Licence Informatique, 3ème année

La Rochelle Université

Rapport de stage

Outils de visualisation de données de GALACTIC

Louna DENIS

2026

Utilisation du portrait d'Évariste Galois réalisé par M. Yann Gautreau
aimablement autorisée dans un cadre universitaire

Table des matières

Liste des figures

Abstract

Résumé en français

Le sujet de ce stage est de concevoir des **outils de visualisation de données**, destinés à être intégrés au projet **GALACTIC** du L3I de La Rochelle.

Le projet **GALACTIC** est un *framework* Python d'analyse de données, conçu au sein du **L3I de La Rochelle**.
GALACTIC traite plus précisément le domaine de l'**Analyse Formelle de Concepts**, appliquée à des données complexes et hétérogènes.
La complexité de ces données rend l'analyse particulièrement difficile, car de nombreux choix doivent être faits par l'analyste, devant être éclairés par les données.

Ainsi nait le besoin pour GALACTIC de pouvoir visualiser les données traitées dans une forme plus facile à analyser par un humain. La création d'outils permettant de **visualiser des données de GALACTIC**, en générant divers **diagrammes**, est l'objet de ce stage.
Ces outils sont principalement réalisés via les modules Python *Seaborn* et *Matplotlib*.

Ce stage traite des premiers travaux de visualisation pour GALACTIC, afin de représenter des données numériques, catégorielles, et booléennes. Un point clé des visualisations souhaitées est de représenter les **prédicats** calculés par GALACTIC, des indications de la sémantique des données pour les ***clusters*** de données générés pas GALACTIC.

Plusieurs tentatives de visualisation des données se sont succédé, sur des données numériques et booléennes, menant à la visualisation des données numériques par des histogrammes des effectifs, indiquant également leurs prédicats.

Abstract in English

This internship's subject is to create **data visualization tools**, with the purpose to be integrated to the GALACTIC project of the L3I of La Rochelle.

The **GALACTIC** project is a Python framework for data analysis, created in the **L3I of La Rochelle**. GALACTIC handles more specifically the domain of **Formal Concept Analysis**, applied to complex and heterogeneous data. The complexity of such data makes the analysis especially difficult, as numerous choices must be made by the analyst, needing to be guided by data.

As such, the need for GALACTIC to visualize data in a form easier to analyze by humans is born. The creation of tools to **visualize data from GALACTIC**, by generating multiple **diagrams**, is the object of this internship. These tools are mainly made with the Python modules *Seaborn* and *Matplotlib*.

This internship handles the first visualization tools for GALACTIC, to represent numerical, categorical and boolean data. A key point for the visualizations is to represent the **predicates** generated by GALACTIC, indications of the data semantics for **clusters** of data generated by GALACTIC.

Several data visualizations have been tried, on numerical and boolean data, leading to visualization of numerical data with distribution histograms, showing the data's predicates as well.

1 Introduction

1.1 Présentation du L3I

Le **L3I**, ou **Laboratoire Image, Informatique et Interaction**, est un laboratoire de recherche affilié à **La Rochelle Université**. Les recherches du laboratoire se centrent autour de la **gestion intelligente et interactive des contenus numériques**.

Le L3I est organisé en 3 équipes:

- **L'équipe IC, Images et Contenus**, concentre ses recherches sur le traitement et l'analyse de contenus de sources variées (documents, fichiers audio, images de capteurs…), et sur l'extraction et la structuration de données provenant de grandes bases de données.
- **L'équipe eAdapt, Dynamique des Systèmes et Adaptativité**, s'intéresse au pilotage par les données des systèmes numériques, afin de concevoir des systèmes plus adaptatifs et sécurisés.
- **L'équipe MC, Modèles et Connaissances**, concentre ses recherches sur les modèles et raisonnement pour la construction de systèmes intelligents, ainsi que l'étude de séquences temporelles.

Le cadre de ce stage se déroule au sein de l'équipe **Modèles et Connaissances**. La **figure 1** décrit l'organigramme du L3I.

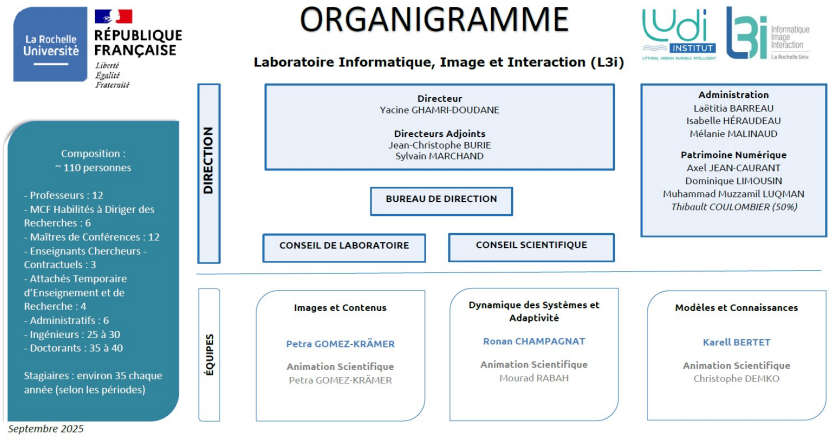

FIGURE 1 – Organigramme du L3I

1.2 Présentation de GALACTIC

L'outil **GALACTIC** (**GA**lois **LA**ttices, **C**oncept **T**heory, **I**mplicational systems and **C**losures) est un ***framework*** **Python** utilisant l'**Analyse Formelle des Concepts** (AFC), une méthode de pattern mining, afin de générer des *clusters* sémantiques de **données hétérogènes et complexes**. Il est composé à la fois d'une librairie extensible, à laquelle les utilisateurs peuvent ajouter des modules, mais également d'une application en ligne de commande permettant de générer des squelettes de code.

Le principe de l'AFC est de générer des *clusters* de données, composés d'objets mais aussi de leurs attributs booléens, ce qui permet de conserver la sémantique des données, la raison pour laquelle les objets ont été placés dans le même *cluster*. L'ensemble de ces *clusters* forment un treillis, une structure algébrique hiérarchique dans laquelle deux groupes quelconques ont toujours un unique plus petit sous-groupe commun.

L'AFC classique a comme limitation qu'elle ne peut être réalisée qu'avec des attributs booléens. C'est l'un des deux verrous scientifiques que GALACTIC a levé: le *framework* permet de traiter des données booléennes, numériques, catégorielles et séquences de manière hétérogène. Les attributs des *clusters* sont remplacés par des prédicats, décrivant les valeurs communes à tous les objets d'un même *cluster*. L'autre verrou scientifique levé par GALACTIC est que les prédicats n'ont pas à être pré calculé, mais sont à la place calculés pendant leur génération sous forme d'un treillis.

Ainsi, l'analyse fournie par GALACTIC est réalisée en "boîte blanche": l'utilisateur de l'outil, un analyste de données, peut influer sur l'analyse et observer son fonctionnement, contrairement aux réseaux de neuronnes par exemple. Il peut comprendre comment les *clusters* se sont formés grâce aux prédicats, et peut influer sur le treillis en utilisant différentes **stratégies**, afin d'orienter la génération des clusters dans la direction voulue.

L'architecture de GALACTIC est composée d'un noyau en plusieurs couches, codé en Rust et en Python, sur laquelle plusieurs modules, écrits en Python, viennent se greffer:

- **Caractéristiques**, permettant d'extraire et de stocker tout type de données;
- **Descriptions**, définissant des prédicats décrivant un *cluster* de données;
- **Stratégies**, qui découpent / affinent un *cluster* de données en sous-*cluster*;
- **Mesures**, ou méta-stratégies, qui permettent de sélectionner / filtrer les sous-*clusters* d'un *cluster* en fonction d'une mesure;
- **Lecture de données**, qui permet la lecture de tous types de fichiers de données.

Chaque module peut se voir ajouter des fonctionnalités par les utilisateurs, permettant par exemple de rajouter des types de données que GALACTIC ne traite pas encore. La **figure 2** décrit le schéma de l'architecture de GALACTIC.

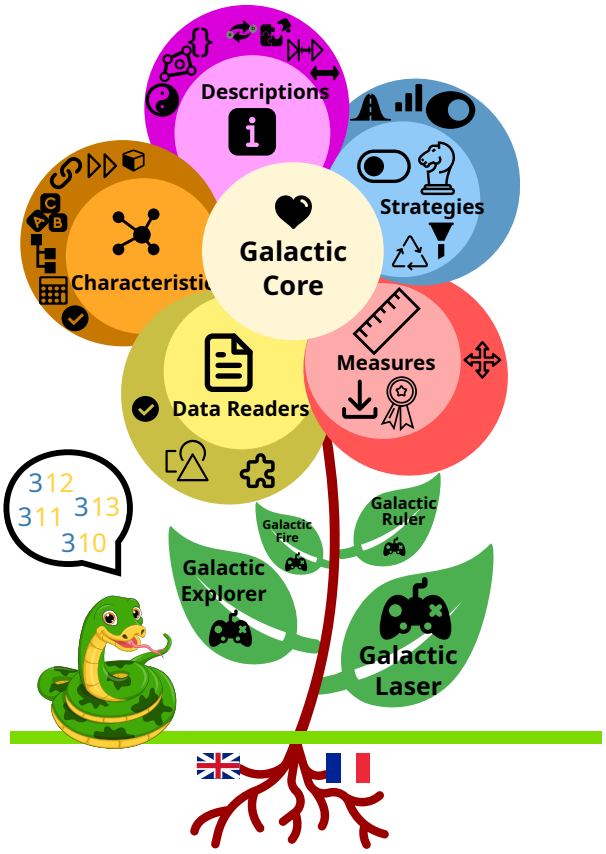

FIGURE 2 – Fleur de GALACTIC

GALACTIC est pendant le déroulé de ce stage en pleine restructuration de code, afin de parvenir à une version deux. Cette nouvelle version est prévue pour fin juin 2026, pour une présentation à la PFIA (Plate-Forme Intelligence Artificielle)[1] de 2026, une série de conférences sur l'IA, rassemblant chercheurs, étudiants et industriels du domaine, dont la prochaine édition est programmée du 29 juin au 3 juillet 2026 à Arras.

1.3 Objectif du stage

L'objectif de ce stage est donc de concevoir des outils de visualisation de données pour le projet GALACTIC, sous la forme de diagrammes. Ces diagrammes ont pour but de mettre en évidence un *cluster* ciblé par rapport à l'ensemble des données, en montrant les prédicats indiquant l'appartenance à ce *cluster*. Ces visualisations se concentrent sur les données numériques, catégorielles et booléennes.

Certaines d'entre elles ont déjà été réalisées:

- Deux visualisations de données numériques, avec un diagramme en barres et un diagramme en lignes. Ces visualisations peuvent seulement représenter un petit *cluster*, sans indiquer ses prédicats.
- Une visualisation de données booléenne en table, pouvant également représenter un petit *cluster* sans ses prédicats.
- Une visualisation de données numériques en histogramme des occurences, pouvant représenter l'ensemble des données numériques et les prédicats correspondant à un *cluster*, sans pour autant représenter le *cluster* en lui-même.

Une amélioration de l'histogramme des données numériques, et des visualisations de données catégorielles et booléennes en barres, avec vision des prédicats, sont prévues pour la fin de ce stage.

Dû au travail en parallèle sur le fonctionnement principal de GALACTIC, la visualisation n'a pas encore pu lui être directement liée. Ainsi, les données testées sont des jeux de données obtenus sur Internet, comme le jeu de données Iris. Les prédicats sont entrés à la main, afin de simuler le résultat des visualisations.

1.4 Outils utilisés

Dans le cadre de ce stage, plusieurs outils ont été utilisés.

Gitlab[2]: Outil de gestion de projet basé sur Git, utilisé pour gérer le projet GALACTIC ainsi que la visualisation.

Python[3]: Langage de programmation principal de la visualisation.

Matplotlib[4]: Module Python, utilisé pour la génération de tous type de diagrammes dans des fenêtres dédiées.

Seaborn[5]: Module Python, se reposant sur *Matplotlib* pour simplifier les commandes de diagrammes. Utilisé en tandem avec ce dernier.

Pandas[6]: Module Python, permettant de stocker des données de manière pratique et efficace. Utilisé pour tester la visualisation sur des données réelles, en attendant l'intégration à GALACTIC. Possède des fonctions de visualisation basées sur *Matplotlib*.

Numpy[7]: Module Python, contenant un grand nombre de fonctions mathématiques complexes. Il est notamment utilisé par *Matplotlib* pour calculer certains diagrammes, ces calculs pouvant être utilisés indépendamment.

2 Développement

2.1 Visualisations de petits clusters

2.1.1 Diagrammes en barres - données numériques

La première tentative de visualisation numérique fut de représenter les données par un diagramme en barres. Plus précisément, seul un *cluster* du jeu de données est représenté. L'idée est de placer en ligne des diagrammes en barres, un par attribut. On retrouve donc les objets du *cluster* en abscisse, et leurs valeurs pour l'attribut correspondant en ordonnée, indiquées par la hauteur des barres. La **figure 3** montre un exemple d'un tel diagramme.

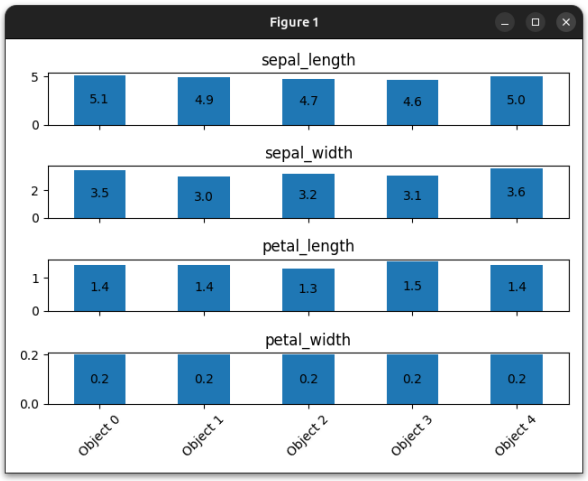

FIGURE 3 – Diagramme en barres pour 5 objets et 4 attributs

Plusieurs axes ont été générés à l'avance grâce à la fonction *subplots* de *Matplotlib*, et passés en paramètres de la fonction de visualisation numérique.

> **Architecture d'un diagramme *Matplotlib***
> Les diagrammes *Matplotlib* sont composés d'une **Figure** principale, correspondant à la fenêtre, auquel on associe un ou plusieurs **Axes**, chacun pouvant contenir un diagramme. Tous les éléments visibles de ce diagramme sont des **Artists** (les lignes, le texte…).

On utilise la fonction *plot* de *Matplotlib*, en utilisant son API explicite pour préciser quel axe contient quel diagramme.

> **Les deux modes de *Matplotlib***
> Les diagrammes de *Matplotlib* peuvent être créés de deux manières différentes. - **La manière implicite**, la plus courante, laisse la librairie se charger des détails. Il suffit d'appeler les fonctions via le module *pyplot*. Elle est plus facile d'utilisation, et s'inspire de l'utilisation du logiciel *Matlab*. - **La manière explicite** permet à l'utilisateur de préciser quel objet de *Matplotlib*, comme un *Axes*, effectue quelle action. Elle offre plus de liberté au programmeur, et se rapproche de la programmation objet.

Chaque diagramme partage son axe x *(on parle ici de l'axe au sens mathématique, à ne pas confondre*

avec Axes*)*. Ce partage se fait grâce au paramètre *sharex* de la fonction *plot*, ce qui permet aux barres de chaque diagramme d'être alignées en colonnes, et donc de partager les mêmes valeurs.

Cependant, cette visualisation a un inconvénient: elle ne peut afficher qu'un petit nombre de données avant de devenir illisible. En effet, une barre par objet est beaucoup trop quand on doit afficher 30 objets ou plus, comme visible sur la **figure 4**.

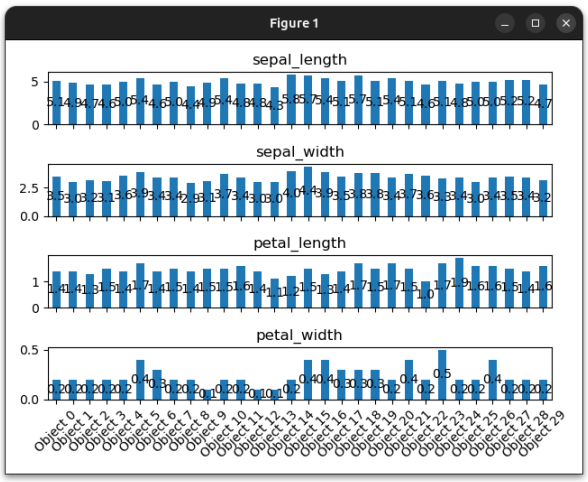

FIGURE 4 – Diagramme en barres pour 30 objets et 4 attributs

Ainsi, cette visualisation permet de représenter des petits *clusters* de manière détaillée, mais n'est pas adaptée à de plus grands *clusters*.

2.1.2 Diagrammes en lignes - données numériques

Pensée comme un tandem de la visualisation en barres, cette visualisation place également les attributs d'un *cluster* dans des diagrammes en ligne, alignés en colonne par ses objets. La différence est que les valeurs sont représentées par des points, mis en évidence, et reliés par des lignes servant seulement à indiquer les différences de valeurs avec les autres objets du *cluster*. Un exemple de ce diagramme est représenté par la **figure 5**.

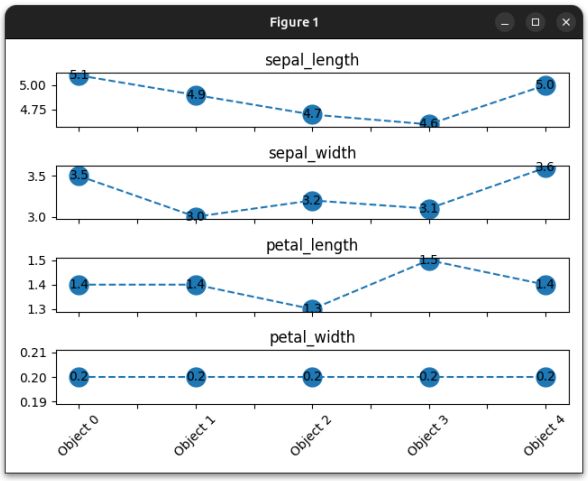

FIGURE 5 – Diagramme en lignes pour 5 objets et 4 attributs

Cependant, étant donné que ce diagramme est organisé de la même manière que le diagramme en barres, il souffre du même défaut: la visualisation d'un grand nombre d'objets, comme on peut l'observer sur la **figure 6**. Tout comme le diagramme en barres précédent, il est donc plus adapté pour une visualisation détaillée de petits *clusters*.

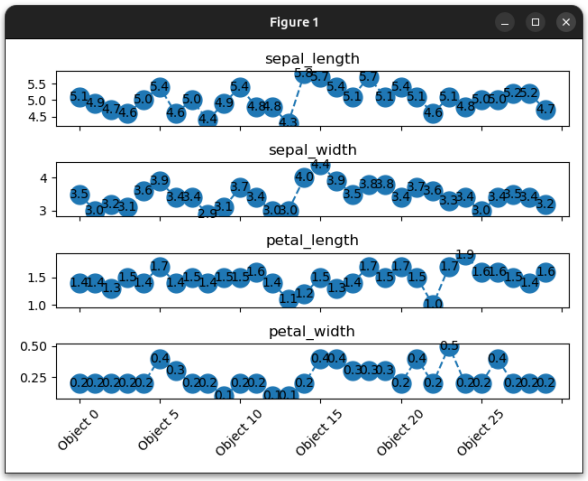

FIGURE 6 – Diagramme en ligne pour 30 objets et 4 attributs

2.1.3 Table pour données booléennes

Pour la visualisation de données booléennes, le choix le plus évident est de représenter un *cluster* par une table. Les attributs sont en colonnes et les objets en lignes. On colore les cases si leur valeur est *True*, et on remplace les mots booléens de la table par des *X* si *True*. Un exemple de cette table est montré sur la **figure 7**.

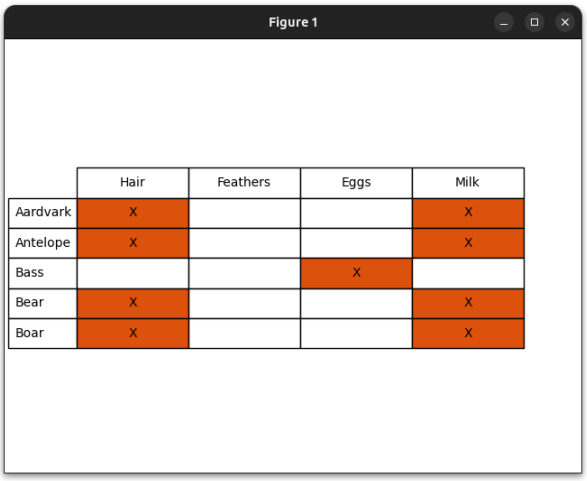

FIGURE 7 – Table booléenne de 5 objets et 4 attributs

Cependant, cette visualisation se heurte aux mêmes problèmes que les deux précédentes: la taille du *cluster*.
Les tables *Matplotlib* ne sont pas conçues pour de grands nombres de données, et l'on peut en effet voir sur la **figure 8** que la table n'est pas visible en entier quand elle contient trop d'objets. En effet, les tables *Matplotlib* ne disposent pas de fonctionnalités de "*scrolling*", qui permettraient de descendre dans la table, et donc d'afficher tous les objets.

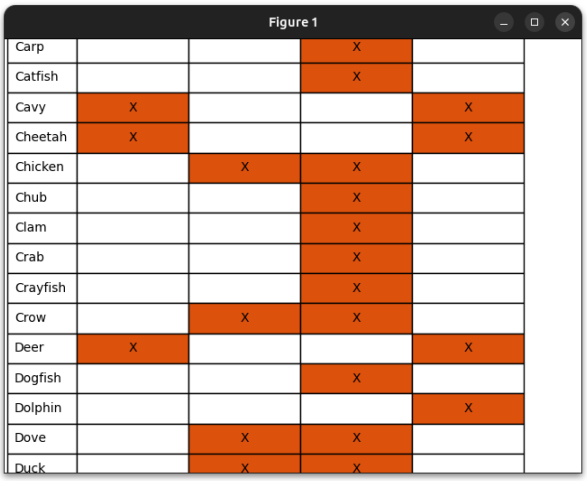

FIGURE 8 – Table booléenne pour 30 objets et 15 attributs

2.1.4 Table pour données booléennes - autres méthodes

Étant donné que les tables de *Matplotlib* seules ne suffisent pas, il a fallu chercher d'autres moyens d'afficher des tables avec un grand nombre de valeurs. Une idée fut d'utiliser le module *plottable*, un module Python spécialisé dans la création de tables. En changeant des paramètres de la *Figure*, tels que *figsize* ou *dpi*, il est possible de "dézommer" le tableau, et ainsi voir toutes les données. Cependant, même ainsi, le tableau n'est pas très lisible avec des grandes valeurs. On peut voir sur la **figure 9** que

la table montre cette fois tous les objets, mais les titres de lignes et de colonnes sont trop petits pour être lisibles.

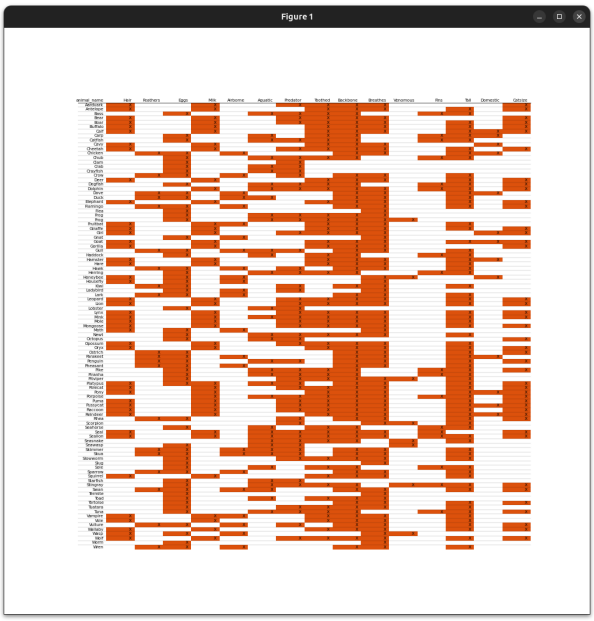

FIGURE 9 – Table booléenne plottable pour 101 objets et 15 attributs

Une autre idée fut d'utiliser la fonction *table* directement intégrée dans *Pandas*, le module Python qui stocke les données. Mais le résultat souffre des mêmes problèmes que *plottable*, comme on peut le voir sur la **figure 10**.

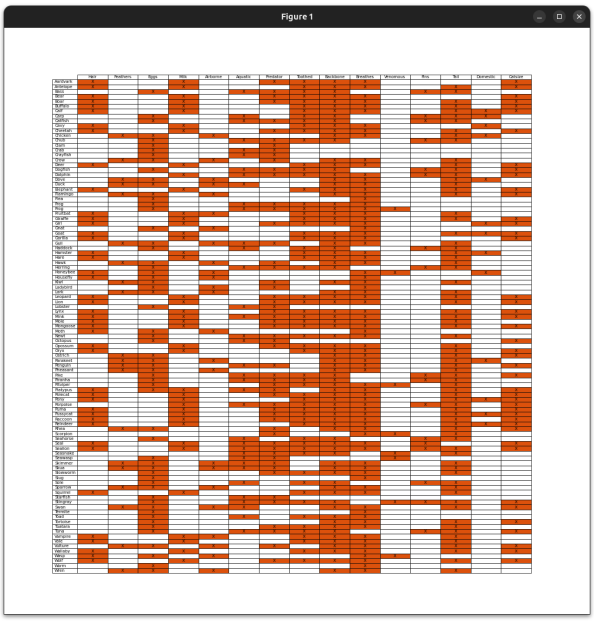

FIGURE 10 – Table booléenne pandas pour 101 objets et 15 attributs

2.2 Visualisations groupées

Un *cluster* de données pour GALACTIC contient le plus souvent plusieurs types de données. Pour pouvoir visualiser un *cluster* dans sa totalité, il est donc nécessaire de créer plusieurs diagrammes.

Pour cela, on va séparer les données, en testant le type de chacune des colonnes du *DataFrame*, qui représentent les attributs. Pour afficher plusieurs diagrammes à la fois, et notamment les données numériques qui contiennent elles-mêmes plusieurs diagrammes, on utilise la fonction *subplot2grid* de *Matplotlib*, qui permet de préciser la position de chaque diagramme. Puis, on appelle les fonctions de chaque type de données, en leur fournissant le ou les axes correspondants aux diagrammes. Le résultat de ce groupement est visible sur la **figure 11**.

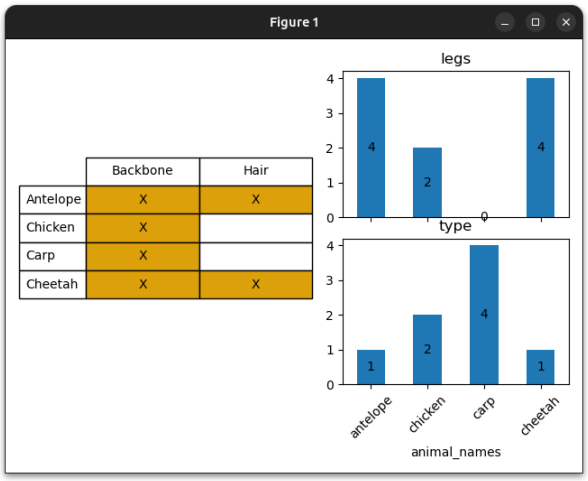

FIGURE 11 – Visualisation groupée booléenne et numérique (barres) pour 4 objets, 2 attributs booléens et 2 attributs numériques

Cependant, cette visualisation groupée a été abandonnée, car séparer les différentes visualisations en différentes fenêtres est plus clair pour l'utilisateur, et laisse plus de place à chaque type de données, au lieu de les serrer les unes contre les autres.

2.3 Visualisations en histogramme

Pour régler les problèmes de taille de données, il a fallu choisir un type de diagramme qui ne changerait pas beaucoup même avec d'énormes quantités de données. La solution trouvée pour les données numériques fut d'utiliser un histogramme des occurences.

En effet, un histogramme des occurences permet de représenter la distribution d'un jeu de données: la largeur de chaque barre indique quelles valeurs numériques sont ciblées, et la hauteur compte le nombre de ces valeurs ciblées dans le jeu de données. On perd en précision, car on ne connaît pas la hauteur de chaque valeur individuellement, seulement par groupes, mais cette perte de précision permet d'indiquer une tendance dans la distribution des données, information importante pour éclairer l'utilisateur dans son analyse.

Un autre intérêt de l'histogramme est que le nombre de barres est calculé automatiquement par des règles mathématiques, apportant différents paramétrages, et assurant que le nombre de barres ne soit jamais trop important.

Ainsi, l'histogramme des occurences se trouve être un très bon diagramme pour représenter un *cluster* parmi l'ensemble des données.

2.3.1 Histogramme sans prédicats - données numériques

On place chaque attribut numérique des données dans un diagramme différent. Ces diagrammes n'ont pas besoin d'être placés en lignes, car ils ne partagent pas les mêmes valeurs en abscisse.

L'histogramme est tracé grâce au module *Seaborn*, par la fonction *histplot*, ce qui permet de simplifier le code par rapport aux commandes *Matplotlib*. On place sur l'axe des abscisses les valeurs des données, et on laisse l'axe des ordonnées être calculé par *Seaborn*. *Seaborn* calcule automatiquement des *bins* à partir des données, des barres contenant une partie des valeurs en abscisse. L'ordonnée indique la hauteur de chacun de ces *bins*, représentant le compte dans les données de l'ensemble des valeurs contenues dans chaque *bin*.

Pour aider à la représentation, une courbe est tracée par-dessus l'histogramme, pour mieux visualiser la distribution des données. Cependant, cette courbe est générée par le paramètre *kde* de la fonction *histplot*, qui ne représente pas une simple courbe suivant les barres, mais une *kernel density estimation*[8], une courbe de densité des données, qui obéit à ses propres règles mathématiques et peut donc diverger de la vision de l'histogramme. Un exemple de *kde* pour un histogramme est montré sur la **figure 12**. Malgré ce souci de courbe, l'histogramme se révèle être une bonne solution pour la visualisation de l'ensemble des données numériques. Il est donc temps de rajouter les prédicats sur les histogrammes.

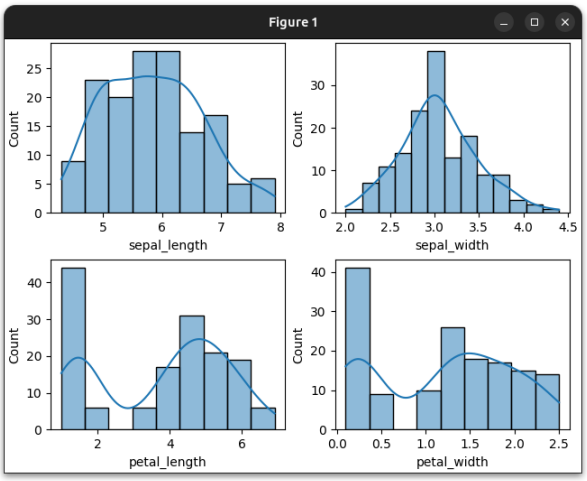

FIGURE 12 – Histogramme numérique simple, avec *kde*, représentant les 150 objets du jeu de données Iris et ses 4 attributs numériques

2.3.2 Histogramme avec visualisation de prédicats - données numériques

Pour l'usage de GALACTIC, les informations les plus importantes à visualiser sont les prédicats du *cluster* mis en évidence.

Les données numériques ont deux prédicats par attribut: une borne inférieure et une borne supérieure, les objets se situant entre ces deux bornes dans chacun des attributs font donc partie du *cluster* représenté par ces prédicats. Pour l'histogramme actuel, on va simplement colorer la zone des prédicats de chaque attribut, sans tenir compte du *cluster*. En effet, un *cluster* nécessiterait que chaque objet en faisant parti ait toutes ses valeurs dans la zone des prédicats de chaque attribut, ce qui est calculé par GALACTIC et ne peut donc être facilement simulé.

Un problème qui se pose est que le fonctionnement d'un histogramme implique qu'on ne peut pas s'assurer que les prédicats soient placés exactement aux bornes d'une barre. Ce problème fait que les *bins* contenant un prédicat sont entièrement colorées, même si certaines de ses valeurs ne font pas partie du prédicat. Pour régler ce problème, on va pré calculer les *bins* de l'histogramme avec la fonction *histogram_bin_edges* du module *Numpy*. On va ensuite rajouter un *bin* pour chaque prédicat à la liste retournée par la fonction, forçant l'histogramme à une séparation exacte, mais altérant l'équilibre de l'histogramme (toutes les barres ne sont plus de la même largeur). La **figure 13** montre un exemple d'histogramme dont on peut voir les prédicats.

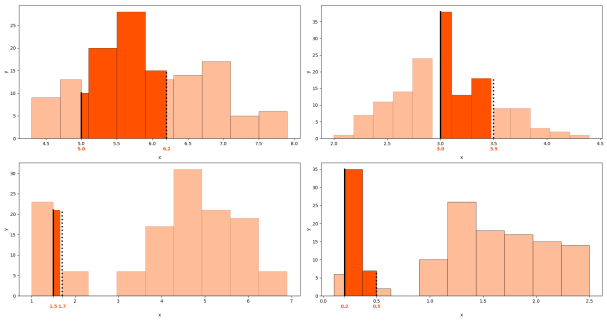

FIGURE 13 – Histogramme numérique avec prédicats, représentant les 150 objets du jeu de données Iris. Les prédicats sont sepal_length >= 5, sepal_length < 6.2, sepal_width >= 3, sepal_width < 3.5, petal_length >= 1.5, petal_length < 1.7, petal_width >= 0.2, petal_width < 0.5

2.3.3 Histogramme avec choix de l'inclusion stricte ou non des prédicats - données numériques

Dans son fonctionnement de base, les barres de l'histogramme incluent la valeur du prédicat inférieur, mais pas la valeur du prédicat supérieur. Ainsi, seul le prédicat inférieur est inclus dans la zone des prédicats, menant à un décalage entre les données réelles et le nombre compté sur l'histogramme. De plus, on veut dans le cadre de GALACTIC avoir le choix de l'inclusion ou non de chaque prédicat dans la zone, car un prédicat numérique peut aussi bien être *strictement plus grand que 5* que *plus grand ou égal à 5*.

Pour palier à ce problème, il n'existe malheureusement pas de paramètre de *Matplotlib* permettant d'inclure ou non ces bornes. Selon les cas d'inclusion ou non, on va donc manuellement changer l'histogramme en modifiant les hauteurs de certains *bins* proches des prédicats. Ainsi, dans l'exemple de l'inclusion de la borne supérieure, on va enlever à la hauteur du *bin* de la borne (qui n'est pas inclus dans la zone des prédicats), et ajouter cette même valeur au *bin* juste avant. Les **figures 14 et 15** représentent le même *cluster* et les mêmes objets, d'abord en incluant seulement la borne inférieure, puis en incluant seulement la borne supérieure des données.

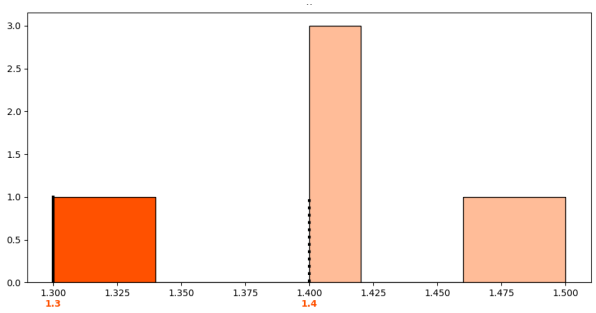

FIGURE 14 – Prédicat sans changement de hauteur, 5 objets sur 1 attribut. Les prédicats sont supérieur ou égal à 1.3 et inférieur à 1.4

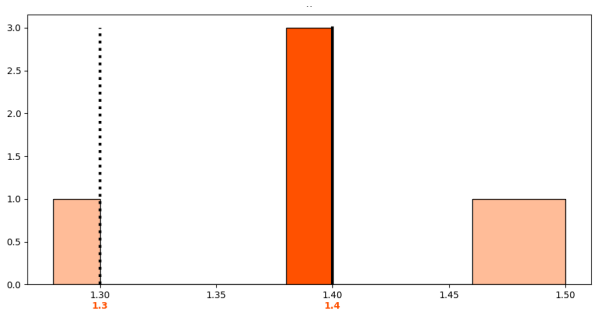

FIGURE 15 – Prédicat avec changement de hauteur, 5 objets sur 1 attribut. Les prédicats sont supérieur à 1.3 et inférieur ou égal à 1.4

Afin d'indiquer ce changement de lecture de l'histogramme, on rajoute des lignes délimitant les prédicats, en pointillé si le prédicat n'est pas inclus, en trait plein sinon. La hauteur de ces lignes n'est pas arbitraire: elle correspond à la hauteur de la barre la plus proche qui se trouve dans la zone du prédicat. Si un des prédicats d'un attribut correspond à une des bornes du diagramme de ce même attribut, et que ce prédicat n'est pas inclus, on ajoute une barre à l'histogramme.

Si les deux prédicats d'un attribut sont les mêmes, on ajoute une ligne de la taille du nombre d'éléments de la valeur du prédicat, que l'on colore de la couleur des barres de la zone des prédicats. La **figure 16** représente un exemple d'histogramme de ce cas particulier.

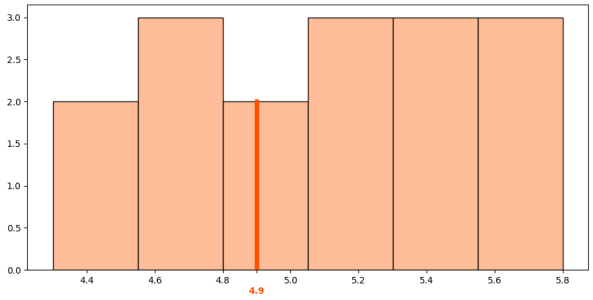

FIGURE 16 – Histogramme avec prédicats égaux: inférieur ou égal à 4.9 et supérieur ou égal à 4.9

On rajoute également à l'histogramme une meilleure version de la courbe du premier histogramme, tracée en prenant les hauteurs de l'histogramme comme données, au lieu d'un *kde*.
Si l'on utilise seulement les points de l'histogramme, la courbe ne sera qu'une liste de points liés par des segments.
On utilise donc l'interpolation pour générer des points entre les points déjà établis, en utilisant la fonction *smoothstep* afin d'adoucir la courbe sans dépasser les points originaux.
Un histogramme avec cette courbe est montré sur la **figure 17**.

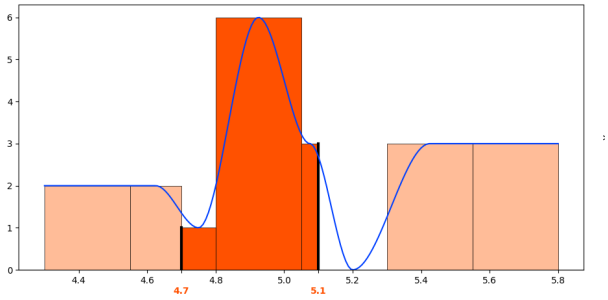

FIGURE 17 – Histogramme avec courbe. Les prédicats sont supérieur ou égal à 4.7 et inférieur ou égal à 5.1

3 Visualisations restantes

GALACTIC a pour objectif de traiter des données hétérogènes, contenant tout types de données. Seulement représenter l'ensemble des données et les prédicats pour des données numériques ne suffira donc pas.
Cependant, elle se révèle comme étant potentiellement la plus compliquée à traiter des données "classiques" (numérique, booléen, catégoriel), et l'avancement sur les autres types de données devrait être plus rapide, car il se trouve que les manières choisies pour les représenter sont très similaires.

3.1 Visualisation numérique - ajout de clusters

La visualisation numérique est pour l'instant la plus avancée de toutes, mais elle requiert tout de même des améliorations. Notamment, il serait intéressant que la visualisation affiche le *cluster* en plus des prédicats, car chaque objet du *cluster* doit valider tous les prédicats sur tous ses attributs. C'est cependant un objectif difficile à réaliser sans une intégration à GALACTIC, car il faudrait pouvoir récupérer les prédicats et *clusters* après une analyse pour être sûr de la validité des résultats.

3.2 Visualisation catégorielle

Pour les données catégorielles, l'objectif est de les représenter dans un diagramme en barres.
Ainsi, on retrouverait chaque attribut en abscisse, ayant une barre pour chacune des valeurs possibles, et en ordonnée le compte de chacune de ces valeurs.

Le prédicat des données catégorielles étant un ensemble de valeurs catégorielles, il suffira pour le représenter de colorer certaines barres selon leur inclusion dans la liste du prédicat.

3.3 Visualisation booléenne

La visualisation booléenne se trouve être un cas particulier de la visualisation catégorielle, où chaque attribut n'a que deux valeurs possibles, *true* et *false*. Le prédicat booléen est un ensemble d'attributs, indiquant quels attributs doivent avoir la valeur *true* pour qu'un objet fasse partie du *cluster*. Il suffit donc de reprendre la même visualisation que pour les données catégorielles, en colorant les barres représentées par le prédicat.

4 Conclusion

Ce début de stage au L3I fut une très intéressante première expérience dans le monde professionnel de l'informatique et de la recherche.
En travaillant sur le projet GALACTIC, j'ai pu travailler dans une petite équipe motivée, travaillant sur un projet concret ayant pour but d'être utilisé professionnellement dans un milieu universitaire. Cela permet un point de vue différent sur la programmation, qui renforce l'importance d'un code clair et compréhensible, car mon code devra être réutilisé dans le futur par d'autres étudiants ou chercheurs.

Sur le point technique, j'ai pu largement approfondir mes connaissances du module *Matplotlib*, aussi bien ses forces et ses faiblesses, ainsi que d'appendre le fonctionnement du module *Seaborn*.
J'ai également pu apprendre beaucoup sur la visualisation de données, notamment les enjeux et les principaux obstacles de ce type de projet.

Enfin, et potentiellement le plus important, j'ai pu apprendre à travailler à un rythme plus professionnel.
N'ayant pas l'habitude de coder de manière aussi cadrée et régulière, ce fut une expérience des plus intéressantes qui m'a beaucoup appris sur la gestion du temps et des objectifs.

Je ne suis cependant pas à la fin de mon stage, et il me reste ainsi beaucoup à apprendre.
De nombreux objectifs sont encore à atteindre, afin de pouvoir ajouter ma modeste contribution au projet GALACTIC.

Références

4.1 [1] Plate-Forme Intelligence Artificielle (PFIA) 2026

Lien: https://pfia26.cril.fr/

Dernière consultation: 29/05/2026

4.2 [2] Documentation de Gitlab

Lien: https://docs.gitlab.com/

Dernière consultation: 29/05/2026

4.3 [3] Documentation de Python

Lien: https://www.python.org/

Dernière consultation: 29/05/2026

4.4 [4] Documentation de Matplotlib

Lien: https://matplotlib.org/stable/users/explain/figure/api_interfaces.html#api-interfaces

Dernière consultation: 29/05/2026

4.5 [5] Documentation de Seaborn

Lien: https://seaborn.pydata.org/tutorial.html

Dernière consultation: 29/05/2026

4.6 [6] Documentation de Pandas

Lien: https://pandas.pydata.org/docs/

Dernière consultation: 29/05/2026

4.7 [7] Documentation de Numpy

Lien: https://numpy.org/doc/stable/

Dernière consultation: 29/05/2026

4.8 [8] Kernel Density Estimation (KDE)

Lien: https://en.wikipedia.org/wiki/Kernel_density_estimation

Dernière consultation: 29/05/2026