objectif est de permettre une manipulation rapide et intuitive des données, avec des fonctionnalités de base efficaces, capables de gérer des jeux de données de tailles variées. Cette application se voudra légère, facile à prendre en main et évolutive, afin de pouvoir intégrer à terme des outils supplémentaires.

Ce mémoire s'organise autour de trois grands axes :

- La méthodologie, qui reviendra sur les consignes de développement, les outils utilisés et la planification du projet
- L'architecture de l'application, qui décrira la structure du code, les différents modules et leur interaction
- Un guide utilisateur, qui expliquera comment utiliser l'application. Il sera présent en annexe.

6 Méthodologie

6.1 Spécifications fonctionnelles

Pour le développement, le gestionnaire de projet Python **uv** sera utilisé. Il permettra de travailler dans un environnement virtuel isolé, évitant ainsi les problèmes de dépendances ou de conflits entre bibliothèques. L'application sera développée avec Dash, un framework permettant la conception d'applications web interactives, tout en offrant une grande flexibilité de personnalisation.

De plus, l'application, conçue de manière modulaire, sera organisée en quatre fichiers principaux :

- **app.py** : Contient l'application Dash elle-même. L'exécution de ce script lancera l'interface web. À terme, une version exécutable (.exe) sera générée pour faciliter l'utilisation de l'outil, notamment pour les utilisateurs non familiers avec Python.
- **layouts.py** : Définit la structure visuelle de l'application. Il regroupe tous les composants statiques de l'interface (titres, boutons, menus déroulants, zones de texte, etc.) ainsi que la disposition générale des éléments sur la page.
- **callbacks.py** : Regroupe l'ensemble des callbacks Dash. Il définit les interactions dynamiques entre les composants de l'interface, comme par exemple les liens entre les menus de sélection, les tableaux de données et les graphiques.

- **data_manager.py** : Centralise toutes les fonctions liées au traitement des données. Il assure le chargement des fichiers (CSV, Excel), les opérations de filtrage et de tri, ainsi que le calcul de statistiques de base nécessaires à la visualisation.

Cette organisation modulaire permettra de maintenir un code propre, lisible et facilement extensible.

6.2 fonctionnalités de base

6.2.1 Importation de fichiers

L'application devra permettre l'importation de plusieurs fichiers aux formats CSV ou Excel. L'utilisateur pourra importer ces fichiers un par un, les visualiser via un menu dédié, les afficher librement dans l'interface, ou les retirer selon ses besoins.

6.2.2 Filtrage et tri des données

L'utilisateur pourra appliquer des filtres sur les colonnes du jeu de données actif, afin de n'afficher que les lignes correspondant à des critères spécifiques. De plus, il sera possible de trier les données selon une ou plusieurs colonnes, dans un ordre croissant ou décroissant.

Une fois un filtre ou un tri appliqué, il apparaîtra dans un historique situé dans un menu déroulant où l'utilisateur pourra suivre l'ajout des filtres et des tris et les enlever à son bon vouloir.

TABLE 1 – Filtres disponibles dans Datazen

Type de filtre	Description
Recherche exacte	Filtre les données pour ne conserver que les valeurs exactement égales à la recherche.
Recherche "contient"	Filtre les données contenant une chaîne de caractères spécifique dans une colonne.
Comparaison	Permet de filtrer selon des opérateurs : supérieur, inférieur, supérieur ou égal, inférieur ou égal, égal, différent.
Filtrage par type de colonne	Sélectionne uniquement les colonnes d'un certain type : quantitatif, qualitatif ou booléen.
Filtrage par sélection de colonnes	Permet de ne garder que les colonnes spécifiées par l'utilisateur.
Gestion des valeurs manquantes	Supprime ou remplace les valeurs manquantes dans les données.
Traitement des outliers	Permet de détecter et supprimer ou modifier les valeurs aberrantes.

TABLE 2 – Tris disponibles dans Datazen

Type de tri	Description
Tri alphabétique	Trie les colonnes de type texte selon l'ordre alphabétique croissant ou décroissant.
Tri numérique	Trie les colonnes de type numérique ou booléen selon l'ordre croissant ou décroissant.

6.2.3 Modification des données et sauvegarde

L'application permettra à l'utilisateur de modifier les données directement dans l'interface. Les modifications pourront être effectuées sur des cellules individuelles. Une fois les modifications apportées, l'utilisateur pourra choisir de les sauvegarder dans le fichier d'origine à l'aide d'un bouton dédié (attention, il sauvegardera aussi l'état après filtrage et triage).

6.2.4 Fusion de fichiers

L'application proposera une fonctionnalité de fusion de fichiers. L'utilisateur pourra sélectionner deux fichiers importés pour les fusionner. Si une fusion est effectuée, elle sera placée dans la même section que les fichiers importés, et l'utilisateur pourra directement travailler dessus.

TABLE 3 – Types de fusion disponibles dans Datazen

Type de fusion	Description
Concaténation (lignes)	Combine deux fichiers en ajoutant les lignes de l'un à la fin de l'autre, en supposant que les colonnes sont identiques.
Concaténation (colonnes)	Combine deux fichiers en ajoutant les colonnes de l'un à côté de celles de l'autre, en supposant que les lignes correspondent.
Merge inner	Fusionne deux fichiers en ne conservant que les lignes présentes dans les deux fichiers.
Merge outer	Fusionne deux fichiers en conservant toutes les lignes des deux fichiers, en remplissant avec des valeurs manquantes quand nécessaire.
Merge left	Fusionne en conservant toutes les lignes du fichier de gauche, en ajoutant les données correspondantes du fichier de droite.
Merge right	Fusionne en conservant toutes les lignes du fichier de droite, en ajoutant les données correspondantes du fichier de gauche.

6.2.5 Exportation de données

L'utilisateur pourra exporter les données traitées dans différents formats (CSV, Excel) via un bouton d'exportation. Idéal pour transformer un Excel en CSV ou inversement, ou pour exporter les données après traitement.

6.2.6 Visualisation des données

L'utilisateur sera capable de visualiser dynamiquement ses données (filtrées, triées ou modifiées) sous forme de tableaux, de graphiques interactifs et d'indicateurs.

TABLE 4 – Visualisations disponibles dans Datazen

Catégorie	Type de données	Visualisation
Tableaux	Quantitatives	Statistiques descriptives (moyenne, somme, écart-type, et plus), matrice de corrélation
	Qualitatives	Tableau de fréquences
Graphiques standards	Quantitatives	Histogramme, Boxplot, Violinplot
	Qualitatives	Barplot, Camembert (Pie chart)
Graphiques avancés	Quantitatives	Nuage de points (scatter plot), Droite de régression, Boxplot, Violinplot
	Qualitatives	Boxplot, Violinplot

6.3 Technologies utilisées

6.3.1 Python

Python est un langage de programmation créé en 1991 par Guido van Rossum. Ce langage est connu dans le monde du développement pour sa simplicité et sa lisibilité, permettant une application dans divers secteurs tels que le développement web, l'analyse de données, l'intelligence artificielle, et bien d'autres. Il est également très populaire dans le domaine de la science des données et de l'apprentissage automatique grâce à sa vaste bibliothèque de modules et de frameworks.

Un module en Python est simplement un fichier/script contenant du code Python (souvent des fonctions, des variables et des classes) qui peut être utilisé dans d'autres scripts Python. Dit comme ça, cela ne semble pas très utile, mais il va nous permettre, en combinant plusieurs fichiers de ce type, de créer des packages Python.

Un package Python est un ensemble de modules regroupés dans un répertoire. Il permet d'organiser le code de manière structurée et de faciliter la réutilisation des modules. Un package peut contenir plusieurs sous-packages, chacun avec ses propres modules. Comme dit précédemment, le but d'un package est de permettre à un utilisateur d'utiliser des technologies déjà codées par un autre utilisateur Python sans avoir à les faire lui-même. En plus de cela, cela permet, lors de l'utilisation de packages reconnus, d'éviter tout problème lié à des erreurs de codage ou des bugs. Cela permet donc, dans notre cas, de rendre notre code plus robuste et de gagner du temps tout en utilisant des technologies avancées comme Dash.

Pour finir, notre projet, étant aussi modulaire, s'orientera vers une architecture de type MVC (Modèle-Vue-Contrôleur). Cette architecture est très utilisée dans le développement d'applications web et permet de séparer la logique métier (`data_manager.py`), la présentation (`layout.py`) et la gestion des interactions (`callback.py`). Cela facilite la maintenance et la lisibilité.

6.3.2 Dash

Comme cité précédemment, nous utiliserons Dash pour toute la partie graphique de l'application. Mais pourquoi choisir cet outil plutôt qu'un affichage classique dans la console ? En effet, un des principaux enjeux en data science est la présentation des résultats de manière claire, fluide et interactive. L'utilisation du terminal est certes puissante, mais atteint vite ses limites dans le cadre d'un projet comme celui-ci.

Dash permet justement de faire le lien entre développement web et code Python. Il a été conçu pour permettre à des développeurs Python de créer des applications web interactives sans avoir à maîtriser le JavaScript ou les technologies front-end classiques. En d'autres termes, Dash génère automatiquement le HTML et le CSS nécessaires, tout en offrant une personnalisation poussée via le CSS si besoin.

Techniquement, Dash repose sur trois piliers :

- **Flask**, pour gérer le backend (les routes, le serveur local, etc.)
- **Plotly.js**, pour les graphiques interactifs
- **React.js**, pour l'interface utilisateur dynamique côté navigateur

L'utilisateur n'a ainsi pas besoin de recharger la page grâce à un système de callbacks déclaratifs, qui permettent de réagir automatiquement aux interactions sur le site.

Parallèlement, plusieurs outils sont disponibles pour accélérer le développement, comme le module `dash_core_components`, qui permet d'ajouter rapidement des éléments d'interface tels que des menus déroulants, des graphiques interactifs, des sliders, etc. À la manière de *Bootstrap* pour le HTML, ces composants offrent un gain de temps considérable pour construire une interface utilisateur complète et réactive.

6.3.3 Pandas

Pour remplir l'une des premières fonctionnalités majeures de notre application à savoir la lecture, l'importation, le filtrage et le tri de fichiers CSV ou Excel nous utiliserons la bibliothèque Pandas.

Cette bibliothèque a été choisie car elle est largement reconnue et approuvée par la communauté des développeurs, en particulier dans le domaine de la data science. Avec plus de 3 400 contributeurs et plus de 334 millions de téléchargements mensuels sur PyPI, Pandas représente une solution fiable, robuste et bien documentée. Elle nous permettra ainsi de gagner un temps précieux dans le développement, sans compromis important sur la performance ou la fiabilité.

Elle nous offrira notamment les fonctionnalités suivantes :

- Lecture et écriture de fichiers CSV ou Excel de manière simple et rapide.
- Manipulation aisée des données sous forme de tableaux.
- Filtrage, tri, regroupement et transformation des données.
- Intégration fluide avec d'autres bibliothèques comme Plotly, Dash.

Enfin, l'un des atouts majeurs de Pandas réside dans sa capacité à manipuler des structures de données complexes, telles que les séries temporelles ou encore les fichiers Excel comportant plusieurs feuilles.

6.3.4 Plotly

Une des fonctionnalités annexes, mais non des moindres, est l'utilisation du package Plotly pour gérer la partie génération de graphiques de notre application.

Cette bibliothèque est largement reconnue dans le monde du développement, avec 255 contributeurs et environ 25 millions de téléchargements mensuels. Son utilisation vise, comme pour Pandas, à obtenir un gain de temps conséquent lors du développement, sans compromettre la performance ni la fiabilité.

Plotly permet de créer une large gamme de graphiques, allant des histogrammes, courbes, boîtes à moustaches, aux graphiques plus complexes comme les heatmaps, les graphiques 3D, ou encore les cartes géographiques.

De plus, Plotly offre une interactivité avancée aux utilisateurs, leur permettant de zoomer, survoler, sélectionner ou filtrer les données directement depuis les graphiques, ce qui améliore significativement l'expérience utilisateur.

Enfin, son intégration native avec le framework Dash facilite la création d'applications web interactives, rendant Plotly particulièrement adapté aux projets nécessitant une visualisation dynamique et riche en fonctionnalités.

6.3.5 Le cache

Nous reviendrons plus en détail plus tard dans le mémoire sur les raisons d'utiliser un cache pour notre application. Cependant, il est nécessaire de présenter rapidement celui que l'on va utiliser.

Globalement, le cache est un espace de stockage temporaire qui permet de conserver des données fréquemment utilisées pour accélérer les accès ultérieurs. Dans le contexte de notre application, il sera utilisé pour stocker les résultats des opérations de traitement de données, afin d'éviter de recalculer les mêmes résultats à chaque interaction de l'utilisateur.

Afin d'éviter le développement d'un cache maison, nous utiliserons le module `flask_caching` qui est un module de cache pour Flask, le framework web sur lequel Dash est construit. Il permet de stocker en mémoire les différents jeux de données importés.

TABLE 5 – Types de cache dans `flask-caching`

Type	Description rapide
<code>simple</code>	Cache en mémoire, non persistant.
<code>filesystem</code>	Sauvegarde les données en cache sur le disque local.
<code>redis</code>	Utilise une base Redis pour un cache rapide et distribué. Stocké dans la RAM
<code>memcached</code>	Cache distribué rapide mais sans persistance. aussi dans la RAM

Le cache choisi sera le `filesystem`, ce choix est motivé par deux raisons essentielles. Premièrement, contrairement aux autres types de caches du module, c'est le seul qui stocke les données en dehors de la RAM, ce qui permet, pour les petites configurations, de ne pas saturer la mémoire vive de l'ordinateur. Deuxièmement, le fait de stocker les données sur le disque permet de les conserver afin de réduire les temps de chargement en cas d'importation répétée d'un même jeu de données.

```
1 cache.init_app(  
2     app.server ,  
3     config={  
4         "CACHE_TYPE": "filesystem",  
5         "CACHE_DIR": "flask_cache",  
6         "CACHE_DEFAULT_TIMEOUT": 0,  
7     },  
8 )
```

Ce code permet d'initialiser le cache et se situe dans le fichier `app.py`. Il configure le cache pour utiliser le type `filesystem`, en spécifiant le répertoire où les données en cache seront stockées. Le paramètre `CACHE_DEFAULT_TIMEOUT` est défini à 0, ce qui signifie que les données en cache ne seront pas expirées automatiquement. Cela permet de conserver les données en cache indéfiniment, tant que l'application est en cours d'exécution.

6.4 Environnement de développement

La combinaison des outils précédemment cités permet de garantir un projet solide et fiable. Il convient toutefois de définir l'environnement de développement utilisé.

Le projet sera initialisé à l'aide de l'outil **uv**, et structuré sous forme de package. Cette organisation modulaire permettra d'utiliser les différentes fonctionnalités (importation, tri, visualisation, etc.) de manière indépendante, en dehors même de l'application principale.

Une fois le développement terminé, l'ensemble du code sera publié en **open source**, afin de permettre à toute personne intéressée de réutiliser les fonctions du package dans ses propres projets.

Le développement sera entièrement réalisé sur **Visual Studio Code**, un environnement complet et adapté au développement Python et web. L'intégralité du projet sera disponible sur github : <https://github.com/alexisvincent37/Datazen>.

6.5 Methodologie de développement

Chaque élément de l'application sera développé un par un. Cette méthode peut présenter certains inconvénients, auxquels nous reviendrons plus tard. Cependant, chacun de ces éléments suivra un protocole précis une fois établi.

- **Tests unitaires** : Chaque fonction développée fera l'objet de tests unitaires afin de vérifier son bon comportement et d'éviter des bugs imprévus durant le développement. L'outil `pytest` sera utilisé pour réaliser ces tests et sera exécuté régulièrement afin de maintenir un bon taux de couverture et de garantir la stabilité du code.
- **Black** : L'outil de formatage `black` sera utilisé régulièrement pour assurer une lisibilité optimale du code. Il permettra de garantir le respect des normes de style en vigueur, notamment celles de la convention PEP8.
- **Gestion d'erreurs** : Une fois les fonctionnalités développées ainsi que les callbacks, une gestion des erreurs sera effectuée en cas de nécessité.
- **Documentation** : Toutes les fonctions du projet, en particulier celles constituant le `data_manager.py`, seront documentées à l'aide de docstrings afin de faciliter la compréhension et la maintenance du code, notamment en cas de reprise par un autre développeur. Un fichier README sera également rédigé pour présenter à

la fois le package et l'application Dash, et sera accessible sur la page GitHub du projet.

De plus l'application sera développée pas à pas c'est à dire que chaque fonctionnalité qu'elles soit calculatoire ou graphiques seront développées simultanément. Cela permettra d'assurer une cohérence entre les différentes parties de l'application et de faciliter les tests et l'intégration.

6.6 Planification du projet

Le projet a été réalisé en 1 mois et 3 semaines, le développement a donc été divisé en plusieurs étapes.

- **Semaine 1** : Documentation, rédaction du cahier des charges, mise en place de l'environnement de développement, création du package, installation des dépendances et configuration de l'application Dash. Choix du type d'interface web et des composants à utiliser. Début du développement de l'interface utilisateur.
- **Semaine 2** : Développement de la fonctionnalité d'importation de fichiers, création des callbacks associés, tests unitaires et gestion des erreurs. Début du développement de la fonctionnalité de filtrage et de tri des données ainsi que la modification unitaire des données.
- **Semaine 3** : Finalisation du développement des filtres, mise en place d'un historique des filtres et tris appliqués, et mise en place d'un système de sauvegarde des données modifiées.
- **Semaine 4** : Développement d'une fonctionnalité d'archivage et de suppression des filtres et tris appliqués, ainsi que début de l'ajout de la fonctionnalité de fusion de données.
- **Semaine 5** : Fin du développement des fonctionnalités de fusion de données, ajout de la fonctionnalité d'exportation des données, mise en place de l'export (CSV ou Excel) des données traitées. Préparation de la section liée à la visualisation.
- **Semaine 6** : Développement de la visualisation des données, création des graphiques interactifs et des tableaux dynamiques.
- **Semaine 7** : Changement de l'architecture de stockage et des calculs des données, mise en place du cache pour améliorer les performances de l'application. Finalisation de la documentation et des tests unitaires.
- **Semaine 8** : Tests sur des cas d'usage et de performance globale. Optimisation du code et rédaction du mémoire.

6.7 Type d'interface

Comme dit précédemment, l'application sera une application web. Cependant, il existe plusieurs types d'interfaces web. En effet, on pourrait partir sur une interface à plusieurs onglets, fenêtres, ou encore une interface unique avec des sections dynamiques.

Cependant, notre but ici, rappelons-le, est de créer une application simple et intuitive. Nous choisirons donc d'opter pour une interface de type **dashboard** ou **Power BI**, séparant clairement les différentes zones de l'application, à savoir : l'importation, le traitement et la visualisation des données.

Ces différentes sections seront présentées sous la forme de volets dépliables, permettant à l'utilisateur d'avoir un affichage clair de ce qu'il est en train de faire, tout en le guidant au mieux dans la recherche des différents outils mis à sa disposition.

Par ailleurs, comme vous l'avez compris, une interface de type **dashboard** restreint de manière significative l'espace disponible. Il faut donc trouver une solution pour afficher tous les éléments nécessaires à son utilisation. C'est pour cela que des pop-ups seront développés pour les différentes fonctionnalités, comme l'importation de données ou l'application des filtres, afin de toujours garder une interface claire et non surchargée.

6.7.1 Thème de l'interface

Afin d'orienter le design de l'interface de mon application, je me suis appuyé sur les résultats de l'étude menée par Qiangqiang Fan, Jinhan Xie, Zhaoyang Dong et Yang Wang. Cette étude analyse l'impact de la couleur du texte et de l'éclairage ambiant sur la performance cognitive et la fatigue visuelle. Les résultats montrent que des couleurs claires comme le jaune ou le blanc, surtout en mode sombre (polarité négative), entraînent une augmentation significative de l'accommodation pupillaire, du taux de clignement et de l'indice de fatigue visuelle. Bien que certaines couleurs favorisent les performances cognitives, elles génèrent aussi une charge oculaire plus élevée. En m'appuyant sur ces observations, j'ai fait le choix d'une interface sombre avec des couleurs de texte modérées, pour limiter la fatigue visuelle lors d'un usage prolongé, notamment dans des environnements peu éclairés. Ce choix vise à améliorer l'ergonomie de l'application tout en assurant un confort oculaire et une bonne lisibilité.

7 Architecture de l'application

7.1 Préambule

Dans cette section, nous allons détailler l'intégralité de l'architecture de l'application : comment chaque module s'articule avec les autres, ainsi que leur fonctionnement individuel. Nous aborderons également les différents problèmes rencontrés durant le développement, ainsi que les solutions apportées lorsque cela a été possible.

Enfin, cette section retracera un à un les éléments essentiels présents dans le code, afin de faciliter la compréhension pour toute personne souhaitant reprendre ou maintenir ce projet sans se perdre dans sa structure.

7.2 Organisation du code

Voici un aperçu de la racine du projet, ainsi que l'explication de l'utilité des fichiers principaux :

	.git	24/07/2025 03:45	Dossier de fichiers	
	.pytest_cache	24/07/2025 00:59	Dossier de fichiers	
	.venv	24/07/2025 01:08	Dossier de fichiers	
	__pycache__	23/07/2025 23:56	Dossier de fichiers	
	assets	23/07/2025 23:45	Dossier de fichiers	
	flask_cache	24/07/2025 02:36	Dossier de fichiers	
	src	23/07/2025 23:45	Dossier de fichiers	
	test	23/07/2025 23:45	Dossier de fichiers	
	.coverage	23/07/2025 22:51	Fichier COVERAGE	52 Ko
	.gitignore	24/07/2025 03:49	Fichier source Git l...	1 Ko
	.python-version	05/06/2025 14:02	Fichier PYTHON-...	1 Ko
	app	24/07/2025 00:41	Application	99 457 Ko
	app	24/07/2025 01:02	Fichier source Pyt...	1 Ko
	LICENSE	23/07/2025 23:38	Fichier	2 Ko
	pyproject	24/07/2025 00:58	Fichier source Toml	1 Ko
	README	24/07/2025 03:45	Fichier source Mar...	5 Ko
	uv.lock	24/07/2025 00:58	Fichier LOCK	110 Ko

- **app.py** : Point d'entrée de l'application, il configure et lance le serveur Dash. Une version .exe est aussi disponible pour une utilisation simplifiée (adaptée aux utilisateurs non familiers avec Python).
- **src/datazen/layouts.py** : Définit la structure de l'interface utilisateur, incluant les composants statiques.
- **src/datazen/callbacks.py** : Contient les fonctions de rappel qui gèrent les interactions dynamiques entre les composants de l'interface.
- **src/datazen/data_manager.py** : Regroupe les fonctions de traitement des données, comme l'importation, le filtrage et la fusion.
- **assets/styles.css** : Fichier de style CSS pour personnaliser l'apparence de l'application.
- **src/datazen/README.md** : Documentation du projet, expliquant son installation et son utilisation.
- **flask_cache/** : Répertoire contenant les données mises en cache.
- **test/** : Répertoire contenant les tests unitaires pour vérifier le bon fonctionnement des différentes fonctionnalités de l'application.

7.3 app.py

Voici un aperçu de la structure du fichier app.py :

```

1 from src.datazen.layouts import *
2 from src.datazen.data_manager import cache
3 from src.datazen.callbacks import *
4 from dash import Dash, html
5
6 app = Dash(external_stylesheets=external_stylesheets,
7           suppress_callback_exceptions=True)
8
9 cache.init_app(
10     app.server,
11     config={
12         "CACHE_TYPE": "filesystem",
13         "CACHE_DIR": "flask_cache",
14         "CACHE_DEFAULT_TIMEOUT": 3600,
15     },
16 )
17
18 app.layout = html.Div(
19     [
20         menudata,
21         importpopup,
22         table_container,
23         visualization_container,
24         text_filter_popup,
25         in_text_filter_popup,

```

```

26     comparaison_filter_popup ,
27     types_columns_popup ,
28     keep_columns_popup ,
29     Na_popup ,
30     outlier_popup ,
31     sort_abc_popup ,
32     sort_123_popup ,
33     concatenate_popup ,
34     merge_popup ,
35     export_popup ,
36 ],
37 id="app_container",
38 )
39
40 server = app.server
41
42 if __name__ == "__main__":
43     app.run(debug=True, host="0.0.0.0", port=8050)

```

Ce fichier initialise l'application Dash. Tout d'abord, nous importons nos différents modules ainsi que Dash lui-même. Ensuite, vient la configuration de l'application, où nous commençons par importer notre fichier CSS. Nous indiquons également à Dash que nous souhaitons supprimer les exceptions de callbacks. En effet, l'application étant conçue sous forme de dashboard et divisée au maximum en appels de callbacks, il est fréquent que plusieurs callbacks aient la même sortie, ce qui est par défaut bloqué par Dash afin d'éviter tout conflit.

Juste après, comme présenté précédemment, nous initialisons le cache en précisant son type, le répertoire de stockage des données mises en cache, ainsi que la durée de vie par défaut du cache.

Puis vient la partie graphique. L'application est composée de plusieurs sections, chacune représentant un élément de l'interface utilisateur. Ces éléments sont importés depuis le fichier `layouts.py` et organisés dans une structure HTML.

Nous notons, comme évoqué précédemment, la présence de trois grandes sections principales : `menudata`, `table_container` et `visualization_container`. Les autres éléments sont, comme dit, de simples popups destinées à alléger au maximum l'affichage et à offrir à l'utilisateur une vision claire de ses actions.

Voici la description de chaque section :

Nom	Description
menudata	Menu latéral gauche contenant les fichiers importés et le bouton d'importation
importpopup	Popup d'importation de fichiers
table_container	Conteneur principal permettant la visualisation des données sous forme de DataFrame, incluant les boutons de manipulation ainsi que l'historique des données
visualization_container	Conteneur pour les visualisations graphiques, incluant des valuebox, tableaux, graphiques unicolonnes, et une section avancée pour créer des graphiques à partir de deux colonnes
text_filter_popup	Popup pour la recherche exacte de texte dans une colonnes
in_text_filter_popup	Popup pour la recherche d'une suite de caractères dans une colonne
comparaison_filter_popup	Popup pour le filtrage par comparaison (numérique)
types_columns_popup	Popup pour le choix des types de colonnes (quantitatif, qualitatif, booléen)
keep_columns_popup	Popup pour la conservation des colonnes sélectionnées
Na_popup	Popup pour le traitement des valeurs manquantes
outlier_popup	Popup pour le traitement des valeurs aberrantes
sort_abc_popup	Popup pour le tri alphabétique
sort_123_popup	Popup pour le tri numérique
concatenate_popup	Popup pour la concaténation de colonnes ou de lignes
merge_popup	Popup pour la fusion de tables
export_popup	Popup pour l'exportation des données

TABLE 6 – Liste des composants et popups dans l'application

La ligne `server = app.server` permet de faire le lien entre l'application Dash et le serveur Flask sous-jacent, en exposant ce dernier sous un nom que les serveurs WSGI comme Gunicorn peuvent reconnaître et utiliser pour déployer correctement l'application.

Ces types de serveurs sont souvent utilisés pour déployer des applications Dash en production, car ils offrent une meilleure gestion des performances et de la scalabilité comparativement au serveur de développement intégré de Dash. Notamment grâce à la présence de workers, qui permettent de gérer plusieurs requêtes simultanément, améliorant ainsi la réactivité de l'application. Cependant, durant la phase de développement de l'application, la gestion multi-workers n'a pas été implémentée, mais elle pourrait l'être à l'avenir pour optimiser les performances.

Pour finir, nous pouvons constater que l'application, via le script `app.py`, se lance en mode debug, ce qui permet de voir les erreurs directement dans le navigateur et de recharger automatiquement l'application à chaque modification du code. Le `host` est défini sur « 0.0.0.0 », ce qui permet d'accéder à l'application depuis n'importe quelle adresse IP locale (comme `localhost:8050`).

L'exécutable, quant à lui, est lancé sans mode debug, car il est destiné à être utilisé par des utilisateurs non familiers avec Python. Il est donc préférable de ne pas afficher les erreurs à l'écran.

8 Composants

Dans cette section, nous allons décrire les différents composants de l'application, en mettant l'accent sur leur rôle et leur fonctionnement.

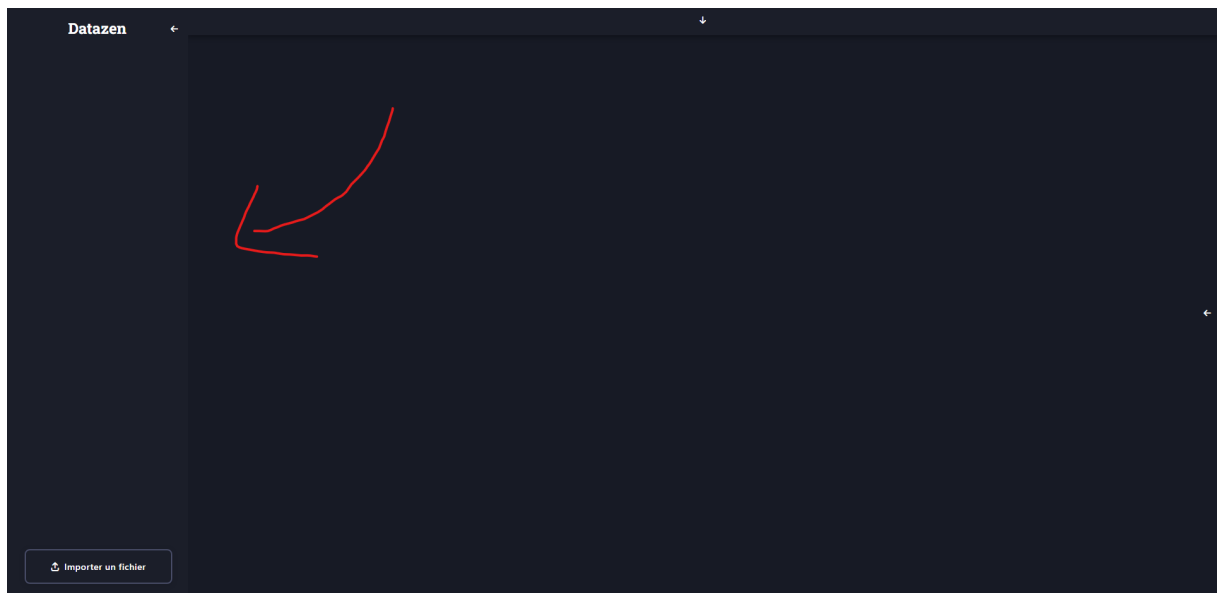
8.1 `menudata`

Le composant `menudata` est le menu latéral gauche de l'application. Il permet à l'utilisateur de visualiser les fichiers importés, d'accéder aux boutons d'importation et de gestion des données, ainsi que de naviguer vers les différentes sections de l'application.

Une fois l'application lancée, cette page s'affiche avec le menu et le bouton d'importation bien mis en évidence, afin d'indiquer à l'utilisateur instinctivement par où commencer.

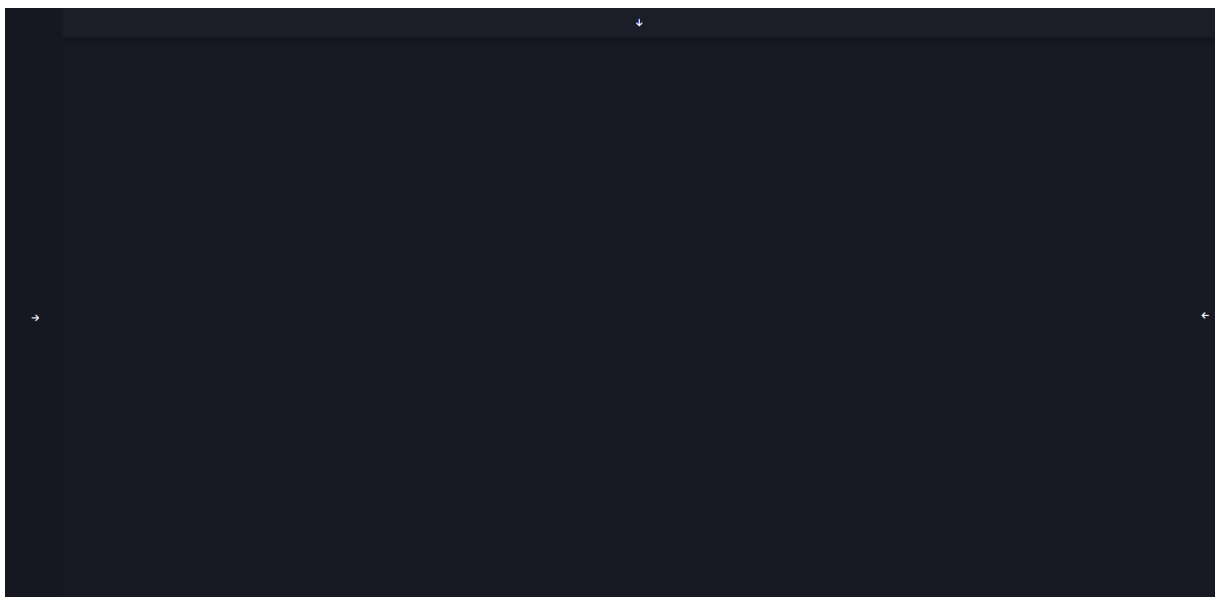
Comparé à d'autres applications, j'ai fait le choix de limiter au maximum les informations disponibles à l'utilisateur pour le guider et lui offrir une sorte de marche à suivre explicite, afin d'éviter toute surcharge visuelle, comme on peut le constater dans des applications telles qu'Excel, où les menus et boutons sont nombreux et dispersés partout.

Voici un aperçu de ce composant :



Un bouton de fleche en haut a droite de ce menu permet de le réduire ou de l'agrandir, afin de laisser plus de place à la visualisation des données.

Voici un aperçu du code de ce composant une fois fermé :



Ce composant est particulièrement simple :

```

1  menudata = html.Div(
2      children=[
3          html.Div(
4              [
5                  "Datazen",
6                  html.Button(
7                      html.I(className="fa-solid fa-arrow-left"),
8                      id="sidebar_button"
9                  ),

```

```

10         ],
11         id="left_menu_header",
12     ),
13     html.Div([], id="datastorage"),
14     html.Div(
15         children=[
16             html.I(className="fa-solid fa-arrow-up-from-bracket"),
17             "Importer un fichier",
18         ],
19         id="importbutton",
20     ),
21     dcc.Store(id="stored-data"),
22 ],
23 id="menudata",
24 )

```

On voit tout de même plusieurs éléments importants dans ce code. Tout d'abord, le titre de l'application, qui est affiché en haut du menu. Ensuite, le bouton d'importation, qui permet à l'utilisateur de charger un fichier dans l'application. Enfin, le composant `dcc.Store` est utilisé pour stocker les données importées et les rendre accessibles aux autres composants de l'application.

La div `datastorage` est vide au départ, mais elle sera remplie dynamiquement avec les fichiers importés par l'utilisateur. Cela permet de garder une trace des fichiers chargés et les sélectionner pour les visualiser ou de les supprimer.

8.1.1 `dcc.Store`

Il est important de s'arrêter sur ce composant, car il a été central dans le développement de l'application. Le composant `dcc.Store` est utilisé pour stocker des données dans l'application Dash. Il permet de conserver des informations entre les différentes interactions de l'utilisateur, sans avoir à les recalculer à chaque fois.

Le problème est qu'il ne stocke les informations qu'au format JSON. Or, dans le cadre de cette application, les jeux de données manipulés sont souvent volumineux, sous forme de tableaux complexes (DataFrames pandas). La conversion répétée entre ces tableaux et le format JSON, nécessaire pour le stockage dans `dcc.Store`, est très coûteuse en termes de calcul. En effet, la transformation de listes ou d'objets complexes en JSON, puis leur reconversion en DataFrame, prend du temps et impacte directement les performances de l'application.

De plus, puisque les données sont stockées dans la mémoire du navigateur, qui est généralement limitée, l'utilisation de `dcc.Store` pour des jeux de données lourds peut entraîner une augmentation importante de la taille des fichiers JSON échangés. Cela peut

provoquer un ralentissement notable, voire des plantages, notamment lors de l'exécution de nombreux callbacks qui manipulent ces données. Ces limitations doivent donc être prises en compte pour garantir la fluidité et la stabilité de l'application.

Nous reviendrons plus tard sur la solution apportée à ces problèmes et aux différentes versions de stockages des données.

8.1.2 Importation de fichiers

Pour importer un fichier dans l'application, il suffit simplement de cliquer sur le bouton d'importation. Cela ouvre une fenêtre de dialogue permettant à l'utilisateur de sélectionner un fichier CSV ou Excel à importer, via un composant `dcc.Upload`. Une fois le fichier chargé, une interface personnalisée s'affiche automatiquement. Celle-ci permet à l'utilisateur de configurer plusieurs paramètres essentiels pour un import propre et cohérent avec le format du fichier.

Pour les fichiers CSV, il est possible de définir le séparateur de colonnes (comme la virgule, le point-virgule, ou la tabulation), le caractère utilisé pour les décimales (point ou virgule), ainsi que la ligne à utiliser comme en-tête (utile lorsque les fichiers contiennent des lignes d'information avant les noms de colonnes).

Pour les fichiers Excel, l'utilisateur peut sélectionner la feuille de calcul à importer, choisir le caractère de décimale, et également indiquer la ligne d'en-tête à prendre en compte. Cette étape permet d'assurer que les colonnes soient correctement interprétées et que les données soient bien structurées dès l'import.

Les sélections sont possibles grâce à des composants Dash appelés `dcc.Dropdown` et `dcc.Input`, qui permettent de créer des menus déroulants et des champs de saisie pour les paramètres d'importation.

En cas d'erreurs de type de fichier importé, un message d'erreur s'affiche pour informer l'utilisateur que le fichier n'est pas au format attendu et après un délai de 5 scd grâce à un `dcc.Interval`, le `dcc.Upload` est réinitialisé pour permettre un nouvel essai.

8.1.3 Affichage des popups

Les popups sont des fenêtres modales qui s'affichent au-dessus de l'interface principale pour permettre à l'utilisateur d'interagir avec certaines fonctionnalités spécifiques de l'application. Elles sont utilisées pour des actions précises, comme l'importation de fichiers, la configuration de filtres, ou encore le traitement des valeurs manquantes.

Un popup ne possède pas de classe CSS propre, mais il dispose d'un identifiant unique

(id), ce qui permet de le différencier clairement des autres composants de l'application. Chaque popup est défini dans le fichier `layouts.py` et activé via un callback bien spécifique.

Voici un exemple de callback permettant de modifier dynamiquement la classe CSS du composant `menudata` afin de masquer ou d'afficher le menu latéral gauche :

```
1 @callback(
2     Output("menudata", "className"),
3     Input("sidebar_button", "n_clicks"),
4     State("menudata", "className"),
5     prevent_initial_call=True,
6 )
7 def toggle_sidebar(n_clicks, active_class):
8     if n_clicks is None:
9         return no_update
10
11     current_class = active_class if active_class else ""
12
13     if "closed" in current_class:
14         return current_class.replace("closed", "").strip()
15     else:
16         return current_class + "_closed"
```

Pour bien comprendre ce fonctionnement, il est utile de rappeler comment un callback est structuré dans Dash. Un callback est une fonction automatiquement appelée lorsqu'un ou plusieurs `Input` changent. Elle peut également utiliser un ou plusieurs `State` pour lire la valeur d'un composant sans déclencher le callback.

Dans cet exemple, le déclencheur est le nombre de clics sur le bouton `sidebar_button` (grâce à l'`Input`), tandis que le `State` permet de lire la classe CSS actuelle du menu latéral sans que cela n'entraîne d'exécution.

La fonction `toggle_sidebar` est alors exécutée : si aucun clic n'a eu lieu (`n_clicks is None`), on ne fait rien. Sinon, on regarde si la classe CSS actuelle contient `"closed"`. Si oui, on l'enlève (le menu s'ouvre). Sinon, on l'ajoute (le menu se ferme).

Ce principe est réutilisé de manière similaire dans tous les popups de l'application, chacun ayant un id unique, et un système de visibilité contrôlé via des classes CSS et des callbacks spécifiques.

On gardera le même principe pour les popups, cependant comme on ouvre les popups la classe rajoutée sera `"open"`.

Voici un exemple de popup d'importation de fichier :

```
1 @callback(
2     Output("importpopup", "className"),
3     Output("import-feedback-csv", "children"),
4     Output("import-feedback-excel", "children"),
5     Input("importbutton", "n_clicks"),
6     Input("importpopup_close_button", "n_clicks"),
7     Input("import-validate-button-csv", "n_clicks"),
8     Input("import-validate-button-excel", "n_clicks"),
9     State("importpopup", "className"),
10    State("import-data", "contents"),
11    State("import-data", "filename"),
12    State("import-separator", "value"),
13    State("import-decimal", "value"),
14    State("import-header", "value"),
15    State("import-sheetname", "value"),
16    State("import-decimal-excel", "value"),
17    State("import-header-excel", "value"),
18    prevent_initial_call=True,
19 )
20 def toggle_import_popup(
21     openclick,
22     closeclick,
23     importvalidation_csv,
24     importvalidation_excel,
25     active_class,
26     contents,
27     filename,
28     sep,
29     decimal_csv,
30     header_csv,
31     sheet_name,
32     decimal_excel,
33     header_excel,
34 ):
35     current_class = active_class or ""
36
37     if ctx.triggered_id == "importbutton":
38         if "open" not in current_class:
39             new_class = (current_class + "▢open").strip()
40             return new_class, "", ""
41             return current_class, "", ""
42
43     elif ctx.triggered_id == "importpopup_close_button":
```

```

44     new_class = current_class.replace("open", "").strip()
45     return new_class, "", ""
46
47     elif ctx.triggered_id == "import-validate-button-csv":
48         result = import_csv(contents, filename,
49                             sep, decimal_csv, header_csv)
50         error = result.get("error")
51         if error is None:
52             new_class = current_class.replace("open", "").strip()
53             return new_class, "", ""
54         return current_class,
55             error or "Une erreur inconnue est survenue.", ""
56
57     elif ctx.triggered_id == "import-validate-button-excel":
58         result = import_excel(
59             contents, filename, sheet_name,
60             header=header_excel,
61             decimal=decimal_excel
62         )
63         error = result.get("error")
64         data = result.get("data")
65         if error is None and data is not None:
66             new_class = current_class.replace("open", "").strip()
67             return new_class, "", ""
68         return current_class, "", error
69
70     return no_update, no_update, no_update

```

Le callback ci-dessus gère l'ouverture et la fermeture du popup d'importation, ainsi que la validation des données importées. Il utilise plusieurs `Input` pour détecter les actions de l'utilisateur, comme le clic sur le bouton d'importation, le bouton de fermeture du popup, ou les boutons de validation pour les fichiers CSV et Excel.

On observe l'ajout d'un nouvel élément central beaucoup utilisé dans les callbacks : `ctx.triggered_id`. Il s'agit d'une variable contextuelle qui permet de savoir quel élément a déclenché le callback. Cela est particulièrement utile lorsque plusieurs `Input` sont présents, car cela permet de savoir précisément quelle action a été effectuée par l'utilisateur, et ainsi d'adapter le comportement du callback en fonction du bouton cliqué.

Ce callback permet donc la gestion des boutons principaux du popup, ainsi que la validation des données importées. Il appelle les fonctions `import_csv` et `import_excel` pour traiter les fichiers importés, en passant les paramètres nécessaires comme le séparateur de colonnes, le caractère décimal, et la ligne d'en-tête. Si une erreur est détectée lors de l'importation, un message d'erreur approprié est affiché à l'utilisateur.

Le problème, c'est que simplement appeler ces fonctions pour effectuer une vérification basique n'est pas forcément optimal. Nous y reviendrons plus en détail à la fin de ce mémoire.

Ensuite dans un souci de synthèse nous allons faire la liste des callback qui gèrent le popup d'importation de fichier :

Nom du Callback	Description
toggle_button_and_type_error_import	Gère l'ouverture/fermeture du bouton d'importation (dans le popup) et affiche les erreurs liées au type de fichier importé.
reset_button_after_interval	Réinitialise l'état du bouton d'importation après un délai ou lors de la fermeture du popup.
update_import_title	Met à jour le titre général du popup avec le nom du fichier importé pour les fichiers CSV.
update_import_content_class_csv	Met à jour l'affichage du contenu spécifique à l'import CSV selon l'état du popup.
update_import_title_excel	Met à jour le titre du popup lors de l'import d'un fichier Excel, affiche le nom du fichier.
update_import_content_class_excel	Gère l'affichage et la fermeture du contenu spécifique à l'import Excel, vérifie la validation.

TABLE 7 – Liste des callbacks gérant le popup d'importation

8.1.4 Logique d'importation

En admettant que, malgré toutes les vérifications, une importation ait été effectuée, les données importées sont stockées dans le composant `dcc.Store` nommé `stored-data`. Ce composant permet de conserver les données en mémoire et de les rendre accessibles aux autres composants de l'application.

Chaque fichier est stocké de la manière suivante :

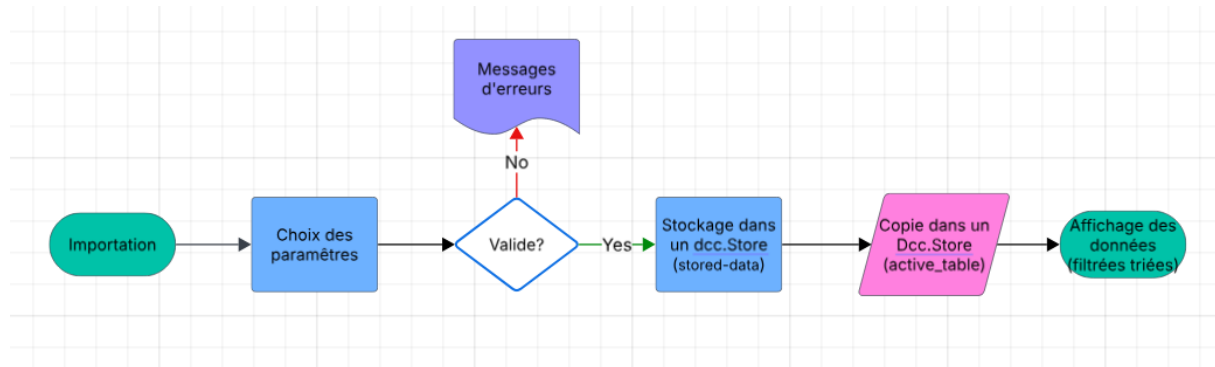
```

1 {
2   "data": {
3     "colonne1": [valeurs],
4     "colonne2": [valeurs],
5     ...
6   },
7   "file": {
8     "id": "uuid4",
9     "name": "nom_du_fichier"
10  }
11 }
```

Cette structure permet de conserver les données sous forme de dictionnaire, avec les noms des colonnes comme clés et les valeurs correspondantes dans des listes. Le champ "file" contient des informations sur le fichier importé, comme son nom et un identifiant

unique généré avec `uuid4` pour éviter les conflits entre plusieurs fichiers. Le problème de générer un identifiant aléatoire à chaque fichier importé est qu'il peut entraîner des incohérences si un même fichier est importé plusieurs fois. Donc il n'est pas possible d'éviter les doublons de cette manière.

La logique souhaitée pour l'importation des données est la suivante :



Cependant, comme évoqué précédemment, les données sont stockées dans le composant `dcc.Store`, qui utilise le format JSON. Cela rend leur manipulation lourde et coûteuse en termes de performances. De plus, ce composant utilise la mémoire du navigateur, ce qui limite fortement la taille des données pouvant être traitées efficacement.

Après plusieurs tests, l'import d'un fichier contenant environ 40 000 lignes et 13 variables l'importation prenait 13,64 secondes, et son affichage complet dans l'interface nécessitait près d'une minute. L'application étant censée répondre aux lenteurs des outils traditionnels comme Excel, de tels résultats étaient non recevables.

Le moindre ajout d'interaction avec ces données entraînait des ralentissements supplémentaires, notamment parce que :

- les données étaient dupliquées dans plusieurs `dcc.Store`,
- chaque opération nécessitait une reconversion du format JSON vers un `pandas.DataFrame`, ce qui est extrêmement coûteux en temps de calcul.

La solution mise en place pour remédier à ces problèmes est l'utilisation d'un système de cache. Ce cache a été choisi car il ne repose pas sur la mémoire vive, mais sur un stockage local directement sur l'ordinateur. Cela permet non seulement d'accélérer l'exécution, mais aussi d'éviter de réimporter les mêmes fichiers à chaque utilisation de l'application.

Une limitation subsiste cependant : le cache n'est pas automatiquement relu au démarrage. Il est donc toujours nécessaire de réimporter les fichiers souhaités manuellement, mais sans devoir en refaire le traitement complet.

Pour remédier également aux limites des identifiants UUID, un algorithme de hachage a été mis en place. Celui-ci génère un identifiant unique pour chaque fichier importé, basé sur son contenu. Ainsi, même si deux fichiers ont le même nom, ils seront considérés

comme différents si leur contenu diffère.

L'algorithme de hachage utilisé est le SHA-256, reconnu pour sa robustesse et sa résistance aux collisions. Il produit un condensé de 256 bits (32 octets) et est largement utilisé dans les systèmes où la fiabilité du hachage est cruciale.

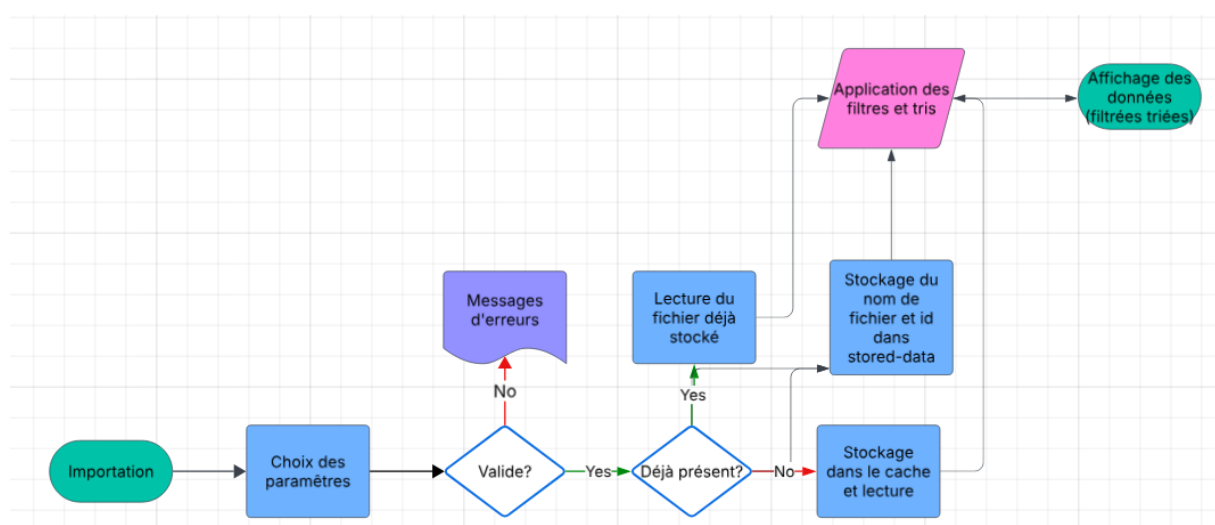
```
1 def id_hash(contents: str) -> str:
2     h = hashlib.sha256()
3     h.update(contents.encode("utf-8"))
4     return h.hexdigest()
```

Nous utilisons le module `hashlib` pour implémenter cet algorithme. Lorsqu'un fichier est lu via le composant `dcc.Upload`, deux éléments sont récupérés : le nom du fichier sous forme de chaîne de caractères, et son contenu encodé en base64. Ce format permet de représenter des données binaires sous forme ASCII, facilitant ainsi leur stockage et leur transmission.

Notre algorithme de hachage va donc prendre ce contenu encodé en base64 et le convertir en une chaîne de caractères unique. Cette chaîne servira d'identifiant pour le fichier importé, ce qui permet de le retrouver facilement dans le cache et d'éviter les doublons.

Il est préférable d'utiliser le contenu du fichier plutôt que son nom, car deux fichiers peuvent avoir le même nom tout en contenant des données différentes. Le contenu, lui, garantit l'unicité de l'identifiant généré.

La nouvelle logique d'importation des données est donc la suivante :



Avec cette méthode, l'importation du même fichier d'exemple prend désormais environ 4 secondes s'il n'est pas encore dans le cache, et seulement 3 secondes s'il y est déjà. Ce petit gain peut encore être optimisé si l'on trouve un moyen de vérifier la présence d'un

fichier dans le cache sans appeler les fonctions d'importation complètes.

Pourquoi gagne-t-on autant de temps ? Les deux raisons principales sont les suivantes :

- Les données ne sont plus stockées dans des `dcc.Store` JSON coûteux à manipuler ;
- Le fichier est converti directement en objet `DataFrame` via le cache, sans reconversion répétée.

En effet, on ne peut pas stocker un objet complexe comme un `pandas.DataFrame` directement dans le cache. C'est pourquoi on utilise le module `pickle` pour sérialiser l'objet : on le transforme en binaire avant de le stocker. `Pickle` est une bibliothèque Python qui permet de sérialiser et désérialiser des objets Python. La sérialisation consiste à convertir un objet en une suite d'octets pouvant être stockée ou transmise, et la désérialisation permet de le reconstruire. C'est une opération peu coûteuse en temps de calcul, ce qui accélère considérablement l'affichage et le traitement des données dans l'interface.

Pour finir voici un aperçu de la structure du cache :

```
1 {  
2     "Hash1": "pickle_data1",  
3     "Hash2": "pickle_data2",  
4 }
```

Les fonctions pour utiliser le caches sont les suivantes :

```
1 def set_df_to_cache(file_id: str, df: pd.DataFrame):  
2     data_pickle = pickle.dumps(df)  
3     cache.set(file_id, data_pickle, timeout=0)  
4  
5  
6 def get_df_from_cache(file_id: str) -> pd.DataFrame | None:  
7     pickled_df = cache.get(file_id)  
8     if pickled_df is None:  
9         return None  
10    return pickle.loads(pickled_df)
```

Du package `flask_caching`, nous avons utilisé la méthode `set` pour stocker les données dans le cache, et la méthode `get` pour les récupérer. Le paramètre `timeout` permet de définir la durée de vie du cache, ici fixée à 0, ce qui signifie que les données en cache ne seront pas expirées automatiquement.

Pour finir, nous y reviendrons plus en détail, mais maintenant le `dcc.Store stored-data` ne contient plus que les identifiants des fichiers importés, ainsi que leur nom. Cela permet de garder une trace des fichiers chargés sans alourdir la mémoire avec les données elles-mêmes.

Nous le verrons plus tard, car cela peut sembler un peu flou sur les schémas précédents,

mais à l'origine, lors de la première version où l'on stockait les données, on les copiait pour les mettre dans un autre `dcc.Store` appelé `active_table`, ayant pour objectif de contenir l'`id`, les filtres, les tris, et les données de la table active. Son but était de dissocier les données d'origine des données affichées.

Maintenant, avec cette nouvelle méthode, le composant `active_table` stocke uniquement l'`id`, les filtres et les tris.

8.1.5 Fonctions d'importation

Pour finir avec l'importation de fichiers, nous allons rapidement décortiquer leur fonctionnement. Les fonctions d'importation sont définies dans le fichier `data_manager.py`. Elles sont appelées depuis le callback d'importation ainsi que lors des vérifications, et permettent de traiter les fichiers importés en fonction de leur type (CSV ou Excel).

Il existe une fonction dédiée pour chaque type de fichier :

```
1 def import_csv(  
2     contents: str, filename: str, sep: str, decimal: str,  
3     header: int  
4 ) -> dict:  
5     try:  
6         _, content_string = contents.split(",", 1)  
7         decoded = base64.b64decode(content_string).decode("utf-8")  
8         if filename.endswith(".csv"):  
9             df = pd.read_csv(  
10                 io.StringIO(decoded), sep=sep, decimal=decimal,  
11                 header=header  
12             )  
13             return {"filename": filename, "panda_data": df}  
14     except Exception as e:  
15         return {"filename": filename, "panda_data": None,  
16                 "error": str(e)}
```

Cette fonction prend en entrée le contenu du fichier, son nom, le séparateur de colonnes, le caractère décimal et la ligne d'en-tête. Elle décode le contenu encodé en base64, puis utilise `pandas.read_csv` pour lire le fichier CSV et le convertir en `DataFrame`. Si une erreur survient, elle est capturée et renvoyée dans un dictionnaire avec le nom du fichier.

Comme mentionné précédemment, il est nécessaire de décoder la base64 car le contenu provient du composant `dcc.Upload`, qui encode les fichiers de cette manière pour les transmettre à l'application. Le module `base64` permet de décoder cette chaîne avant de la traiter avec `pandas`.

La ligne `_, content_string = contents.split(",", 1)` peut sembler étrange au premier abord. Elle utilise une syntaxe de décomposition de tuple pour séparer la chaîne `contents` en deux parties : la première (avant la virgule) est ignorée (représentée par

le tiret bas `_`), et la seconde (après la virgule) est stockée dans `content_string`. Cette séparation est nécessaire car les données transmises incluent un préfixe MIME (comme `data:text/csv;base64,`), et seule la partie après la virgule contient les données encodées du fichier.

Pour les fichiers Excel, la logique est similaire :

```
1 def import_excel(  
2     contents: str, filename: str, sheet_name, header=0, decimal="."  
3 ) -> dict:  
4     try:  
5         _, content_string = contents.split(",", 1)  
6         decoded = base64.b64decode(content_string)  
7         if filename.endswith(".xlsx") or filename.endswith(".xls"):  
8             df = pd.read_excel(  
9                 io.BytesIO(decoded),  
10                 sheet_name=sheet_name,  
11                 header=header,  
12                 decimal=decimal,  
13             )  
14             return {"filename": filename, "panda_data": df}  
15     except ValueError as e:  
16         if "Worksheet not found" in str(e) and "not found" in str(e):  
17             return {  
18                 "filename": filename,  
19                 "panda_data": None,  
20                 "error": f"Feuille Excel introuvable : '{sheet_name}'",  
21             }  
22         return {  
23             "filename": filename,  
24             "panda_data": None,  
25             "error": f"Erreur de valeur : {str(e)}",  
26         }  
27     except Exception as e:  
28         return {"filename": filename, "panda_data": None, "error": str(e)}
```

La seule différence notable est l'utilisation de `pandas.read_excel` au lieu de `read_csv`. Cette fonction permet notamment de spécifier la feuille à lire, ainsi que les paramètres d'en-tête et de caractère décimal. En cas d'erreur, un message spécifique est renvoyé si la feuille demandée est introuvable, ou un message générique pour les autres types d'erreurs.

8.1.6 Résumé et comparaison de l'importation des fichiers

L'importation repose sur un système optimisé combinant un identifiant de contenu (SHA-256) et un cache local. Cette approche accélère considérablement le chargement et la manipulation des données tout en réduisant l'usage de la mémoire vive et en évitant les doublons inutiles.

Critère	Avant (dcc.Store + UUID)	Après (Cache + SHA-256)
Stockage	Mémoire du navigateur	Stockage local (disque)
Format des données	JSON (pandas → JSON → pandas)	Pickle (pandas → binaire → pandas)
Gestion des doublons	Impossible (UUID aléatoire)	Identifiant unique basé sur le contenu (SHA-256)
Temps d'importation (40k lignes)	~13,6 s	~4 s (ou 3 s si en cache)
Temps d'affichage complet dans l'interface	~60 s	Quelques secondes
Coût en performance lors des interactions	Très élevé (reconversion JSON fréquente)	Faible (accès direct au DataFrame)
Risque de surcharge mémoire	Élevé (limite navigateur)	Faible (stockage disque)

8.1.7 Stockage des fichiers dans l'UI

Une fois un fichier importé, il est affiché dans le menu latéral gauche, dans le composant `datastorage`. Chaque fichier est représenté par un bouton contenant son nom (pointant vers son identifiant) et un second bouton sous forme de croix, permettant de le supprimer de l'interface.

Le callback `manage_files` gère cela. Il permet aussi et nous présenterons cela juste après que, lorsque l'on clique sur la petite croix, le fichier soit retiré à la fois de l'interface, mais aussi de la table active (`active_table`).

Pour finir voici un exemple de ce bouton :

```
1      [
2          html.Button(
3              f["name"],
4              className="data_added",
5              id={"type": "data_added", "index": f["id"]},
6          ),
```

```

7         html.Button(
8             html.I(className="fa-solid fa-xmark"),
9             className="data_remove",
10            id={"type": "data_remove", "index": f["id"]},
11        ),
12    ]

```

Il est intéressant de noter que l'on utilise un dictionnaire pour l'id du bouton, ce qui permet de le rendre unique et de le différencier des autres boutons. Cela est particulièrement utile pour les callbacks, car cela permet de savoir quel fichier a été sélectionné ou supprimé.

8.2 table_container

Maintenant que des données ont été importées, il est temps de les visualiser. Pour ce faire, il suffit, comme expliqué précédemment, de cliquer sur un nom de fichier dans le menu latéral gauche.

Une fois cela fait, un tableau s'affiche, représentant le jeu de données. Il est possible de modifier les cellules directement dans le tableau et de naviguer grâce à la pagination.

En parallèle, une fois le bouton cliqué, un `dcc.Store` est créé (`active_table`), contenant l'identifiant du fichier, les filtres et les tris.

Voici le contenu de `active_table` :

```

1 {
2     "id": "hash_du_fichier",
3     "filters": {},
4     "sorts": []
5 }

```

On utilise un dictionnaire plutôt qu'une liste pour stocker les filtres en générant un identifiant uuid4 comme clé afin de pouvoir cumuler plusieurs filtres sur une même colonne.

De plus, les boutons d'outils (cachés par défaut) sont affichés. Si l'on supprime le fichier, le tableau est vidé et les boutons d'outils sont à nouveau cachés.

Le callback permettant d'initialiser le jeu de donnée actif est `display_table`, et celui permettant de gérer les boutons d'outils est `toggle_btns`.

Datazen

used_car_price_data

Importer un fichier

make_year	mileage_kmpl	engine_cc	fuel_type	owner_count	price_usd	brand	transmission	color
2001	8.17	4000	Petrol	4	8587.64	Chevrolet	Manual	White
2014	17.59	1500	Petrol	4	5943.5	Honda	Manual	Black
2023	18.09	2500	Diesel	5	9273.58	BMW	Automatic	Black
2009	11.28	800	Petrol	1	6836.24	Hyundai	Manual	Blue
2005	12.23	1000	Petrol	2	4625.79	Nissan	Automatic	Red
2002	13.77	2500	Diesel	4	6856.03	BMW	Manual	Gray
2023	12.5	1000	Petrol	3	6806.77	BMW	Manual	Red
2015	17.46	2000	Diesel	5	7701.84	BMW	Manual	Blue
2001	14.99	5000	Diesel	2	8730.52	Tesla	Manual	Silver
2020	19.59	4000	Petrol	4	10812.16	Chevrolet	Automatic	White
2013	21.25	1800	Petrol	2	10070.16	BMW	Automatic	Red
2017	20.91	5000	Diesel	5	9638.93	Toyota	Manual	Silver
2005	15.95	2500	Petrol	3	5543.9	Kia	Manual	Black
2005	19.42	800	Petrol	4	5855.47	Hyundai	Automatic	Silver
2018	23.18	3000	Diesel	3	9773.77	Hyundai	Automatic	Blue
2015	20.23	4000	Diesel	5	9468.89	Kia	Manual	White
1998	27.29	1800	Petrol	3	6491.29	Tesla	Automatic	Gray
2002	22.91	2000	Diesel	5	6202.99	Kia	Automatic	Red
2018	26.59	3000	Diesel	3	9921.66	Kia	Manual	Blue
1997	5.54	1500	Diesel	2	1223.8	Chevrolet	Automatic	Black
2016	18.48	1000	Diesel	2	5591.11	Tesla	Automatic	Black
2015	16.38	5000	Electric	5	13054.69	BMW	Automatic	Gray
1996	17.38	3000	Petrol	2	5209.92	Chevrolet	Manual	White
2018	13.88	1000	Petrol	3	4916.24	Tesla	Automatic	Gray
2006	19.88	1800	Petrol	3	5906.84	Kia	Manual	Gray
2000	8.96	2000	Petrol	4	2626.29	Volkswagen	Automatic	Gray

1

385

Nous y reviendrons plus tard, mais une petite flèche est visible en haut, permettant d'afficher l'historique des filtres appliqués ainsi que les tris effectués.

8.2.1 Sauvegarde

Le premier bouton visible est celui de la sauvegarde. Il permet d'écraser les données présentes dans le cache avec les données actuellement affichées dans le tableau. Cela permet de conserver les modifications effectuées par l'utilisateur, comme les filtres ou les tris, et de les réutiliser lorsque l'utilisateur souhaite changer de fichier dans l'interface. Une petite animation de changement d'icône a lieu à chaque clic pour informer l'utilisateur que la sauvegarde a bien eu lieu, grâce à un `dcc.Interval`.

Le callback permettant de gérer la sauvegarde est `save_table_data`. Il est déclenché lorsque l'utilisateur clique sur le bouton de sauvegarde. Il récupère les données actuellement affichées dans le tableau, les sérialise en utilisant `pickle`, et les stocke dans le cache avec l'identifiant du fichier comme clé.

8.2.2 Les filtres et tris

Les filtres permettent à l'utilisateur de restreindre les données affichées dans le tableau en fonction de certains critères. Ils sont appliqués en temps réel et modifient immédiatement l'affichage des données, à l'aide du callback `update_table_with_filters_and_sort`, faisant appel à deux fonctions : `apply_filters` et `apply_sort`, appliquant respectivement les filtres et le tri sur le jeu de données.

Pour afficher la liste des filtres disponibles, il suffit de passer la souris sur le bouton **Filtres**. La classe va s'élargir grâce à une combinaison de `hover` et `transform` en CSS. La liste des filtres s'y trouve : chacun est un bouton cliquable ouvrant un popup pour

configurer et appliquer un filtre.

Le fonctionnement est similaire pour le tri. Un bouton **Tri** est placé juste en dessous de celui des filtres.

The screenshot shows the Datazen web application interface. On the left, there's a sidebar with a search bar containing 'used_car_price_data...' and a button 'Importer un fichier'. The main area displays a table with columns: make_year, mileage_kmpl, engine_cc, fuel_type, owner_count, price_usd, brand, transmission, and color. The table contains 30 rows of car data. On the right, a 'Filtres' (Filters) sidebar is open, listing various filter options: Recherche exacte, Recherche (in), Comparaison, Types colonnes, Colonnes gardées, Valeurs manquantes, and Valeur aberrante. At the bottom right, there are navigation arrows and a page number '1'.

Comme mentionné au début du rapport, il existe actuellement sept filtres et deux types de tris dans **Datazen**. Pour rappel, leurs caractéristiques sont résumées dans les tableaux suivants :

TABLE 8 – Filtres disponibles dans Datazen

Type de filtre	Description
Recherche exacte	Filtre les données pour ne conserver que les valeurs strictement égales à la recherche.
Recherche « contient »	Filtre les données contenant une chaîne de caractères spécifique dans une colonne.
Comparaison	Permet de filtrer selon des opérateurs : supérieur, inférieur, supérieur ou égal, inférieur ou égal, égal, différent.
Filtrage par type de colonne	Sélectionne uniquement les colonnes d'un certain type : quantitatif, qualitatif ou booléen.
Filtrage par sélection de colonnes	Permet de ne garder que les colonnes spécifiées par l'utilisateur.
Gestion des valeurs manquantes	Supprime ou remplace les valeurs manquantes dans les données.
Traitement des outliers	Permet de détecter et supprimer ou modifier les valeurs aberrantes.

TABLE 9 – Tris disponibles dans Datazen

Type de tri	Description
Tri alphabétique	Trie les colonnes de type texte selon l'ordre alphabétique croissant ou décroissant.
Tri numérique	Trie les colonnes de type numérique ou booléen selon l'ordre croissant ou décroissant.

8.2.3 Exemple de filtre

Il n'est pas nécessaire d'expliquer le fonctionnement de tous les filtres individuellement, car ils suivent tous la même logique de conception : clic sur un bouton, ouverture d'un popup spécifique au filtre, saisie des paramètres, puis application immédiate sur les données via le même pipeline de traitement.

Nous allons donc nous concentrer sur un filtre représentatif : le traitement des valeurs aberrantes (ou *outliers*).

```

1 outlier_popup = html.Div(
2     children=[
3         html.Button(
4             html.I(className="fa-solid fa-xmark"),
5             id="outlier_popup_close_button",
6             className="filter_close_button",
7         ),
8         html.H2("Valeurs aberrantes", id="outlier_popup_title"),
9         dcc.Dropdown(
10             id="outlier_column_dropdown",
11             placeholder="Choisir une colonne...",
12             style={
13                 "backgroundColor": "#2A2D3A",
14                 "border": "1px solid #2A2D3A",
15                 "color": "#000000",
16                 "width": "200px",
17             },
18         ),
19         dcc.Dropdown(
20             id="outlier_action_dropdown",
21             options=[
22                 {'label': 'Supprimer les lignes', 'value': 'drop'},
23                 {'label': 'Remplacer par la médiane', 'value': 'median'},
24                 {'label': 'Winsoriser (clipping)', 'value': 'winsorize'},
25             ],
26             placeholder="Choisir une action...",
27             style={

```

```

28         "backgroundColor": "#2A2D3A",
29         "border": "1px_solid_#2A2D3A",
30         "color": "#000000",
31         "width": "200px",
32     },
33 ),
34 html.Button(
35     "Valider",
36     className="import-validate-button",
37     id="import-validate-button-outlier",
38 ),
39 ],
40 id="outlier_popup",
41 )

```

Tout d'abord, nous définissons le popup dédié aux valeurs aberrantes (*outliers*). Il contient un bouton de fermeture, un titre, deux menus déroulants permettant de sélectionner la colonne cible ainsi que l'action à effectuer, et enfin un bouton de validation.

Il est important de noter que, pour chaque filtre ou tri, les composants sont adaptés selon les besoins spécifiques. Par exemple, le filtre de recherche nécessite uniquement un **dropdown** pour sélectionner la colonne et un champ de saisie pour la valeur recherchée, tandis que le filtre de comparaison inclut un **dropdown** pour choisir l'opérateur (supérieur, inférieur, égal, etc.) ainsi qu'un champ de saisie pour la valeur.

Voici comment est codé le filtre dans le fichier `data_manager.py` :

```

1  def filter_outlier(df: pd.DataFrame, col: str, action: str) -> pd.DataFrame:
2      df = df.copy()
3      Q1 = df[col].quantile(0.25)
4      Q3 = df[col].quantile(0.75)
5      IQR = Q3 - Q1
6      lower_bound = Q1 - 1.5 * IQR
7      upper_bound = Q3 + 1.5 * IQR
8
9      outliers = (df[col] < lower_bound) | (df[col] > upper_bound)
10
11     if action == "drop":
12         df = df.loc[~outliers]
13     elif action == "median":
14         median_val = df[col].median()
15         df.loc[outliers, col] = median_val
16     elif action == "winsorize":
17         df.loc[df[col] < lower_bound, col] = lower_bound
18         df.loc[df[col] > upper_bound, col] = upper_bound
19

```

```
return df
```

Cette fonction prend en entrée un dataframe (désérialisé depuis le cache), le nom de la colonne à traiter et l'action à effectuer. Elle calcule les bornes inférieure et supérieure en utilisant l'écart interquartile (IQR) pour identifier les valeurs aberrantes. Selon l'action choisie, elle supprime les lignes contenant des *outliers*, remplace ces valeurs par la médiane de la colonne, ou applique un clipping (winsorisation) pour les ramener dans les bornes définies.

Le clipping est une technique qui consiste à limiter les valeurs d'une variable à un certain intervalle. Dans ce cas, les valeurs inférieures à la borne inférieure sont remplacées par cette borne, et celles supérieures à la borne supérieure sont remplacées par cette dernière. Cela permet de conserver les données tout en réduisant l'impact des valeurs extrêmes.

Pour chaque filtre ou tri, il sera nécessaire de mener ces opérations en premier lieu, à savoir créer l'interface et la fonction de traitement.

```

1  def apply_filters(df: pd.DataFrame, filters: dict) -> pd.DataFrame:
2
3  for f_id, f in filters.items():
4      if f["type"] == "text":
5          df = filter_text(df, f["col"], f["value"])
6      elif f["type"] == "in_text":
7          df = filter_in_text(df, f["col"], f["value"])
8      elif f["type"] == "comparaison":
9          df = filter_comparaison(df, f["col"], f["value"],
10                                 f["operator"])
11      elif f["type"] == "types_columns":
12          df = filter_types_columns(df, f["col_type"])
13      elif f["type"] == "keep_columns":
14          df = filter_keep_columns(df, f["columns"])
15      elif f["type"] == "na":
16          df = filter_na(df, f["col"], f["action"])
17      elif f["type"] == "outlier":
18          df = filter_outlier(df, f["col"], f["action"])
19  return df

```

L'étape suivante consiste à intégrer notre fonction de filtre dans la fonction globale appelée par le callback, afin d'appliquer directement les filtres sur le `DataFrame`. On conservera la même logique pour les tris, en utilisant la fonction `apply_sort`.

Pour résumer, un filtre se compose d'une partie visuelle (le popup) et d'une partie fonctionnelle (les fonctions de traitement). Cependant, il est nécessaire de faire le lien entre les deux, c'est-à-dire de faire en sorte que lorsque l'on clique sur le bouton de

validation du popup, la fonction de traitement soit appelée avec les paramètres saisis dans celui-ci.

Pour cela, un filtre est toujours composé de trois callbacks :

- Un callback pour ouvrir et fermer le popup, en utilisant la classe CSS "open" pour rendre le popup visible ou non.
- Un callback pour remplir les dropdowns du popup ; ici, on remplit le dropdown pour qu'il ne contienne que les colonnes numériques, car le filtre des valeurs aberrantes ne peut être appliqué qu'à ces colonnes (ce qui évite la gestion d'erreurs dans la fonction de traitement).
- Un callback pour stocker le filtre (ou le tri) dans le `dcc.Store active_table`.

Ces callbacks sont respectivement :

- `toggle_popup_outlier` : gère l'ouverture et la fermeture du popup du filtre des valeurs aberrantes lorsque l'utilisateur clique sur le bouton correspondant, ainsi que la fermeture via la petite croix.
- `update_outlier_column_dropdown` : remplit le dropdown des colonnes avec les colonnes numériques du tableau actif. Il se déclenche quand les données de la table active changent.
- `store_outlier_filter` : stocke le filtre des valeurs aberrantes dans le `dcc.Store active_table`. Il crée un uuid4 pour pouvoir cumuler plusieurs fois le même filtre sur différentes colonnes ou la même pour certains et le stock sous la forme `<'uuid4'>:{"type": "outlier", "col": "nom_colonne", "action": "drop"}`. Il se déclenche quand l'utilisateur clique sur le bouton de validation du popup.

Ce processus est similaire pour tous les filtres et tris, avec des variations dans les paramètres spécifiques à chaque type de filtre ou de tri.

8.2.4 Historique et application des filtres et tris

En cliquant sur la flèche en haut du tableau, il est possible d'ouvrir l'historique des filtres et tris appliqués. Cet historique est alimenté à chaque fois qu'un filtre ou un tri est ajouté au `dcc.Store active_table`.



Dans cet exemple, on utilise un jeu de données comportant plusieurs modèles de voiture. Grâce à cet historique, il est possible de suivre en temps réel les filtres et tris appliqués. On peut aussi les supprimer en cliquant sur la croix à droite de chaque ligne.

Cet historique est alimenté par deux callbacks :

- `update_filter_list` : ce callback est déclenché à chaque fois qu'un filtre est ajouté. Il met à jour l'historique en ajoutant une nouvelle entrée.
- `update_sort_list` : ce callback est déclenché à chaque fois qu'un tri est ajouté. Il met à jour l'historique en ajoutant une nouvelle entrée.

Les entrées sont donc un texte en fonction de chaque filtre ou tri présent dans le `dcc.Store active_table`. Il est nécessaire, lors de l'ajout d'un filtre ou d'un tri, de prendre en charge ce dernier dans sa fonction respective.

Même principe, mais avec les callbacks `remove_filter` et `remove_sort`, qui sont déclenchés lorsque l'utilisateur clique sur la croix à droite de chaque ligne de l'historique.

Ces callbacks suppriment le filtre ou le tri correspondant du `dcc.Store active_table` et mettent à jour l'affichage du tableau en conséquence, en déclenchant le callback `update_table_with_filters_and_sort`.

8.2.5 Application des filtres et tris

Comme dit précédemment, pour appliquer un filtre ou un tri.

Le callback `update_table_with_filters_and_sort` est appelé. Il est déclenché par les changements dans le `dcc.Store active_table`, qui contient l'identifiant du fichier, ainsi que les filtres et tris appliqués. Voyons cela plus en détail.

```

1 @callback(
2     Output("table_container", "children", allow_duplicate=True),
3     Output("active_table", "data", allow_duplicate=True),
4     Input("active_table", "data"),

```

```

5     prevent_initial_call=True,
6 )
7 def update_table_with_filters_and_sort(active_table):
8     if not active_table:
9         return no_update, no_update
10
11     file_id = active_table.get("id")
12     filters = active_table.get("filters", {})
13     sort_info = active_table.get("sort", [])
14
15     if file_id is None:
16         return no_update, no_update
17
18     pickled_df = cache.get(file_id)
19     if pickled_df is None:
20         return "Cache expire ou fichier introuvable,
21 rechargement du fichier.", no_update
22
23     df = pickle.loads(pickled_df)
24
25     df_filtered = apply_filters(df, filters)
26
27     df_sorted = apply_sort(df_filtered, sort_info)
28
29     sorted_data = df_sorted.to_dict("records")
30
31     table = dash_table.DataTable(
32         id={"type": "data_viewer", "index": file_id},
33         data=sorted_data,
34         columns=[{"name": col, "id": col} for col in df_sorted.columns],
35         page_size=26,
36         editable=True,
37         style_table={
38             "margin-top": "2.5vh",
39             "overflowX": "auto",
40             "borderRadius": "10px",
41             "boxShadow": "0 4px 10px rgba(0, 0, 0, 0.1)",
42             "border": "1px solid #374151",
43             "height": "90vh",
44             "maxWidth": "70vw",
45         },
46         style_header={
47             "backgroundColor": "#1f2937",
48             "color": "white",
49             "fontWeight": "bold",

```

```

50         "fontSize": "16px",
51         "borderBottom": "2px_solid_#374151",
52         "textAlign": "center",
53         "position": "sticky",
54         "top": 0,
55         "zIndex": 1000,
56         "minWidth": "150px",
57     },
58     style_cell={
59         "color": "white",
60         "padding": "6px",
61         "textAlign": "center",
62         "fontFamily": "Segoe_UI,_sans-serif",
63         "fontSize": "12px",
64         "backgroundColor": "transparent",
65         "whiteSpace": "normal",
66         "overflow": "visible",
67         "textOverflow": "unset",
68         "minWidth": "150px",
69     },
70 )
71
72     return table, active_table

```

Depuis le début, c'est cette fonction qui initialise le tableau. Elle est appelée à chaque fois que le `dcc.Store active_table` est modifié, c'est-à-dire lorsqu'un filtre ou un tri est ajouté ou supprimé, mais aussi lorsqu'on clique sur un fichier dans le menu latéral gauche.

Elle commence par vérifier si `active_table` est vide, auquel cas elle ne fait rien. Ensuite, elle récupère l'identifiant du fichier ainsi que les filtres et tris appliqués. Si l'identifiant est `None`, elle ne fait rien non plus.

Ensuite, elle récupère les données en cache à partir de l'identifiant du fichier. Si le fichier n'est pas trouvé dans le cache, elle renvoie un message d'erreur.

Après cela, elle déséréalise les données en utilisant `pickle.loads` pour obtenir un `DataFrame pandas`.

Elle applique ensuite les filtres et tris en utilisant les fonctions `apply_filters` et `apply_sort`. Enfin, elle crée le tableau avec les données triées et filtrées.

8.2.6 Conclusion filtre et tri

Tous ces éléments permettent à l'utilisateur de manipuler les données de manière intuitive et efficace. Les filtres et tris sont appliqués en temps réel, offrant une expérience utilisateur fluide. L'historique des filtres et tris permet de garder une trace des modifications apportées aux données, facilitant ainsi la navigation et l'analyse.

8.3 Fusion de données

Le bouton de fusion de données se situe en dessous du bouton de tri. Deux fonctionnalités sont disponibles : la concaténation et la fusion (*merge*).

Une fois le bouton cliqué, de la même manière que pour les filtres et tris, un popup s'affiche, permettant de choisir les options de fusion selon la méthode souhaitée.

Dans le fichier `data_manager.py`, deux fonctions distinctes permettent de gérer la fusion et la concaténation des données.

En ce qui concerne les callbacks, on retrouve :

- un callback pour afficher le popup de fusion,
- un callback pour remplir les **dropdowns** selon les options disponibles,
- un dernier callback, avec le suffixe `add_file`, qui exécute la fonction de fusion lors du clic sur le bouton de validation et affiche le résultat sous le nom `concat_<uuid4>` ou `merge_<uuid4>` dans le `storagedata` (là où les fichiers sont stockés).

8.3.1 Exportation des données

Le dernier bouton est celui d'exportation. Une fois cliqué, un popup s'ouvre comme pour les autres boutons et demande à l'utilisateur de choisir le format d'exportation (CSV ou Excel).

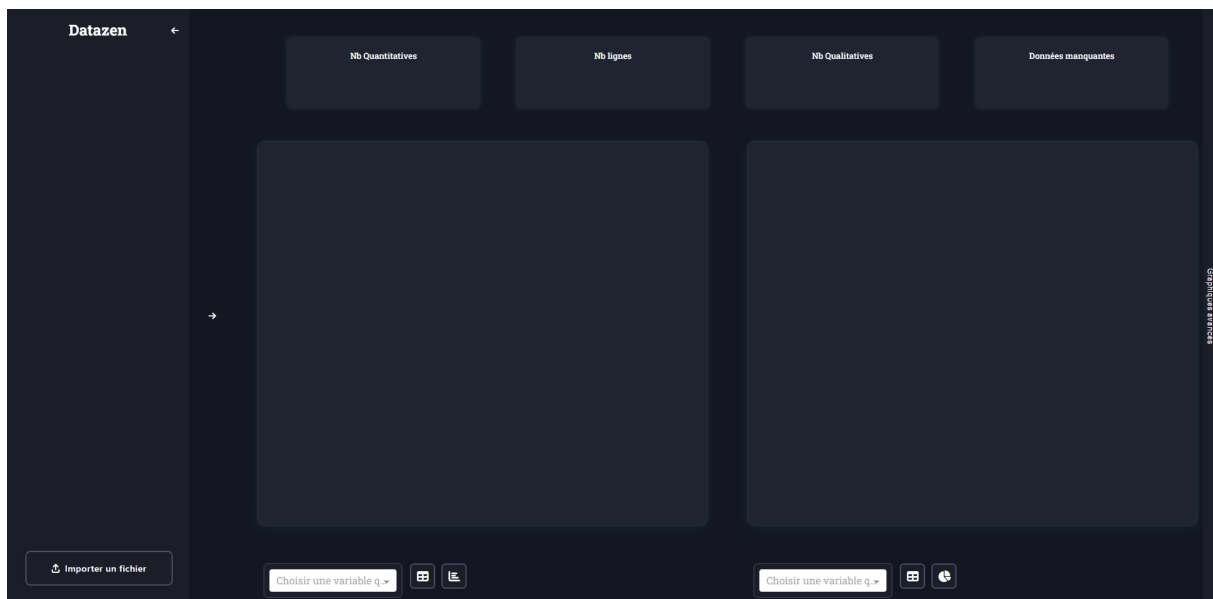
Le callback `export_data` est appelé lorsque l'utilisateur clique sur le bouton de validation d'exportation. Il récupère les données actuellement affichées dans le tableau (c'est-à-dire après application des filtres et tris), les sérialise selon le format choisi (CSV ou Excel), puis les propose en téléchargement direct dans le navigateur grâce aux composants `dcc.Download` et `send_data_frame`.

8.4 Visualization__container

Une fois les données traitées comme souhaité, vient alors la phase de visualisation. **Datazen** offre des outils simples mais efficaces pour visualiser rapidement les données.

Pour accéder à l'espace de visualisation, il suffit de cliquer sur la petite flèche située à droite de l'écran, ce qui permet de fermer l'espace de traitement des données afin de

consacrer l'intégralité de l'écran à la visualisation.



On peut observer plusieurs éléments :

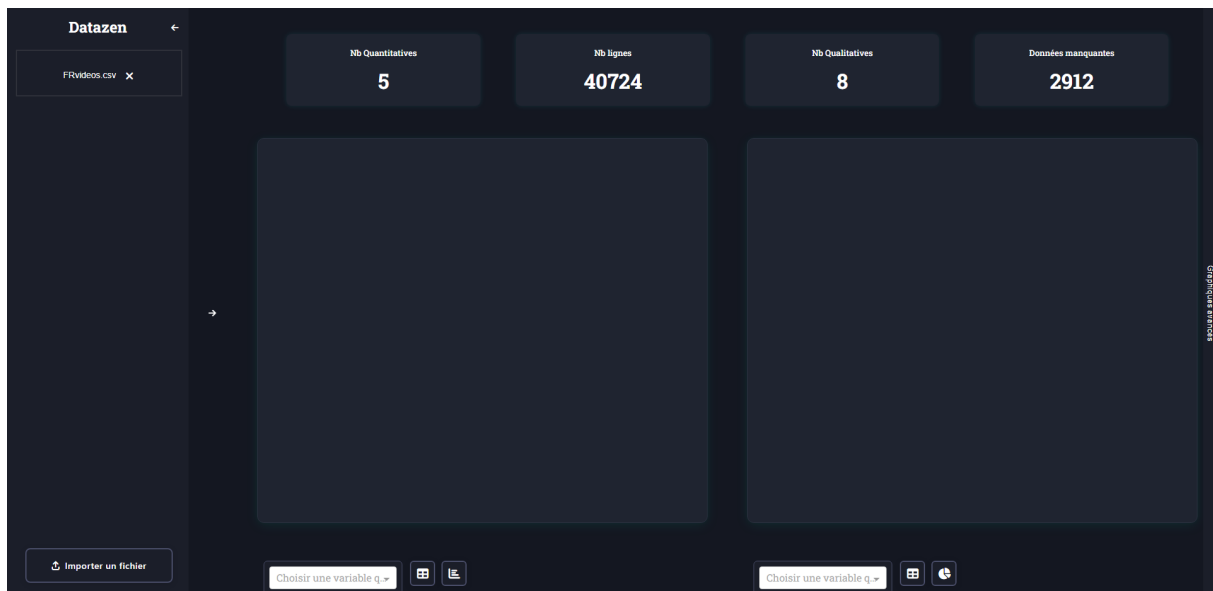
- Quatre *value boxes* permettant de visualiser rapidement des informations clés du jeu de données actif : le nombre de variables quantitatives, le nombre de variables qualitatives, le nombre de lignes, ainsi que le nombre de valeurs manquantes.
- Deux encadrés qui accueilleront les tableaux et graphiques.
- Deux menus déroulants accompagnés de deux boutons permettant de choisir respectivement l'affichage de tableaux ou de graphiques.

Implicitement, la partie gauche est dédiée à la visualisation des variables quantitatives, tandis que la partie droite est réservée aux variables qualitatives.

Dès qu'un jeu de données est sélectionné et que le `dcc.Store active_table` est modifié, que ce soit par initialisation ou par application de filtres, les callbacks liés aux menus déroulants mettent à jour leurs options en remplissant les colonnes quantitatives à gauche et qualitatives à droite.

Les *value boxes* sont également mises à jour en fonction des modifications apportées au jeu de données actif. Ces *value boxes* sont gérées par des callbacks dédiés et des fonctions de calcul situées dans le fichier `data_manager.py`.

Ainsi, le suivi constant de l'état de `active_table` permet de mettre à jour les valeurs affichées dans les *value boxes*, mais aussi de remplir dynamiquement les menus déroulants pour les graphiques et tableaux.



A noter que chaque composant, chaque tableau, etc., réutilise les fonctions de filtre et de tri (`apply_filters` et `apply_sort`) pour appliquer les filtres et tris sur les données avant de les afficher. Le callback `update_table_with_filters_and_sort` est uniquement dédié à l'espace de traitement des données, autrement dit, il ne gère que le tableau affiché.

8.5 Tableaux et graphiques

Une fois les données traitées, il est possible de les visualiser sous forme de tableaux ou de graphiques.

Pour cela, il suffit de sélectionner une colonne quantitative ou qualitative dans les menus déroulants situés en bas, puis de cliquer sur le bouton correspondant (tableau ou graphique).

Si le bouton *tableau* est cliqué, un tableau de synthèse s'affiche : pour les variables quantitatives, il présente les statistiques descriptives (nombre, moyenne, médiane, écart-type, minimum, Q1, Q2, Q3, maximum ainsi que le nombre de valeurs manquantes) ; pour les variables qualitatives, la répartition des catégories est affichée.

De plus, un tableau de corrélation est affiché pour les variables quantitatives, permettant de visualiser les relations entre elles.

Pour les graphiques, il suffit de cliquer sur le bouton dédié. Un menu déroulant apparaît alors dans la zone graphique, permettant de choisir le type de graphique souhaité.

Voici les graphiques disponibles pour les variables quantitatives :

- Histogramme
- Boîte à moustaches
- Graphique violon (*violin plot*)

Voici les graphiques disponibles pour les variables qualitatives :

- Diagramme circulaire
- Diagramme en barres

Attention, il se peut que les graphiques ne s'actualisent pas automatiquement lors du changement de jeu de données ; il faut alors recommencer la manipulation.

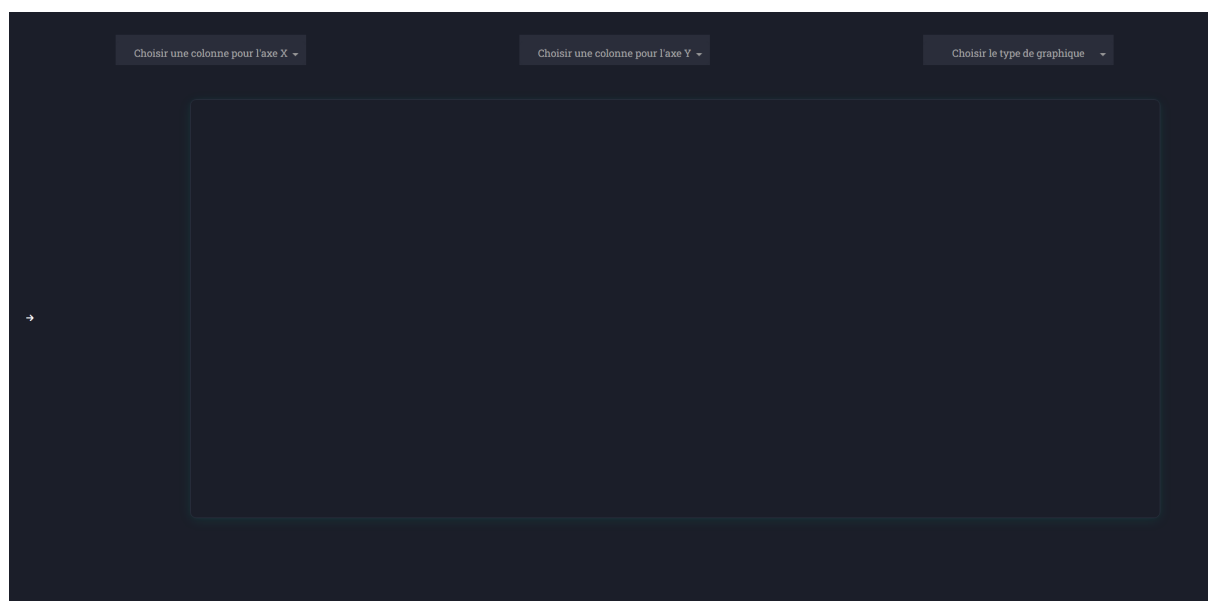
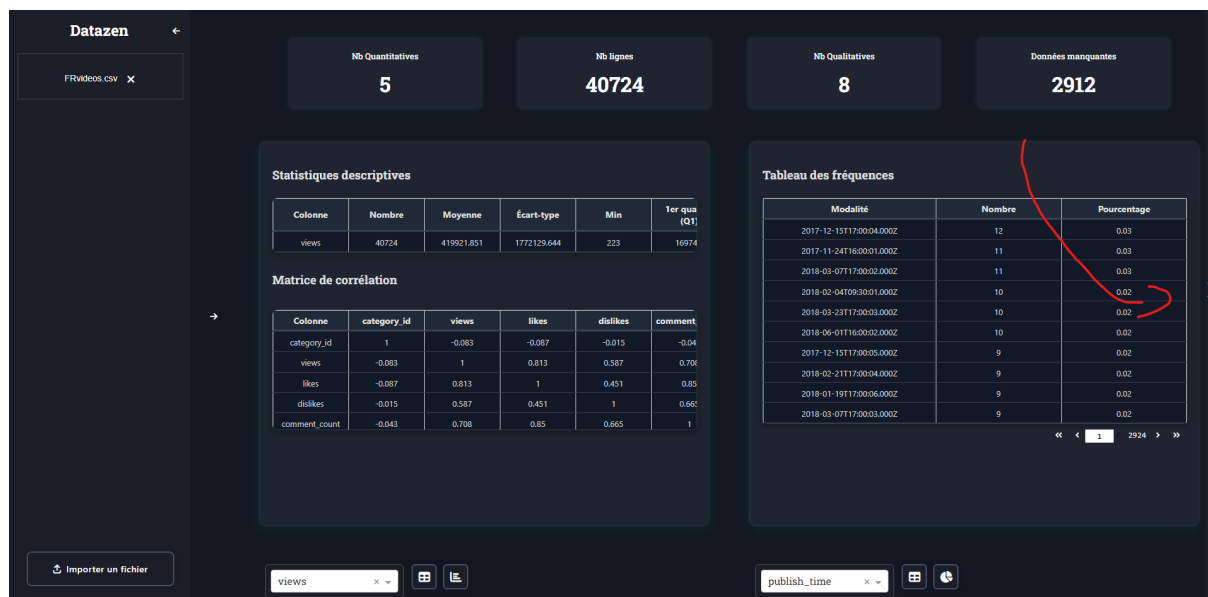
Les callbacks liés à cette section sont les suivants :

- `update_valuebox_quantitative_count` : Met à jour le compteur des variables quantitatives affiché dans `valuebox_quantitative_count`.
- `update_valuebox_qualitative_count` : Met à jour le compteur des variables qualitatives affiché dans `valuebox_qualitative_count`.
- `update_nrow` : Met à jour le nombre de lignes des données filtrées affiché dans `valuebox_nrow_data_count`.
- `update_missing_count` : Met à jour le nombre de valeurs manquantes affiché dans `valuebox_missing_count`.
- `update_dropdowns` : Met à jour les options des dropdowns pour variables quantitatives et qualitatives.
- `update_quantitative_tables` : Génère les tableaux statistiques descriptifs et matrice de corrélation pour la variable quantitative sélectionnée.
- `update_qualitative_tables` : Génère le tableau des fréquences pour la variable qualitative sélectionnée.
- `show_quanti_graph_dropdown_section` : Affiche le dropdown pour choisir le type de graphique quantitatif.
- `update_quanti_graph` : Met à jour le graphique quantitatif selon le type sélectionné (histogramme, boxplot, violon).
- `show_quali_graph_dropdown_section` : Affiche le dropdown pour choisir le type de graphique qualitatif.
- `update_quali_graph` : Met à jour le graphique qualitatif selon le type sélectionné (camembert, bar chart).

8.5.1 Graphique avancé

Une fonctionnalité pour générer des graphiques avancés est également disponible. Elle permet de créer des représentations plus complexes en combinant deux variables, qu'elles soient quantitatives ou qualitatives.

Pour cela, il suffit de cliquer sur le bouton *Graphique avancé* situé à droite de l'espace de visualisation.



On peut alors voir apparaître une fenêtre dédiée à la création de ces graphiques avancés. Il est possible de choisir une variable pour chacun des deux dropdowns, ainsi que le type de graphique : nuage de points, courbe de régression, boxplot ou violinplot.

Les dropdowns sont remplis dynamiquement, toujours en fonction des modifications de `active_table`. Celui de gauche correspond à l'abscisse (X) et celui de droite à l'ordonnée (Y).

Les callbacks liés à cette fonctionnalité sont les suivants :

- **toggle_advanced_graphs_button** : Active ou désactive l’affichage du panneau des graphiques avancés en modifiant la classe CSS et le contenu du bouton (texte ou icône).
- **update_graph_dropdowns** : Récupère le DataFrame en cache grâce à un identifiant, applique les filtres éventuels, et met à jour les options des menus déroulants X et Y avec les colonnes filtrées.
- **update_advanced_graph** : Gère la création et l’affichage du graphique demandé (scatter, régression linéaire, boxplot, violinplot) selon les colonnes X et Y sélectionnées et le type choisi. Gère aussi les erreurs (cache expiré, colonnes manquantes, type non supporté, erreurs lors de la génération) en affichant un message approprié.

Il est toujours possible d’exporter les graphiques grace à plotly, en cliquant sur le bouton d’exportation situé en haut à droite de chaque graphique.

8.6 Conclusion

L’interface utilisateur de **Datazen** est conçue pour être intuitive et facile à utiliser. Elle permet aux utilisateurs de charger, traiter, visualiser et exporter des données de manière fluide. Les fonctionnalités de filtrage et de tri offrent une flexibilité importante pour explorer les données, tandis que les outils de visualisation permettent de créer des graphiques et des tableaux pertinents pour l’analyse.

9 Discussion et optimisation

L’application remplit bien ses fonctions de base et constitue une bonne alternative à des outils comme **Excel**, **Power BI**, etc. Cependant, plusieurs améliorations restent envisageables.

Tout d’abord, il serait important d’améliorer la gestion du responsive sur les petits écrans. En l’état, l’interface est optimisée principalement pour des écrans de taille moyenne à grande, ce qui limite son usage sur tablette ou mobile.

Ensuite, l’ajout de filtres combinables avec une logique « ou » (au lieu de l’actuelle logique « et ») permettrait de faire des analyses plus souples et plus poussées. On pourrait également imaginer des filtres plus avancés comme des **group_by**, pour effectuer des regroupements dynamiques sur certaines colonnes.

Concernant l’importation de données, il serait intéressant de supporter davantage de formats de fichiers. De plus, il faudrait dissocier la vérification d’erreurs de la fonction d’importation actuelle, qui peut provoquer des lenteurs suite à leur appel.

Une autre piste d’amélioration importante concerne la sauvegarde du contexte de

travail. Actuellement, un simple rafraîchissement de la page entraîne la perte totale de l'analyse en cours. Il serait donc pertinent d'intégrer un système de sauvegarde automatique ou manuelle permettant de reprendre une session ultérieurement.

Enfin, dans la partie visualisation, l'ajout de nouveaux types de graphiques serait un vrai plus. Il serait aussi intéressant de permettre à l'utilisateur de personnaliser certains paramètres graphiques tels que la taille ou la couleur des points, la palette, l'opacité, etc.

Toutes ces pistes ouvrent la voie à une version plus avancée et robuste de **Datazen**, tout en gardant sa simplicité d'utilisation.

10 Bilan personnel

Ce projet m'a permis de mettre en pratique les compétences acquises lors de mon cursus universitaire. Il m'a également confronté à la réalité du développement d'une application. J'ai compris l'importance de bien prévoir l'architecture du code, de documenter les fonctions et de tester régulièrement les fonctionnalités. Une bonne préparation en amont peut éviter des problèmes, comme celui du changement du système de stockage des données, qui a pris du temps car il a été codé tard dans le projet.

Pour finir, cette application, bien que simple, offre des outils utiles que je serai amené à réutiliser à l'avenir, ainsi qu'à l'améliorer.

11 Conclusion

Ce mémoire a présenté la conception et le développement d'une application modulaire en Python, visant à offrir une solution simple et accessible pour le traitement et la visualisation de données tabulaires. Face aux limites des outils classiques comme Excel, notamment en termes de prise en main et de performance, l'application propose une alternative légère, intuitive et évolutive, adaptée aussi bien aux débutants qu'aux utilisateurs plus avancés.

L'utilisation de technologies approuvées telles que **Dash**, **Pandas** et **Plotly** a permis de concilier interactivité, efficacité et simplicité, tout en garantissant une architecture propre et maintenable. À terme, ce projet pourra être enrichi par l'intégration de nouvelles fonctionnalités, toujours dans l'objectif de faciliter la manipulation des données sans nécessiter de compétences techniques approfondies.

12 Annexes

12.1 Guide utilisateur

Ce chapitre est dédié à la présentation de l'interface utilisateur de **Datazen**. Pour rendre le guide plus interactif, nous allons utiliser des captures d'écran pour illustrer les différentes fonctionnalités et les appliquer à des cas d'usage concrets.

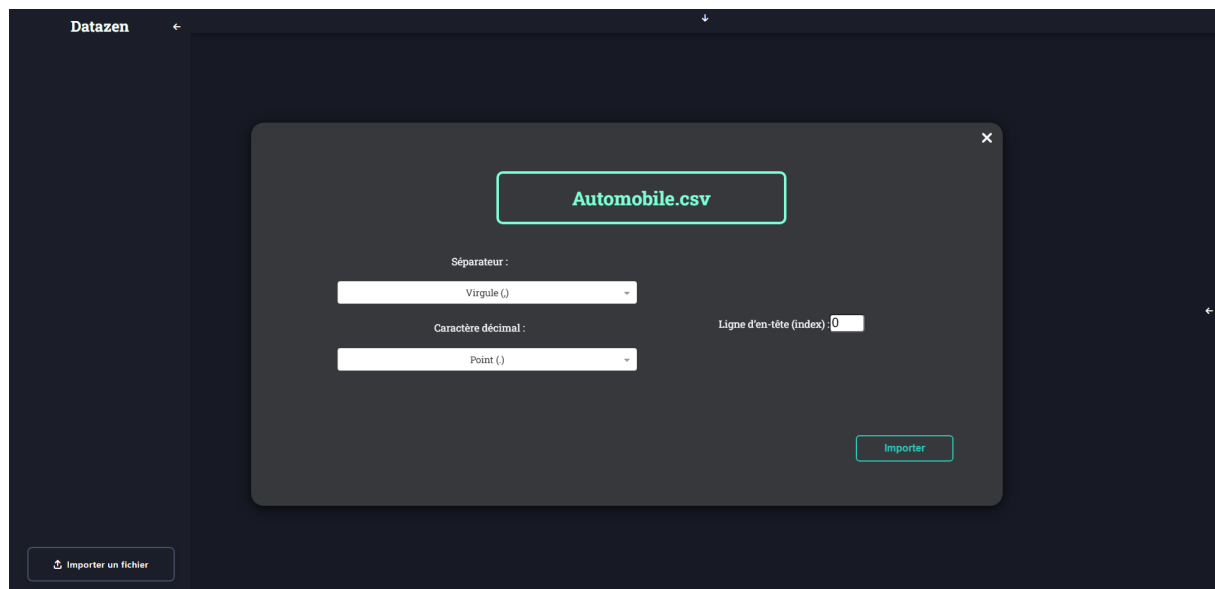
Tout les jeux de données utilisés dans ce chapitre sont disponibles dans le dossier **data** du dépôt **GitHub**.

12.1.1 Cas données : automobile

Nous allons importer un jeu de données contenant des informations sur les voitures (**automobile.csv**). Ce jeu de données est idéal pour illustrer les fonctionnalités de **Datazen**, notamment les filtres et tri.

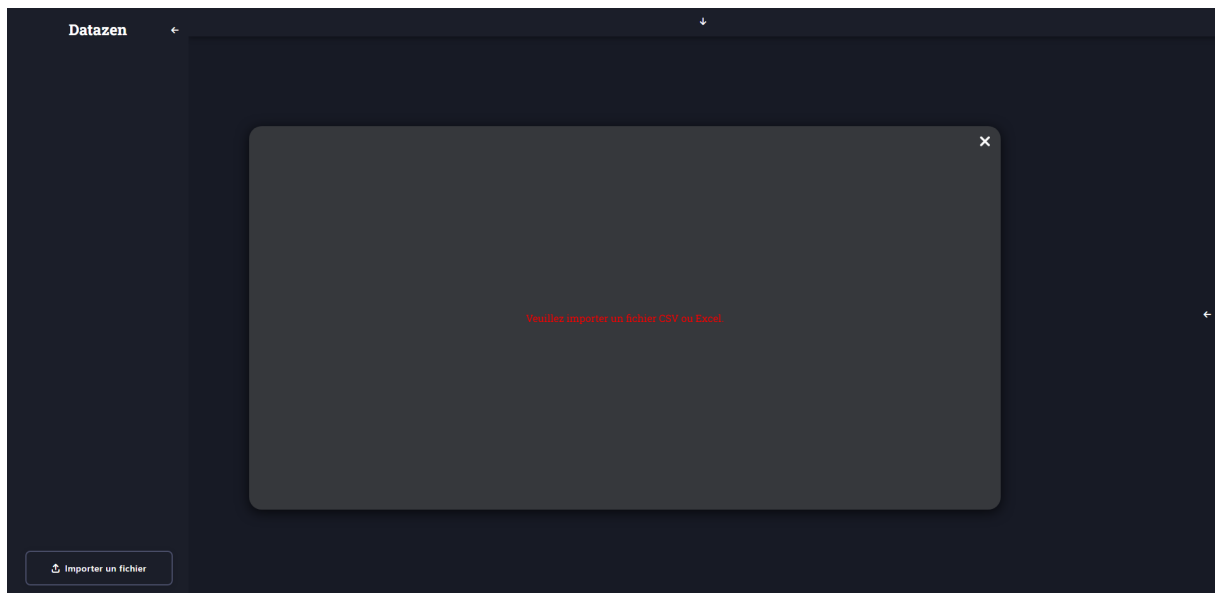
Pour ce faire, comme expliqué précédemment, il suffit de cliquer sur le bouton *Importer un fichier* dans le menu latéral gauche. Une fenêtre s'ouvre alors, permettant de sélectionner le fichier à importer.

Une fois le fichier sélectionné, le popup d'importation CSV s'ouvre, nous permettant de choisir les options d'importation.



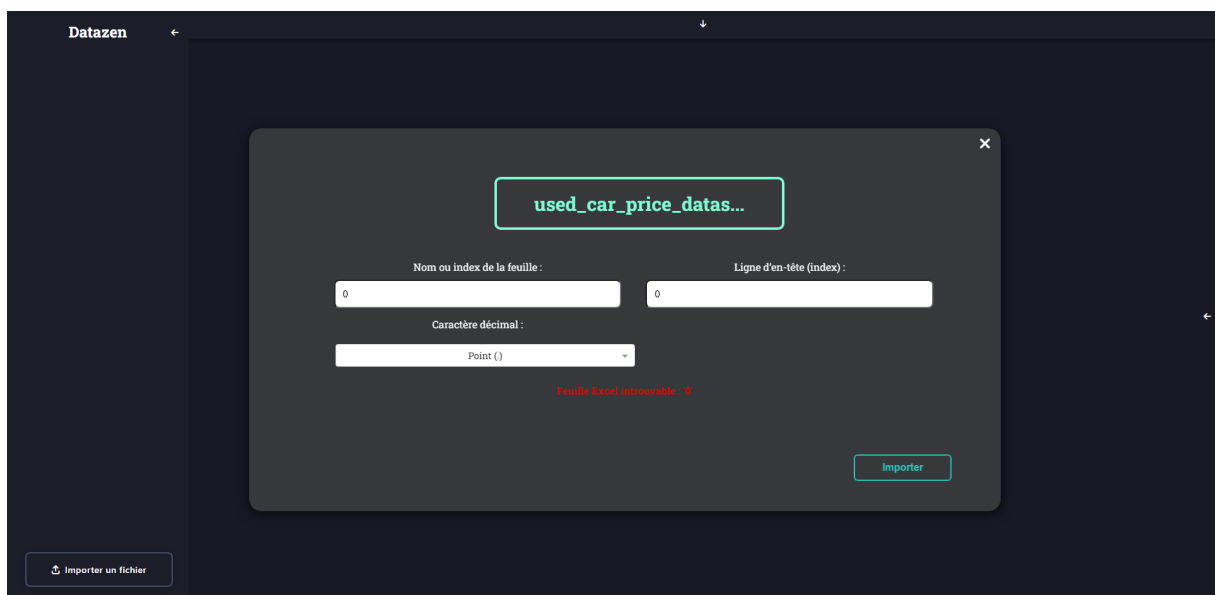
Il nous suffit de choisir les options correspondant à notre jeu de données, puis de cliquer sur *Valider*.

En cas d'erreur, nous aurons cette fenêtre :



Il suffit alors d'attendre 5 secondes pour effectuer une nouvelle tentative ou de fermer le popup.

L'importation Excel fonctionne de la même manière, mais avec des options spécifiques à ce format de fichier :



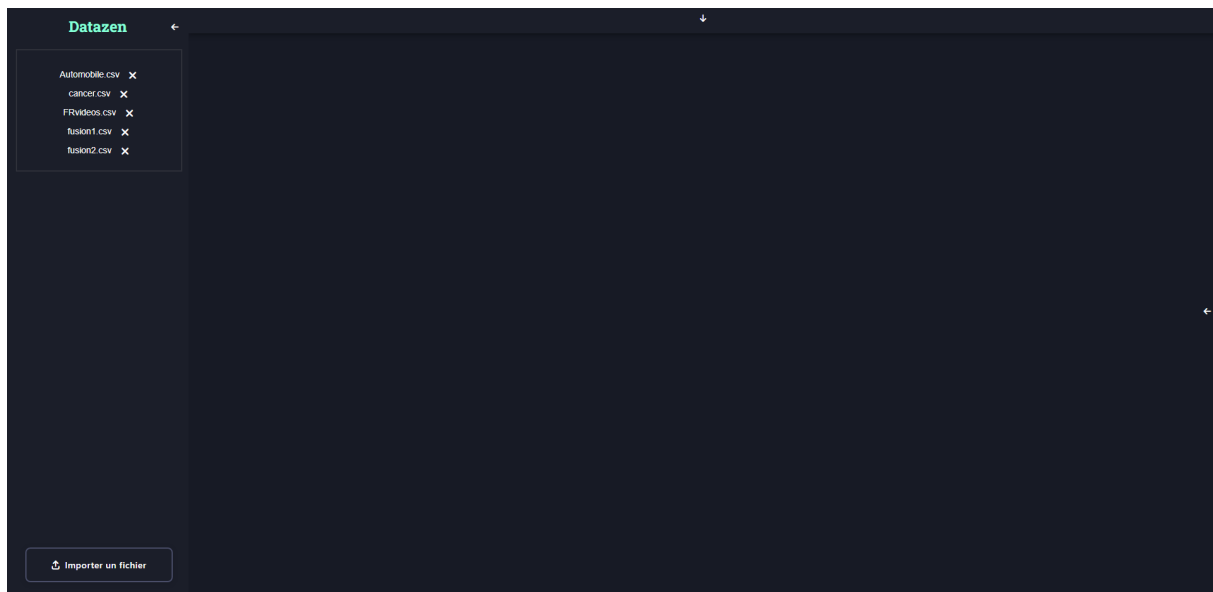
On peut voir que si la feuille choisie n'existe pas, un message d'erreur s'affiche.

Nous allons maintenant importer tous les jeux de données nécessaires à la suite de ce guide, à savoir :

- `automobile.csv` : informations sur les voitures
- `FRvideos.csv` : données de vidéos YouTube en France
- `cancer.csv` : informations sur des données comportementales et médicales sur des patients afin de prédire le risque d'avoir un cancer du poumon
- `fusion1.csv` : données pour la fusion

— **fusion2.csv** : données pour la fusion

Une fois les fichiers importés, nous pouvons les voir s'ajouter un à un dans le menu latéral gauche.



Pour afficher les données, il suffit de cliquer sur le nom du fichier dans le menu latéral gauche. Le tableau s'affiche alors dans la partie centrale de l'interface, avec les données du fichier sélectionné.

Nous pouvons aussi fermer le menu latéral gauche en cliquant sur la petite flèche située en haut à droite, ce qui permet de consacrer l'intégralité de l'écran à la visualisation des données.

The screenshot shows the Datan interface with a table of data. The table has 9 columns: 'name', 'mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model_year', and 'origin'. The data is sorted by 'name' in ascending order. The table contains 32 rows of data, including cars like 'chevrolet chevelle malibu', 'buick skylark 320', 'plymouth satellite', 'amc rebel sst', 'ford torino', 'ford galaxie 500', 'chevrolet impala', 'plymouth fury iii', 'pontiac catalina', 'amc ambassador dpl', 'dodge challenger se', 'plymouth 'cuda 340', 'chevrolet monte carlo', 'buick estate wagon (sw)', 'toyota corona mark ii', 'plymouth duster', 'amc hornet', 'ford maverick', 'datsun pl510', 'volkswagen 1131 deluxe sedan', 'peugeot 504', 'audi 100 ls', 'saab 99e', 'bmw 2002', 'amc gremlin', and 'ford f250'. The table is displayed in a dark theme with white text. On the right side of the table, there are several icons for filtering and sorting the data.

name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
chevrolet chevelle malibu	18	8	307	130	3504	12	70	usa
buick skylark 320	15	8	350	165	3693	11.5	70	usa
plymouth satellite	18	8	318	150	3436	11	70	usa
amc rebel sst	16	8	304	150	3433	12	70	usa
ford torino	17	8	302	140	3449	10.5	70	usa
ford galaxie 500	15	8	429	198	4341	10	70	usa
chevrolet impala	14	8	454	220	4354	9	70	usa
plymouth fury iii	14	8	440	215	4312	8.5	70	usa
pontiac catalina	14	8	455	225	4425	10	70	usa
amc ambassador dpl	15	8	390	190	3850	8.5	70	usa
dodge challenger se	15	8	383	170	3563	10	70	usa
plymouth 'cuda 340	14	8	340	160	3609	8	70	usa
chevrolet monte carlo	15	8	400	150	3761	9.5	70	usa
buick estate wagon (sw)	14	8	455	225	3086	10	70	usa
toyota corona mark ii	24	4	113	95	2372	15	70	japan
plymouth duster	22	6	198	95	2833	15.5	70	usa
amc hornet	18	6	199	97	2774	15.5	70	usa
ford maverick	21	6	200	85	2587	16	70	usa
datsun pl510	27	4	97	88	2130	14.5	70	japan
volkswagen 1131 deluxe sedan	26	4	97	46	1835	20.5	70	europa
peugeot 504	25	4	110	87	2672	17.5	70	europa
audi 100 ls	24	4	107	90	2430	14.5	70	europa
saab 99e	25	4	104	95	2275	17.5	70	europa
bmw 2002	26	4	121	113	2254	12.5	70	europa
amc gremlin	21	6	199	90	2648	15	70	usa
ford f250	10	8	360	215	4615	14	70	usa

Supposons que maintenant, nous voulons consacrer notre analyse aux voitures de la marque Toyota avec une puissance comprise entre 90 et 150 chevaux. Il nous suffit d'appliquer les filtres suivants :

-
- recherche (in) : toyota dans "name"
 - comparaison : puissance supérieure ou égale à 90 dans "horsepower"
 - comparaison : puissance inférieure ou égale à 150 dans "horsepower"

À ce jour, l'application ne comporte pas de logique "ou", il n'est donc pas possible d'étudier à la fois les Toyota et les voitures de la marque Ford par exemple en même temps.

Nous allons donc appliquer ces filtres un à un, en passant notre souris sur le bouton *Filtres* pour faire apparaître la liste des filtres disponibles.

Commençons par le filtre de recherche, en cliquant sur le bouton *Recherche (in)* :

name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
chevrolet chevelle malibu	18	8	307	130	3504	12	70	usa
buick skylark 320	15	8	350	165	3693	11.5	70	usa
plymouth satellite	18	8	318	150	3436	11	70	usa
amc rebel sst	16	8	304	150	3433	12	70	usa
ford torino	17	8	302	140	3449	10.5	70	usa
ford galaxie 500	15	8	429	198	4341	10	70	usa
chevrolet impala	14	8	454	220	4354	9	70	usa
plymouth fury ii	14	8	440	215	4312	8.5	70	usa
pontiac catalina	14	8	455	225	4425	10	70	usa
amc ambassador dpl	15	8	390	190	3950	8.5	70	usa
dodge challenger se	15	8	390	190	3943	10	70	usa
plymouth cuda 340	14	8	340	160	3609	8	70	usa
chevrolet monte carlo	14	8	455	225	3686	10	70	usa
buick estate wagon (sw)	14	8	455	225	3686	10	70	usa
toyota corona mark ii	22	6	198	95	2822	15.5	70	japan
plymouth duster	18	6	199	97	2774	15.5	70	usa
amc hornet	21	6	200	85	2567	16	70	usa
ford maverick	22	4	97	88	2130	14.5	70	japan
datsun pl510	26	4	97	46	1815	15	70	europa
volkswagen 1171 deluxe sedan	25	4	110	87	2612	17.5	70	europa
peugeot 504	24	4	107	90	2430	14.5	70	europa
saab 99e	25	4	104	95	2375	17.5	70	europa
bmw 2002	26	4	121	113	2234	12.5	70	europa
amc gremlin	21	6	199	90	2648	15	70	usa
ford f250	10	8	360	215	4615	14	70	usa

À noter que cette fonction ne prend pas en compte la casse, c'est-à-dire que si l'on cherche "Toyota" ou "toyota", cela fonctionnera de la même manière.

Une fois le bouton *Valider* cliqué, le filtre est appliqué et le tableau est mis à jour. On peut voir que seules les voitures de la marque Toyota sont affichées.

De plus, le filtre sera affiché dans l'historique des filtres appliqués, accessible en cliquant sur la flèche en haut du tableau.

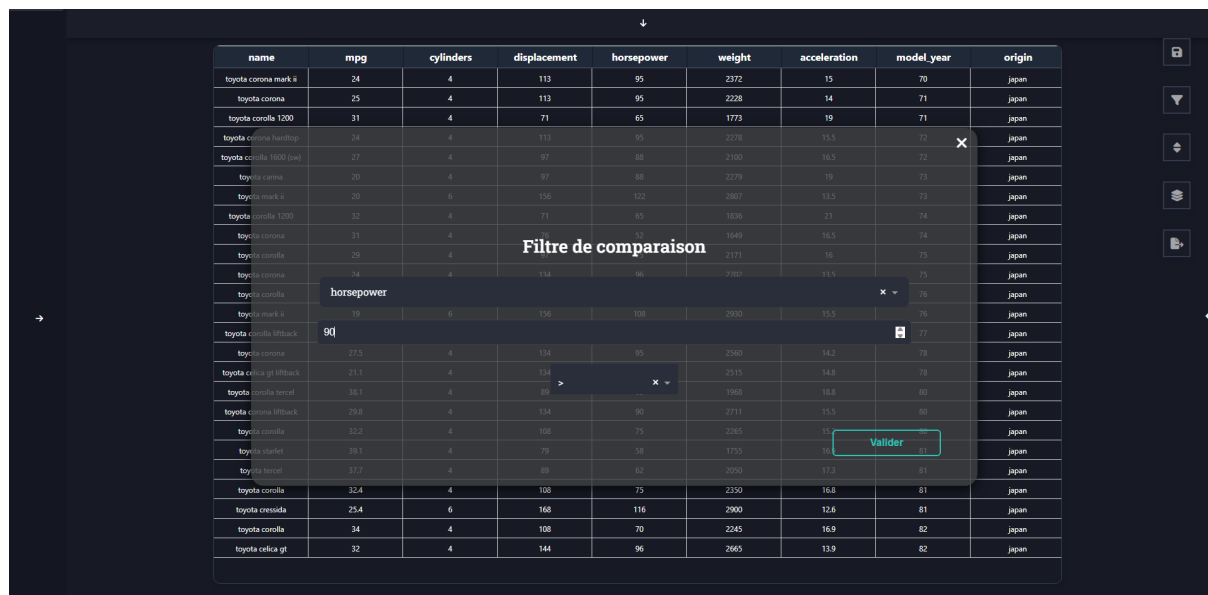
name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
toyota corona mark ii	24	4	113	95	2372	15	70	japan
toyota corona	25	4	113	95	2228	14	71	japan
toyota corolla 1200	31	4	71	65	1773	19	71	japan
toyota corona hardtop	24	4	113	95	2278	15.5	72	japan
toyota corolla 1600 (sw)	27	4	97	88	2100	16.5	72	japan
toyota carina	20	4	97	88	2279	19	73	japan
toyota mark ii	20	6	156	122	2807	13.5	73	japan
toyota corolla 1300	32	4	71	65	1836	21	74	japan
toyota corona	31	4	76	52	1649	16.5	74	japan
toyota corolla	29	4	97	75	2171	16	75	japan
toyota corona	24	4	134	96	2702	13.5	75	japan
toyota corolla	28	4	97	75	2155	16.4	76	japan
toyota mark ii	19	6	156	108	2930	15.5	76	japan
toyota corolla liftback	26	4	97	75	2265	18.2	77	japan
toyota corona	27.5	4	134	95	2560	14.2	78	japan
toyota celica gt liftback	21.1	4	134	95	2515	14.8	78	japan
toyota corolla tercel	38.1	4	89	60	1968	18.8	80	japan
toyota corona liftback	29.8	4	134	90	2711	15.5	80	japan
toyota corolla	32.2	4	108	75	2265	15.2	80	japan
toyota starlet	39.1	4	79	58	1755	16.9	81	japan
toyota tercel	37.7	4	89	62	2050	17.3	81	japan
toyota corolla	32.4	4	108	75	2350	16.8	81	japan
toyota cressida	25.4	6	168	116	2900	12.6	81	japan
toyota corolla	34	4	108	70	2245	16.9	82	japan
toyota celica gt	32	4	144	96	2665	13.9	82	japan

Voici les Toyota présentes dans ce jeu de données.

Si l'on va directement dans la zone de visualisation, on peut voir qu'il y en a précisément 25.

Maintenant, appliquons le filtre de comparaison pour la puissance supérieure ou égale à 90 chevaux.

Nous cliquons sur le bouton *Comparaison* :



name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
toyota corona mark ii	24	4	113	95	2372	15	70	japan
toyota corona	25	4	113	95	2228	14	71	japan
toyota corolla 1200	31	4	71	65	1773	19	71	japan
toyota corona hardtop	24	4	113	95	2378	15.5	72	japan
toyota corolla 1600 (sed)	27	4	97	88	2180	16.5	72	japan
toyota corolla	26	4	97	88	2279	19	73	japan
toyota mark ii	20	6	156	122	3807	13.5	73	japan
toyota corolla 1200	32	4	71	65	1838	21	74	japan
toyota corona	31	4	71	67	1849	16.5	74	japan
toyota corolla	29	4	71	67	2171	16	75	japan
toyota corona	34	4	134	96	2782	13.9	75	japan
toyota corolla	30	4	134	96	2782	13.9	75	japan
toyota mark ii	19	6	156	108	2900	15.5	76	japan
toyota corolla liftback	30	4	134	95	2560	14.2	76	japan
toyota corona	27.5	4	134	95	2560	14.2	76	japan
toyota celica gt liftback	21.1	4	134	95	2515	14.8	78	japan
toyota corolla liftback	38.1	4	88	62	1988	18.8	80	japan
toyota corona liftback	29.8	4	134	90	2711	15.5	80	japan
toyota corolla	32.2	4	108	75	2385	16.9	81	japan
toyota starlet	38.1	4	79	58	1725	19	81	japan
toyota tercel	37.7	4	88	62	2050	17.3	81	japan
toyota corolla	32.4	4	108	75	2350	16.8	81	japan
toyota cressida	25.4	6	168	116	2900	12.6	81	japan
toyota corolla	34	4	108	70	2245	16.9	82	japan
toyota celica gt	32	4	144	96	2665	13.9	82	japan

Nous choisissons la colonne "horsepower", l'opérateur "supérieur" et nous saisissons la valeur 90.

Une fois le bouton *Valider* cliqué, le filtre est appliqué et le tableau est mis à jour. On peut voir que seules les voitures de la marque Toyota avec une puissance supérieure ou égale à 90 chevaux sont affichées.

Il suffit maintenant de faire la même chose pour la puissance inférieure ou égale à 150 chevaux.

Nous cliquons à nouveau sur le bouton *Comparaison* :

The screenshot shows a web application interface for comparing cars. A table lists various Toyota models with their specifications. A modal titled 'Filtre de comparaison' is open, allowing users to filter the table. The modal has a dropdown menu for 'horsepower' with the value '150' selected. Below the dropdown is a radio button for '≤' (less than or equal to). A 'Valider' button is at the bottom right of the modal.

name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
toyota corona mark ii	24	4	113	95	2372	15	70	japan
toyota corona	25	4	113	95	2228	14	71	japan
toyota corona hardtop	24	4	113	95	2278	15.5	72	japan
toyota mark ii	20	6	156	122	2887	13.5	73	japan
toyota corona	24	4	134	96	2702	13.5	75	japan
toyota mark ii	19	6	156	108	2930	15.5	76	japan
toyota corona	27.5	4	134	95	2580	14.2	78	japan
toyota celica gt liftback	21.1	4	134	95	2515	14.8	78	japan
toyota creosida	25.4	6	116	116	2900	12.6	81	japan
toyota celica gt	22	4	134	95	2695	13.9	82	japan

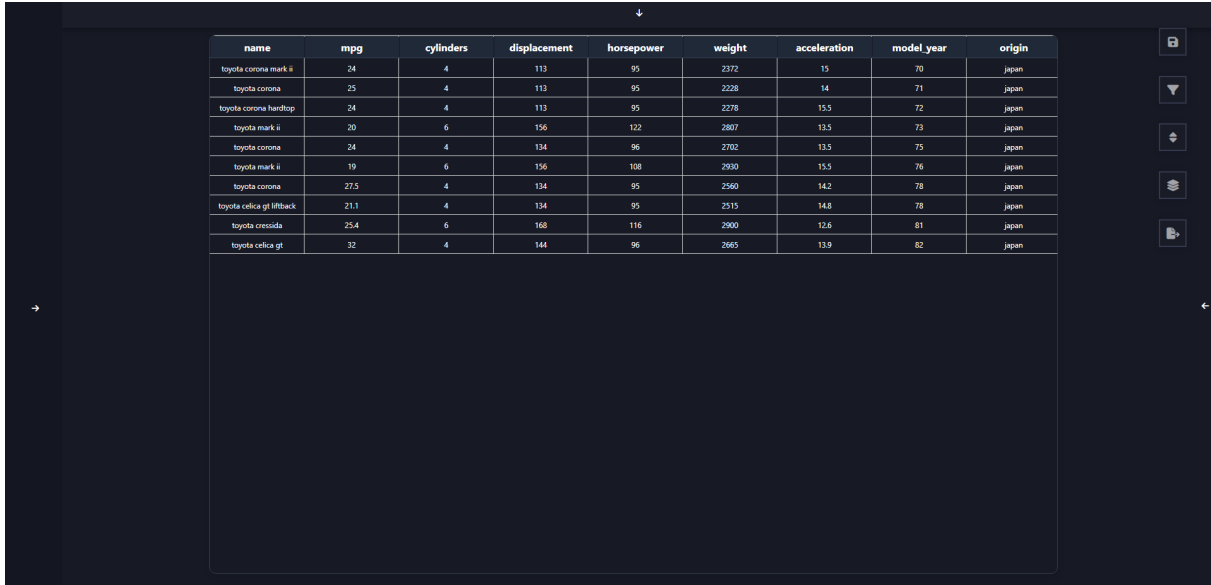
Nous choisissons la colonne "horsepower", l'opérateur "inférieur ou égal" et nous saisissons la valeur 150.

Une fois le bouton *Valider* cliqué, le filtre est appliqué et le tableau est mis à jour. On peut voir que seules les voitures de la marque Toyota avec une puissance comprise entre 90 et 150 chevaux sont affichées.

Si nous déroulons l'historique des filtres appliqués, nous pouvons voir que les trois filtres sont présents.

The screenshot shows the 'Historique' (History) section of the application. It has two main panels: 'Filtres actifs' (Active filters) and 'Tri actif' (Active sorting). The 'Filtres actifs' panel lists three filters: 'name contient 'toyota'', 'horsepower plus grand que 90', and 'horsepower plus petit ou égal à 150'. Each filter has a red 'x' icon to remove it. The 'Tri actif' panel is currently empty.

Une fois filtré, voici nos données :



name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
toyota corona mark ii	24	4	113	95	2372	15	70	japan
toyota corona	25	4	113	95	2228	14	71	japan
toyota corona hardtop	24	4	113	95	2278	15.5	72	japan
toyota mark ii	20	6	156	122	2807	13.5	73	japan
toyota corona	24	4	134	96	2702	13.5	75	japan
toyota mark ii	19	6	156	108	2930	15.5	76	japan
toyota corona	27.5	4	134	95	2560	14.2	78	japan
toyota celica gt liftback	21.1	4	134	95	2515	14.8	78	japan
toyota cressida	25.4	6	168	116	3000	12.6	81	japan
toyota celica gt	32	4	144	96	2665	13.9	82	japan

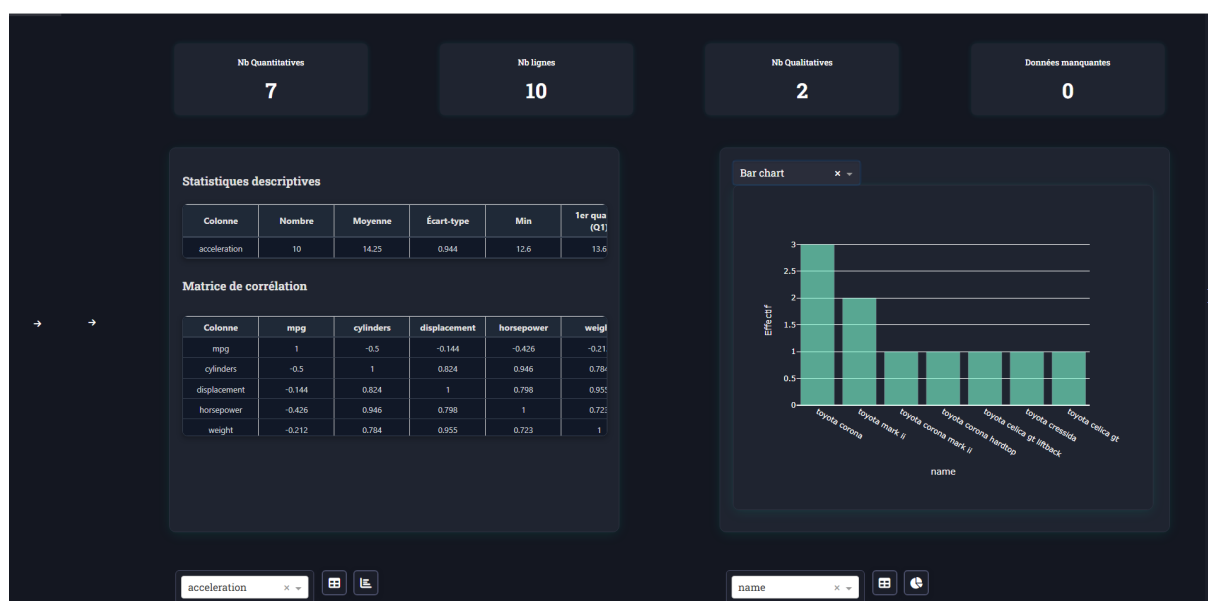
En se rendant dans la partie de visualisation, on peut voir que le nombre de lignes est de 10, le nombre de variables quantitatives est de 7, le nombre de variables qualitatives est de 2 et le nombre de valeurs manquantes est de 0.

Il est aussi possible maintenant d'étudier chaque colonne indépendamment. Pour l'exemple, nous allons regarder l'accélération des voitures. On utilise alors la partie gauche de l'espace de visualisation, dédiée aux variables quantitatives.

Et pour une variable qualitative, on utilise la partie droite, dédiée aux variables qualitatives. Nous regarderons ici la marque des voitures.

Comme ces deux parties sont dissociées, il est possible d'afficher des tableaux dans une partie et des graphiques dans une autre.

Voici le résultat. Pour rappel, on choisit d'abord la colonne, puis le bouton souhaité. À gauche, le tableau ; à droite, le graphique :



Pour la partie quantitative, on observe un tableau statistique regroupant les statistiques de base citées précédemment, à savoir : le nombre de valeurs, la moyenne, la médiane, l'écart-type, le minimum, Q1, Q2, Q3, le maximum et les valeurs manquantes. Il est possible de scroller horizontalement pour voir toutes les colonnes.

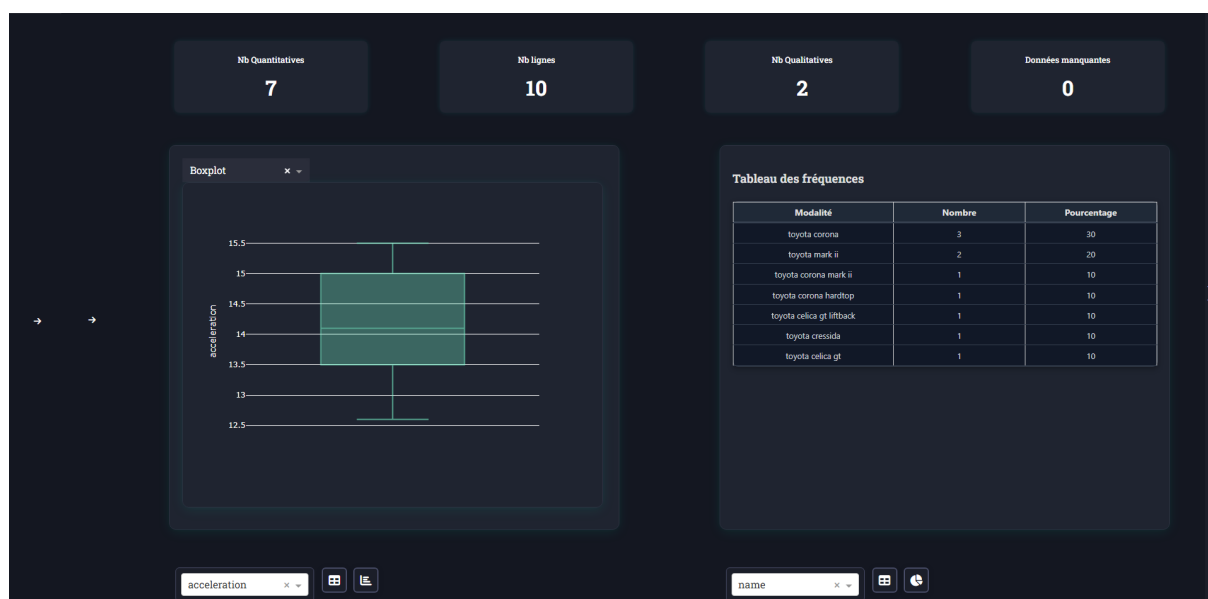
En dessous, une matrice de corrélation entre les variables quantitatives est affichée.

Nous pouvons voir ici, par exemple, que la cylindrée de la voiture est extrêmement positivement corrélée avec la puissance, ce qui est logique.

Pour la partie qualitative, j'ai fait le choix d'afficher un graphique en barres pour représenter les marques de voitures. On peut donc connaître précisément, avec ce graphique, la répartition des marques de voitures dans le jeu de données de la marque Toyota.

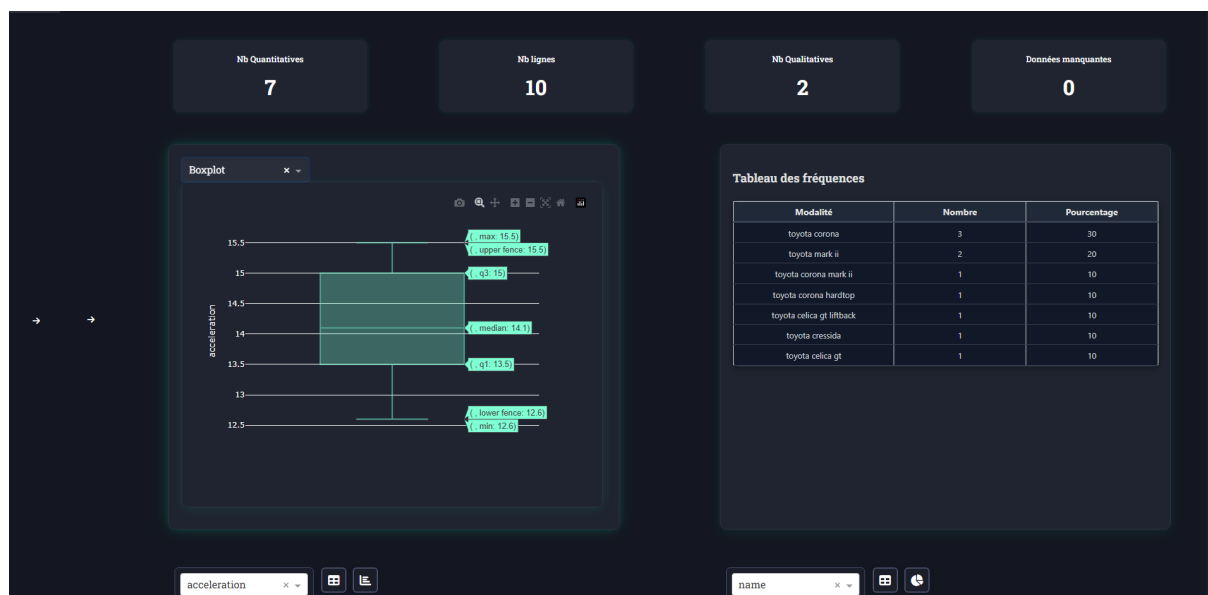
On peut aussi faire l'inverse, et afficher un graphique pour la partie quantitative et un tableau pour la partie qualitative.

Voici le graphique pour la partie quantitative, avec un histogramme de l'accélération des voitures :



Le barplot pour l'accélération ne montre pas de valeurs aberrantes, mais si c'était le cas, il serait possible de les traiter en cliquant sur le bouton *Valeurs aberrantes* dans la partie traitement des données.

En passant la souris sur le graphique, il est possible de voir des informations supplémentaires sur le graphique.



Pour revenir au tableau, on a pu noter une accélération moyenne de 14,25 secondes.

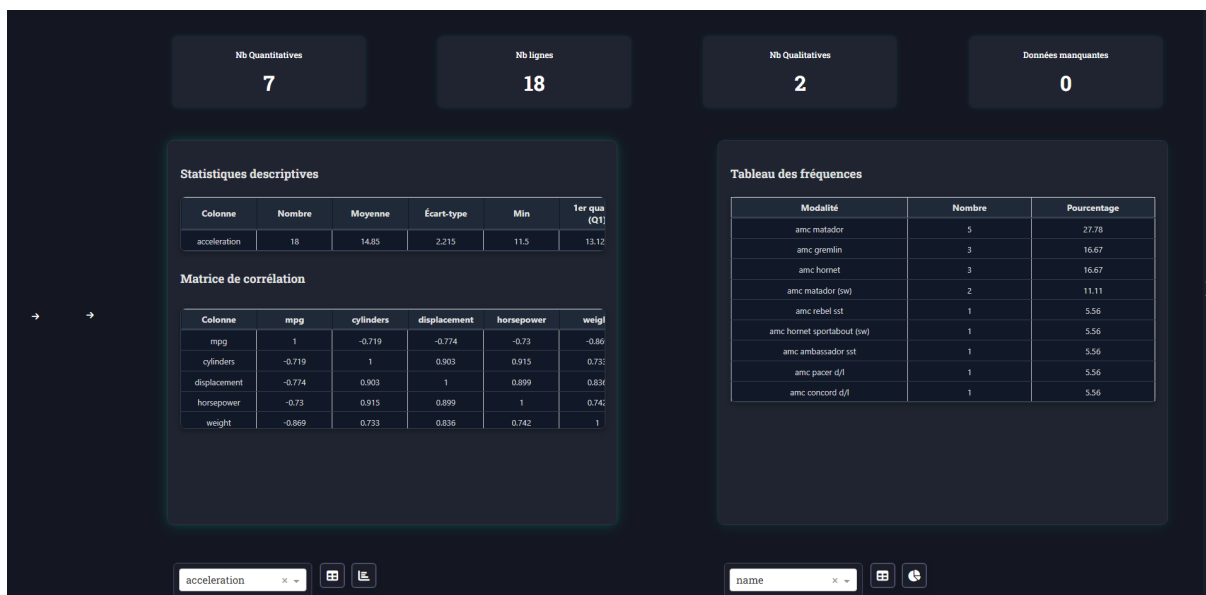
Il peut être intéressant de comparer cela à une autre marque de voiture, en gardant la même logique de puissance.

Pour cela, il suffit de retirer le filtre dans l'historique et d'en appliquer un nouveau, par exemple sur la marque **amc** (marque américaine).



name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
amc rebel sst	16	8	304	150	3433	12	70	usa
amc hornet	18	6	199	97	2774	15.5	70	usa
amc gremlin	19	6	232	100	2634	13	71	usa
amc matador	18	6	232	100	3388	15.5	71	usa
amc hornet sportabout (sw)	18	6	258	110	2962	12.5	71	usa
amc ambassador sst	17	8	304	150	3672	11.5	72	usa
amc matador (sw)	15	8	304	150	3892	12.5	72	usa
amc matador	14	8	304	150	3672	11.5	73	usa
amc hornet	18	6	232	100	2945	16	73	usa
amc gremlin	18	6	232	100	2789	15	73	usa
amc hornet	19	6	232	100	2901	16	74	usa
amc matador	16	6	258	110	3632	18	74	usa
amc matador (sw)	14	8	304	150	4257	15.5	74	usa
amc matador	15	6	258	110	3730	19	75	usa
amc gremlin	20	6	232	100	2914	16	75	usa
amc matador	15.5	8	304	120	3962	13.9	76	usa
amc pacer d/i	17.5	6	258	95	3182	17.8	76	usa
amc concord d/i	18.1	6	258	120	3410	15.1	78	usa

Voici la nouvelle moyenne d'accélération, qui est de 14,85 secondes.



Nb Quantitatives
7

Nb lignes
18

Nb Qualitatives
2

Données manquantes
0

Statistiques descriptives

Colonne	Nombre	Moyenne	Écart-type	Min	1er qua (Q1)
acceleration	18	14.85	2.215	11.5	13.12

Matrice de corrélation

Colonne	mpg	cylinders	displacement	horsepower	weight
mpg	1	-0.719	-0.774	-0.73	-0.86
cylinders	-0.719	1	0.903	0.915	0.73
displacement	-0.774	0.903	1	0.899	0.83
horsepower	-0.73	0.915	0.899	1	0.74
weight	-0.86	0.73	0.83	0.74	1

Tableau des fréquences

Modalité	Nombre	Pourcentage
amc matador	5	27.78
amc gremlin	3	16.67
amc hornet	3	16.67
amc matador (sw)	2	11.11
amc rebel sst	1	5.56
amc hornet sportabout (sw)	1	5.56
amc ambassador sst	1	5.56
amc pacer d/i	1	5.56
amc concord d/i	1	5.56

12.1.2 Valeurs aberrantes

Dans cette section, nous utiliserons le jeu de données **FRvideos.csv** pour illustrer la gestion des valeurs aberrantes.

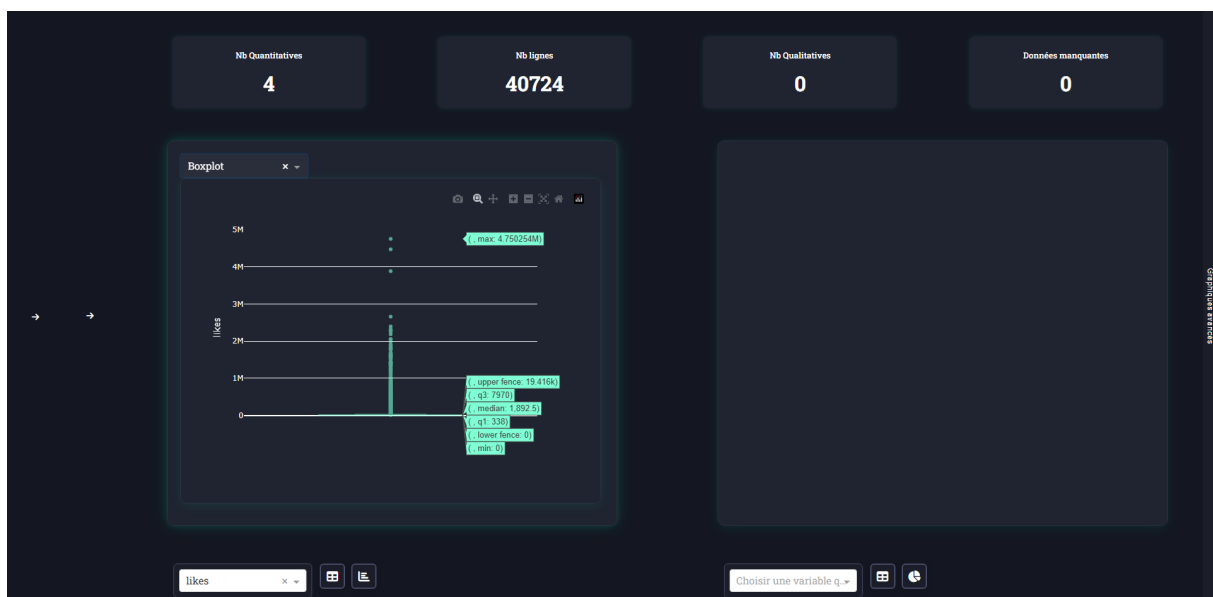
Pour éclaircir notre affichage, nous allons conserver uniquement les colonnes qui nous intéressent, avec le filtre *Colonnes gardées*.

Voici le jeu de données avant application des filtres :

views	comments_disabled	comment_count	likes	dislikes
212702	false	3817	29282	1108
432721	false	1161	14053	576
482153	false	9580	76203	477
925222	false	4309	85016	550
141695	false	481	8091	72
141253	false	417	14354	202
187654	false	2419	9286	1381
91051	false	701	1674	1903
2340441	false	7575	200598	6018
635236	false	1483	5945	722
294065	true	0	0	0
1504950	false	2913	108635	1562
335121	false	3004	23410	2034
53248	false	234	5164	43
55313	false	396	4153	97
177608	false	237	8163	95
372013	false	813	23645	411
79611	false	25	56	4
255349	false	950	20069	245
653398	false	3976	27773	2778
152390	false	1335	4666	1668
321090	false	13845	36515	3354
2418783	false	12703	97192	6146
72519	false	362	689	23
166132	false	1042	13738	156
240158	false	935	6765	160

L'affichage est plus clair grâce à la sélection des colonnes.

Pour savoir s'il est pertinent de traiter les valeurs aberrantes, nous allons analyser, à l'aide d'un boxplot, la colonne **likes**, qui représente le nombre de likes d'une vidéo.



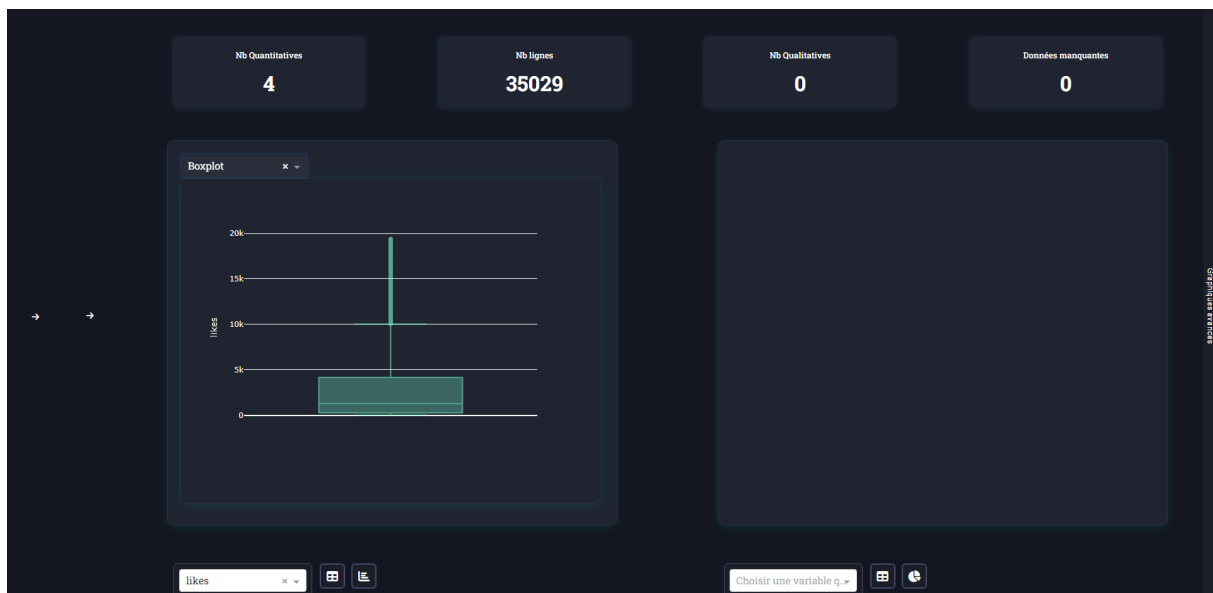
Les nombreux points uniques représentent des valeurs aberrantes, ce qui est logique : certaines vidéos peuvent avoir énormément de likes, tandis que d'autres n'en ont presque pas.

Pour les traiter, nous allons utiliser le bouton *Valeurs aberrantes*, dans la partie traitement des données.

Nous allons choisir de les supprimer, mais il est également possible de les agréger à la médiane ou de les clipper.

views	comments_disabled	comment_count	likes	dislikes
212702	false	3817	26282	1108
432721	false	1161	14053	576
482153	false	9580	76203	477
675222	false	4303	85016	550
141690	false	481	8091	72
141233	false	417	14334	202
167654	false	2419	9386	1381
91051	false	701	1674	1903
2340941	false	7375	200598	6018
635236	false	1483	5945	722
294065	false	0	0	0
1504950	false	2913	108835	1562
335121	false	234	23410	2034
53248	false	234	3164	43
33813	false	813	4132	97
172608	false	234	8163	95
373013	false	813	23645	411
79611	false	25	56	4
233349	false	950	20009	245
630386	false	3916	27773	219
152390	false	3335	4666	1608
321090	false	13645	36315	3354
2418783	false	12703	97192	6146
72519	false	362	689	23
166132	false	1042	13738	156
240158	false	935	6765	160

Une fois le traitement appliqué, nous obtenons le graphique suivant :



On observe toujours la présence de quelques valeurs aberrantes, mais elles sont acceptables, car elles sont inférieures au seuil :

$$Q_3 + 1,5 \times IQR$$

contrairement aux valeurs initiales.

Le problème avec les valeurs aberrantes est qu'elles peuvent fausser les résultats de nos analyses, notamment en classification supervisée, avec des modèles comme la LDA ou les K plus proches voisins, qui s'appuient fortement sur la distribution des données. Il est donc important de les traiter de manière appropriée.

12.1.3 Graphiques avancés

Pour aller plus loin dans l'analyse des données, nous pouvons utiliser des graphiques avancés. Ces derniers permettent de visualiser les relations entre plusieurs variables et d'identifier des patterns ou des anomalies.

Voici les graphiques avancés que nous pouvons créer :

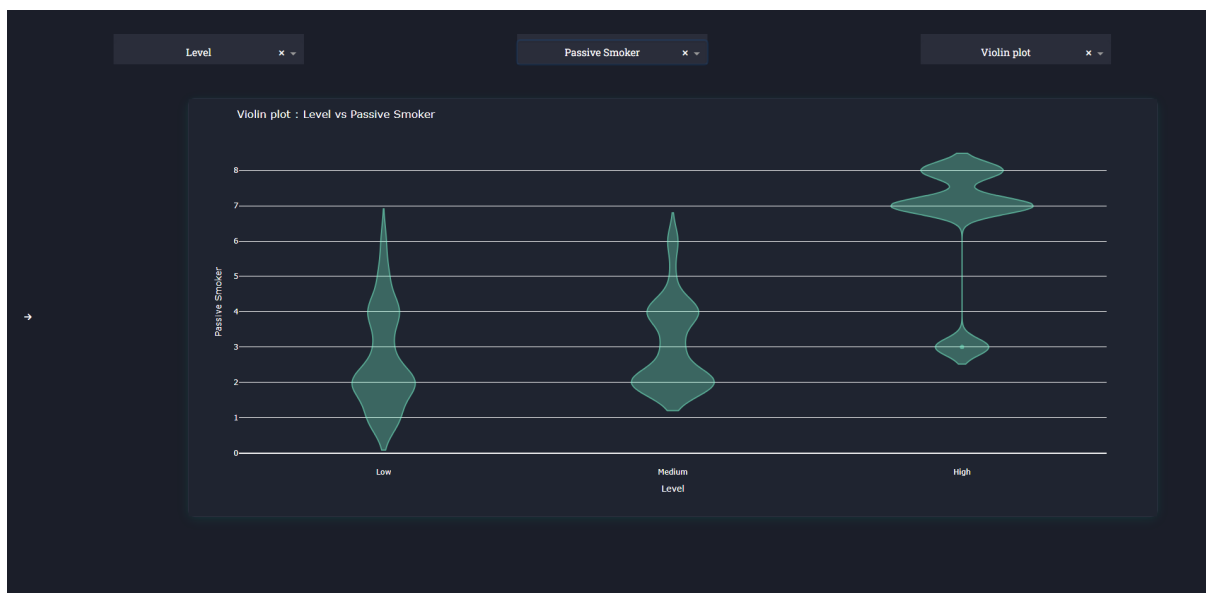
- **Nuage de points** : pour visualiser la relation entre deux variables quantitatives.
- **Courbe de régression** : pour modéliser la relation entre deux variables quantitatives et prédire une variable en fonction de l'autre.
- **Boxplot** : pour visualiser la distribution d'une variable quantitative en fonction d'une variable qualitative.
- **Violinplot** : similaire au boxplot, mais avec une représentation de la densité de probabilité de la variable quantitative.

Pour cette section, nous utiliserons le jeu de données `cancer.csv`, qui contient des informations sur des patients et leur risque de développer un cancer du poumon.

Pour ouvrir le panneau des graphiques avancés, il suffit de cliquer sur le bouton *Graphiques avancés* situé à droite de l'espace de visualisation.

Une fois le panneau ouvert, nous pouvons choisir les variables à analyser ainsi que le type de graphique souhaité dans les menus déroulants situés en haut du panneau.

Voici un exemple de graphique avancé que nous pouvons créer avec ce jeu de données :



Ici, un graphique en violon a été créé pour représenter le risque d'avoir un cancer du poumon en fonction du score de fumage passif, compris entre 0 et 8 ou plus.

Plus on est proche de 8, plus le niveau de fumage passif est élevé.

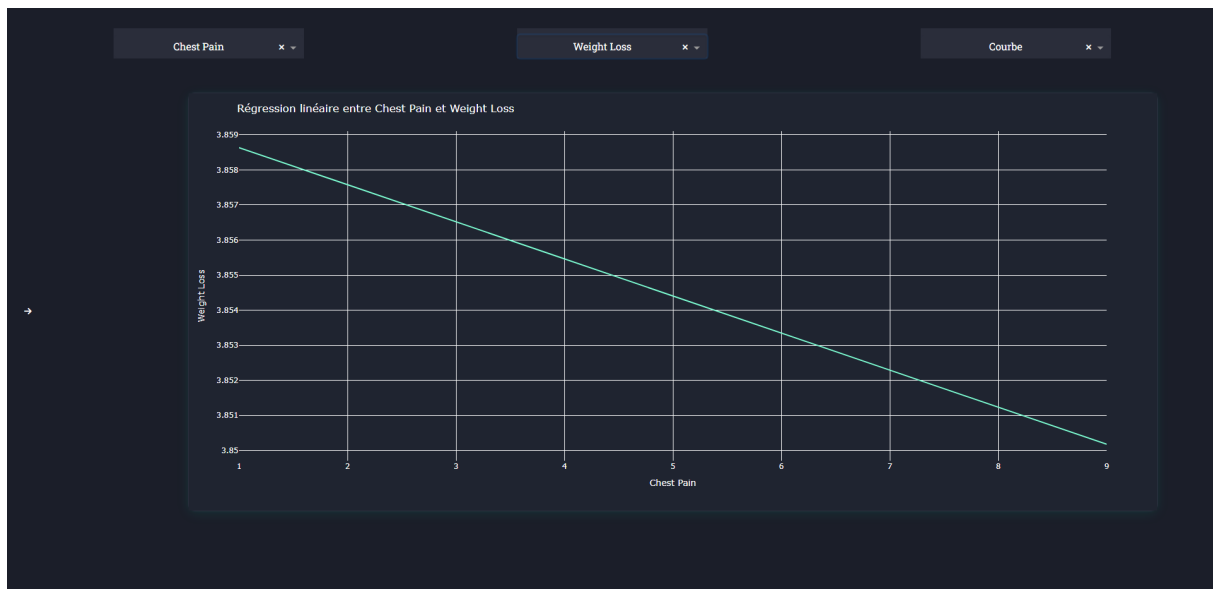
Les résultats du graphique montrent clairement une information contenue dans cette variable. En effet, pour un risque élevé, on observe que la densité de l'effectif la plus importante se situe entre un score de 6 et 8, alors que pour un faible risque d'avoir un cancer du poumon, cette densité se trouve plutôt entre 0 et 3.

Ce type de graphique est très utile en classification supervisée, car il permet de visualiser les relations entre les variables, de détecter des motifs (patterns) et d'identifier les variables qui contiennent de l'information pertinente pour mieux prédire la classe cible (ici, le risque d'avoir un cancer du poumon).

Un dernier exemple pourrait être le taux d'obésité en fonction du niveau de fumage passif :



On voit ici une droite fortement croissante, ce qui peut indiquer que plus le niveau de fumage passif est élevé, plus le taux d'obésité est important.



Le lien entre les douleurs thoraciques et le niveau de perte de poids est décroissant mais très faible, ce qui peut indiquer que ces deux variables ne sont pas liées.

Il existe de nombreux autres exemples de graphiques avancés que l'on peut créer avec **Datazen**. Il suffit de choisir les variables et le type de graphique souhaité dans les menus déroulants.

12.1.4 Fusion de données

Dans cette section, nous allons explorer la fonctionnalité de fusion de données de **Datazen**. Cette fonctionnalité permet de combiner plusieurs jeux de données en un seul, soit par concaténation, soit par fusion.

Pour y accéder, il suffit de cliquer sur le bouton *Fusion de données* situé juste en dessous du bouton de tri dans la partie traitement des données.

Nous utiliserons les jeux de données `fusion1.csv` et `fusion2.csv` pour illustrer cette fonctionnalité.

Respectivement :

Datazen

fusion1.csv

fusion2.csv

Import un fichier

employee_id	salary	bonus
1	50000	5000
2	60000	6000
4	55000	4000
5	65000	7000

Datazen

fusion1.csv

fusion2.csv

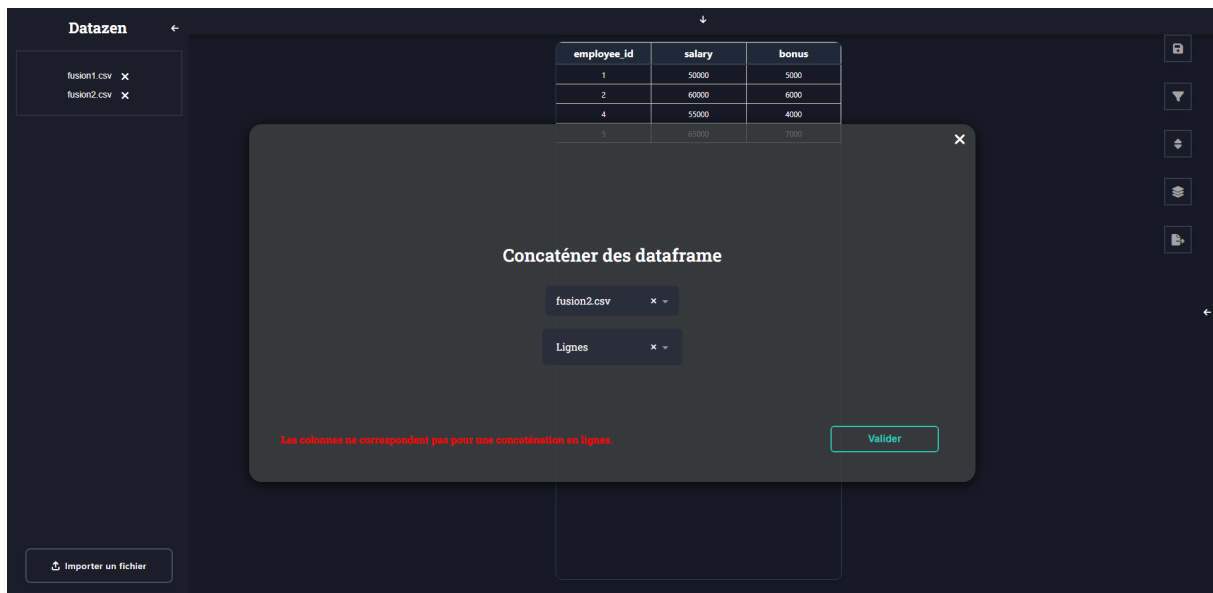
Import un fichier

employee_id	name	department
1	Alice	HR
2	Bob	Engineering
3	Charlie	Marketing
4	David	Sales

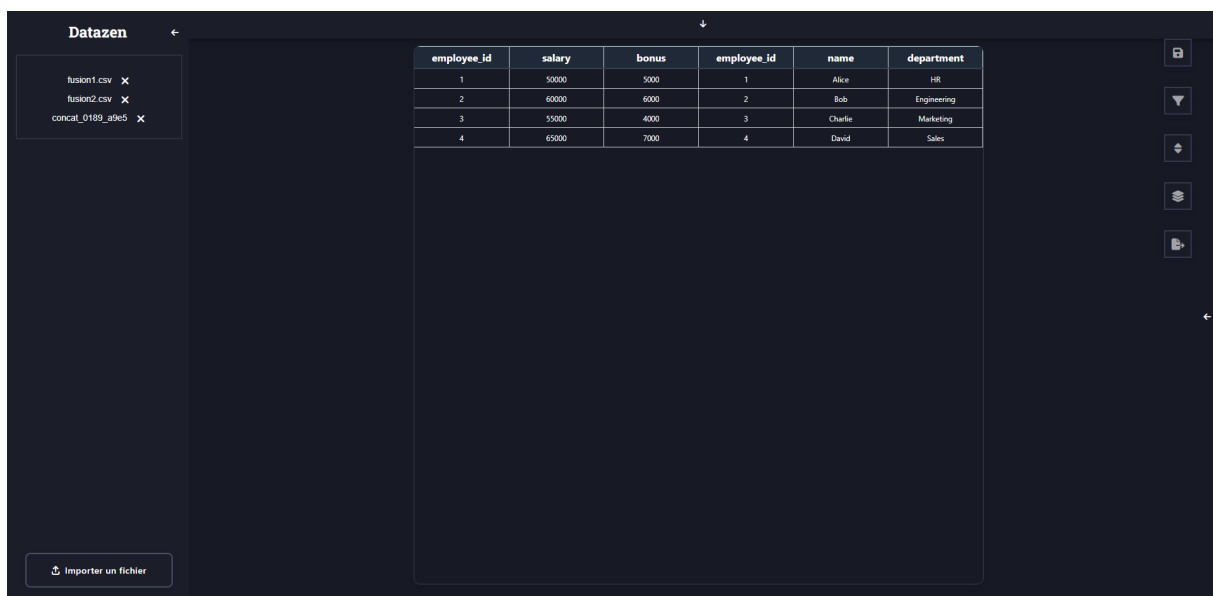
Concaténation de données

Ces deux jeux de données comportent le même nombre de lignes, il est donc possible de les concaténer par colonnes.

Pour ce faire, il faut d'abord choisir l'un des deux fichiers comme table active, puis cliquer sur le bouton *Fusion de données* :



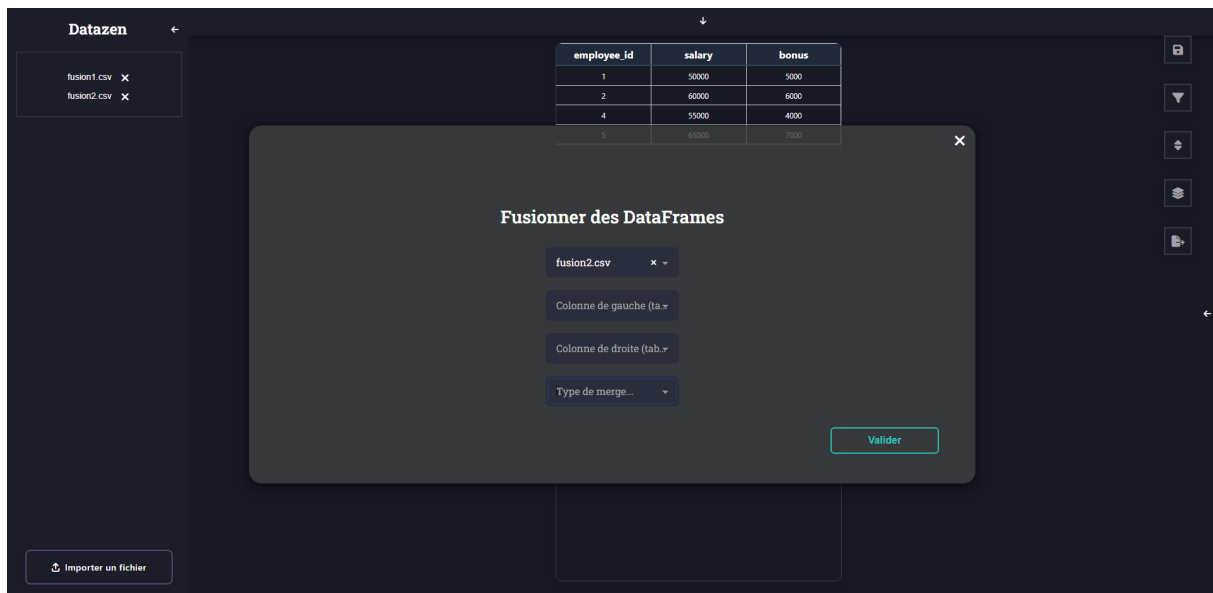
Si la concaténation n'est pas possible, un message d'erreur s'affiche.



Ici, la concaténation en colonnes a bien été appliquée : on voit le nouveau jeu de données s'ajouter dans le menu latéral gauche, et on peut dès lors interagir avec lui ou l'exporter via le bouton d'exportation situé juste en dessous.

Fusion par clé (merge)

On observe que ces deux jeux de données possèdent une colonne en commun, `employee_id`. Il est donc possible de les fusionner en utilisant cette colonne comme clé.



On commence par sélectionner le jeu de données à fusionner, puis la colonne du jeu actif contenant les identifiants. Ensuite, on sélectionne la colonne correspondante dans le second jeu, puis on choisit la méthode de fusion parmi :

- **inner** : conserve uniquement les lignes dont les identifiants sont présents dans les deux jeux de données.
- **outer** : conserve toutes les lignes des deux jeux de données, en remplissant les valeurs manquantes par des NaN.
- **left** : conserve toutes les lignes du jeu de données actif, en ajoutant les colonnes correspondantes du second jeu lorsque les identifiants correspondent.
- **right** : conserve toutes les lignes du jeu de données à fusionner, en ajoutant les colonnes du jeu actif si les identifiants correspondent.

Comme ici les identifiants ne sont pas exactement les mêmes dans les deux fichiers, nous choisissons la méthode **inner**, afin de ne conserver que les lignes ayant des identifiants communs.

Une fois le bouton *Valider* cliqué, le jeu de données fusionné s'affiche dans le menu latéral gauche :

employee_id	salary	bonus	name	department
1	50000	5000	Alice	HR
2	60000	6000	Bob	Engineering
4	55000	4000	David	Sales

Tout comme avec la concaténation, ce nouveau jeu de données peut être exploré, analysé ou exporté.

12.2 Guide d'installation de Datazen

12.2.1 Installation rapide

1. Cloner le dépôt GitHub

Pour récupérer le code source de Datazen, utilisez la commande suivante dans un terminal :

```
1 git clone https://github.com/alexisvincent37/Datazen
```

2. Se placer dans le répertoire cloné

```
1 cd "votre_chemin/Datazen"
```

3. Créer un environnement virtuel (recommandé)

```
1 python -m venv .venv
```

Activation de l'environnement :

— Sous Windows :

```
1 .venv\Scripts\activate
```

— Sous Linux / macOS :

```
1 source .venv/bin/activate
```

4. Installation des dépendances

```
1 pip install auto-py-to-exe coverage dash dash-design-kit
2 flask-caching openpyxl pandas plotly pytest pytest-cov
3 statsmodels xlswriter
```

12.3 Lancement de l'application

Méthode 1 : exécuter l'application en Python

Sous PowerShell (Windows) :

```
1 $env:PYTHONPATH="src"; python app.py
```

Sous Linux/macOS :

```
1 PYTHONPATH=src python app.py
```

L'application sera alors disponible à l'adresse suivante : <http://localhost:8050>.

Lancer l'application en mode production avec Gunicorn :

```
1 PYTHONPATH=src gunicorn app:server --workers 7 --bind 0.0.0.0:8050
```

(Adaptez le nombre de workers selon les capacités de votre machine.)

Méthode 2 : utilisation de l'exécutable

Un exécutable nommé `app.exe` est disponible à la racine du projet. Il a été généré via l'outil `auto-py-to-exe`.

- Il permet de lancer directement l'application sans installer Python.
- Le mode `debug` est désactivé (`debug=False`) pour une utilisation stable.
- Idéal pour une utilisation quotidienne ou non technique.

13 Bibliographie

Références

- [1] Python Software Foundation, Documentation officielle,
<https://docs.python.org/3/>
- [2] Dash, Documentation officielle,
<https://dash.plotly.com/>
- [3] Pandas, Documentation officielle,
<https://pandas.pydata.org/>

-
- [4] Plotly, Documentation officielle,
<https://plotly.com/python/>
 - [5] Flask, Documentation officielle,
<https://flask.palletsprojects.com/>
 - [6] Flask-Caching, Documentation officielle,
<https://flask-caching.readthedocs.io/>
 - [7] Pickle — Python object serialization, Documentation officielle,
<https://docs.python.org/3/library/pickle.html>
 - [8] Statista, Infographie sur le volume des données numériques,
<https://fr.statista.com/infographie/17800/big-data-evolution-volume-donnees-numeri>
 - [9] Qiangqiang Fan, Jinhan Xie, Zhaoyang Dong et Yang Wang, « The Effect of Ambient Illumination and Text Color on Visual Fatigue under Negative Polarity », *Sensors*, vol. 24, art. 3516, 2024,
<https://doi.org/10.3390/s24113516>