

# Rapport Projet Intégré & Gestion de projet

Groupe Interface-Data Vis-Analyse

**TD 2**

**Master 1**

**Promotion 2020-2021**

**7 décembre 2020**

**Master 1 Informatique  
Parcours Informatique**

**UE** Projet Intégré

**UCE** Gestion de Projet

**Responsables**

Mme. Fadila Bentayeb

Mr. Omar Boussaid

## Sommaire

<b>Titre</b>	<b>1</b>
<b>Sommaire</b>	<b>2</b>
<b>1 Introduction Générale</b>	<b>4</b>
<b>2 Description du Rendu</b>	<b>5</b>
<b>3 GROUPE INTERFACE</b>	<b>6</b>
3.1 Noms et prénoms des participants . . . . .	6
3.2 Introduction . . . . .	6
3.3 Gestion de Projet . . . . .	6
3.3.1 Organisation de l'équipe . . . . .	6
3.3.2 Répartition des tâches et organisation des Sprints . . . . .	7
3.3.3 Organisation du Sprint . . . . .	8
3.3.4 Le Burn Down Chart . . . . .	8
3.3.5 Mock-up ou prototype de l'application . . . . .	10
3.4 Projet Intégré . . . . .	11
3.4.1 Conception des interfaces Graphique . . . . .	11
3.4.2 Pré-traitement des données . . . . .	23
3.5 Conclusion . . . . .	29
<b>4 GROUPE DATA VISUALISATION 1</b>	<b>30</b>
4.1 Noms et prénoms des participants . . . . .	30
4.2 Introduction . . . . .	30
4.3 Les données & les outils . . . . .	30
4.4 Gestion de projet & Modèle SCRUM . . . . .	31
4.4.1 Les rôles . . . . .	31
4.4.2 Product Backlog & User stories . . . . .	31
4.4.3 Le burndown chart . . . . .	32
4.4.4 Trello : une aide précieuse . . . . .	32
4.4.5 Les sprints . . . . .	33
4.4.6 Conclusion sur la méthode SCRUM . . . . .	39
4.5 Projet Intégré . . . . .	40
4.5.1 Introduction . . . . .	40
4.5.2 Méthode pour exécuter le programme . . . . .	40
4.5.3 Le chargement des données . . . . .	40
4.5.4 L'interface . . . . .	41
4.5.5 Les requêtes . . . . .	43
4.5.6 Le graphe . . . . .	44
4.6 Conclusion . . . . .	46
<b>5 GROUPE DATA VISUALISATION 2</b>	<b>47</b>
5.1 Noms et prénoms des participants . . . . .	47
5.2 Introduction . . . . .	47
5.3 Gestion de Projet . . . . .	47
5.3.1 L'approche SCRUM . . . . .	47
5.3.2 Gestion du projet selon la vision du Scrum Master . . . . .	48
5.3.3 Vision du livrable pour le client selon le Product Owner . . . . .	52
5.3.4 Table des annexes . . . . .	55

5.4	Projet Intégré . . . . .	67
5.4.1	Présentation et importation des données . . . . .	67
5.4.2	Création d'une interface et requêtes associées . . . . .	67
5.4.3	Création des requêtes . . . . .	72
5.5	Visualisation des données . . . . .	74
5.5.1	Premières visualisations . . . . .	74
5.5.2	Visualisation en fonction de l'interface . . . . .	76
5.6	Utilisation de l'application . . . . .	84
5.6.1	Exécutable . . . . .	84
5.6.2	Distribution Python . . . . .	85
5.7	Conclusion . . . . .	86
<b>6</b>	<b>GROUPE ANALYSE</b>	<b>87</b>
6.1	Noms et prénoms des participants . . . . .	87
6.2	Introduction . . . . .	87
6.3	Gestion de Projet . . . . .	87
6.3.1	Introduction . . . . .	87
6.3.2	Besoin du client . . . . .	87
6.3.3	Fonctionnement de l'équipe . . . . .	88
6.3.4	Réalisation du projet . . . . .	89
6.3.5	Conclusion . . . . .	95
6.4	Projet Intégré . . . . .	97
6.4.1	Introduction . . . . .	97
6.4.2	Nettoyage des données . . . . .	97
6.4.3	Analyse des communautés d'auteurs – co-auteurs . . . . .	98
6.4.4	Création de thème à partir des titres des publications . . . . .	109
6.4.5	Analyse des thèmes de chaque auteur . . . . .	132
6.4.6	Conclusion . . . . .	134
<b>7</b>	<b>Intégration</b>	<b>135</b>
7.1	Gestion de Projet . . . . .	135
7.2	Projet Intégré . . . . .	135
7.2.1	Intégration de l'interface . . . . .	135
7.2.2	Intégration des fonctionnalités . . . . .	136
<b>Bibliographie</b>		<b>137</b>

## 1 Introduction Générale

Les services d'un réseau d'information sont un moyen de diffusion d'information mis à profit pour récolter des données sur un sujet précis.

Le Digital Bibliography & Library Project (DBLP) est un site internet répertoriant un catalogue de bibliographies en informatique.

Initialement connu sous le nom de DataBase systems and Logic Programming, DBLP a été conçu au départ comme un catalogue de bibliographies sur les bases de données et la programmation logique. Il s'est ensuite étendu à tout le domaine informatique en changeant d'acronyme, signifiant maintenant Digital Bibliography & Library Project.

Ces services de réseau d'information ne cessent de croître avec le nombre de données qui circule quotidiennement. Il s'avère alors intéressant de représenter ces données afin d'en tirer des analyses pertinentes sur ces grandes quantités d'information partagées en temps réel par de nombreux utilisateurs.

Dans le cadre de la première année du Master Informatique de l'Université Lyon 2, il nous a été donné de répondre à ce besoin de représentation et d'analyse en mettant en place un projet suivant deux optiques : la gestion du projet, et le développement de la solution. Dans la première nous avons porté notre attention sur la façon d'organiser et de répartir le travail au sein de l'équipe, et dans la deuxième nous avons mis à profit cette organisation pour fournir un produit utilisable.

Le projet est constitué de différentes étapes à savoir, la création d'une interface et la gestion des données utilisées, la visualisation des données sous forme de graphes dynamiques et enfin l'analyse des données bibliographiques.

## 2 Description du Rendu

Les consignes du projet nous ont été délivrées **le Mardi 30 septembre 2020**. Nous avions au total 16 séances de Travaux dirigés pour la réalisation de ce projet.

Ce rapport est une description détaillée de la réalisation du projet du Groupe de TD 2. Il contient deux parties essentielles : la première relève de la gestion de projet de chaque équipe, et la deuxième relève d'une description sur la conception logiciel du projet.

**2.0.0.1 Date limite du rendu :** Lundi 7 décembre 2020

**2.0.0.2 Structure du rapport**

- Noms et prénoms des participants :
- Introduction
- **Partie 1:** Gestion de projet
- **Partie 2:** Projet Intégré
- Conclusion et perspectives

### 3 GROUPE INTERFACE

#### 3.1 Noms et prénoms des participants

- **Sara Lehtihet** (Scrum Master)
- **Remy Kurdoncuff** (Product Owner)
- **Thomas Giovannini**
- **ZineEddine Lakhdari**
- **Thibaut Martinet**
- **Elliot Jacquemet**

#### 3.2 Introduction

Lorsque nous avons décidé de réaliser la partie interface de ce projet, nous avons rapidement pris conscience que notre travail se diviserait en deux missions distinctes mais tout aussi importantes.

La première était la récupération des données, le nettoyage de celles-ci, puis la conversion en données exploitables pour les autres groupes.

La seconde était le développement de l'interface graphique qui accueillerait l'analyse et les visualisations des autres groupes.

Nous avons décidé de diviser notre groupe en deux sous-groupes, un pour chacune de ces missions.

#### 3.3 Gestion de Projet

Lors de la première séance de TD Gestion de projet qui a eu lieu le Jeudi 01 octobre 2020, nous avons défini le scrum master et le product owner.

Rôle	Nom
Scrum Master et membre de l'équipe IHM	Sara Lehtihet
Product Owner et membre de l'équipe de développement	Remy Kurdoncuff
Membre de l'équipe de développement	Thibaut Martinet
Membre de l'équipe IHM	Thomas Giovannini
Membre de l'équipe de développement	ZineEddine Lakhdari
Membre de l'équipe IHM	Elliot jacquemet

**Table 1.** Membres de l'équipe Scrum

##### 3.3.1 Organisation de l'équipe

Comme on peut le voir dans la table 1, voici comment nous nous sommes répartis en deux groupes :

- **demi-groupe 1:** S'occupe de la partie traitement des données, et se compose de :
  - Thibaut Martinet
  - Remy Kurdoncuff
  - ZineEddine Lakhdari
- **demi-groupe 2 :** S'occupe de la partie conception des interfaces graphiques, et se compose de :
  - Sara Lehtihet
  - Thomas Giovannini

- Elliot Jacquemet

Nous organisons une réunion au début de chaque séance qui dure en moyenne 15 minutes afin de regarder l'avancement personnel de chaque membre du groupe. Cette réunion est dirigée par le Scrum Master (Sara) qui supervise le bon déroulement des backlogs et assure le respect des users stories et des sprints. Le product owner (Remy) quant à lui, son rôle est d'interagir avec le client et d'apporter au reste de l'équipe ses attentes.

Nous faisons des réunions quotidienne au début de chaque séance de TD que celle ci soit en projet intégré ou en gestion de projet et celle dans le but d'assurer le bon avancement de nos deux demi groupes.

Voici un aperçu des points traité lors des mélée quotidienne :

#### **TD Projet intégré : 30/09/2020**

- Exposition du sujet du projet.
- Création des groupes.
- Choix du sujet parmis interface, visualisation et analyse.

#### **TD Gestion de Projet : 01/10/2020**

- Définition du master Scrum et du product owner
- Définition de l'IDE a utilisé.
- Documentation sur DBLP.
- Conception d'une maquette pour l'interface.

#### **TD Projet intégré : 07/10/2020**

- Définition et répartition des tâches.
- Téléchargement des fichiers XML.
- Début de conception des interfaces Graphique.

### **3.3.2 Répartition des tâches et organisation des Sprints**

Code	Tâche	Nom étudiant	Etat
T1	Télécharger le fichier XML	ZineEddine	✓
T2	Parcourir intelligemment le fichier XML	Thibaut	✓
T3	Transformer le fichier XML en Json	Remy	✓
T4	Choisir une organisation de fichier CSV	Thibaut, ZineEddine, Remy	✓
T5	Extraction des années en Json	ZineEddine	✓
T6	Extraction des auteurs en Json	Remy	✓
T7	Extraction des mots clés en Json	Remy	✓
T8	Extraction des publications en Json	Thibaut	✓
T9	Organiser l'écriture de données dans le fichiers CSV	Thibaut, ZineEddine, Remy	✓
T10	Optimiser le parcours du fichier XML	Thibaut	✓
T11	Écriture des fichiers simples	Thibaut	✓
T12	Écriture du fichier publication_author	Remy	✓
T13	Écriture du fichier publication_year	ZineEddine	✓
T14	Écriture du fichier publication_keywords	Thibaut	✓
T15	Nettoyage des keywords entièrement numériques	Remy	✓
T16	Optimiser le parcours du fichier XML	Thibaut	✓
T17	Rédaction du rapport	Thibaut, ZineEddine, Remy	✓

**Table 2.** Table des tâches du sous groupe Data

Code	Tâche	Nom étudiant	État
T18	Conception de l'interface principale avec TKinter	Sara	✓
T19	Conception de l'interface visualisation avec TKinter	Thomas	✓
T20	Conception du pop-up et barre de chargement avec Tkinter	Elliot	✓
T21	Documentation sur les librairies qui existent	Sara, Thomas, Elliot	✓
T22	Recherche de fonctionnalité dans la doc de pyqt5	Sara, Thomas, Elliot	✓
T23	Conception de l'interface Principale avec PyQt5	Sara	✓
T24	Conception de l'interface visualisation avec PyQt5	Thomas	✓
T25	Conception du pop-up et barre de chargement avec PyQt5	Elliot	✓
T26	Intégration et mise à jour de l'interface Visualisation	Thomas, Elliot	✓
T27	Conception de l'interface analyse avec PyQt5	Sara, Thomas	✓
T28	Intégration et des données à l'interface principale	Elliot, Thibaut	✓
T29	Conception de l'interface analyse avec PyQt5	Sara, Thomas	✓
T30	Intégration des fonctionnalité de l'interface Analyse	Sara	✓
T31	Intégration de l'interface visualisation 2	Thomas	✓
T32	Rédaction du rapport	Sara, Thomas, Elliot	✓

**Table 3.** Table des tâches du sous groupe Conception de l'interface Graphique

Sprint	Tâche
Sprint 1	T1,T2,T3,T18,T19,T20,T21,T22
Sprint 2	T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T23,T24,T25,T16,T27,T28,T29,T30,T31,T32

**Table 4.** Table contenu des Sprint

### 3.3.3 Organisation du Sprint

Dans le backlog sprint, que l'on peut voir à la figure 1, chaque tâche est représentée par un post-it, et la position de ce dernier dans le tableau indique son avancement.

Nous avons organisé toutes nos tâches en deux Sprints ; le premier est identifié par une barre de couleur jaune, et le deuxième par une barre de couleur orange.

Le logiciel que l'on a utilisé pour le suivi des sprints est Trello[13].

**Le sprint backlog** est une partie des user stories du backlog produit que l'équipe s'engage à livrer d'ici la fin du sprint.

**Le backlog produit** est une liste priorisée des fonctionnalités à développer ou améliorer dans le cadre d'un service / produit informatique (logiciel, application mobile, etc.).

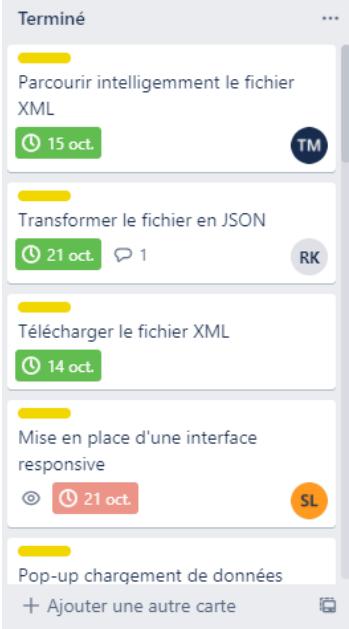
### 3.3.4 Le Burn Down Chart

Nous avons réalisé un burn down chart, que l'on peut voir à la figure 2, qui nous a permis de représenter sous forme graphique l'évolution de la quantité de travail restante pour une période donnée.

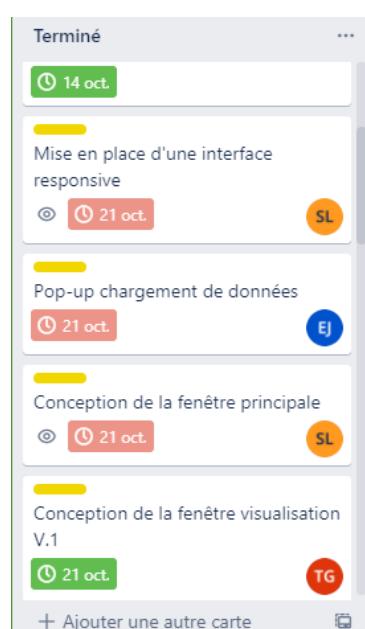
L'unité de temps est le nombre de séances de TD, qui durent 1h45 ; la quantité de travail étant quant à elle exprimée en nombre de tâches à réaliser.

Le graphique se lit de gauche à droite, le point de départ étant situé à l'extrême gauche, sur le jour 0, le point final situé le plus à droite représentant le dernier jour du sprint.

Le scrum master a mis à jour le burn down chart à chaque fin de sprint, cela permet d'obtenir rapidement et facilement un indicateur fiable de l'état d'avancement des développements.



**(a) Sprint 1**

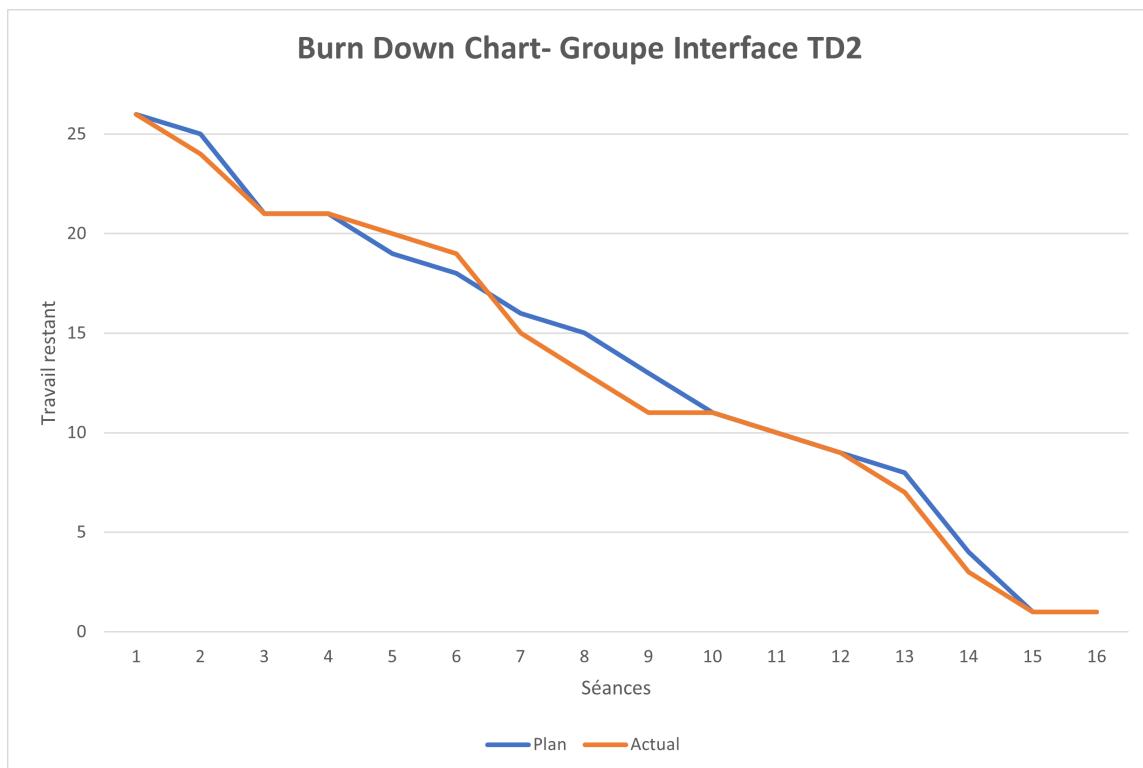


**(b) Suite Sprint 1**



**(c) Sprint 2**

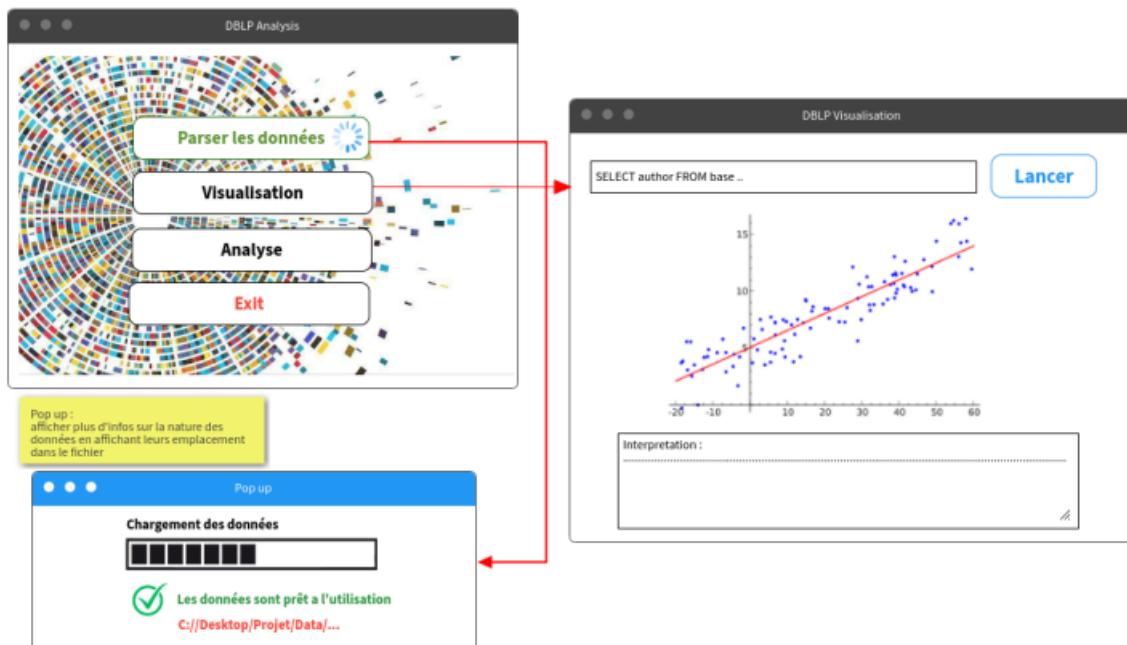
**Figure 1.** Exemple des Sprints



**Figure 2.** Burn Down Chart

### 3.3.5 Mock-up ou prototype de l'application

Nous avons réalisé un prototype d'interface utilisateur en ligne sur le site MockFlow[11]. Le but de ce prototype est de permettre de se projeter et de tester aussi bien le fond (le contenu) que la forme (le design). En ajoutant les bonnes couleurs et les bonnes images, nous pourrons avoir un aperçu plus réaliste de votre futur application.



**Figure 3.** maquette de l'interface Graphique

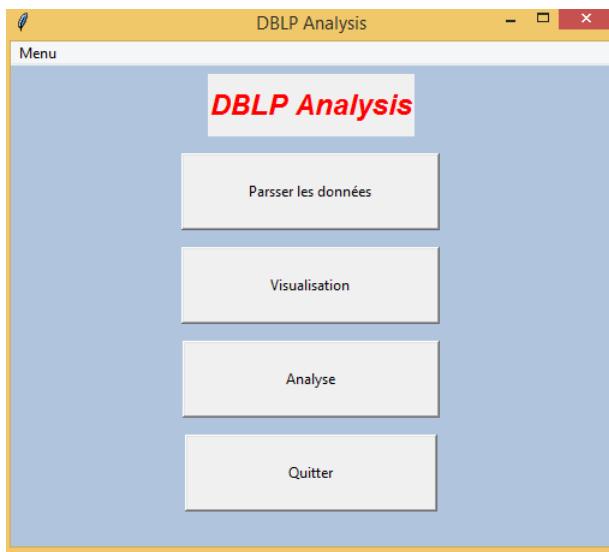
## 3.4 Projet Intégré

### 3.4.1 Conception des interfaces Graphique

**3.4.1.1 Conception de l'interface Principale avec Tkinter** Il a été primordial de faire un prototype car celui-ci servira de base à la phase de conception de notre application.

La conception de l'interface principale de l'application qui a pour objectif de fournir une porte d'accès aux fonctionnalités des autres groupes visualisations et analyse.

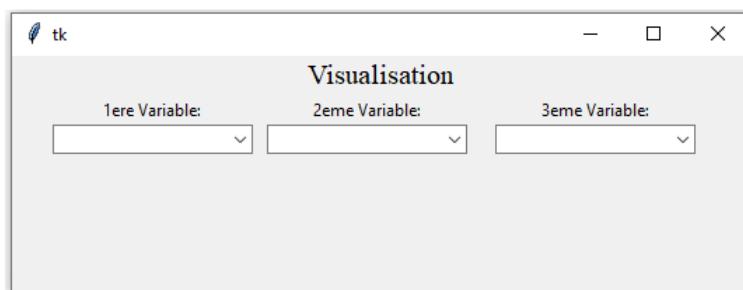
Nous avons donc pris comme patron le prototype qu'on avait fait et nous avons utilisé pour cela la bibliothèque Tkinter [5] de python pour ça mise en forme.



**Figure 4.** Interface Principale avec Tkinter

**3.4.1.2 Conception de l'interface Visualisation avec Tkinter** La fenêtre visualisation devait se composer comme suit : tout d'abord des filtres qui permettent à l'utilisation de d'affiner son analyse, ensuite une partie visualisation qui affiche un graphique et enfin une partie analyse pour le groupe analyse.

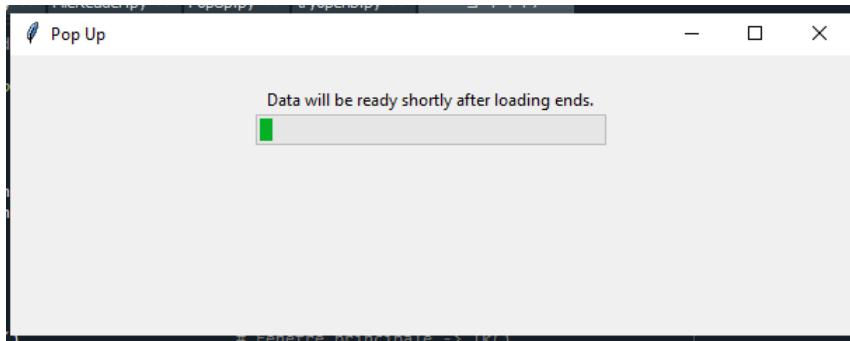
Pour créer la première version de la fenêtre visualisation nous avons utilisé la librairie tkinter [5] de python. Cette librairie permet lorsqu'on exécute le code, de créer une interface graphique.



**Figure 5.** Interface visualisation avec Tkinter

**3.4.1.3 Conception du Pop-up avec tkinter** Pour notre première version de la pop up, nous avions choisi d'utiliser la librairie graphique interne de Python : Tkinter. Cette librairie a une fonction « `TopLevel()` » imitant le comportement d'une pop classique (contrairement à la fonction « `Tk()` » qui imite le comportement d'une fenêtre classique).

On a donc réalisé une première version de la pop-up avec une barre de chargement et un court texte indiquant que la fin du chargement correspond aux données :



**Figure 6.** Pop-up et barre de chargement avec Tkinter

**3.4.1.4 Documentation sur les librairies existantes** Pendant le développement des trois fenêtres avec la bibliothèque Tkinter [5], on s'est vite rendu compte des limites de celle ci. En effet, le placement des objets dans l'espace n'est pas du tout intuitif et l'esthétique est très limitée. On a donc interrompu le développement sur tkinter le **21 octobre 2020** pour changer de librairie.

On a décidé de chercher une autre technologie robuste, fiable, et efficace qui nous permettra de faire un design plus moderne.

Sara nous a présenté la librairie « **PyQt5** » ainsi que l'IDE **Qt Designer** permettant de rapidement créer des interfaces agréables à l'œil de l'utilisateur (format .ui que l'on convertit grâce à Conda). Nous avons donc réalisé une nouvelle version sur cet outil.

**3.4.1.5 Recherche de fonctionnalité dans la doc de PyQt5** **Qt** est un ensemble d'outils d'aide à la création de logiciels.

L'objectif de Qt est de faciliter le travail des développeurs de différentes manières :

- en étant multiplateforme : les applications créées avec Qt sont compatibles avec différents systèmes d'exploitation et le développeur peut se concentrer sur son applicatif plutôt que sur les détails spécifiques des plateformes visées.
- en étant natif : les applications ne subissent pas de pénalité en terme de performances et d'intégration à la plateforme (ergonomie, design, accès au système).
- en étant complet : tous les ingrédients de base pour la réalisation d'applications sont intégrés, y compris des couches graphiques pour les interfaces utilisateurs.

On utilisant la librairie PYQT5 et en passant par une application qui s'appelle QtDesigner, on peut directement développer l'interface graphique et ensuite convertir cette interface en code.

**3.4.1.6 installation de PYQT5 & QT Designer sur Windows** En exécutant cette commande dans une invite de commande, PYQT5 et QT Designer seront installés avec la distribution **pip** de Python :

```
pip install PyQt5
pip install PyQt5-tools #permet de télécharger les outils dont qt5 designer
```

Qt Designer est un programme livré avec Qt qui permet de dessiner nos fenêtres visuellement. Plus encore, Qt Designer nous permet aussi de modifier les propriétés des widgets, d'utiliser des layouts, et d'effectuer la connexion entre signaux et slots.

Il nous permet donc de gagner du temps et d'éviter les tâches répétitives d'écriture du code de génération de la fenêtre.

**Remarque :** Voici le Chemin d'accès à QT5 Designer :

C:/Python/Python37-32/Lib/site-packages/pyqt5\_tools/Qt/bin/designer.exe

L'IDE **Qt Designer** permet la création d'interface graphique en y déposant les différentes composantes graphique qu'il propose par simple glisser-déposer. On peut voir l'interface du logiciel à la figure 7.

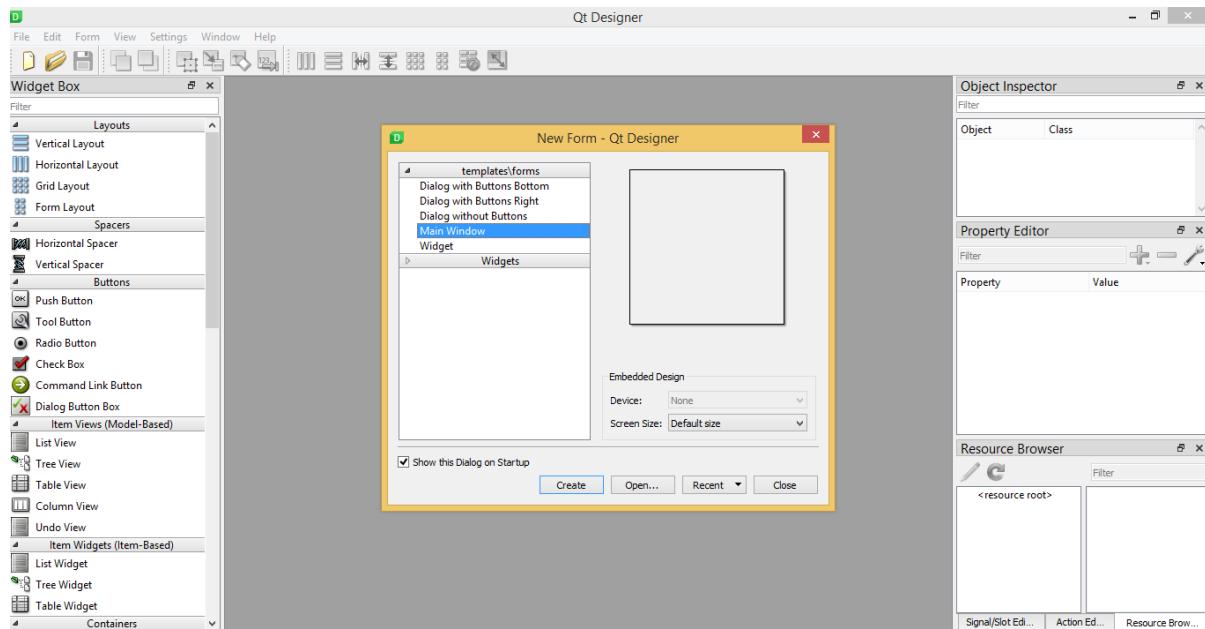


Figure 7. Qt Designer IDE

**Remarque :** Possibilité d'installer Qt et Qt Designer depuis le site officiel [12].

Mais le mieux serait de les installer depuis une invite de commande pour éviter les erreurs lors de la génération du fichier .py.

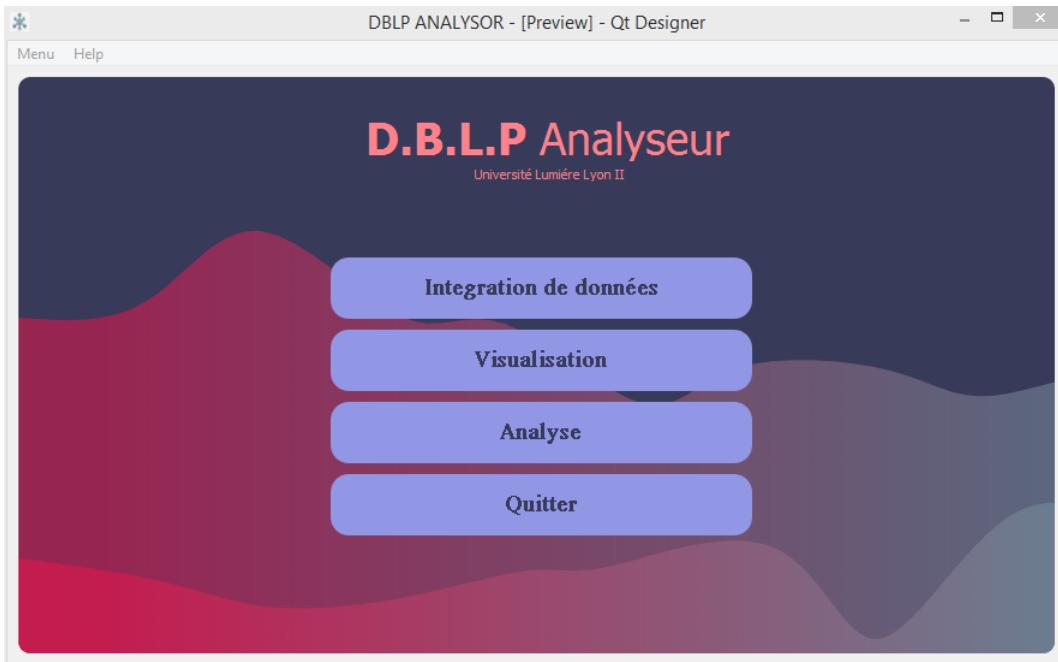
Qt designer nous permet de concevoir des interfaces graphique en y déposant les composantes graphiques depuis la palette des widget box que cet IDE propose et nous permet d'enregistrer ces interfaces sous format .ui.

On peut générer le fichier .py depuis le fichier .ui à l'aide de la commande suivante, toujours dans une invite de commande :

```
pyuic5 -x interface.ui -o interface.py -x
```

**3.4.1.7 Conception de l'interface principale avec PyQt5** A l'aide de l'IDE Qt Designer nous avons pu refaire la conception de l'interface principale pour un rendu plus ergonomique, voici donc la version v.2 de l'interface principale avec cet outil :

**Remarque :** Nous nous sommes inspirer du site [9] pour avoir l'ensemble des couleurs utilisé dans nos interface.



**Figure 8.** Interface Principale avec Qt Designer

Pour avoir ce rendu là, il nous a fallu créer une nouvelle fenêtre brute "Main Window", ou nous ajoutant les composants graphique tel que des boutons ou label pour le titre.

En premier lieu, nous avons changer la couleur de la fenêtre en violet et cela en faisant un clic droit sur celle-ci et en accédant a l'option "change stylesheet" et en y insérant le code css suivant :

```

1 {
2     background-color: rgb(56, 58, 89);
3 }
```

Ensuite, nous avons ajouter une image de fond sous forme de "Waves" que nous avons customiser sur le site [14]

Pour ce faire, voici les étapes à suivre :

1. Accès a "change stylesheet" depuis un clic droit sur la fenêtre principale.
2. Sélectionné "add Ressources /Image".
3. sélectionné "Edit resource" qui se présente sous forme d'un craillon.
4. sélectionné "new resource file" qui se trouve en bas à gauche de la fenêtre sous forme d'une page blanche.
5. Nommé et enregistré le fichier(qr).
6. sélectionné "add prefix" en bas à droite, cela créera un prefixe nommé "new prefix" .
7. sélectionné "New prefix" et cliqué sur " add files" et y importer l'image souhaiter.
8. ces manipulation rajoute la ligne de code suivante :

```

1  {
2      background-image: url(:/newPrefix/svg.png);
3  }

```

Voici le script css qui nous a permis de faire la mise en forme des boutons ( buttons arrondis, couleurs des boutons, text..)

```

1 {
2     QPushButtton{
3         background-color:rgb(145, 150, 229);
4         border-radius: 15px;
5         color:rgb(56, 58, 89);
6         opacity: 0.9;
7     }
8 }

```

**Génération du fichier.py à partir du fichier.ui :** En utilisant la commande décrite précédemment, nous obtiendrons le code python de cette interface

**Remarque Importante :** Lorsque nous utilisons des images dans notre interface avec qt designer, celle ci sera sous forme de ressource sous format **.qrc**

Le fichier Python de cette ressource (.qrc) doit obligatoirement être générer pour éviter des erreurs lors de la compilation du script python propre à cette interface.

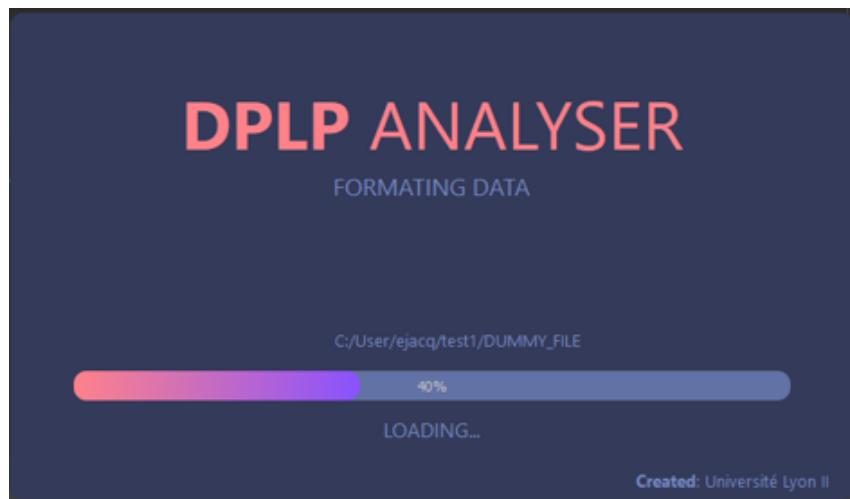
Pour cela nous avons exécuter cette commande dans le CMD :

```
pyrcc5 resource.qrc -o resource.py
```

**3.4.1.8 Conception du pop-up avec PyQt5** L'interface de chargement 9 doit contenir plusieurs éléments essentiels. Une barre de chargement représentant l'avancement du traitement des données et le chemin où les données sont téléchargées.

Nous avons décidé de réaliser notre projet à l'aide de Qt Designer lors de la séance du 21/10/2020 (QtPy étant une autre librairie graphique). La Popup a été terminé pour la séance du 05/11/2020.

Comme stipulé précédemment, la barre de chargement doit représenter l'avancement du parseage des données. Pour cela, le sous-groupe data a implémenté une fonction permettant de vérifier le nombre de publications traitées. On se base donc là-dessus en faisant un rapport avec le nombre total de publications. Lorsque le chargement se termine, la fenêtre se ferme automatiquement pour nous laisser revenir au menu principal. Ces fonctionnalités ont été ajoutées lors des dernières séances du 02/12/2020. Nous avons aussi rendu la fenêtre sans bordures afin de donner un effet de popup (importance moindre par rapport au menu principal).

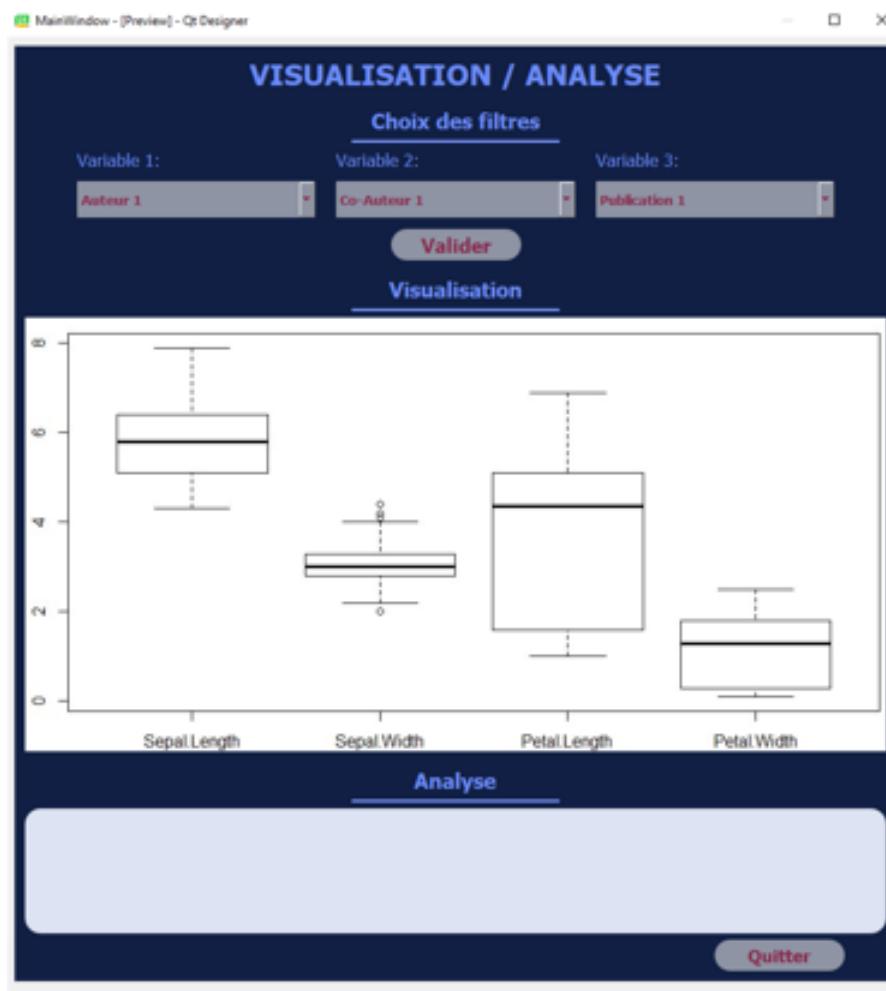


**Figure 9.** Pop-Up de téléchargement des données avec PyQt5

**3.4.1.9 conception de l'interface Visualisation :** La fenêtre visualisation devait de composer comme suit : tout d'abord des filtres qui permettent à l'utilisateur d'affiner son analyse, ensuite une partie visualisation qui affiche un graphique et enfin une partie analyse pour le groupe analyse. L'idée était donc de mettre le travail des groupes visualisations et du groupe analyse sur la même page.

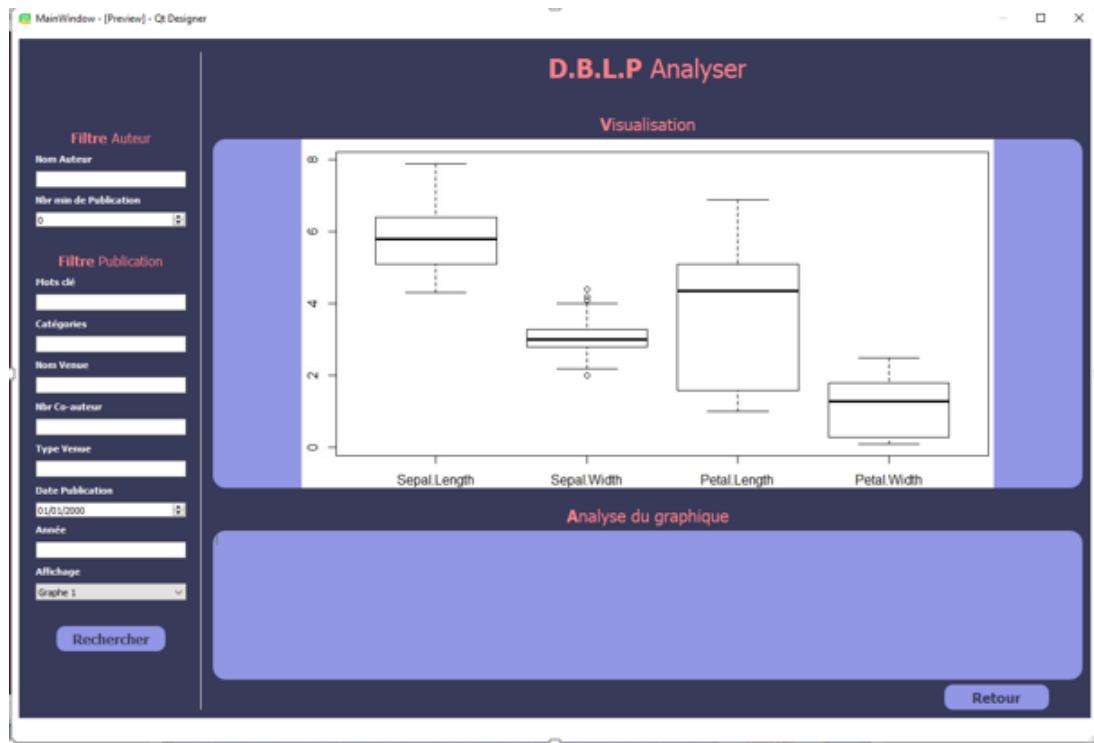
La deuxième version [10](#) (avec QT5) de l'interface graphique a été terminée le 5 novembre.

On peut donc clairement voir les filtres, la partie visualisation où l'image d'un graphique s'affiche pour la démo ainsi qu'une zone de texte pour l'analyse.



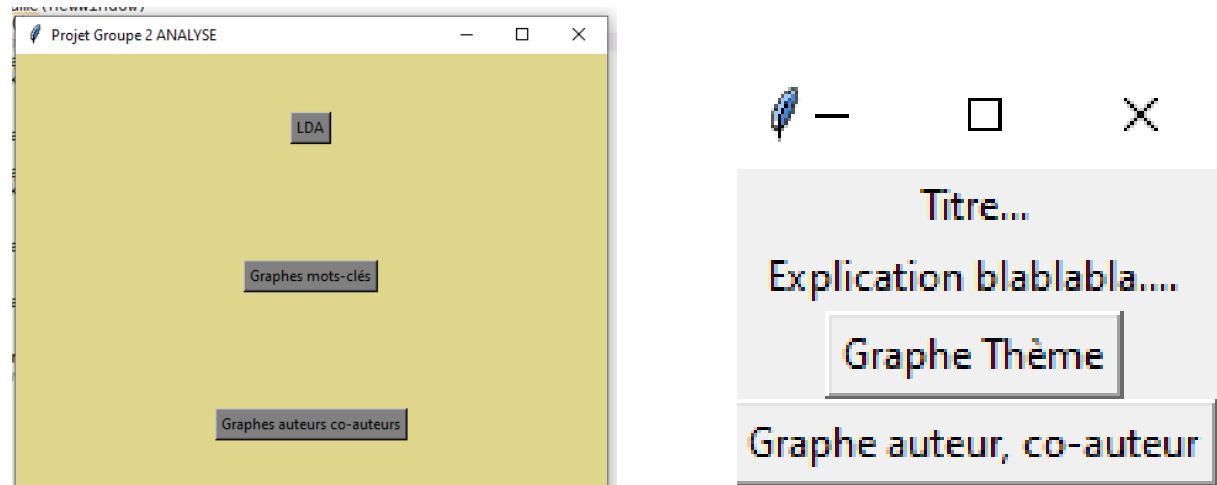
**Figure 10.** Interface visualisation version final

**3.4.1.10 Intégration et mise à jour de l'interface visualisation** Après consultation avec les groupes visualisation, nous avons décidé de changer la partie filtre. En effet, ces derniers n'étaient pas suffisamment nombreux et mal placés. Aussi, après consultation avec le groupe analyse, nous avons décidé de faire une fenêtre propre pour leur travail. Ainsi la zone de texte pour l'analyse présent dans la V2 de l'interface visualisation servira à écrire l'analyse du graphique affiché dans la V3 [11](#). La V3 a été terminé le 25 novembre.



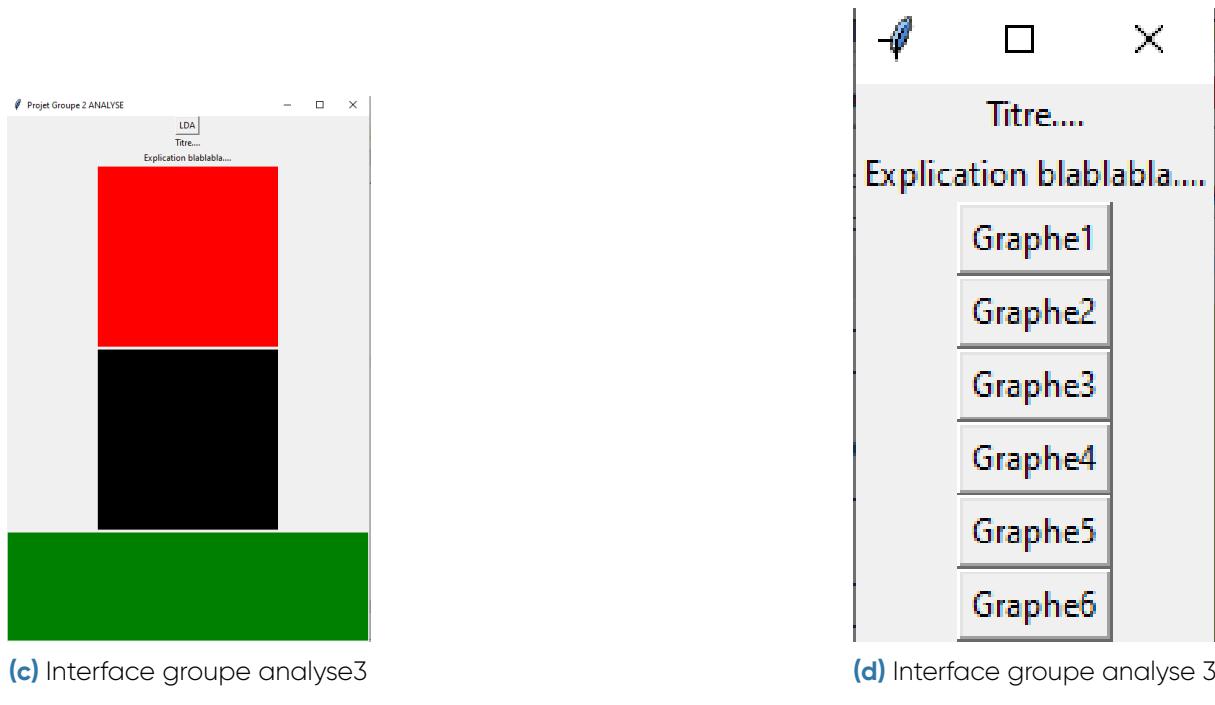
**Figure 11.** Interface visualisation version 3

**3.4.1.11 Conception de la fenêtre Analyse** Concernant l'interface du groupe analyse, ils l'ont développé avec la librairie TKinter. Celle-ci se composait de 4 fenêtres : une fenêtre d'accueil, une fenêtre pour la LDA, une pour les graphes mots-clés et enfin une pour les graphes auteurs-coauteurs.



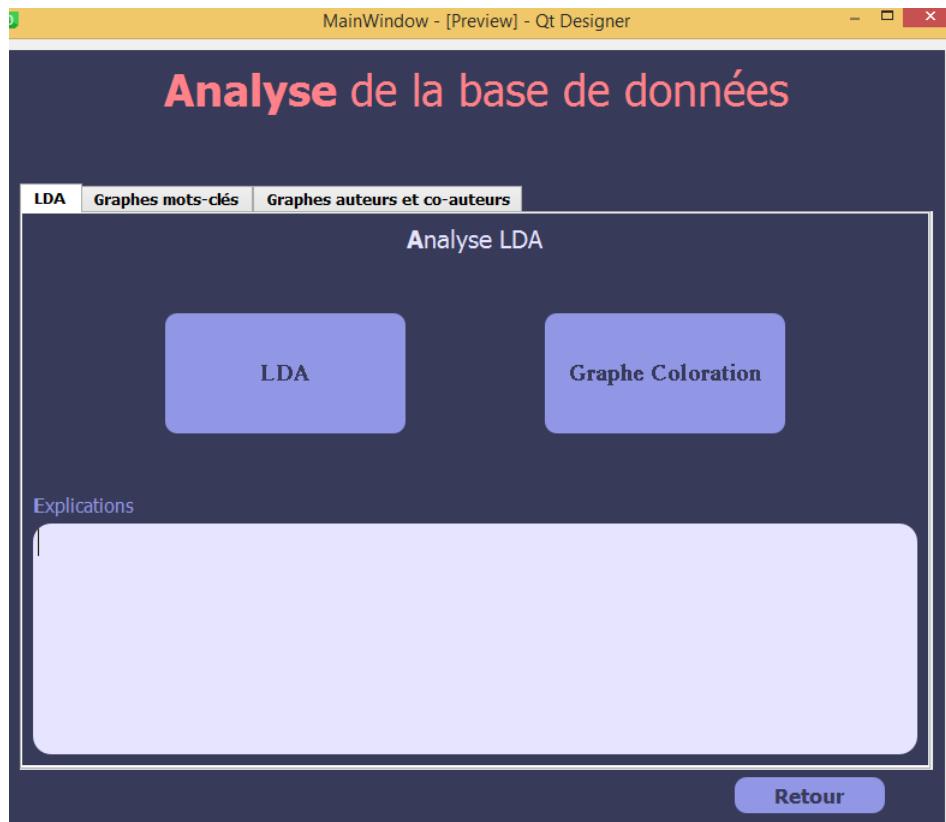
**(a)** Interface groupe analyse 1

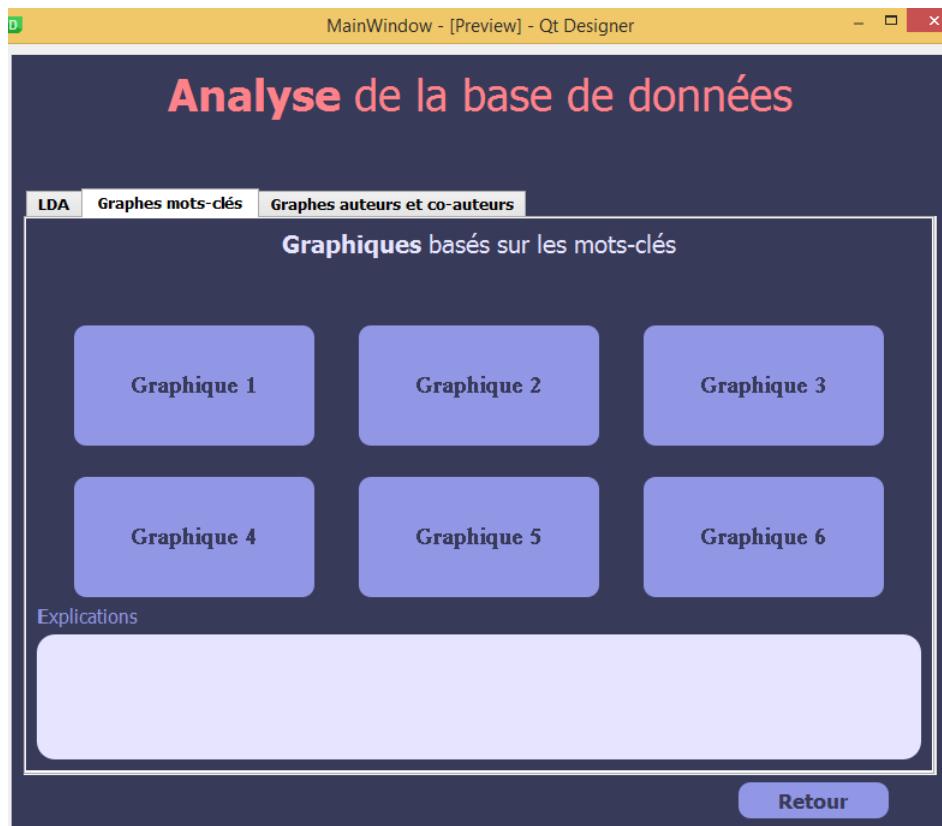
**(b)** Interface groupe analyse 2



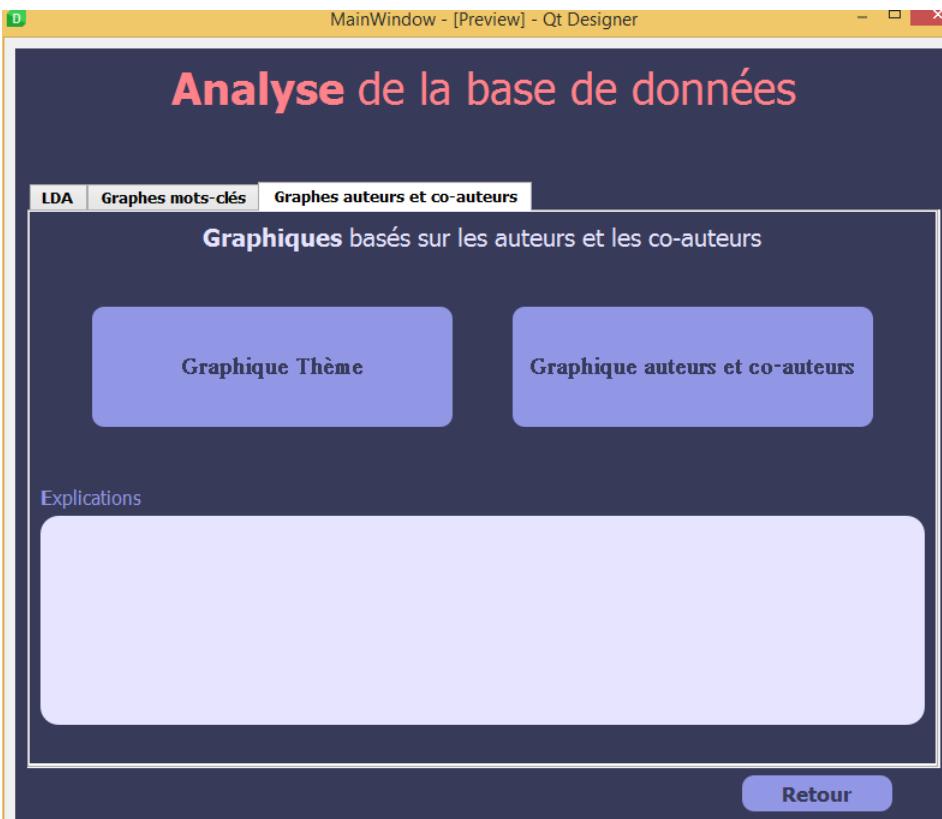
**Figure 12.** IHM du groupe analyse avant intégration

Nous avons donc décidé d'utiliser le widget **TabWidget** qui permet d'avoir plusieurs fenêtres en simultané sur la même interface. Pour changer d'affichage, il suffit de cliquer sur l'onglet correspondant. La conception de cette interface [13](#), en se basant sur le travail du groupe analyse, nous a pris environ 2 heures.





(b) Interface analyse 2



(c) Interface analyse 3

**Figure 13.** Interface analyse

**3.4.1.12 Intégration des données à l'interface** Nous avons intégré le parsing des données à l'ouverture du pop-up. Nous avons ainsi incrémenté la barre de chargement par rapport au nombre de publications traitées (on programme le timer de la barre de chargement de manière à s'actualiser à chaque nouvelle publication traitée grâce à la fonction iter()). Enfin lorsque la totalité des données sont traitées (barre de chargement à 100%), nous créons les fichiers csv dans le répertoire racine puis on ferme le pop-up.

Nom	Modifié le
.idea	04/12/2020
_pycache_	04/12/2020
venv	04/12/2020
exemple_boite_a_moustache_2	12/11/2020
svg	26/10/2020
download_data	12/11/2020
FileReader	04/12/2020
get_authors	13/11/2020
get_keywords	29/11/2020
get_publication	18/11/2020
get_year	14/11/2020
iconintegrate	03/11/2020
Image	12/11/2020
img	03/11/2020
interface_visu_v2	01/12/2020
isValid_keyword	27/11/2020
main	04/12/2020

Nom	Modifié le
.idea	04/12/2020 14:46
_pycache_	04/12/2020 15:58
venv	04/12/2020 14:34
author	04/12/2020 15:59
keyword	04/12/2020 15:59
publication	04/12/2020 15:59
publication_author	04/12/2020 15:59
publication_keywords	04/12/2020 15:59
publication_year	04/12/2020 15:59
year	04/12/2020 15:59
exemple_boite_a_moustache_2	12/11/2020 10:59
svg	26/10/2020 12:16
download_data	12/11/2020 11:05
FileReader	04/12/2020 14:31
get_authors	13/11/2020 09:25

Figure 14. Intégration data & UI

**3.4.1.13 Intégration des fonctionnalité de l'interface Analyse** Nous avons intégré les fonctionnalités de l'interface analyse en nous basant sur ce que le groupe nous a envoyé.

Cette intégration consiste à attribuer une fonction à chaque bouton qui permet d'ouvrir des pages HTML. Celles-ci représentent le résultat du parcours des fichiers CSV du groupe analyse.

Nous avons également intégré leurs explications dans des widgets du type "EditText". On peut voir l'aperçu de l'interface après l'intégration à la figure 15.

## Analyse de la base de données

[LDA](#) [Graphes mots-clés](#) [Graphes auteurs et co-auteurs](#)

**Graphes basés sur les mots-clés**

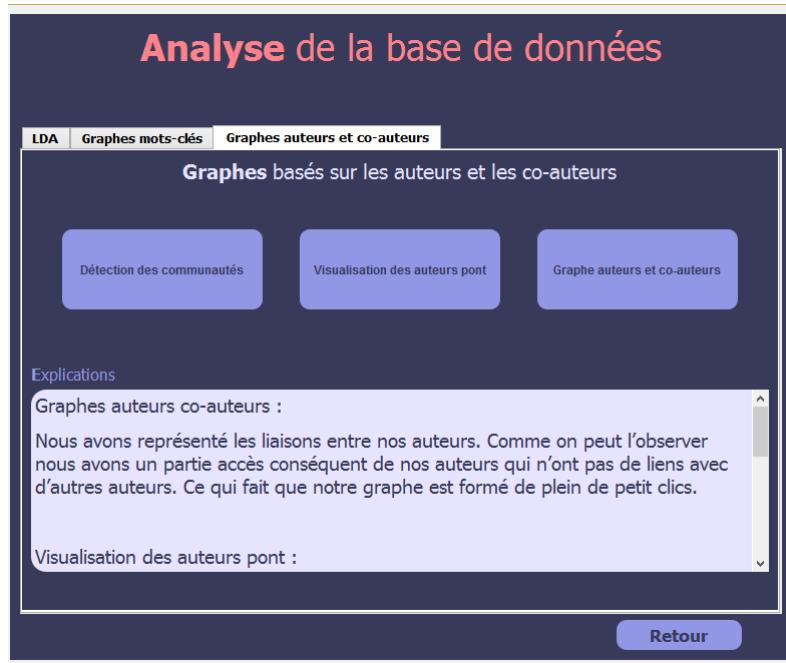
- Graphe des mots clefs avec la totalité des arêtes
- Graphe des mots clefs sans les arêtes avec un poids de 1
- Graphe des mots clefs sans les arêtes avec un poids inférieur ou égale à 5
- Graphe des mots clefs sans les arêtes avec un poids supérieur ou égale à 5
- Graphe des mots clefs sans les arêtes avec un poids supérieur à 1

**Explanations**

L'objectif de cette partie était de créer des graphes sur les mots clefs pour obtenir des groupes de mots clefs permettant d'attribuer des classes aux auteurs. Les graphes présents dans cette partie sont réalisés avec le même échantillon qui au départ contient 61714 lignes.

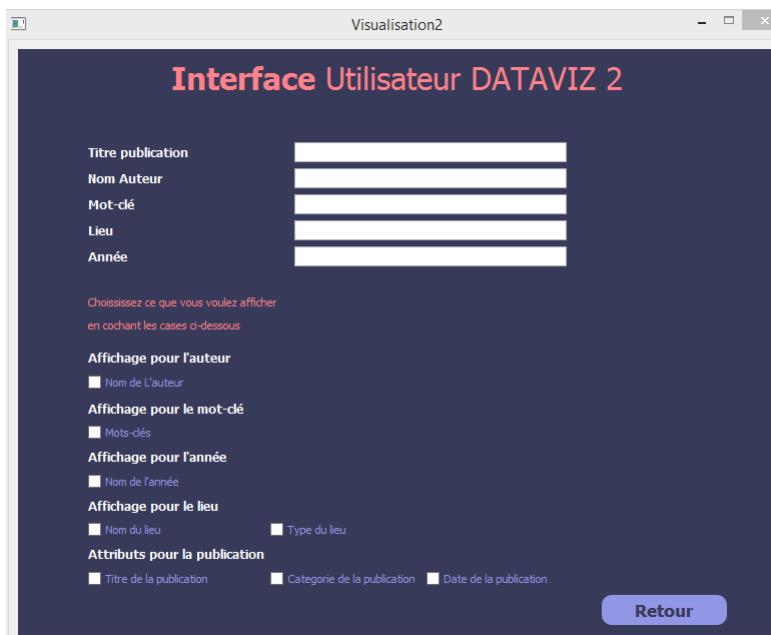
[Retour](#)

Figure 15. Intégration fonctionnalité de l'interface Analyse



**Figure 16.** Intégration 2 fonctionnalité de l'interface Analyse

**3.4.1.14 Intégration de l'interface visualisation 2** Nous avons développé une interface identique à celle du groupe visualisation 2. Celle-ci n'est pas fonctionnelle, c'est juste pour l'aspect visuel. Le développement de cette interface a duré environ 1 heure.



**Figure 17.** Intégration interface Visualisation Groupe 2

### 3.4.2 Pré-traitement des données

**3.4.2.1 Finalité :** L'objectif de notre travail a été de récupérer les données du site, de les pré-traiter, et de les enregistrer dans un certain format qui permet aux autres groupes d'y accéder efficacement.

Pour cela, nous avons d'abord suivi les fichiers exemples que Mme Bentayeb nous a envoyé lors de la première séance de TD. Puis nous avons vu avec les autres groupes comment améliorer et compléter ces données pour leurs travaux.

Les données de dblp sont au format xml. Actuellement, le fichier contient 7914203 enregistrements d'articles, conférences, pages web d'auteur, etc.

Et chaque enregistrement contient certaines informations, mais d'un type d'enregistrement à l'autre, certaines informations ne sont pas présentes.

Voici les différents types d'enregistrements :

Occurrence	Type	Description
34.105%	inproceedings	conférence
33.034%	www	page web d'un auteur
30.242%	article	article scientifique
0.982%	phdthesis	thèse de doctorat
0.827%	incollection	partie de livre (avec son propre titre)
0.231%	book	livre scientifique
0.574%	proceedings	débats et/ou séance de questions après une conférence
0.0002%	mastersthesis	mémoire de master (il y en a 12 en tout)

Et voici les informations les plus courantes, les autres n'apparaissant que dans moins de 20% des enregistrements :

Occurrence	Nom du champ	Description
99.999%	@key	id de la publication
99.999%	@mdate	date de dernière modification de l'enregistrement
99.994%	title	titre de la publication
99.036%	author	nom(s) de(s) auteur(s) de la publication
67.119%	url	url où trouver la table des matières
66.965%	year	année de parution de la publication
64.723%	ee	url(s) où trouver la version électronique de la publication
58.92%	pages	bornes de pagination si le document n'est qu'une partie d'une publication
35.527%	booktitle	raccourcis du nom de la conférence, du séminaire ou du livre dans lequel la publication est parue
34.933%	crossref	référence vers le journal dans lequel la publication est parue
30.544%	volume	édition du journal dans lequel la publication est parue
30.239%	journal	nom du journal dans lequel la publication est parue
22.169%	number	numéro du journal dans lequel la publication est parue

Voici maintenant le contenu des fichiers csv :

- **publication.csv :**

- id\_publication : identifiant unique
- date\_pub : date de dernière modification de l'enregistrement
- nbr\_authors : nombre de co-auteurs

- article\_title : titre
- categorie : type de publication
- is\_complete (ajouté sur demande des autres groupes) : variable binaire indiquant si l'enregistrement est l'intégralité (1) ou juste une partie (0) de la publication
- **author.csv** :
  - id\_author : identifiant unique
  - name\_author : nom et prénom
  - nbr\_publication : nombre de publications enregistrées
- **keyword.csv** :
  - keyword : mot-clé (unique)
  - nbr\_used : nombre d'occurrences dans les titres de toutes les publications
- **year.csv** :
  - id\_year : identifiant unique
  - year : année
  - nbr\_publication : nombre de publications enregistrées
- **publication\_author.csv** :
  - id\_author : identifiant de l'auteur
  - id\_publication : identifiant de la publication
- **publication\_keywords.csv** :
  - id\_publication : identifiant de la publication
  - keyword : mot-clé
  - nbr\_use\_keyword : nombre d'occurrence du mot-clé dans le titre de la publication
- **publication\_year.csv** :
  - id\_publication : identifiant de la publication
  - id\_year : identifiant de l'année

Et enfin, voici les autres fichiers que l'on génère sur demande des autres groupes :

- **keyword\_fullnumeric\_cleaned.csv** : même chose que le fichier keyword.csv, mais après avoir supprimé les mots-clés entièrement numériques
- **publication\_keywords\_fullnumeric\_cleaned.csv** : même chose que le fichier publication\_keywords.csv, mais après avoir supprimé les mots-clés entièrement numériques
- **keyword\_metric\_system\_cleaned.csv** : même chose que le fichier keyword.csv, mais après avoir supprimé les mots-clés exprimant une valeur avec une unité du système métrique
- **publication\_keywords\_metric\_system\_cleaned.csv** : même chose que le fichier publication\_keywords.csv, mais après avoir supprimé les mots-clés exprimant une valeur avec une unité du système métrique

**3.4.2.2 Téléchargement des données :** Le téléchargement des données a été la première partie essentielle du projet. Le but était de télécharger le fichier source dblp.xml sur le site de dblp[10] que nous utilisons dans notre projet.

Dans cette tâche nous avons utilisé deux modules essentiels : URLLIB[6] et GZIP[2]. URLLIB comprend des fonctions permettant d'obtenir des informations et des données à partir d'url. Dans notre cas, nous l'avons utilisé pour récupérer une archive du site web. Nous avons ensuite pu décompresser cette archive en utilisant le module GZIP. Après ces deux étapes, il ne nous restait plus qu'à stocker le fichier dblp.xml dans un répertoire et à l'utiliser dans les prochaines tâches. Une seule séance a été consacrée à cette partie.

**3.4.2.3 Passage de xml en json :** Après la tâche de téléchargement, il nous a semblé pertinent de transformer les données afin de pouvoir les traiter efficacement. Nous nous sommes naturellement orientés vers le format JSON qui nous semblait être plus cohérent

et plus facile d'usage que le format XML. La première étape de cette tâche consistait à récupérer le XML et à le transformer en dictionnaire, pour cela nous avons utilisé le module XMLTOJSON[7]. Ensuite, nous avons dû mettre ces données au format JSON, ce que nous avons pu faire très facilement grâce au module JSON[3]. Enfin, nous avons créé un fichier JSON et nous avons copié nos données à l'intérieur. Pour finir nous avons récupéré un petit échantillon des données afin de tester nos fonctions en attendant que la tâche de parcours du fichier soit terminée.

```

1  {
2      "article": {
3          "@date": "2020-06-25",
4          "@key": "tr/meltdown/s18",
5          "@publtype": "informal",
6          "author": [
7              "Paul Kocher",
8              "Daniel Genkin",
9              "Daniel Gruss",
10             "Werner Haas 0004",
11             "Mike Hamburg",
12             "Moritz Lipp",
13             "Stefan Mangard",
14             "Thomas Prescher 0002",
15             "Michael Schwarz 0001",
16             "Yuval Yarom"
17         ],
18         "title": "Spectre Attacks: Exploiting Speculative Execution.",
19         "journal": "meltdownattack.com",
20         "year": "2018",
21         "ee": {
22             "@type": "oa",
23             "#text": "https://spectreattack.com/spectre.pdf"
24         }
25     }
26 }
```

Après plusieurs séances, nous nous sommes aperçus que transformer les objets en JSON de cette manière était peut-être trop coûteux, au vu de la quantité de données que nous avions à traiter. Nous avons donc décidé de nous contenter de transformer les éléments en dictionnaires, qui ont une syntaxe proche du JSON ce qui nous permettrait dans le futur de proposer les données sous forme de fichiers JSON plutôt que CSV.

**3.4.2.4 Parcours du fichier xml :** Les données téléchargeables sur le site [www dblp org](http://www dblp org) sont un unique fichier XML, contenant plusieurs gigaoctets de balises successives qui correspondent aux publications. Afin de les traiter efficacement sans avoir de problème de mémoire, il a fallu parcourir intelligemment le fichier, balise par balise, afin de traiter les publications au fur et à mesure que nous les extrayons du fichier.

Cette tâche nous a pris 3 séances, alors que nous avions estimé que cela nous en prendrait 2, notamment parce qu'au départ, nous ne connaissions pas bien les données, et nous avions sous-estimé la complexité et surtout l'irrégularité des différents éléments présents dans les balises. La durée de cette tâche s'explique aussi par un travail assez

lourd d'optimisation de la classe, pour minimiser le temps d'exécution de l'extraction et du traitement.

```

1 <article mdate="2020-06-25" key="tr/meltdown/s18" publtype="informal">
2   <author>Paul Kocher</author>
3   <author>Daniel Genkin</author>
4   <author>Daniel Gruss</author>
5   <author>Werner Haas 0004</author>
6   <author>Mike Hamburg</author>
7   <author>Moritz Lipp</author>
8   <author>Stefan Mangard</author>
9   <author>Thomas Prescher 0002</author>
10  <author>Michael Schwarz 0001</author>
11  <author>Yuval Yarom</author>
12  <title>Spectre Attacks: Exploiting Speculative Execution.</title>
13  <journal>meltdownattack.com</journal>
14  <year>2018</year>
15  <ee type="oa">https://spectreattack.com/spectre.pdf</ee>
16 </article>
```

Nous avons tout d'abord décidé de faire cette classe pour extraire les balises et leurs contenus, et renvoyer les publications une par une, avant d'être traitées par d'autres fonctions, dont la fonction de transformation dans le format json. Mais les performances étaient très mauvaises, et lors de nos recherches pour optimiser, nous avons fini par trouver une manière plus rapide d'extraire les publications et de les transformer en dictionnaires directement dans cette classe.

La classe lit donc le fichier ligne par ligne, détecte la balise ouvrante de la publication dans la chaîne de caractères, et parcourt les balises xml dans la suite de la chaîne de caractères à la recherche de la balise fermante qui va clore la publication. Une fois fait, elle va transmettre la longue chaîne de caractères, avec toutes les balises et leurs contenus qui composent la publication, à une fonction de la librairie LXML[4] qui va transformer cette chaîne de caractères en objet arborescent parcourable. Par la suite, la classe va descendre dans cet objet et récupérer toutes les infos qu'elle va stocker dans un dictionnaire, qu'elle va ensuite renvoyer.

La syntaxe des dictionnaires de python étant sensiblement la même que la syntaxe du format json, il est très facile d'enregistrer les données au format json, si le besoin s'en fait sentir dans le futur.

Pour finir, la classe FileReader s'utilise comme un itérateur, c'est à dire que l'on peut utiliser une boucle for ou utiliser les fonctions iter et next sur une instance de la classe pour recevoir les publications une par une et les envoyer dans les fonctions de traitement avant de passer à la suivante.

**3.4.2.5 Organisation des fichiers csv :** Une fois l'extraction faite, nous avons dû organiser la répartition et le découpage des données en différents fichiers. Mme Bentayeb nous ayant fourni des fichiers de données partielles afin de tester et de comprendre facilement le travail à accomplir, les autres groupes ont basé leurs travaux sur ce format précis, et par gain de temps pour eux, nous avons décidé tous ensemble de générer des fichiers dans le même format.

La réalisation de cette tâche a duré 2 séances, nous avons dû comprendre chaque colonne des fichiers csv fournis par Mme Bentayeb pour pouvoir réfléchir à ce que l'on

id_publication	date_pub	nbr_authors	article_title	categorie
tr/meltdown/s18	2020-06-25	10	Spectre Attacks : ...	article
tr/meltdown/m18	2020-06-25	10	Meltdown	article
tr/acm/CS2013	2019-05-27	0	Computer Science ...	article

**Table 5.** publication.csv

décidait de garder ou non. Nous avions même estimé que cela nous prendrait 3 séances, mais finalement nous avons rapidement compris les données, et nous avons facilement fait le lien entre les publications au format json et les informations des fichiers csv exemplaires.

**3.4.2.6 Extraction des keywords :** Pour construire les fichier CSV, la première étape a été d'extraire des données précises. D'abord, nous avons créé la fonction get-keywords() qui prend en paramètre un dictionnaire contenant une publication. Dans cette fonction, nous pouvons parcourir une publication et en extraire les mots-clés dans le but de créer un fichier qui répertorie chaque mot-clé dans l'ensemble des articles. La difficulté ici était de trouver les bons séparateurs en évitant d'avoir des virgules, des espaces ou des points dans nos mots-clés. La finalité était ici de renvoyer un tableau, contenant tous les mots-clés déjà prétraités, aux fonctions qui écrivent les fichiers CSV. Deux séances ont été consacrées à la réalisation de cette tâche.

**3.4.2.7 Extraction des auteurs :** L'extraction des auteurs nous a aussi posé quelques problèmes, deux séances ont dû être consacrées à cette tâche. En effet, lors de la création de la fonction get-authors(), qui prend en paramètre un dictionnaire contenant une publication, nous nous sommes aperçus que les auteurs n'étaient pas toujours référencés de la même manière. Nous avons dû refaire un travail, d'observation des données et nous avons pris conscience qu'il était nécessaire de traiter les cas de listes d'auteurs et d'auteurs seuls mais aussi des cas de nom d'auteur mal positionné. Cela fait, il nous a été possible de récupérer des listes d'auteurs complètes, utilisées plus tard dans la création des fichiers csv.

**3.4.2.8 Extraction des dates :** Dans cette partie nous avons implémenté une fonction get-year() qui prend en paramètre une publication sous forme de dictionnaire. Cette fonction va parcourir la publication de la manière la plus optimisée possible afin de ne pas rallonger le temps d'exécution du programme qui est déjà très long. Cette fonction nous renvoie l'année de la publication et un identifiant unique, ce résultat, comme les autres servira à l'écriture des fichiers csv. Deux séances ont été consacrées pour la réalisation de cette tâche.

**3.4.2.9 Extraction des publications :** Cette tâche consistait à implémenter une fonction qui récupère différentes informations de la publication passée en paramètre : l'identifiant, la date, le nombre d'auteurs, le titre, et la catégorie. Au vu du nombre d'informations à extraire, nous avions estimé 2 séances pour faire cette fonction. Mais cela a été plus simple que pour les auteurs ou les keywords, car ces données sont plus régulières, et nous avions de toute façon déjà vu comment gérer les différents cas particuliers de format. Cela nous a donc pris une séance.

**3.4.2.10 Écriture des fichiers simples :** Pour l'écriture des fichiers publication.csv, author.csv, keyword.csv et year.csv, nous avons tout d'abord codé une classe mère avec des opérations élémentaires comme par exemple : La modification (incrémentation ou ajout d'une valeur) des données d'un buffer ; en l'occurrence, le buffer est un dictionnaire, par soucis de complexité en temps d'accès et de modification des données La vérification de présence d'une valeur L'écriture du contenu du buffer dans un certain fichier Nous avons ensuite simplement créé

une classe par fichier qui hérite de cette classe mère et qui utilise les opérations élémentaires pour mettre à jour les données à chaque publication. Cette tâche nous a aussi pris une séance.

**3.4.2.11 Préparation à l'écriture des fichiers croisés :** Pour les fichiers croisés, nous avons appliqué la même méthode que pour les fichiers simples, à savoir une classe mère avec les opérations élémentaires. Le code étant très proche de la classe mère des fichiers simples, il ne nous a fallu que quelques minutes pour faire cette classe.

**3.4.2.12 Écriture du fichier publication-author :** La création de la classe qui permet l'écriture du fichier publication-author nous a pris une séance. Elle permet d'écrire un fichier qui donne tous les auteurs de chaque publication. Cela ne nous a posé que très peu de difficultés grâce à l'utilisation de la classe mère d'écriture de fichier croisé : WriteField-cross().

**3.4.2.13 Écriture du fichier publication-year :** La création de la classe qui permet l'écriture du fichier publication-year nous a aussi pris une séance. Elle permet d'écrire un fichier qui donne l'année de chaque publication. Cette fonction utilise elle aussi la classe mère d'écriture des fichiers croisés.

**3.4.2.14 Écriture du fichier publication-keywords :** Cela nous a pris une séance pour développer la classe qui permet d'écrire le fichier publication-keywords.csv. Il a simplement fallu la faire hériter de la classe mère des fichiers croisés, lier la classe au fichier correspondant, et lui faire ajouter les mot-clés aux données grâce aux opérations élémentaires.

**3.4.2.15 Demandes spécifiques pour les données :** Une fois les fichiers de base générés, nous avons beaucoup échangé avec les groupes de Data Visualisation et d'Analyse, afin de savoir s'ils seraient intéressés par des informations supplémentaires dans les différents fichiers.

Ils nous ont donc demandé d'ajouter une colonne dans publication.csv indiquant si la publication est un extrait ou l'article complet. Ils ont également formulé la demande d'avoir des mots-clés avec moins de valeurs aberrantes. Nous nous sommes donc mis d'accord pour nettoyer les mots-clés ne contenant que des chiffres, et les mots-clés exprimant une mesure selon le système international d'unités.

**3.4.2.16 Nettoyage des keywords entièrement numériques :** Comme expliqué précédemment, après discussion avec les groupes de visualisation, nous nous sommes aperçus que notre nettoyage des mots-clés n'était pas suffisant. En effet, de nombreux mots-clés étaient inutilisables par les autres groupes du projet. Après concertation avec Madame Bentayeb et avec les membres des autres groupes, nous en sommes arrivés à la conclusion que certains mots-clés, dans des langues étrangères pouvaient avoir un intérêt. En revanche, il nous a semblé important de supprimer les mots-clés entièrement numériques. Nous avons donc écrit une fonction isNumeric qui prend en entrée un mot-clé et qui renvoie un booléen indiquant si le mot-clé est entièrement numérique ou non. La réalisation de cette tâche n'a duré qu'une séance.

**3.4.2.17 Nettoyage des keywords exprimant une mesure :** Pour retirer les mots-clés exprimant une mesure selon le système international d'unités, nous avons créé une fonction qui prend un mot-clé en paramètre et qui indique s'il suit la syntaxe suivante :

- Au moins un caractère numérique
- Un éventuel préfixe d'ordre de grandeur, comme c (centi), k (kilo), etc.

- Une unité du système international, comme m (mètre), s (seconde), etc.  
ou  
Une unité dérivée du système international, comme W (watt), J (joule), Hz (hertz), etc.  
ou  
Une unité que l'on trouve typiquement dans le milieu informatique, que nous avons dû lister manuellement, comme o (octet), b ou bit, B (byte), etc.
- Une éventuelle suite de caractères numériques, correspondant à un éventuel exposant.

Lors du traitement des mot-clés, on supprime les caractères non alphanumériques, on ne peut donc pas trouver de virgule, ce qui simplifie légèrement l'expression régulière que nous devons appliquer à chaque mot-clés. Cette tâche nous a pris une séance.

### 3.5 Conclusion

Pour conclure, nous pouvons dire que la réalisation de l'interface dans un projet qui regroupe un aussi grand nombre de personnes a été un véritable défi. En effet, le développement de l'interface ne consistait pas uniquement en un travail de création de fichiers et d'interfaces graphiques mais aussi en un travail d'adaptation, que ce soit aux données initiales ou aux travaux des groupes de visualisation et d'analyse.

Comme nous l'avons expliqué dans l'introduction, notre groupe a été scindé en deux, l'adaptation aux données a donc principalement concerné le groupe de pré-traitement des données. Nous avons d'abord été confronté à une quantité de données qui nous paraissait colossale, d'où la nécessité de trouver des manières optimisées pour les traiter. Par ailleurs, les compositions des publications que nous avons dû analyser étaient très variées ce qui nous a demandé d'effectuer un travail d'observation, de nettoyage et de vérification des données.

Par la suite, nous avons été sollicité par les autres groupes travaillant sur le projet. Nous avons été en mesure répondre à des requêtes concernant le nettoyage des données et la création de certains fichiers spécifiques, en reprenant et en intégrant de nouveaux éléments dans notre code. Nous avons également dû récupérer et comprendre les interfaces créées par les groupes de visualisations et d'analyse afin d'intégrer leurs travaux dans une interface globale fonctionnelle.

Les adaptations continues nécessaires à la réalisation d'un tel projet sont difficiles à retranscrire mais sont à l'origine de nombreux échanges avec nos professeurs et nos collègues dans le but de produire un travail qui soit le plus proche possible des attentes de chacun. Plus précisément de satisfaire les attentes des clients concernant l'interface graphique et celles de nos collaborateurs concernant l'organisation des données.

## 4 GROUPE DATA VISUALISATION 1

### 4.1 Noms et prénoms des participants

- **ELEOUET Clément** (Product Owner)
- **HAMZA Assoumani Chissi**
- **LAMOTHE Dorian**
- **LO Mouhamadou**
- **TRIKI Arthur** (Scrum Master)
- **DOROVSKIKH Kirill**

### 4.2 Introduction

Dans le cadre d'un projet au sein du Master 1 Informatique (Université Lumière Lyon 2), les étudiants étaient chargés de réaliser un outil permettant de visualiser et interpréter les données extraites du site DBLP.

Ainsi, chaque TD a été scindé en 4 équipes avec trois tâches principales :

- La réalisation de l'interface de l'outil;
- La visualisation des données;
- L'analyse des données.

Notre groupe, composé de 6 étudiants a choisi de s'occuper de la partie datavisualisation.

Nous disposions ainsi de 16 séances pour créer et intégrer notre outil de visualisation à l'interface globale.

Nous étions ainsi chargé de réaliser un graphe de relation entre les différentes entités de la base de données : les auteurs, leurs publications, etc... Un point d'honneur serait mis à l'interactivité de l'outil et à son intuitivité vis-à-vis d'un utilisateur non-initié.

Ainsi nous verrons par la suite que notre produit final se scinde en trois parties distinctes :

- La création d'une interface;
- La création d'un interpréteur de requêtes;
- La mise en place du graphe et des ses composantes.

### 4.3 Les données & les outils

La préparation des données est en grande partie gérée par l'équipe Interface. En effet, ces derniers sont censés extraire les données au format XML puis nous les transmettre au format CSV.

Le site DBLP (Digital Bibliography & Library Project) est un site web publant un catalogue de bibliographies en informatique.

Les données DBLP se scindent en 4 fichiers représentant les entités :

- Author (id\_author, name\_author, nbre\_publication).
- Publication (id\_publication, date\_pub, nbr\_authors, article\_title, categorie).
- Keyword (id\_keyword, keyword).
- Year (id\_year, year).

On s'appuie également sur 3 fichiers représentant les liens entre les entités :

- Author\_Publication (id\_author, id\_publication, nbre\_publication).

- Keyword\_Publication (id\_keyword,id\_publication).
- Year\_Publication (id\_year, id\_publication).

Le diagramme de relations ci-dessous permet de mieux visualiser les liens entre les différents fichiers :

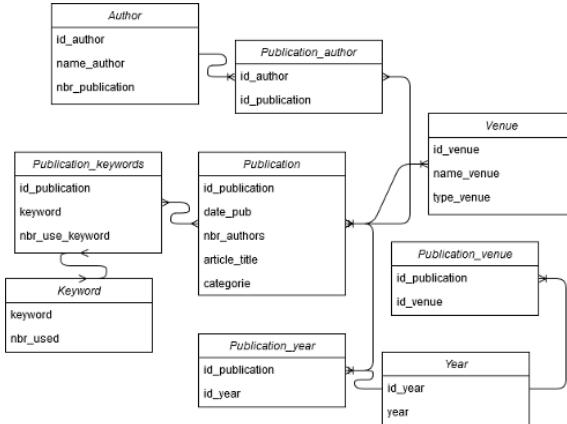


Figure 18

Afin de réaliser les tâches qui nous sont confiées, il a été décidé que l'outil serait développé grâce au langage informatique Python. Nous avons ainsi principalement utilisé l'interface de développement Spyder ainsi que Jupyter Notebook.

## 4.4 Gestion de projet & Modèle SCRUM

### 4.4.1 Les rôles

Dans le cadre de l'enseignement 'Gestion de projet & Génie Logiciel', la méthode SCRUM nous a été présentée et imposée afin de mener à bien ce projet. Ainsi, voici un court résumé des rôles de la méthode SCRUM et de comment ils ont été alloués au sein du groupe :

- Le Scrum Master : il est celui qui veille à la bonne application de la méthode SCRUM tout en assurant la communication au sein de l'équipe de travail. Dans notre groupe c'est Arthur TRIKI qui assure ce rôle.
- Le Product Owner : il donne des directives vis-à-vis des fonctionnalités du produit, en forte collaboration avec le client. Il s'occupe de la gestion du product backlog. Ce rôle est assuré par Clément ELEOUET.
- L'équipe de développement : elle possède des compétences multiples qui lui permettent de réaliser les objectifs mis en place dans les différents backlogs.

### 4.4.2 Product Backlog & User stories

Afin de mieux appréhender les missions confiées par les professeurs et dans le cadre de l'approche SCRUM, nous avons dans un premier temps décidé de réaliser des User Stories qui correspondent aux besoins et attentes du client vis-à-vis des fonctionnalités du produit final. Ces fonctionnalités attendues sont intégrées au sein du product backlog, liste hiérarchisée des exigences initiales du client. Cette liste est amenée à évoluer tout au long du projet en fonction de l'avancement du travail et des problèmes rencontrés.

Voici une liste non-exhaustive de nos user stories :

- En tant qu'utilisateur je veux pouvoir exécuter des requêtes personnalisées et obtenir des résultats numériques et graphiques. J'aimerais néanmoins pouvoir m'appuyer sur des requêtes conseillées par le concepteur.

- En tant qu'utilisateur je veux pouvoir appliquer des filtres à mes requêtes.
- En tant qu'utilisateur je veux pouvoir interagir avec le graphe.
- En tant qu'utilisateur je veux pouvoir exporter les résultats de ma requête (tables, graphiques).
- En tant qu'utilisateur je veux pouvoir choisir ma charte graphique.
- En tant qu'utilisateur je veux pouvoir consulter mes anciennes requêtes.
- En tant qu'utilisateur je veux pouvoir naviguer sur les sommets du graphe en zoomant et dézoomant.
- En tant qu'utilisateur je veux pouvoir obtenir une fiche d'information synthétique sur un auteur, une publication en cliquant dessus.
- En tant qu'utilisateur je veux pouvoir modifier le graphe en ajoutant/supprimant des relations.

Ces user stories permettent à l'équipe de développement d'avoir des axes de réflexion définis avant chaque planification de sprint et de visualiser l'avancée du projet en fonction de la proportion d'exigences satisfaites.

#### 4.4.3 Le burndown chart

Le burndown chart (ou graphique d'avancement) indique l'avancement des tâches au cours d'un sprint et/ou du projet général. Il s'exprime en fonction du tracé de la charge de travail restante en heures (ordonnée), en fonction du temps (en jours).

#### 4.4.4 Trello : une aide précieuse

Afin de gérer au mieux la gestion des tâches, nous avons dès le départ du projet choisi d'utiliser l'outil Trello qui nous a permis d'appliquer un modèle intuitif et strict s'adaptant parfaitement aux directives principales de SCRUM que sont la transparence et l'efficacité.

Ce site nous a permis de hiérarchiser et d'organiser nos tâches à la manière du précepte abordé en cours « To do, Doing, Done ».

Trello nous permettait également d'affecter des membres de l'équipe à une tâche donnée. Ainsi, à chaque sprint review nous savions exactement où nous en étions par rapport au product backlog.

De plus, en affectant à chaque tâche le temps prévu et effectif, cet outil nous a permis de tenir un suivi strict en facilitant la mise en place de burndown chart à chaque sprint.

Voici un extrait de notre Trello :



Figure 19

#### 4.4.5 Les sprints

Le sprint est une des itérations du processus de développement du produit final. Il est le cœur de la méthode SCRUM et se découpe en plusieurs étapes :

- La planification : celle-ci se fait en amont du sprint afin de le préparer au mieux. Le Scrum Master fait un bilan du sprint précédent tout en s'assurant que le développement du produit suit les attentes définies dans le Product Backlog. Enfin, il alloue à chacun les tâches du sprint à venir.
- Le daily scrum meeting correspond à une réunion intermédiaire avant chaque session de travail. Au cours de cette courte réunion (15 min. max) on étudie l'avancement de chacun dans ses tâches et les difficultés rencontrées.
- Le sprint review : il s'agit d'un bilan où chacun présente ses avancées. On peut évoquer la suite du projet, et c'est pourquoi dans notre cas il est fréquent que ce dernier s'emboîte avec la plion du sprint suivant.

Voici donc un récapitulatif détaillé du déroulement de nos sprints au sein duquel nous évoquerons

##### 4.4.5.1 Sprint 1 : Les tâches pour le premier sprint étaient les suivantes :

- Pour le premier sprint, nous nous étions fixés comme objectif initial d'appréhender chacun les données en effectuant des manipulations simples (moyenne, tri etc...)
- Le deuxième grand objectif de ce premier sprint était de trouver des librairies intéressantes pour la phase graphique et d'en fournir une première utilisation basique.
- Une autre tâche était de définir les users stories et de les intégrer au product backlog comme vu précédemment.
- Nous étions aussi censés optimiser l'importation des bases de données en utilisant plusieurs méthodes de chargement.

- Enfin, nous avions prévu de nous intéresser à des méthodes de requête sur les données avec un seul paramètre en entrée (par exemple, afficher les publications d'un auteur donné).

Ce premier sprint, conséquent de par ses nombreux objectifs de prospection et de production, s'étalait sur une période de deux semaines.

	Graphe & Librairies	User Story & Backlog	Chargement des données	Requête & Interpréteur
CLEMENT		X		X
ARTHUR	X			
LO			X	X
HAMZA			X	X
KIRILL	X			
DORIAN		X		X

Figure 20

En résumé, nous avons trouvé plusieurs anomalies dans les données, résultant d'une mauvaise saisie, d'un manque considérable d'informations, etc... Dans notre Sprint Backlog nous avions prévu une marge de temps en prévision du data management. En effet, une base de données n'est jamais ou rarement propre avant traitement et c'est une étape de notre travail.

Pour travailler sur le graphe, nous avons vite fait le choix de NetworkX, librairie spécialisée dans l'affichage de graphes, associée à plotly. Celle-ci nous semblait être le meilleur compromis au vu des exigences du client (interactivité, esthétisme) et des compétences de l'équipe de développement. Voici un exemple du graphe obtenu lors du premier sprint :

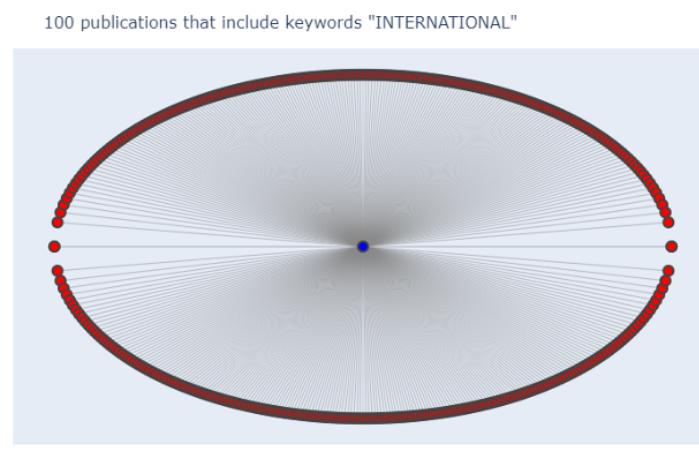
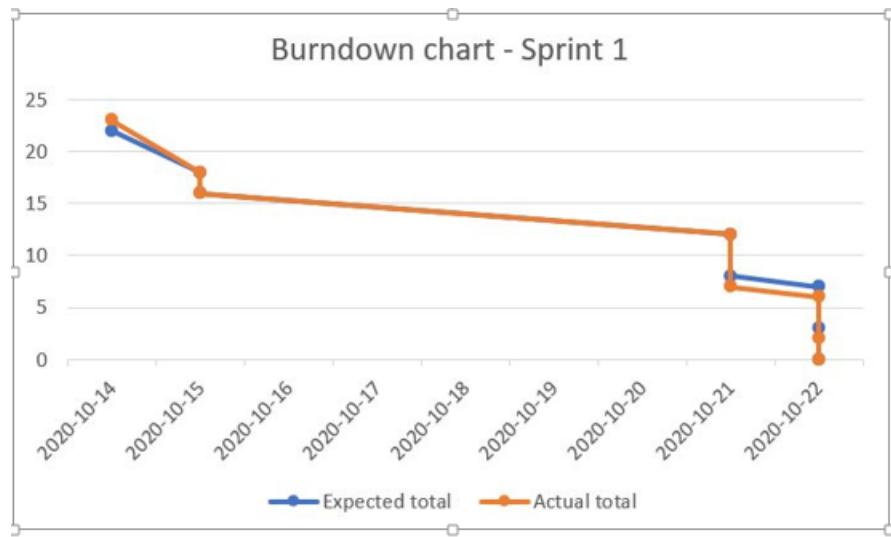


Figure 21. Graphe obtenu lors du Sprint 1

En passant la souris sur les noeuds on obtenait des informations à leur sujet. Néanmoins dans notre sprint review, nous avions prévu de généraliser cette méthode d'affichage de graphe à un plus grand nombre de requêtes.

Concernant les requêtes, nous avons construit, lors du premier sprint une requête simple accompagnée de son interpréteur qui permettait seulement d'afficher les publications d'un auteur renseigné par l'utilisateur. Nous verrons que la méthode employée a été remplacée lors des sprints suivants, ce qui soulève encore là une notion importante de SCRUM qu'est l'adaptabilité.

Vous trouverez ci-dessous le burndown chart du sprint 1 :



**Figure 22.** Burn Down Chart Sprint1

**4.4.5.2 Sprint 2 :** Pour ce deuxième sprint, nous nous étions répartis le travail en trois parties :

- La première équipe devait développer un interpréteur de requête, qui traduisait une requête SQL en une requête simple sur les données avec la librairie pandas.
- Une autre équipe devait étendre les méthodes de requête à l'ensemble des tables et des paramètres qu'est susceptible de rentrer l'utilisateur.
- L'équipe graphique continue de son côté à adapter le code du graphe pour le rendre plus adapté au backlog et users stories.
- Enfin, afin de ne pas prendre de retard, nous avons décidé de faire un rapport sur le sprint précédent.

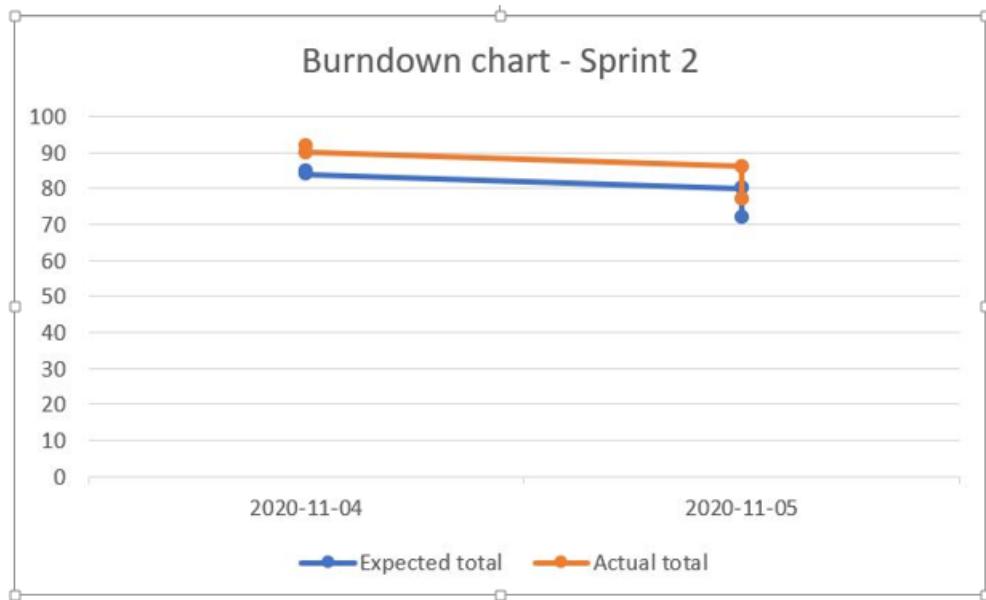
	Graphique	Requêtes	Interpréteur
CLEMENT			X
ARTHUR	X		
LO		X	
HAMZA		X	
KIRILL	X		
DORIAN			X

**Figure 23**

Au milieu de ce sprint, lors de l'une de nos mélées (équivalent du daily scrum meeting), nous avons trouvé la librairie pandasql permettant de simplifier considérablement l'interprétation des requêtes entrées par l'utilisateur.

Ainsi le product owner, a réuni l'ensemble de l'équipe pour décider si il valait mieux mettre de côté l'interpréteur de requêtes, au profit de l'utilisation de cette librairie. Au bout d'un long arbitrage, nous avons décidé d'agir dans ce sens en raison de plusieurs raisons, à la fois orientées client et développeurs.

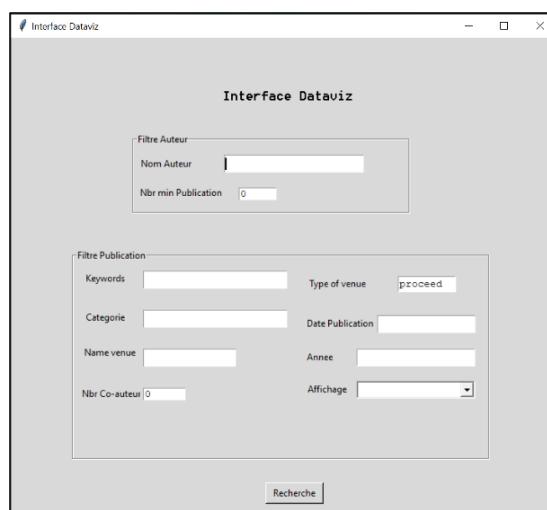
Vous trouverez ci-dessous le burndown chart du sprint 2 :



**Figure 24.** Burn Down Chart Sprint2

**4.4.5.3 Sprint 3 :** Pour le troisième sprint, nous nous sommes une nouvelle fois répartis le travail en 3 équipes. La grande innovation de ce sprint est la création de l'interface dataviz qui permet à l'utilisateur, de manière interactive, de choisir les champs qu'il souhaite voir apparaître sur le graphique. Le deuxième objectif important défini en amont du sprint était de faire communiquer cette interface avec la librairie pandasql, puis au graphe.

Voici un aperçu de la première version de l'interface :



**Figure 25.** Interface Visualisation

Cette Interface Homme-Machine (IHM) a été réalisée à l'aide de la librairie tkinter. Plusieurs

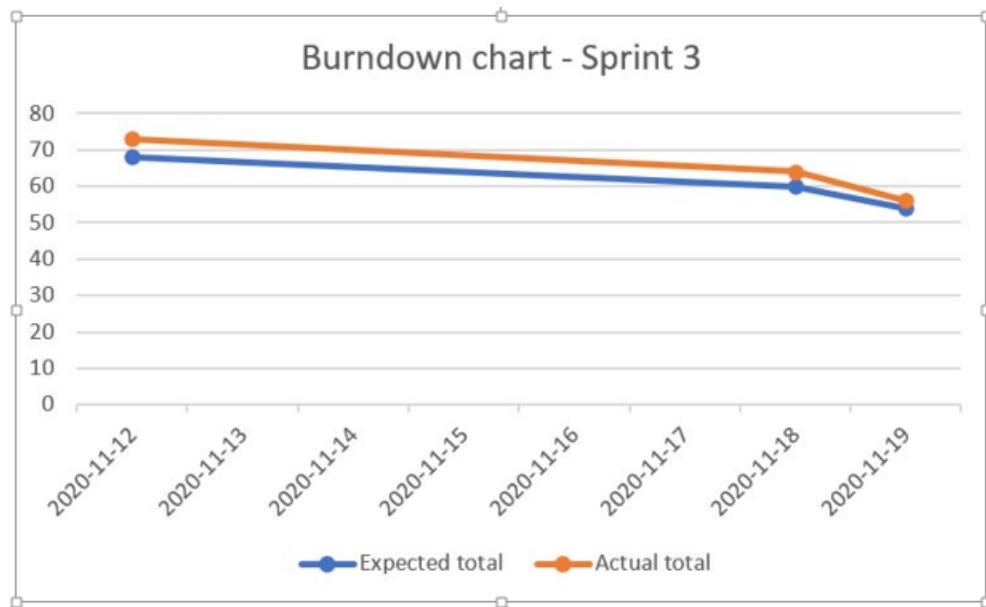
filtres ont été reportés ainsi qu'une option d'affichage permettant à l'utilisateur de choisir si il veut visualiser les publications ou les auteurs.

Nous avons une nouvelle fois voulu répondre aux users stories nous demandant une interactivité et une intuitivité maximale. Ainsi, l'utilisateur applique les filtres qu'il souhaite d'une manière compréhensible.

Ainsi, grâce à une fonction `getElement()`, une autre équipe était censée récupérer les données saisies par l'utilisateur et les transformer en requêtes SQL (fonction `buildQuery()`), grâce à la librairie `pandasql`. Ces éléments seront détaillés dans la partie Projet Intégré du rapport.

Dans un troisième temps, l'équipe en charge de la partie graph en récupérant les données de la requête et les adapter au code de la fonction graphique réalisée au cours du premier sprint.

Voici le burndown chart du sprint 3 :



**Figure 26.** Burn Down Chart Sprint<sup>3</sup>

**4.4.5.4 Sprint 4 :** Le dernier sprint se concentrat principalement sur les items du product backlog les moins importants.

Ainsi il s'agissait principalement d'ajuster les finitions du graphiques et de gérer les éventuelles erreurs dans le code.

Nous avons également ajusté la procédure pour récupérer les données entrées par l'utilisateur.

Nous avons également optimisé le temps de chargement des données grâce à une nouvelle procédure que nous avons dû adapter aux nouvelles données par le groupe interface. Cela soulève un autre point fort du modèle SCRUM qu'est l'adaptabilité maximale aux difficultés et aux demandes du client.

De plus l'équipe de développement a également dû prendre en compte les remarques des professeurs que l'on peut identifier comme les utilisateurs du produit final. Nous avons

ainsi dû prendre faire quelques modifications quant à l'affichage du texte dans le graphe etc...

Dans un autre temps, il nous a aussi fallu modifier notre travail antérieur vis-à-vis du rapport en raison de la structure de ce dernier qui ne correspondait pas aux volontés des professeurs.

Enfin, nous avons dû nous préparer à la soutenance et à la démonstration de notre produit final.

	Graphe	Gestion des erreurs	Rapport	Soutenance	Finitions Interface
CLEMENT		X		X	
ARTHUR			X	X	
LO	X			X	X
HAMZA		X		X	
KIRILL	X			X	
DORIAN			X	X	

Figure 27

Voici le burndown chart du sprint 4 :

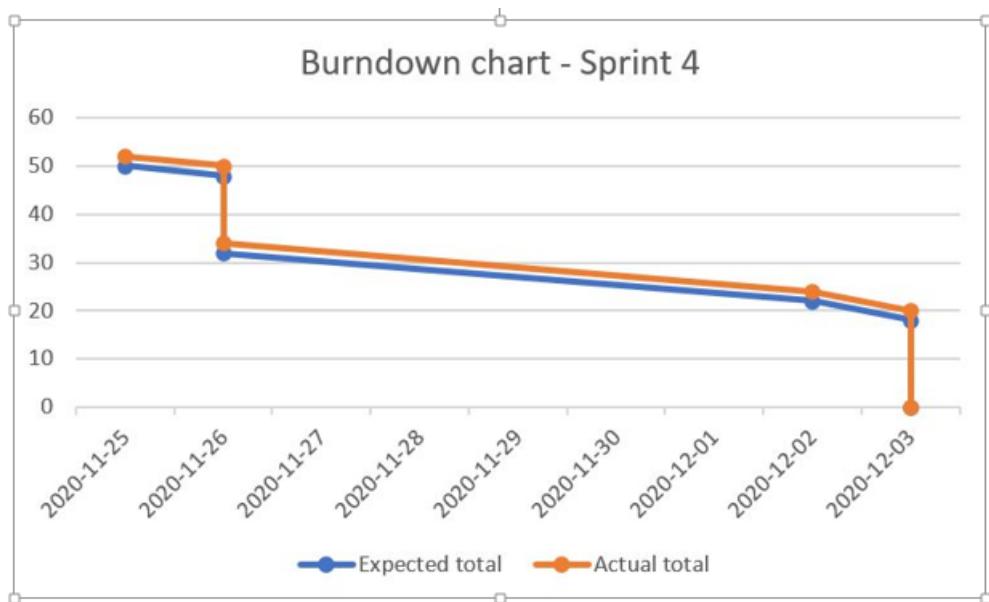
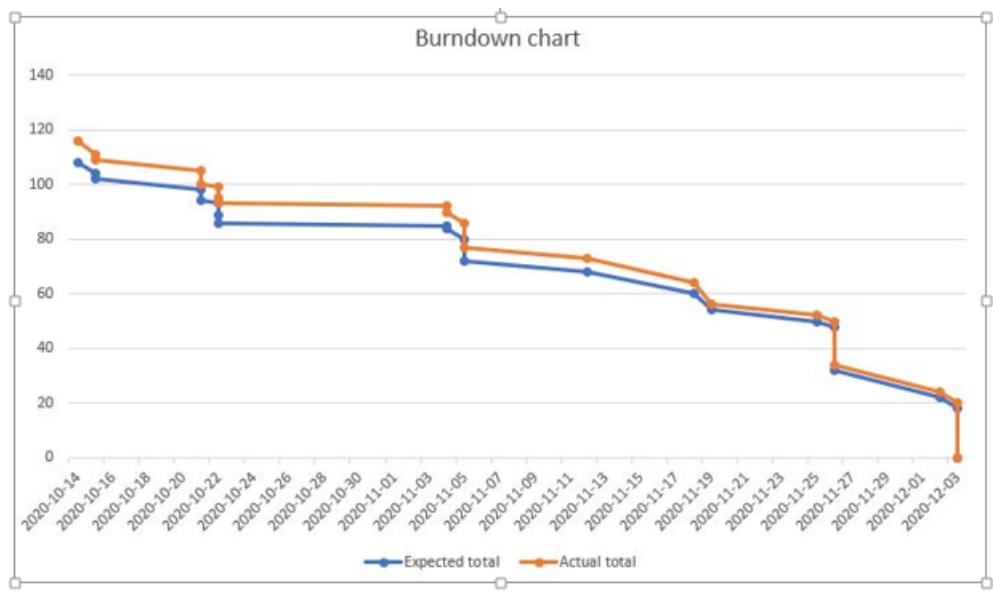


Figure 28. Burn Down Chart Sprint2

Ainsi que le burndown chart du projet au complet :



**Figure 29.** Burn Down Chart Sprint final

#### 4.4.6 Conclusion sur la méthode SCRUM

En conclusion, la méthode SCRUM nous a permis de mieux structurer notre travail tout en divisant de manière optimale les tâches. L'outil Trello nous a été d'une grande aide dans la manière d'appréhender la structure de chaque sprint et du projet global.

Ainsi, chaque membre pouvait consulter l'avancée du projet ce qui montre le soucis de transparence de SCRUM.

Enfin, la structure en sprint a permis à l'ensemble de l'équipe de ne pas se disperser. Les scrum meeting ont permis à chacun d'évoquer ses difficultés et aux membres de s'entraider ce qui est important lorsque ces derniers ne se connaissent pas.

En somme, l'ensemble des composantes de SCRUM ont fait que la productivité du projet était optimale. Les outils comme le burndown chart ont été déterminants dans la planification qui fait tant défaut en temps normal.

## 4.5 Projet Intégré

### 4.5.1 Introduction

Dans cette partie du rapport nous allons expliquer en détail les méthodes employées pour développer le produit final. Ainsi, pour chaque étape, nous présenterons le rendu et commenterons certains morceaux importants du code (fonctions, classe...).

Nous diviserons ce rapport en 4 étapes principales :

- Le chargement des données ;
- L'interface DataViz ;
- Les requêtes ;
- Le graphe.

Notre travail se répartit en 5 fichiers distincts que sont :

- AddTable.py ;
- Requête.py ;
- Interface.py ;
- Interface\_support.py ;
- DisplayGraphe.py .

Les fonctions principales du projet sont :

- Saisie de filtres pour exécuter des requête sur le jeu de données DBLP ;
- Construction d'une requête SQL à partir des filtres ;
- Exécution de la requête et récupération du résultat sous forme de dataframe ;
- Visualisation sous forme de graphe.

### 4.5.2 Méthode pour exécuter le programme

- Télécharger les fichiers du programme
- Importer les modules nécessaires au fonctionnement du programme
- Ouvrir le fichier AddTable.py et modifier le chemin d'accès vers les fichiers du dataset
- Ouvrir le fichier Interface.py et cliquer sur le bouton d'exécution ou bien sur le raccourci

### 4.5.3 Le chargement des données

Comme évoqué plus haut, le fichier AddTable.py permet d'importer les données.

Pour cela nous utiliserons deux librairies principales :

- Pandas ;
- Dataframe\_sql.

Avec pandas nous pouvons importer les données CSV normalement avec la fonction `read_csv()`. Les fichiers étant trop conséquents pour nos machines on a précédemment réalisé un échantillonnage sur les jeux de données initiaux.

La partie la plus intéressante dans l'importation des données est l'utilisation de la librairie `dataframe_sql`.

En effet dans un soucis de performance cette dernière permet d'enregistrer en mémoire les nouvelles tables créées avec des requêtes SQL :

```
table = ds.query("SELECT * FROM author INNER JOIN pub_aut ON author.id_author = pub_aut.id_author")
table = table.drop(columns=["author.id_author","pub_aut.id_author"])
ds.register_temp_table(table,"fusautid_aut")
ds.remove_temp_table("author")
ds.remove_temp_table("pub_aut")
```

**Figure 30**

Ici on voit que l'on crée donc quatre nouvelles tables où les jointures entre les tables sont déjà faites, ce qui permet de gagner du temps lors de l'exécution de la requête dans l'interface.

Ainsi la partie query("SELECT ... FROM ... WHERE") exécute la requête tandis que les fonctions remove\_temp\_table(), register\_temp\_table(), drop() permettent la manipulation de tables afin d'optimiser les données dans un soucis constant de performance.

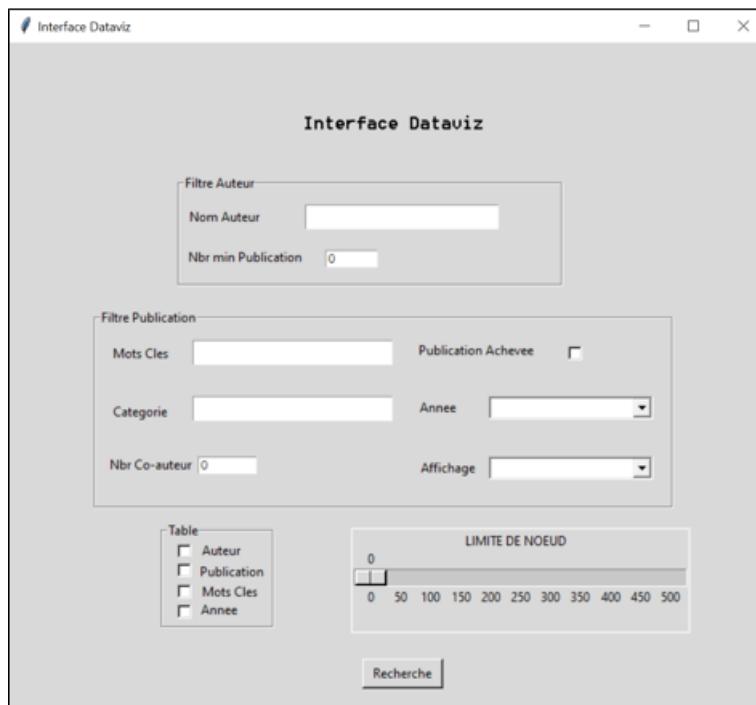
#### 4.5.4 L'interface

**4.5.4.1 Présentation et utilisation** L'interface graphique est une étape non-négligeable de notre travail. En effet, elle répond à plusieurs éléments de notre backlog et donc à de nombreux besoins de l'utilisateur.

Ses principales qualités doivent être l'ergonomie, la lisibilité, la compréhensivité ainsi que la rapidité.

De plus l'interface, fait le lien entre la requête et le graphe.

Ainsi, voici un aperçu du résultats final de notre interface :



**Figure 31.** Aperçu de l'interface Visualisation

Pour faire simple, toutes les cellules de saisie permettent d'appliquer des filtres sur la requête (clause "WHERE"), par exemple, si vous entrez un nom d'auteur dans la cellule prévue à cet effet, le programme retournera les informations liées à cet auteur. Il est possibles d'appliquer plusieurs filtres sur une même requête.

Décortiquons les principales fonctionnalités de l'interface :

- La partie Filtre Auteur permet d'appliquer les filtres sur la table auteur et ses variables nom et nombre de publications. Ainsi on peut renseigner que l'on ne souhaite visualiser que les publications d'un auteur X ayant rédigé un minimum de Y publications.
- La partie Filtre Publication permet à l'utilisateur d'appliquer des filtres sur les publications.
- Les cases à cocher permettent de renseigner les tables dont vous désirez avoir des informations (clause "FROM"). Si l'on ne coche aucune case alors le programme interprétera qu'il faut consulter l'ensemble des tables.
- La jauge permet de réguler le nombre de nœuds maximal à reporter sur le graphe. C'est à la demande d'un professeur que nous avons adapté en jauge ce qui était auparavant une simple saisie numérique.

Voyons maintenant un exemple d'utilisation de l'interface. Si l'on souhaite voir l'ensemble des publications d'un auteur A comprenant le mot clé B, notre requête sera alors de la forme suivante : SELECT \* FROM publication, pub\_aut, authors WHERE name\_author = 'A' AND keyword = 'B'.

Dans ce cas on rentre le nom A dans la cellule 'Nom Auteur' et le mot clé B dans la cellule 'Mots Clés'.

Votons maintenant quelques fonctions du programme en détail pour mieux comprendre la conception de l'outil.

**4.5.4.2 Code & Méthodes** Pour construire l'interface nous avons utilisé les librairies TKinter et TKinter.ttk dans un soucis de facilité de prise en main.

Le fichier Interface.py est le fichier cœur du programme et celui à exécuter comme précisé précédemment. Néanmoins pour construire l'interface on utilise également le fichier Interface\_support.py.

Ce fichier interface comprend la fonction vp\_start\_gui() qui est le point de départ du programme. Cette fonction fait appel à la classe Interface présente elle-même dans ce fichier et qui permet de configurer et paramétrier les différents objets de l'interface.

Ainsi voici quelques extraits commentés de la fonction init de cette classe interface :

```
#champ de saisie pour l'auteur
self.EntryNA = tk.Entry(self.Filtre_auteur, textvariable=IS.nomAut)
self.EntryNA.place(relx=0.331, rely=0.272, height=24, relwidth=0.504
                  , bordermode='ignore')
self.EntryNA.configure(background="white")
self.EntryNA.configure(disabledforeground="#a3a3a3")
self.EntryNA.configure(font="TkFixedFont")
self.EntryNA.configure(foreground="#000000")
self.EntryNA.configure(insertbackground="black")
```

Figure 32

Ici, par exemple, on crée le champ de saisie pour le nom de l'auteur. On remarque qu'on récupère le champ nomAut du modèle IS (interface\_support), exemple de la relation entre les deux fichiers.

Une fois les champs rentrés dans l'interface il nous a fallu créer la fonction qui récupère l'ensemble des éléments entrés par l'utilisateur : getElement().

Cette fonction enregistre les données dans trois dictionnaires différents :

- monDict : récupère les filtres entrés par l'utilisateur
- displayDict : récupère le nombre de filtres saisis ainsi que l'affichage et le nombre de noeuds voulus.
- tableDict : récupère les tables cochées dans la partie prévue à cet effet de l'interface.

Une fois ces éléments récupérés, la fonction getElement() prépare la requête à l'aide de la fonction buildQuery() que nous allons détailler dans la partie suivante.

#### 4.5.5 Les requêtes

Nous avons donc fait le choix de construire la requête sous la forme du langage SQL que nous appliquons au différentes tables créées.

Ainsi, une fois les trois dictionnaires remplis à l'aide des données entrées par l'utilisateur, on cherche à l'aide plusieurs boucles if les noms de colonnes à ajouter à la clause SELECT de la requête. On les récupère dans la variable columns.

```
def buildQuery(displayDict, tableDict):
    if(tableDict["author"]==1 and tableDict["publication"]==0):
        columns = "name_author,nbr_publication"
    elif(tableDict["author"]==1 and tableDict["publication"]==1):
        columns = "name_author,nbr_publication,date_pub,categorie,nbr_authors,article_title"
        if(tableDict["year"]==1):
            columns += ",year"
    elif(tableDict["author"]==0 and tableDict["publication"]==1):
        columns = "date_pub,categorie,nbr_authors,article_title"
        if(tableDict["year"]==1):
            columns += ",year"
    elif(tableDict["author"]==0 and tableDict["publication"]==0):
        columns=""
    if (tableDict["keyword"]==1):
        columns = "date_pub,article_title,id_publication"
        if(displayDict["affichage"]=="author"):
            columns= "name_author,id_publication"
    if (displayDict["affichage"]=="author" and (tableDict["author"]==1 or tableDict["publication"]==1) and tableDict["keyword"]==0):
        columns += ",id_publication"
    if (displayDict["affichage"]=="keyword" and (tableDict["author"]==1 or tableDict["publication"]==1) and tableDict["keyword"]==0):
        columns += ",id_publication"
```

Figure 33

On récupère ainsi les valeurs des checkBox pour savoir à quels jeux de données on s'intéresse. On les reporte dans la clause FROM de la requête. Ainsi cela réduit le temps d'exécution.

Enfin pour la clause WHERE on récupère la valeur des filtres qu'on affecte aux noms de colonnes correspondants.

On rajoute enfin une clause LIMIT à la requête si l'utilisateur a indiqué un nombre de noeuds maximal.

Une fois la requête construite, c'est la fonction run() qui est censée renvoyer le jeu de données correspondant.

Ainsi, cette fonction prend elle aussi les trois dictionnaires en paramètre, en plus de la requête construite précédemment.

Ici aussi, comme lors de l'import de données, on va utiliser la librairie dataframe\_sql qui est venue remplacer pandasql dans notre méthodologie de programmation.

En effet, grâce à ses fonctions query(), nous sommes en capacité de requêter les jeux de données déjà créés au début du programme.

## 4.5.6 Le graphe

**4.5.6.1 Aperçu du graphe et fonctionnalités** Pour programmer le graphe nous avons utilisé la librairie NETWORKX. Cette librairie permet de créer, manipuler et d'étudier les graphes de relations.

Avant de commencer l'analyse de la fonction qui développe le graphe prenons un aperçu de ce dernier :

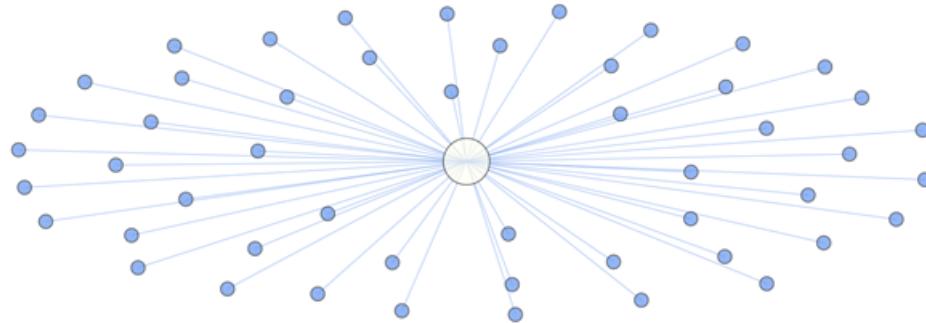


Figure 34

Ici, on voit un nœud central qui correspond au mot-clé linear tandis que les nœuds correspondent aux publications qui le contiennent.

Quand on passe la souris sur les nœuds voici ce qu'il se passe :

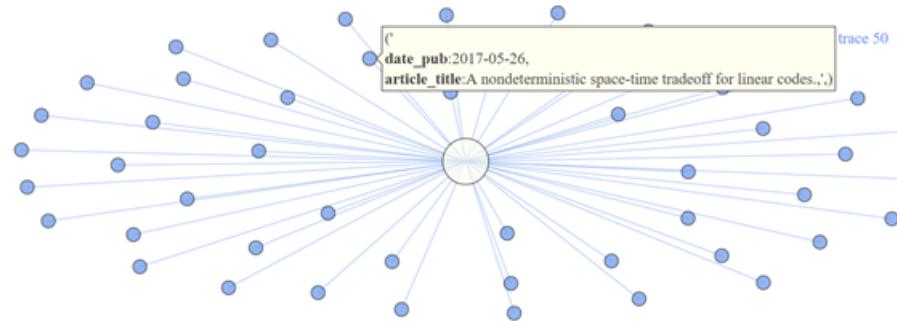


Figure 35

Ainsi on peut accéder aux valeurs correspondant à la publication en question. C'est la notion d'interactivité du graphe qui nous était demandée et qui selon nous répondait à la problématique initiale.

De plus, il est possible de zoomer sur le graphe grâce aux outils du package networkx. Il est également possible de télécharger le graphe en tant que fichier png etc...

Il s'agit d'un graphe simple car selon nous relier les nœuds en fonction d'une autre variable non renseignée par l'utilisateur l'aurait porté à confusion.

Les couleurs du graphe ont été choisies en fonction des remarques faites par les professeurs et par ce qui nous semblait être le plus juste et appréciable par l'utilisateur.

Il est à noter que grâce au package networkx le graphe s'affiche dans une fenêtre web.

#### 4.5.6.2 Code et explications

C'est la fonction display qui permet de créer le graphe et qui est présente dans le fichier displaygraph.py.

Cette dernière fonction prend en paramètre les informations concernant le noeud central (base de la requête) ainsi que le dataframe résultat de la requête. La fonction construit les nœuds, les connexions avec le noeud central et affiche le graphe dans un navigateur.

Ainsi, nous allons l'étudier pas par pas :

```
df = dfResult
# Etape 1 : recuperer les noms de colonnes du dataframe
# ainsi que la valeur du noeud central qui est le parametre de la fonction
colonnes = list()
colonnes = df.columns.tolist()
centralNode = name
```

**Figure 36**

Dans un premier temps, on récupère le noms des colonnes du dataframe ainsi que la valeur du noeud central.

```
node_names = [a for a in range(nbNoeuds)] #Les noms des noeuds seront des numéros les infos apparaîtront avec le curseur sur un noeud
node_weight = 3 #Rendre cette variable constante pour tous les noeuds

node_list = nx.Graph() #Create empty graph
node_list.add_node(centralNode, size = 10) #Add central node(author name)

for i in range(len(node_names)):
    node_list.add_node(node_names[i], size = node_weight) # add node for article
    node_list.add_edge(centralNode, node_names[i], weight = node_weight) #add connection to central node

pos_ = nx.spring_layout(node_list) # generate positions of nodes

edge_trace = {} # create custom edges
```

**Figure 37**

Ensuite, dans cette étape on crée un noeud pour chaque ligne du dataframe.

```
for node in node_list.nodes():
    text = ""
    for i in range(len(df.columns)):
        text += "<br><b>{0}</b>:{1}, ".format(df.columns[i],df[colonnes[i]].iloc[index])
    #data = df[df[colonnes[1]].isin([node])]
    hover_text = ""

    if(node == centralNode):
        hover_text = tuple(["{0}".format(centralNode)])
        color = tuple(['ivory'])
        size = tuple([5 * node_list.nodes()[node]['size']])
    else:
        hover_text = tuple([text])
        color = tuple(['cornflowerblue'])
        size = tuple([5 * node_list.nodes()[node]['size']])
```

**Figure 38**

Cette boucle permet quant à elle d'affecter un texte à chaque noeud.

```
for trace in edge_trace:
    fig.add_trace(trace)

fig.add_trace(node_trace)

#layout customization
fig.update_layout(showlegend = False)

fig.update_xaxes(showticklabels = False)
fig.update_yaxes(showticklabels = False)
plot(fig)
```

**Figure 39**

Enfin, voici la fin du code permettant de tracer le graphe.

#### 4.6 Conclusion

Notre projet portait sur la visualisation de données DBLP à l'aide d'un graphe de relations entre entités.

Au final, ce projet aura été une première pour un bon nombre d'entre-nous en Python. Il nous a permis d'appréhender de nombreuses librairies comme networkx ou dataframe\_sql. Ce projet nous a permis d'appréhender les différents aspects du langage Python avec la manipulation de classe, dictionnaires mais aussi librairies et modules graphiques.

Grâce à ce projet nous avons pu appréhender ce que pouvaient être les exigences du monde professionnel. De la manipulation de gros volumes de données à l'introduction aux graphes de relation nous avons pu nous enrichir tout au long de ce projet.

L'impact du travail à distance s'est fait ressentir dans une moindre mesure car nous pouvions également nous réunir avec des outils comme Discord etc... Cela nous a donc permis d'appréhender ce que pourraient être au futur la situation de nombreux employés en télé-travail.

Il reste selon nous des axes d'améliorations de notre travail, à commencer par l'interactivité du graphe que nous aurions pu pousser plus loin. Néanmoins ce dernier, bien que minimalistique, reste entièrement fonctionnel et facile à prendre en main pour un utilisateur novice.

## 5 GROUPE DATA VISUALISATION 2

### 5.1 Noms et prénoms des participants

- **Baptiste Esposito** (Product Owner)
- **Gwladys Kerhoas** (Scrum Master)
- **David Huynh**
- **Anne-Sophie Koch**
- **Nicolas Mateos**
- **Yvan Bourdin**
- **Elisa Frintz**

### 5.2 Introduction

Le sujet de notre groupe porte également, comme la partie précédente, sur la visualisation des données DBLP sous la forme de graphes. Il s'agit d'une structure s'avérant particulièrement bien adaptée aux réseaux d'informations. L'objectif de notre projet est de mettre en évidence les liens entre les entités principales des fichiers de données afin d'illustrer le réseau d'information de manière exhaustive. Ainsi, notre projet se découpe selon trois perspectives de travail se regroupant par la suite pour générer à l'utilisateur une petite interface de requête.

Nous avons donc travaillé sur la proposition de requête et leur exécution sur les données DBLP. Ensuite, nous avons créé une interface de requêtage accessible pour l'utilisateur afin de lui générer les résultats de requêtes selon ses choix sur l'interface. Enfin, nous avons affiché ces différents résultats sous forme de graphes, révélant ainsi les liens entre les différentes entités de nos données.

Par ailleurs, l'explication de notre outil de visualisation des données est précédée d'une partie développant la gestion de notre projet selon la méthode Scrum. C'est une pratique notamment très utilisée dans le domaine informatique qui nous a permis de mieux appréhender la création de notre outil.

### 5.3 Gestion de Projet

#### 5.3.1 L'approche SCRUM

Agile représente un ensemble de « méthodes et pratiques » basées sur les valeurs et les principes du Manifeste Agile. Ces méthodes reposent sur une approche itérative, incrémentale et flexible concernant les besoins d'un client. Ces méthodes permettent au client de mieux s'intégrer dans l'équipe de développement du projet concerné. Le « mouvement méthodologie agile » repose sur 4 principes : la collaboration entre les membres de l'équipe est préférée aux outils, le développement du produit à la documentation, la collaboration avec le client à la négociation et enfin la flexibilité est préférée à la planification trop rigide. Le centre de gravité n'est donc plus le projet en lui-même mais le produit, rendu fini au client.

La méthode Scrum est utilisée pour implémenter la méthode Agile de développement et de gestion de projet, généralement présentée dans le domaine de l'informatique, et utilisant des termes spécifiques. L'approche SCRUM permet de répondre à la demande d'un client, aux problèmes complexes et changeants, en les divisant en des tâches individuelles, en priorisant chacune de ces tâches et en attribuant à chacune d'entre elles une time-box et une ou plusieurs personnes.

Ainsi, le Scrum Master s'occupe du fonctionnement du projet, le Product owner, des besoins du client et l'équipe de développement, de la réalisation du produit. Aucun matériel n'est nécessaire pour utiliser la méthode Agile. Il faut simplement réussir à bien respecter les principes et les valeurs de Scrum.

Pour l'élaboration de notre projet, nous avons ainsi utilisé la méthode SCRUM qui sera développée par la suite.

### 5.3.2 Gestion du projet selon la vision du Scrum Master

Dans cette première partie, la gestion du projet est développée selon le point de vue du Scrum Master et évoque notamment la gestion de l'avancé du livrable à rendre au client, selon la méthode Scrum.

**5.3.2.1** Intégration de Scrum à notre projet Scrum est constituée de trois éléments importants : les rôles dans l'équipe projet, les sprints ou itérations, ainsi que les règles de SCRUM (product backlog, sprint backlog, definition of done...).

Au sein de notre équipe projet, nous avons attribué les rôles de la manière suivante :

- Product Owner (responsable du product backlog) : Baptiste Esposito
- Scrum Master (vieille à la mise en œuvre de la méthode agile) : Gwladys Kerhoas
- Équipe de Développement : David Huynh, Anne-Sophie Koch, Nicolas Mateos, Yvan Bourdin, Elisa Flintz

Dans le respect des règles Scrum, à chaque début de séance, nous avons appliqué le « Daily Scrum », une réunion quotidienne où chacun fait part aux autres de son avancement et sur les problèmes qu'il a rencontré. De cette façon, nous avons pu voir qui était bloqué sur une tâche en cours et nous avons essayé de trouver qui pouvait lui venir en aide pour qu'il puisse avancer. De plus, cette petite entrevue avant chaque séance de travail nous a permis de fixer les objectifs de la séance et de bien se répartir les tâches pour éviter que plusieurs personnes ne travaillent sur le même item.

Cependant, pour répartir ces items, il a fallu les définir préalablement dans ce que l'on appelle dans la méthode Scrum, notre backlog. Ainsi, le product owner de notre équipe projet, a été chargé de définir les besoins du client au sein d'un carnet produit en exprimant clairement chaque item. Ce travail a été le point de départ de notre projet et nous a permis de définir quels étaient les objectifs premiers du produit à rendre. Pour prioriser les items du product Backlog, nous avons choisi d'associer à chaque item, une étiquette de couleur pour quantifier la difficulté d'un item illustrant la time-box, ou du temps nécessaire pour la réalisation de la tâche. Ci-dessous, il s'agit d'un extrait de notre backlog avec les items priorisés.



**Figure 40.** Extrait du product backlog

Par ailleurs, le Scrum Master a produit au début du projet une prévision<sup>46</sup> sur les times box associé à chacun de ces items du product backlog. Cette démarche fait partie des principes de la méthode Scrum et permet de comparer le temps passé sur les tâches avec la trajectoire idéale. En effet, le Scrum Master alloue une unité de temps maximale ou « time box », fixe pour un item qui doit être respecté par la suite lors de la conception du produit. Le but d'une time box est donc de définir et de limiter le temps consacré à un item.

Lorsqu'on additionne l'ensemble des times box prévues, on obtient un total général de 179 heures pour tous les membres de l'équipe projet. Ainsi, chaque personne travaillera environ 26 heures sur la conception de ce produit, soit 3 heures par semaine. Il s'agit là d'une prévision et non de la réalité du temps passé par chaque membre de l'équipe. L'objectif, au niveau de la gestion du projet, était donc de se rapprocher au mieux de cette prévision.

### 5.3.2.2 Planification des tâches

Après avoir réalisé les prévisions sur les items de notre backlog, il fallait associer ces tâches aux membres de l'équipe pour chaque nouveau sprint.

L'un des principes les plus importants dans la méthode Scrum est le principe de « transparence ». Certaines informations doivent être accessibles par tous, comme la tâche en cours de chacun, son état d'avancement et le sprint actuel de l'équipe. Cela permet ainsi à toute l'équipe de se placer temporellement dans l'avancé du livrable et également pour que chacun se positionne sur une tâche en particulier à effectuer. D'où l'importance que ces informations soient visibles en permanence.

Ainsi, dès le début de notre projet, nous avons utilisé le tableau Trello<sup>47</sup> comme tableau Scrum pour organiser notre backlog, les tâches du sprint en cours et leur état d'avancement. Trello est accessible par tous les membres de l'équipe et actualise les modifications en temps réel, permettant un gain de temps considérable sur la gestion du projet.

Nous avons essayé de reprendre le modèle « To do, In progress, Done » à l'intérieur de chacun de nos sprints. Ainsi, le « To do » est modélisé par le sprint en cours annoncé par le

Scrum Master à chaque début de séance. Ensuite, pour simuler « In progress », nous avons ajouté des étiquettes pour attribuer à un membre de l'équipe une tâche. Enfin, une étiquette « Done » est attribuée sur les tâches qui sont finies pour une nouvelle fois visualiser l'avancé du produit et reproduire le modèle Scrum.

De plus sur un item en particulier, on peut y spécifier un détail ou des informations en commentaires ou encore des pièces jointes. On peut également ajouter des checklists pour montrer l'avancé du travail sur une tâche spécifique.

Au sein de notre équipe projet, nous avons remarqué que lorsque les tâches sont clairement définies pour le sprint sur lequel on travaille, l'attribution des tâches se fait plus rapidement par la suite et le projet avance plus vite.

**5.3.2.3 Les Sprints** Dans la méthode agile de gestion de projet, Scrum utilise des sprints comme des intervalles de temps pendant lesquels l'équipe va compléter un certain nombre de tâches du backlog que nous avons prédefinis en amont. Le Scrum Master, avec l'aide de l'équipe de développement, a ainsi défini les sprints<sup>48</sup> de ce projet. L'intervalle de temps dépend des besoins de l'équipe mais communément, il s'agit d'un intervalle de deux semaines qui est instauré.

Un autre principe tout aussi important que la « transparence » est le principe d'itération dans les sprints. La méthode agile de gestion de projet et le framework Scrum est basé sur une méthode itérative de livrables du produit. Au lieu d'attendre que le projet soit finalisé dans son entièreté, on délivre au client un produit utilisable mais pas final.

A la fin de chaque sprint, le projet doit avoir atteint un stade d'avancement suffisamment complet pour pouvoir montrer quelque chose au client. Cela permet de collecter un retour de la part du client plus tôt afin de continuer à être guider sur le développement du produit et avoir une confirmation sur la validité du livrable. De cette manière, nous sommes sûr de répondre réellement aux attentes du client. Ici, dans le cadre de notre projet, cette fin d'itération pouvait être modélisée par les entretiens avec les enseignants sur l'avancement de notre projet. Nous avons ainsi pu leur montrer nos différentes visualisations ou l'interface pour confirmer que nous répondions bien aux besoins du client.

Au-delà de l'importance des itérations et des améliorations sur le produit, Scrum s'attache à améliorer le processus à chaque nouveau cycle. Ainsi, chaque sprint se termine avec une rétrospective de celui-ci permettant à l'équipe de discuter des améliorations possibles à effectuer sur le prochain sprint. Dans cette dynamique, nous avons pu observer quels points étaient à améliorer et par exemple, lors du sprint 1, nous avons été trop ambitieux sur la prévision de nos sprints ou encore nous étions légèrement en retard sur la préparation de notre oral et sur le rapport à la fin du sprint 3.

**5.3.2.4 Le Sprint Backlog et le Burndown Chart** Le sprint Backlog est l'un des artefacts de Scrum. Il existe trois artefacts autour du backlog : le product backlog correspondant à l'ensemble des items à réaliser lors du projet, le sprint backlog incluant seulement les items à réaliser pendant le sprint en cours et enfin, l'increment backlog (items déjà réalisés, « done »).

Lors de notre projet, nous avons tenu un suivi des itérations de chaque sprint<sup>49</sup> ainsi qu'un suivi des itérations individuel<sup>50</sup> pour chaque membre de l'équipe. Le suivi des itérations correspondant au sprint backlog nous a donc permis de vérifier si le temps passé sur les tâches suivait bien notre trajectoire idéale.

Pour correspondre parfaitement avec notre prévision du backlog, nous sommes partis avec un total du nombre d'heures arrondi à 180 heures ; le total général prévisionnel étant égal à 179 heures précisément. Ensuite, on décrémente de séance en séance de 12 heures, correspondant à un cours de 105min pour 7 personnes au total. En réalité, le sprint Backlog sert à calculer la somme totale de travail restant à n'importe quel moment du sprint.

A la fin du projet, on peut voir sur le sprint backlog que nous avons dépassé le temps de 5,5 heures. Ce dépassement peut paraître anodin à première vue mais il représente tout de même une perte de temps que l'on aurait pu maîtriser lors de nos sprints. Notamment, ce surplus d'heures peut s'expliquer par la mauvaise gestion du sprint 1 où nous ne savions pas réellement comment s'y prendre pour implémenter le code. La mise en marche de la création du code n'a pas été évidente car la notion de graphe et d'interface sur Python était une chose nouvelle pour notre groupe projet. Cependant, nous avons par la suite tenté de rattraper ce retard dans le sprint 3 où notre compréhension concernant les attentes du client était meilleure.

De plus, le burndown chart est une représentation du sprint backlog prenant en ordonné, la quantité de travail restant à effectuer et en abscisse, la durée du projet. Il permet alors de modéliser et mieux comprendre l'avancée du projet.

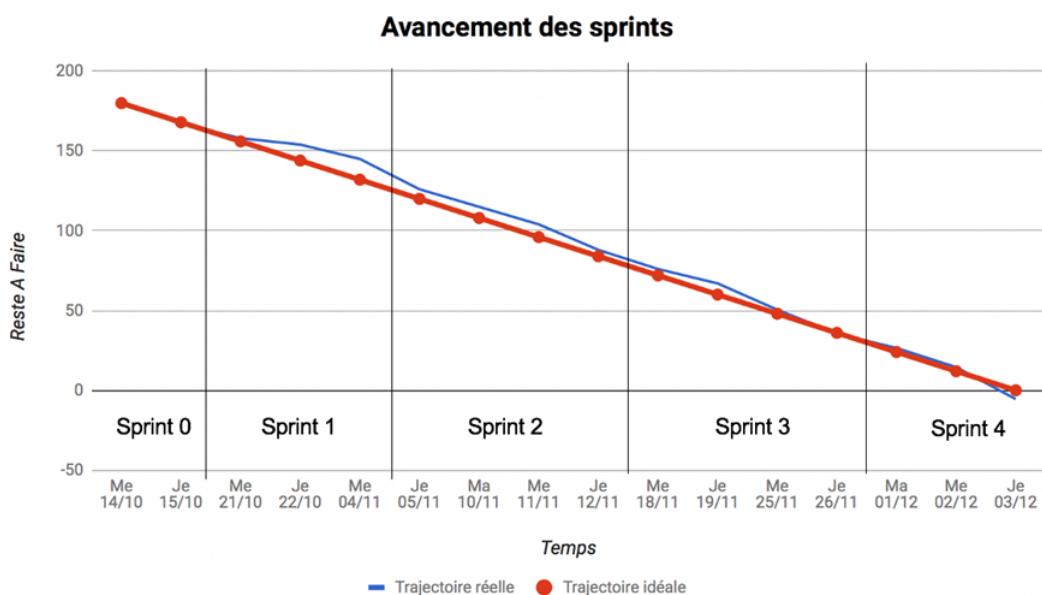


Figure 41. Burndown Chart

Sur ce Burndown Chart, on peut à nouveau voir nos conclusions sur le retard concernant le sprint 1. Par ailleurs, le Scrum Master a également détaillé chacun de ces sprints avec un burndown chart<sup>51</sup> pour mettre en évidence les points d'écart entre la trajectoire idéale et la trajectoire réelle. Ainsi, le rôle du Scrum master est d'accompagner la progression de l'équipe en assurant la transparence du sprint backlog.

**5.3.2.5 Stratégie de travail** Nous avons choisi de banaliser l'après-midi du mardi pour avancer ensemble sur le projet et pour faire des points trois fois par semaine, en comptabilisant les séances de projet intégré et de gestion de projet, soit le mardi, le mercredi, et le jeudi.

Concernant notre stratégie de travail, nous avons évolué selon deux sous-groupes différents : un premier groupe travaillant sur l'interface pour l'utilisateur et les requêtes

associées et un sous-groupe pour la partie visualisation des données. De plus, nous avons explorer diverses possibilités de libraries de visualisation de graphe pour ensuite ne garder que celle qui nous semblait la meilleure et la plus pertinente pour le client.

En effet, lors de notre projet, nous étions au total 7 personnes à travailler ensemble. Nous pouvions ainsi multiplier les idées mais l'enjeu était, là, de ne pas trop se disperser et de garder en tête le livrable à rendre selon les attentes du client. Cependant, notre important nombre de personnes dans notre projet pouvait être plus dur pour trouver un accord entre chaque membre du groupe sur un questionnement.

### 5.3.3 Vision du livrable pour le client selon le Product Owner

La réflexion sur la prise en main de l'interface et la compréhension des graphes, par le client, lie la partie Gestion de projet avec celle du Projet intégré. Les différents éléments cités dans cette partie seront expliqués en détails dans le projet intégré, ici nous allons expliquer les choix faits pour l'utilisateur.

Pour ce qui est de l'interface, les multiples barres de recherches permettent de guider l'utilisateur. Le but étant que, s'il recherche une œuvre, les informations de l'œuvre apparaissent. Si un nom d'auteur est le même que le titre d'une publication on ne peut pas produire le graphe voulu.



**Figure 42.** Extrait de la partie supérieure de l'interface

Les cases à cocher résultent de la même réflexion. Un utilisateur peut choisir les informations à afficher et c'est l'utilisateur qui crée la pertinence de son graphe.



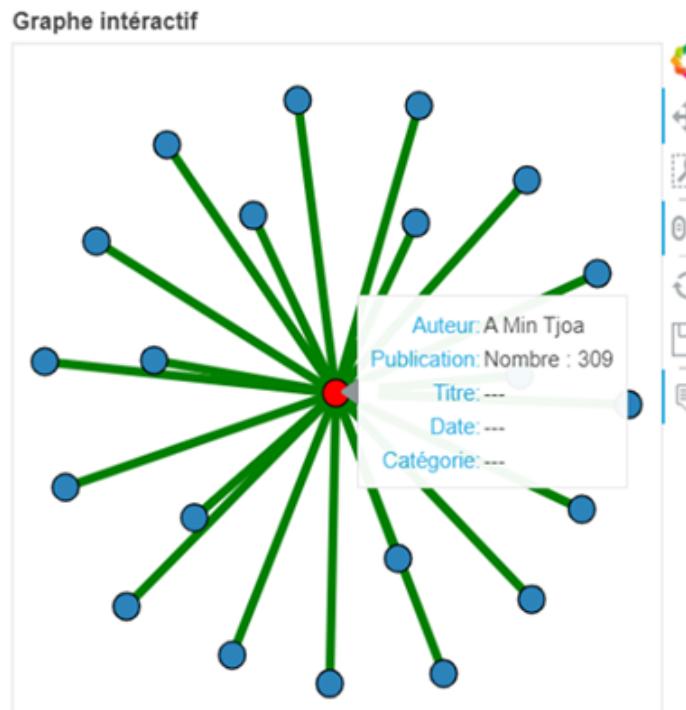
**Figure 43.** Extrait de la partie inférieure de l'interface

La seconde partie en lien avec le client est l'affichage du graphe. Notre but étant de donner un accès visuel aux données, il nous faut des graphes lisibles. La partie interface

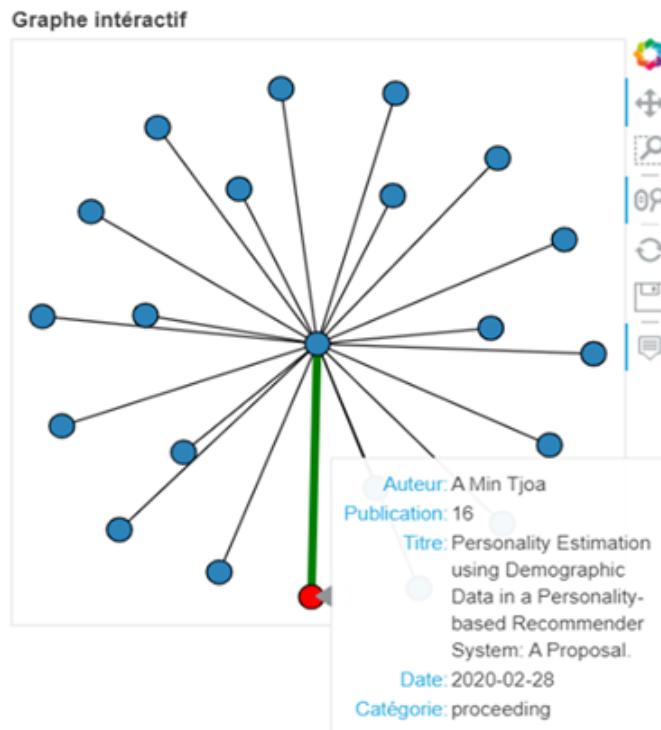
s'assure de créer les requêtes et les réponses appropriées à la demande. Ici, dans le cadre des graphes dynamiques, on ne gère que le nombre d'information affichée. Pour pallier aux contraintes posées pour faire apparaître les données, il vous faudra passer la souris sur le nœud.

L'idée des graphes générés en fonction des requêtes demandées par le client est de pouvoir afficher un nœud principal au centre ainsi que des nœuds secondaires (ou sommets). Le nœud central correspond aux informations liées au nom saisi dans le champ de texte de l'interface. Quant aux sommets, ils dépendent des cases cochées par le client dans la partie inférieure de l'interface. Un exemple ci-dessous avec le nom de l'auteur « A Min Tjoa » saisi, va créer un graphe avec les informations de cet auteur au centre et celles de ses publications autour.

Les graphes créés sont interactifs notamment via la barre d'outils qui se trouve sur la droite du graphe. Comme dit précédemment, on peut passer la souris sur un nœud pour afficher les informations de celui-ci. Si les options sont activées, on peut également se déplacer dans le graphe, zoomer de différentes manières (molette souris ou sélection d'une zone d agrandissement) ou encore réinitialiser le graphe à sa position de départ. Enfin, il est possible de sauvegarder le graphe pour le télécharger en format PNG. Dernier point, le nœud survolé et les arêtes qui lui sont associées sont mis en valeur pour permettre une meilleure lisibilité.



**Figure 44.** Exemple des informations d'un nœud central



**Figure 45.** Exemple des caractéristiques d'un nœud secondaire

#### **5.3.4 Table des annexes**

- Annexe 1 : Prévision du Product Backlog
- Annexe 2 : Utilisation de Trello
- Annexe 3 : Détails des sprints
- Annexe 4 : Sprint Backlog
- Annexe 5 : Sprint Individuel
- Annexe 6 : Burndown Chart pour chaque graphique
- Annexe 7 : Compte rendu des séances

Prévision Backlog	Sprint	Dates prévues	Items	Heures prévues	
				Sous-tâches	
			Mise en place du projet	Constitution équipe projet - Reflexion sur le sujet	5
				Etude des données	7
				Création Product Backlog	5
				Recherche librairies et idées pertinentes de graphes	4
				<b>Total</b>	<b>21</b>
			Importation et premier affichage de graphes	Importation des données sous Python Test des graphes avec le jeu de données initial	3
				Recherche de requête à implémenter	24
				Gestion du projet, Méthode Agile	9
				Rapport - Traitement des données	4
				<b>Total</b>	<b>42</b>
			Création du graphe général	Constitution du graphe général Ajout des requêtes associées aux graphes	6
				Requêtes dynamiques liées à l'interface	8
				Rapport - Construction graphe et requêtage	11
				<b>Total</b>	<b>32</b>
			Dynamisation des graphes	Rendre les graphes dynamiques (interroger l'affichage), Rassembler les différentes parties du projet entre elles	15
				Rapport Projet Intégré/Gestion de Projet	21
				<b>Total</b>	<b>6</b>
			Rendu du produit	Rapport final Présentation projet à l'oral	28
				<b>Total</b>	<b>14</b>
				<b>Total général</b>	<b>179</b>

**Figure 46.** Annexe 1 : Prévision du Product Backlog

**Informations générales**

- Liens utiles**: 4
  - 3/5 Choisir de filtrer les publications selon leur catégorie
  - 2/5 Librairies :
  - Donner des étiquettes aux missions en fonction de la difficulté/du temps : Code couleur
  - Compte rendu des séances Projet Intégré et Gestion de Projet

**Backlog (liste des items)**

- 3/5 Choisir sur un auteur pour accéder à ses publications
- 2/5 Cliquer sur une publication pour connaître son contenu
- 3/5 Cliquer sur une publication pour connaître la liste des co-auteurs de celle-ci
- 2/5 Cliquer sur un mot-clé pour accéder à toutes les publications comportant ce mot-clé
- 2/5 Cliquer sur un événement pour connaître la liste des auteurs présents
- 3/5 Filtrer les publications selon leur année

**SPRINT 0 (14/10/20 - 20/10/20)**

- 3/5 Présentation de l'équipe
- 2/5 Constitution de l'équipe Scrum (10min)
- 2/5 Elaboration du Product Backlog
- 2/5 Priorisation du Product Backlog
- 2/5 Définir les time box associées à chaque items
- 2/5 Planification des sprints
- 3/5 Sprint Burndown Chart (visualisation)/diagramme de la

**SPRINT 1 (21/10/20 - 03/11/20)**

- 2/5 Importation des données sur Python
- 3/5 Afficher quelques graphiques généraux (visualisations générales)
- 3/5 Proposer des requêtes
- 2/5 Proposer des graphes
- 4/5 Se coordonner avec l'équipe Interface
- 2/5 Rapport - Création du graphe général
- 3/5 Création de l'interface avec la librairie Tkinter

**SPRINT 2 (04/11/20 - 17/11/20)**

- 4/5 Creation du graphe général avec les bibliothèques Python
- 3/5 Creation des graphes associés aux requêtes
- 3/5 Requêtes ""dynamiques"" (liées à l'interface)

Figure 47. Annexe 2 : Utilisation de Trello

## Rapport Projet Intégré & Gestion de projet

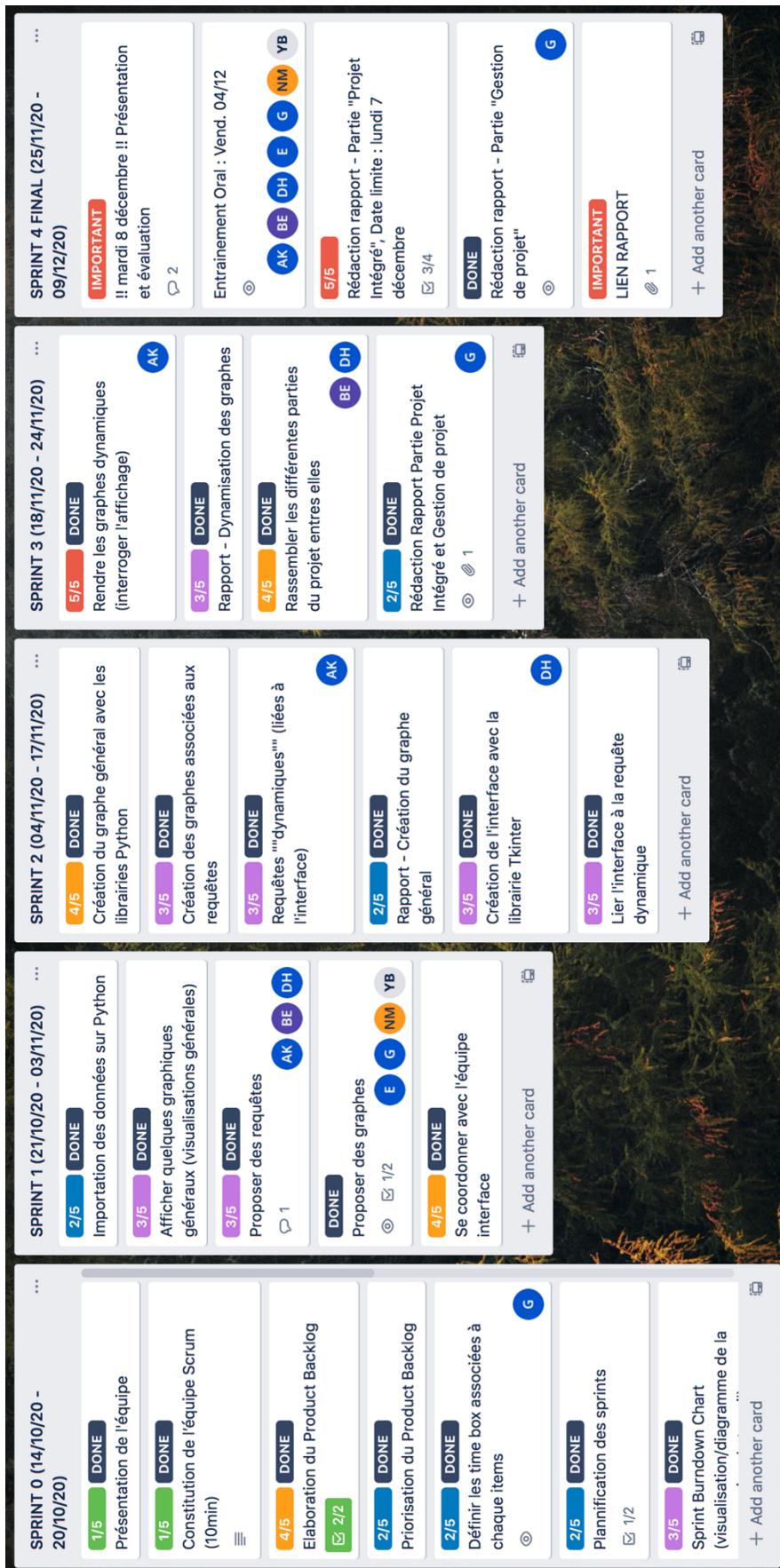


Figure 48. Annexe 3 : Détails des sprints

Sprint Backlog			Nombre d'heures à multiplier par le nombre de personnes sur la tâche												
SPRINT	ITEMS	Sous-Tâche	RESTE À FAIRE												
Sprint 0	Mise en place du projet	Constitution équipe Scrum	8,5												
		Elaboration du Product Backlog	4												
		Priorisation du Product Backlog	1												
		Définir les time box associées à chaque items	1												
		Sprint Burndown Chart	1												
		Lister les librairies à utiliser	2												
		Diagramme UML des données	1												
		Mise en place du Git	1												
		Idée de graphe pertinent	2												
Sprint 1	Importation et premier affichage de graphes	Importation des données sur Python	3												
		Analyses des fichiers csv et premières visualisations	6												
		Rapport - Rédaction du traitement des données	1												
		Création de graphe	6												
		Création des requêtes associées aux graphes	2												
Sprint 2	Création du graphe général	Coordination avec le groupe interface	1												
		Constitution du graphe général	9												
		Ajout des requêtes associées aux graphes	6												
		Requêtes dynamiques liées à l'interface	4												
Sprint 3	Dynamisation du graphe général	Rapport - Traitement des données	2												
		Rapport - Construction de graphe et requête	2												
		Rendre les graphes dynamiques (interroger l'affichage)	9												
Sprint 4	Rendu du produit	Rassembler les différentes parties du projet entre elles	6												
		Rapport Projet Intégré/Gestion de Projet	6												
		Rapport final	3												
Préparation présentation oral du produit			9												
Sprint			13												
Jours d'itération			7												
Me 14/10			0												
Me 15/10			1												
Me 21/10			2												
Me 22/10			1												
Me 04/11			3												
Me 10/11			4												
Me 11/11			4												
Me 12/11			4												
Me 18/11			4												
Me 19/11			4												
Me 25/11			4												
Je 26/11			4												
Ma 01/12			4												
Ma 02/12			4												
Trajectoire réelle			4												
Trajectoire idéale			4												

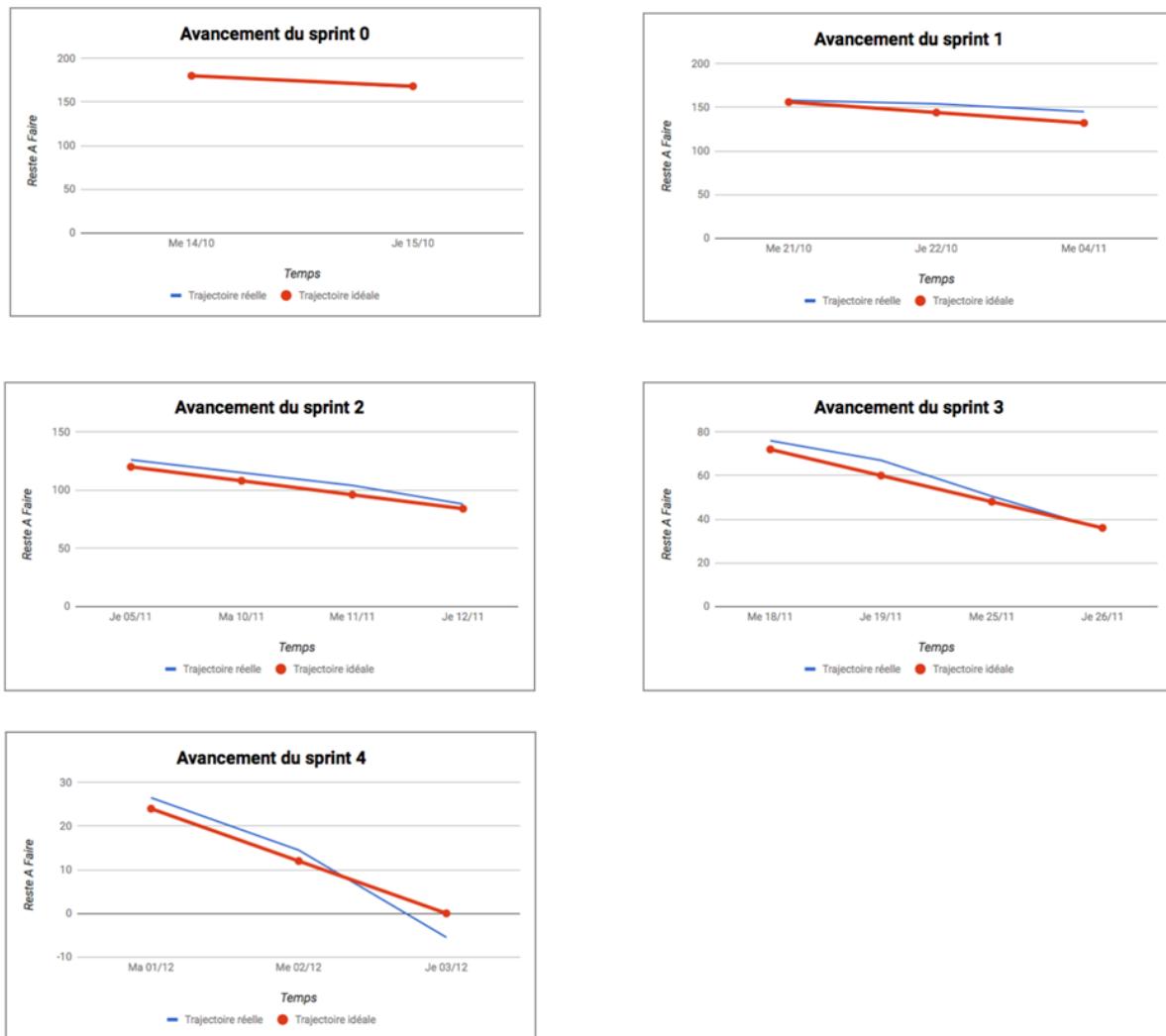
Figure 49. Annexe 4 : Sprint Backlog

# Rapport Projet Intégré & Gestion de projet

---

<b>Sprint Individuel</b>								
<b>Nombre d'heures travaillées par séance et en dehors des séances</b> <b>1 séance = 1,5</b>								
<b>SPRINT 0</b>								
	Anne-Sophie	Baptiste	David	Elisa	Nicolas	Yvan	Gwladys	Total
Me 14/10	1,5	1,5	1,5	1,5	1,5	1,5	1,5	10,5
Je 15/10	1,5	1,5	1,5	1,5	1,5	1,5	1,5	10,5
<b>Total</b>	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>	<b>21</b>
<b>SPRINT 1</b>								
	Anne-Sophie	Baptiste	David	Elisa	Nicolas	Yvan	Gwladys	Total
Me 21/10	1,5	1,5	1,5	1,5	1,5	1,5	1,5	10,5
Je 22/10	1,5	1,5	1,5	1,5	1,5	1,5	1,5	10,5
Me 04/11	1,5	1,5	1,5	1,5	1,5	1,5	1	10
<b>Total</b>	<b>4,5</b>	<b>4,5</b>	<b>4,5</b>	<b>4,5</b>	<b>4,5</b>	<b>4,5</b>	<b>4</b>	<b>31</b>
<b>SPRINT 2</b>								
	Anne-Sophie	Baptiste	David	Elisa	Nicolas	Yvan	Gwladys	Total
Je 05/11	1,5	1,5	3	2	2,5	2	2	14,5
dim. 08/11			1		1,5			2,5
Ma 10/11	4	2	3	3	2	3	2	19
Me 11/11		1	2,5	1	4	1	3	12,5
Je 12/11	1	1,5		1				2,5
sam. 14/11			1					1
dim. 15/11			1					1
Ma 17/11		1		1		2	2	6
<b>Total</b>	<b>6,5</b>	<b>7</b>	<b>11,5</b>	<b>7</b>	<b>10</b>	<b>8</b>	<b>9</b>	<b>59</b>
<b>SPRINT 3</b>								
	Anne-Sophie	Baptiste	David	Elisa	Nicolas	Yvan	Gwladys	Total
Me 18/11		1,5	1,5	2,5	1,5	3,5	1,5	12
Je 19/11	1	1,5	1	1	1	1	1	7,5
Ve 20/11								1
Di 22/11			1					1
Lu 23/11						1	1	2
Ma 24/11	1	1	1	1	1,5	1	2	8,5
Me 25/11	1	1,5	1	1	1,5	3,5		9,5
Je 26/11	4	1,5		1	1	2		9,5
Di 29/11		1				1		2
Lu 30/11		1	2	1	1,5	2		3
Ma 01/12	2		1,5	2,5	3	3,5		
<b>Total</b>	<b>9</b>	<b>9</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>19,5</b>	<b>8,5</b>	<b>53</b>
<b>SPRINT 4</b>								
	Anne-Sophie	Baptiste	David	Elisa	Nicolas	Yvan	Gwladys	Total
Me 02/12	1,5	1,5	2	2	2,5	2	3	14,5
Je 03/12	3	2,5	1,5	2	3	5	2	19
Ve 04/12	4	3	1,5	1		5	2	16,5
<b>Total</b>	<b>8,5</b>	<b>7</b>	<b>5</b>	<b>5</b>	<b>5,5</b>	<b>12</b>	<b>7</b>	<b>50</b>

**Figure 50.** Annexe 5 : Sprint Individuel



**Figure 51.** Annexe 6 : Burndown Chart pour chaque graphique

## Annexe 7 : Compte rendu des séances

### Séance 1 (07/10/2020) – Projet intégré

Il faut bien étudier les données avant de commencer à faire du code sur Python. On peut faire un schéma relationnel pour bien comprendre les données. Cependant, il n'y a pas de SGBD à utiliser pour ce projet. De plus, les données ont déjà été nettoyé.

Pour visualiser les données, il faut réfléchir à la meilleure visualisation nécessitant d'être dynamique (recherche de librairie python) et donc de rechercher des requêtes pertinentes associé au graphe : tout doit être automatique.

A la fin du projet, il faut rendre un rapport par groupe de TD comportant chacune des parties des différents groupes projet.

### Séance 2 (14/10/2020) – Projet intégré

Explication des différents sujets données :

Interface et gestion de données	Visualisation dynamique des données	Analyse des données
- Création d'une interface	- Création de graphe	- Identifier des classes de thème
- Nettoyage des données	- Requête paramétrable (à la demande au niveau de l'affichage)	- Prédiction, clustering
- Participation à l'intégration de l'ensemble des données	- Visualisation des données	- Méthode sophistiquée, de prédiction

Préalablement à la constitution des graphes, il faut visualiser les données à l'aide d'histogramme ou de camembert par exemple.

Le groupe « Interface » s'occupe du traitement des données pour l'ensemble du groupe TD. Cependant, si nous avons besoin d'une certaine forme de données, c'est à nous de pré-traiter nos données pour obtenir le bon format pour la visualisation souhaitée.

Pour notre sujet portant sur la visualisation des données, il faut chercher des types de questions pour y répondre à l'aide du graphe (requête type/simple). On peut s'inspirer des réseaux sociaux pour construire des graphes homogènes/hétérogènes.

Dans le graphe dynamique, il faut pouvoir zoomer sur un auteur ou sur un sous graphe et par exemple, connaître quel auteur travaille avec quel autre auteur.

### Séance 1 (15/10/2020) – Gestion de projet

Définition des différents rôles attribués dans l'équipe projet :

**Product Owner :** celui qui indique exactement ce qu'il veut comme produit final, donc c'est lui qui dicte ses règles, qui prend les décisions finales. Les principales tâches ou items sont définis par le product owner pour le product backlog et seront ensuite répartis sur différents sprints. (slide 19 cours)

**Scrum Master :** Il doit définir la time box pour chacun des items, autrement dit il faut construire le Sprint Burndown Chart pour définir les durées de chaque item.

### Séance 3 (21/10/2020) – Projet intégré

Concernant la création des graphes, on peut s'intéresser à un objet en particulier (un auteur, une publication...) et avoir une fenêtre qui s'ouvre pour l'utilisateur pour qu'il puisse définir ce sur quoi il aimera avoir accès (exemple : les auteurs avec qui travaillent tel auteur).

Il faut que l'utilisateur puisse avoir accès à une interface pour générer un affichage de graphe.

De plus, le graphe doit être dynamique et doit être basé sur des requêtes simples ; il ne faut pas chercher à être performant mais à faire de la visualisation de données.

Afin d'avoir un maximum d'idée, le mieux serait que chacun cherche 3-4 requêtes différentes pour ensuite confronter nos idées. L'important est de chercher à rendre visuel ces requêtes par le biais des graphes.

### Séance 2 (22/10/2020) – Gestion de Projet

Nous avons présenté notre avancement sur notre projet. Nous sommes revenus sur le sujet de l'étude pour comprendre de quel groupe provient les données que nous devons traiter et visualiser.

L'équipe « Analyse » produit des résultats sur lesquels nous allons apporter une autre visualisation. Ainsi, lorsqu'on récupère nos données, nous allons afficher nos données sous forme de graphe en s'appuyant sur des filtrages. Ensuite, il faut affiner les affichages et rendre le contenu dynamique à l'aide de nouvelles requêtes.

Nous avons donc redéfini nos sprints qui paraissait trop ambitieux.

Concernant la partie Méthode Agile, il faut donner au Scrum Master l'ensemble des tâches effectuées par les membres de l'équipe projet avec le temps passé sur la tâche, en sachant qu'un cours dur 105min. On peut alors comparer le temps passé sur les tâches avec la trajectoire idéal.

### Séance 4 (04/11/2020) – Projet intégré

Il est important qu'il y ait une interaction avec les utilisateurs car les requêtes ne doivent pas être prête à l'avance (pas de requêtes pré-enregistrées). L'utilisateur est censé savoir écrire une requête et non cliquer sur un bouton lui affichant le résultat d'une requête.

1. Faire en sorte que l'utilisateur puisse écrire une requête entière dans une barre de recherche.
2. Créer un menu pour cocher des fonctionnalités qui puissent être traduites en requête sur Python.
3. Afficher les résultats obtenus avec des graphes.
4. En cliquant sur un sommet, est-ce qu'il est possible d'avoir d'autres infos pertinentes ?

Faire une proposition de menu/ requêtes pertinentes pour la prochaine fois.

### Séance 3 (05/11/2020) – Gestion de Projet

Répartition du travail selon deux groupes :

- David, Anne-Sophie et Baptiste → Recherche de requêtes
- Nicolas, Elisa, Yvan, Gwladys → Recherche de graphes

Concernant la méthode Agile, il faut bien noter le temps réelle passé sur chacune des tâches du projet afin d'avoir le temps passé par chacun le plus proche de la réalité.

Pour la prochaine fois, il faudrait proposer des graphes plus concrets dans la représentation des données pour montrer notre avancé dans le projet.

### Séance 4 (12/11/2020) – Gestion de Projet

**Interface :** YEAR → changer en liste déroulante (liste combo) pour afficher plus d'années

**Graphe :** Il faudrait demander au DataViz TD1, quelle librairie ils ont utilisé pour la visualisation de leur graphe pour faire des zooms sur des nœuds du graphe. Il faut penser à rendre le graphe dynamique.

Pour rendre la visualisation des graphes plus claires, on pourrait penser à n'afficher le titre de la publication seulement qu'au passage du curseur sur le nœud du graphe, les titres des publications étant très longs.

**Livrable pour le 2 décembre :**

- Pas de PowerPoint à réaliser
- Rapport : 2 parties

**Partie Projet intégré :** Explication et détail technique avec beaucoup de précision (noté sur la qualité de description du travail technique pour que les profs puissent donner suite à ce que l'on a fait → ré-exploitation de notre travail doit être possible)

**Partie Gestion de Projet :** Comment on a utilisé la méthode Scrum pour mener à bien notre projet (Burndown, Backlog, Sprint). Noter sur la qualité de la description de notre méthode de gestion de projet.

### Séance 5 (18/11/2020) – Projet intégré

- Ne pas se mettre en retard sur le rapport
  - Faire le lien entre les différentes parties du projet
  - Bien séparé dans le rapport la partie Gestion de Projet de la partie Projet Intégré
- Partie Gestion de Projet : Méthode Agile
- Partie Projet Intégré : Explication des input, output et les méthodes utilisées

### Séance 5 (19/11/2020) – Gestion de Projet

- Date de soutenance va être décalée
- 1ère Partie Rapport : Scrum
- 2ème Partie Rapport : Partie Technique (assez développé) → Il faut avoir suffisamment de détails pour que le projet puisse être repris par la suite :
  - Expliquer le choix des library
  - Expliquer et commenter les algorithmes

- Visualisation des graphes : expliquer et commenter (chercher ou expliquer quelles pourraient être les perspectives d'amélioration du graphe, avoir un esprit critique sur les résultats affichés)

Si les choses à insérer dans le rapport sont trop volumineuses, il ne faut pas hésiter à mettre des choses en annexe.

Interface (à améliorer) :

- Un seul bouton "Valider" pour confirmer la requête et pas plusieurs boutons.
- Pas besoin de cocher les cases car redondance → la partie des cases à cocher est à supprimer.

Graphes :

- Enrichir le code de Yvan et Elisa avec celui de Nicolas pour repartir sur un seul et même code

## Séance 6 (25/11/2020) – Projet intégré

Rapport final (rendu le lundi 7 décembre) :

- Un seul document par groupe TD
- Pour chaque groupe projet, une première partie sur la gestion de projet et une deuxième partie sur le projet intégré (pas obligé de rédiger selon les sprints, il faut être concis)

Oral de présentation : mardi 8 décembre, 12 min de présentation et 8 min de questions

Il faut que quelqu'un du groupe anime la présentation pour rendre le discours prêt et fluide.

Partie Intégration : tous les groupes doivent être impliqués dans l'intégration du projet.

Problème sur l'importance des données : prendre un gros jeu de données sans que ce soit la totalité des données, prendre un échantillon des données pour afficher les graphes.

Interface :

- Point positif : Plus que un seul bouton 'Valider' → plus facile à comprendre pour l'utilisateur
- Il faudrait améliorer la partie inférieure de l'interface → impression un peu brouillon (interface désuet), il faut que ce soit un peu plus jolie

## Séance 6 (26/11/2020) – Gestion de Projet

Il est important de bien s'entraîner pour l'oral de présentation du produit. Pendant les 12 minutes de présentation, il faut vendre le produit et donc trouver ce qui est essentiel à dire. Il faut répartir les rôles pour savoir qui parle sur quelle partie. Il faut également trouver le bon équilibre entre la généralité du produit et les détails importants à noter à l'oral.

Gestion du projet :

Le sprint final doit donc comprendre les répétitions pour l'oral. Nous avons un petit retard à rattraper concernant l'avancée du projet. Il ne faut pas tarder à rassembler toutes les parties du projet.

Affichage des graphes :

- Jauge de réglage dans la présentation ?
- Affichage des graphes sur une page web

## Séance 7 (02/12/2020) – Projet intégré

Questions :

- Est-ce que tous les membres du groupe doivent parler lors de la soutenance ? Oui
- Est-ce qu'il doit y avoir une petite partie sur la gestion de projet lors de l'oral ? On ne sait pas.

Oral : 2 parties :

- Requêtes = quel type de requête ? Qu'est-ce qu'on peut faire avec ? Dans quel but ?
- Graphe = Qu'est-ce qu'on peut modéliser ?

Il faut valoriser le travail réalisé, vendre le produit, être factuel.

Rapport :

- Guide utilisateur pour l'import des données et pour l'exécution du programme
- Détails les méthodes utilisées
- Illustrer nos propos par des exemples
- Avoir un regard critique sur les méthodes utilisées et sur les résultats des graphes

La partie Intégration vaut 10% de la note final de la partie projet intégré. Il faut rendre tout le code .py pour les enseignants.

## Séance 7 (03/12/2020) – Gestion de Projet

Rapport :

Chaque groupe projet doit avoir sa partie gestion de projet dans le rapport final par TD. Il doit donc y avoir une partie commune avant les 4 rapports pour présenter en général le projet. Nous sommes dans le dernier sprint, il faut donc affiner nos parties concernant le code et concernant le rapport. Il faut rendre le rapport plus joli, l'améliorer.

Oral :

Il faut vraiment organiser l'oral afin de montrer le maximum de fonctionnalités du produit en 12 minutes, il ne faut pas avoir de moment de battement lors de l'oral car ce sera du temps perdu pour ne pas vendre le produit.

Intégration du code :

Si l'intégration ne se fait pas entre chaque équipe projet dans le TD, c'est de la faute de tout le monde ; on partage dans la pénalisation ou la bonification si l'intégration est réussie.

## 5.4 Projet Intégré

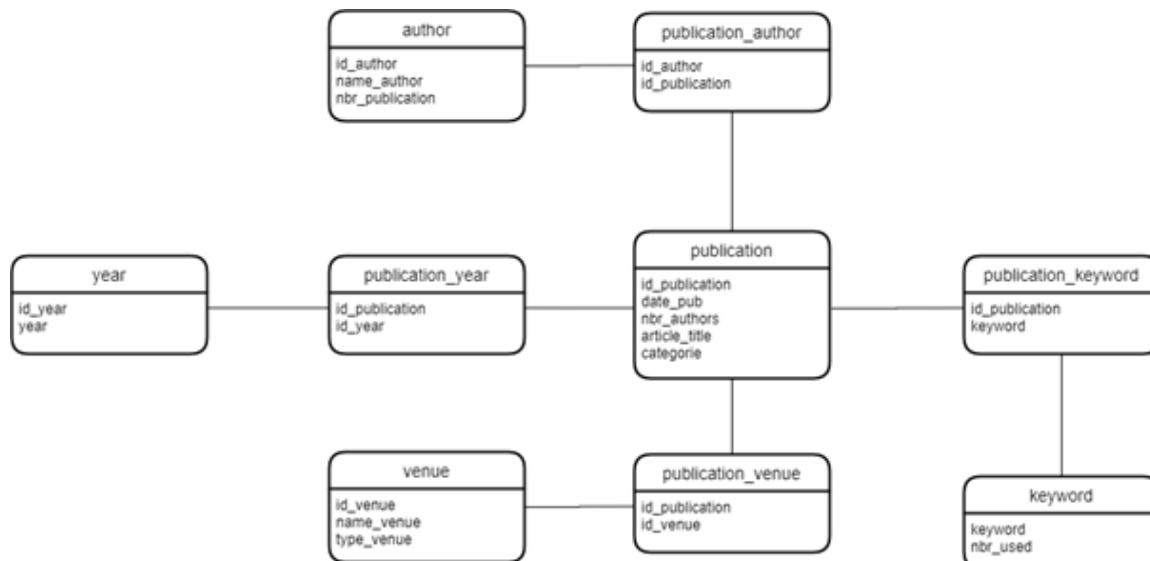
### 5.4.1 Présentation et importation des données

Le Digital Bibliography & Library Project (DBLP) est un site internet répertoriant un catalogue de bibliographies en informatique. L'objectif de notre sujet est de visualiser ces données qui ne cessent de croître quotidiennement.

L'importation des données est un processus de récupération de données à partir d'une ou plusieurs sources différentes. Ici, dans le cadre de notre projet, les données DBLP nous ont donc été fournies par le groupe s'occupant du traitement des données.

Avant de pouvoir construire et afficher le graphe correspondant à ce que l'utilisateur demande depuis l'interface, il faut récupérer les tables de données, étudiées et nettoyées en amont. Celles-ci sont modélisées dans notre étude de cas par des fichiers csv.

Les fichiers sont tout d'abord chargés dans le programme avec la fonction `read_csv` de la bibliothèque Pandas. Ces tables contiennent des informations sur les auteurs, les publications, les années ainsi que les événements et conférences. Elles sont reliées entre elles grâce à d'autres fichiers csv associant les identifiants de ces différents éléments. On peut représenter la structure de données par le diagramme UML ci-dessous.



**Figure 52.** Diagramme UML des données

### 5.4.2 Crédit d'une interface et requêtes associées

Le client a souhaité que l'on développe une petite interface graphique afin que l'utilisation de notre outil soit plus concrète et plus automatique. La création de cette interface ainsi que les requêtes associées vont être développés dans cette partie.

**5.4.2.1 Présentation de l'interface Graphique Dataviz** Lorsque l'utilisateur va exécuter le programme une fenêtre sous forme d'interface graphique « Recherche utilisateur » s'affichera et proposera un choix multiple à l'utilisateur pour ses recherches. Pour cela nous avons utilisé une librairie à l'aide d'un module présent par défaut dans Python : Tkinter. Ce module permet de créer des interfaces graphiques entre le langage Python et la bibliothèque **Tk**.

L'importation de cette bibliothèque se réalise principalement avec le code suivant mais peut s'importer avec d'autres manières différentes :

```
1 from tkinter import *
```

Dans un premier temps, il a fallu définir les caractéristiques principales qui définissent l'interface avec :

- Un nom (fenêtre)
- Un titre (fenetre.title())
- Un canvas (Canvas())

```
##### Fenetre principale avec ces caractéristiques
fenetre = Tk()
fenetre.title("Recherche utilisateur")
canvas = Canvas(fenetre, width = 700, height = 580)
canvas.pack(fill = "both", expand =True)
```

**Figure 53.** Fenetre principale avec ces caractéristiques

Ensuite, nous avons mis à disposition de l'utilisateur cinq zones de textes différentes qui vont lui permettent de lancer sa recherche en fonction du champ principal de chaque table (Publication, Auteur, Keyword, Venue, Year) :

- Nom d'une publication
- Nom d'un auteur
- Nom d'un mot-clef
- Nom d'un lieu
- Nom d'une année



**Figure 54.** Interface utilisateur dataviz 2

```
##### Zone de texte pour les publications/auteurs/keywords/venues/years
ZonedeT_publi = Entry(fenetre, width=30)
ZonedeT_publi.grid(row=1,column=1)

ZonedeT_author = Entry(fenetre, width=30)
ZonedeT_author.grid(row=2,column=1)

ZonedeT_keyword = Entry(fenetre, width=30)
ZonedeT_keyword.grid(row=3,column=1)

ZonedeT_venue = Entry(fenetre, width=30)
ZonedeT_venue.grid(row=4,column=1)

ZonedeT_year = Entry(fenetre, width=30)
ZonedeT_year.grid(row=5,column=1)
```

**Figure 55.** Code programmant les zones de texte

Pour illustrer les différentes zones de textes, nous avons mis différents titres avec la commande « Label » qui permet de les distinguer les unes aux autres.

```
##### Titre des des zones de textes
label_publi = Label(fenetre, text="Titre publication ",fg = "DodgerBlue4",font = ("Arial", 10 , "bold"))
```

**Figure 56.** Exemple du label pour les titres de publications

Les paramètres de cette fonction pour l'exemple sont les suivants :

- Nom de l'interface : « fenetre »
- Texte à afficher : « Titre publication »
- Foreground(fg) : « DodgerBlue4 » = couleur de la police
- Font : (« Arial », 10, « bold ») = Police

Dans un second temps, nous avons créé une partie réservée à des cases à cocher qu'on appellera « Partie Filtrage ». Ces cases à cocher sont créées à l'aide de la commande « Checkbutton », ce qui va permettre à l'utilisateur de choisir ce qu'il souhaite afficher avec le champ rempli dans la zone de texte. Chaque case représente les différentes variables des tables et sont les suivants :

- Auteur : Nom de l'Auteur
- Keyword : Mots clés
- Year : Nom de l'année
- Venue : Nom du lieu et son type
- Publication : Nom publication, catégorie et sa date de publication



**Figure 57.** Partie filtrage avec cases à cocher

```
chk_btn_nomAuteur = Checkbutton(fenetre, text="Nom de l'Auteur", variable=Nom_Auteur_val,font = ("Arial", 9))
```

**Figure 58.** Exemple du checkbutton pour afficher le nom de l'auteur

Les paramètres de cette fonction pour l'exemple sont les suivants :

- Nom de l'interface : « fenetre »
- Texte à afficher : « Nom de l'auteur »
- Variable : **Nom\_Auteur\_val**
- Font : (« Arial », 9) = Police

Nous utilisons, pour le nom de la variable `Nom_Auteur_val = IntVar()` la commande « `IntVar()` » qui va permettre de mémoriser un entier. Lorsque la case est cochée la variable prends en valeur 1 et sera utilisée dans la fonction qui récupère les paramètres des cases à cocher sinon elle est égale à 0.

**5.4.2.2 Fenêtre utilisateur lors d'une recherche :** L'utilisateur souhaite connaître l'année de publication et son auteur quand il rentre le nom d'une publication :

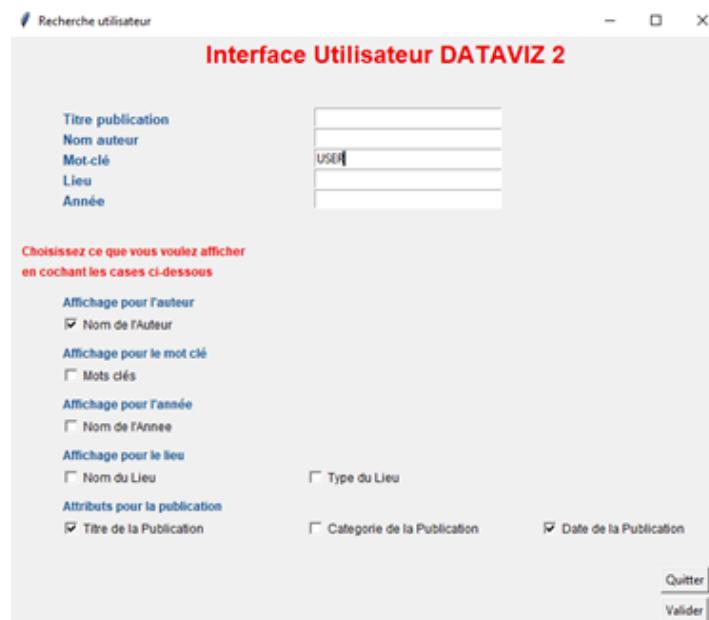


**Figure 59.** Exemple de recherche dans la "Fenêtre utilisateur"

La console python va réaliser la requête programmée en amont par une requête dynamique. Cette dernière va faire la jointure sur les tables que l'on utilise et va afficher le résultat dans la console :

```
1  [['Oshri Halimi' 2018]
2   ['Ron Kimmel' 2018]]
```

**5.4.2.3 Autre exemple :** l'utilisateur souhaite connaître les auteurs qui ont utilisé le mot « USER » avec le nom de la publication et sa date de publication :



**Figure 60**

Résultat de la requête :

```
1  [['Mario Alonso'
2   'Energy-Efficient Downlink Power Control in mmWave Cell-Free and
3   User-Centric Massive MIMO.'
3   '2019-09-16']]
```

Pour valider la demande de l'utilisateur, un bouton « valider » a été mis en place pour valider la demande.

```
##### Boutons d'entrees de valeurs
btn_valider=Button(fenetre, text="Valider", command=btn_entree_valider)
```

**Figure 61**

Les paramètres de cette fonction pour l'exemple sont les suivants :

- Nom de l'interface : « fenetre »
- Texte à afficher : « Valider »
- Command = « **btn\_entree\_valider** »

Ce bouton est piloté par une fonction nommée « btn\_entree\_valider » :

```
#Fonction déclenchée par le bouton valider. Va lancer la recherche sur le 1er champ rempli
def btn_entree_valider():
    name=[]
    typename=[]
    if Zonedet_publi.get():
        name.append(Zonedet_publi.get())
        typename.append("publication")
    if Zonedet_author.get():
        name.append(Zonedet_author.get())
        typename.append("author")
    if Zonedet_keyword.get():
        name.append(Zonedet_keyword.get())
        typename.append("keyword")
    if Zonedet_venue.get():
        name.append(Zonedet_venue.get())
        typename.append("venue")
    if Zonedet_year.get():
        name.append(Zonedet_year.get())
        typename.append("year")
    att=[]
    typeAtt = set()
    set_att_typeAtt_from_intf(att, typeAtt)
    print(request(name, typename, att, typeAtt))
```

**Figure 62.** Fonction "btn\_entree\_valider" qui permet de valider la recherche

**5.4.2.4 Interprétation de la fonction « **btn\_entree\_valider()** :** Avant de retourner le résultat, la fonction vérifie une par une si les différentes zones de textes sont remplies et si c'est le cas, alors le programme lance la fonction qui permet de récupérer les paramètres des différentes cases à cocher avec « `set_att_typeATT_from_intf()` » et donc d'effectuer la requête « `request` » principale du programme.

```
#fonction qui permet remplir les parametres att et typeAtt selon le choix de l'user dans l'interface
def set_att_typeAtt_from_intf(att,typeAtt):
    if Nom_Auteur_val.get() == 1 :
        att.append("name_author")
        typeAtt.add("author")
    if NamePubli_val.get() == 1 :
        att.append("article_title")
        typeAtt.add("publication")
    if date_publication.get() == 1:
        att.append("date_pub")
        typeAtt.add("publication")
    if ListeKeyword_val.get() == 1:
        att.append("keyword")
        typeAtt.add("keyword")
    if lieu_publi_val.get()== 1:
        att.append("name_venue")
        typeAtt.add("venue")
    if T_lieu_publi_val.get()==1:
        att.append("type_venue")
        typeAtt.add("venue")
    if categorie_val.get()== 1:
        att.append("categorie")
        typeAtt.add("publication")
    if choix_annee.get()== 1:
        att.append("year")
        typeAtt.add("year")
```

**Figure 63.** Stockage des paramètres pour les cases à cocher

Cette fonction permet de vérifier l'état de chaque case à cocher de l'interface et ainsi de récupérer les deux paramètres qui sont enclenchés si la case est cochée :

- `att`(élément d'une table)

- typeATT(table)

Pour quitter l'interface il faut simplement cliquer sur le bouton « QUITTER » ou cliquer sur la petite croix située en haut à droite de l'interface :



**Figure 64.** Stockage des paramètres pour les cases à cocher

```
##### Boutons QUITTER
bouton_quitter = Button(fenetre, text="Quitter", command=fenetre.destroy)
```

**Figure 65**

Les paramètres de cette fonction pour l'exemple sont les suivants :

- Nom de l'interface : « fenetre »
- Texte à afficher : « Quitter »
- Command = « fenetre.destroy »

Le bouton « Quitter » est piloté par la commande : « fenetre.destroy » qui va permettre de quitter l'interface lorsque le bouton est activé.

### 5.4.3 Crédit des requêtes

Afin de construire le graphe correspondant à la demande de l'utilisateur, il faut extraire parmi toutes les données celles spécifiques demandées. Cette étape d'extraction s'effectue grâce aux requêtes sur les tables de données. La fonction de requêtage prend en argument la liste des noms recherchés, la liste des types de ces noms, la liste des attributs à afficher et l'ensemble des types de ces attributs, et retourne le tableau des attributs demandés.

Pour cela, on peut tout d'abord remarquer que toutes les tables sont reliées à la table publication. Cela signifie qu'à partir de n'importe quelle publication, on peut connaître tous les attributs liés, comme les noms de ses auteurs, son année de publication, la conférence d'où il est tiré, etc... Cela signifie donc également que pour atteindre n'importe quelle table depuis n'importe où, il faut obligatoirement passer par les publications.

Donc, après avoir extrait la ligne correspondante au premier élément de recherche, on associe cette valeur à sa ligne dans sa table d'ID (par exemple year\_publication pour year) puis on recherche les publications ayant le même ID.

```
#Recherche de la ligne correspondante à la première valeur de recherche
if typename[0]=="author":
    table = author.loc[author.name_author==name[0],]
elif typename[0]=="publication":
    table = publication.loc[publication.article_title==name[0],]
elif typename[0] == "keyword":
    table = keyword.loc[keyword.keyword==name[0],]
elif typename[0]== "venue":
    table = venue.loc[venue.name_venue==name[0],]
elif typename[0] == "year":
    table = year.loc[year.year==int(name[0]),]
else:
    return print("Erreur : le type d'entrée n'existe pas")

#Association avec la table ID et la table publication
table = pd.merge(table, idTable[typename[0]], on=idName[typename[0]])
table = pd.merge(table, publication, on='id_publication')
```

**Figure 66.** Recherche de la ligne correspondante à la première valeur de recherche

Maintenant que l'on connaît les publications en lien avec la recherche, on peut associer à ces publications les autres attributs, en associant tout d'abord les ID des publications avec les ID des tables de relation author\_publication, year\_publication, venue\_publication ou keywords\_publication selon la demande, puis on récupère les lignes correspondantes des tables.

```
#Merge cette ligne avec les tables correspondantes à la recherche
for typ in typeAtt:
    if (typ in intoTable):
        del table[idName[typ]]
        del table[donneesName[typ]]
    table = pd.merge(table, idTable[typ], on='id_publication')
    table = pd.merge(table, donneesTable[typ], on=idName[typ])
```

**Figure 67**

Attention, si le type de la valeur de sortie est la même que celui de sortie (par exemple dans le cas où on souhaiterait les co-auteurs), il peut y avoir une erreur lors du merge (colonne qui change de nom). On supprime alors cette colonne avant d'associer les tables afin d'éviter cela.

Enfin, parmi toutes les lignes récupérées, on retourne seulement les colonnes des attributs demandés.

```
#Sélection plus précise des données avec les autres valeurs de recherche
for i in range(len(typename)):
    if typename[i]=="author":
        val = author.loc[author.name_author==name[i],]
    elif typename[i]=="publication":
        val = publication.loc[publication.article_title==name[i],]
    elif typename[i] == "keyword":
        val = keyword.loc[keyword.keyword==name[i],]
    elif typename[i]== "venue":
        val = venue.loc[venue.name_venue==name[i],]
    elif typename[i] == "year":
        val = year.loc[year.year==int(name[i]),]
    else:
        print("Erreur : Le type d'entrée n'existe pas")
    table = pd.merge(table, idTable[typename[i]], on='id_publication')
    table = pd.merge(table, val, on=idName[typename[i]])

#retour des attributs demandés
return table[attribute]
```

**Figure 68**

Selon ce qui ressortira de cette fonction, il est possible que certaines lignes soient similaires. Lorsque celle-ci est utilisée, il ne faut pas oublier d'utiliser la méthode drop\_duplicates() sur le résultat, ce qui permet de corriger cela.

## 5.5 Visualisation des données

Lors du déroulement du projet nous avons eu pour mission de réaliser des graphes correspondant à une visualisation des données des fichiers csv qui nous ont été transmis. Nous avons donc commencé par chercher les différentes librairies qui allaient nous permettre de construire des graphes. La première librairie que l'on a pu trouver est la suivante :



**Figure 69**

Networkx est une bibliothèque Python servant à étudier des graphes et également des réseaux. C'est donc un logiciel distribué sous la licence BSD. Elle comporte diverses fonctionnalités :

- Classes pour graphes simples et orientés.
- Conversion de graphes depuis et vers divers formats.
- Capacité à construire des aléatoires ou à les construire de manière progressive.
- Capacité à trouver des sous graphes et de réaliser des réseaux en 2D et 3D.

Toutes ces fonctionnalités en font donc un outil efficace pour réaliser les tâches de notre projet intégré.

### 5.5.1 Premières visualisations

Tout d'abord, nous avons utilisé la librairie pandas pour récupérer les données présentes dans les divers fichiers csv. Ce qui nous permet à l'exécution de charger et de lire tout au même moment.

```
author = pd.read_csv("D:/MasterInfo/Projet_Integre/GestionProjet/Gestion_de_projets/dataset/author.csv", engine='python', error_bad_lines=False)
publication_author = pd.read_csv("D:/MasterInfo/Projet_Integre/GestionProjet/Gestion_de_projets/dataset/publication_author.csv", engine='python', error_bad_lines=False)
```

Figure 70

Aussi nous avons utilisé le .head qui permet lorsqu'on en vient à utiliser pandas de récupérer une partition des données pour ne pas trop surcharger le temps de lancement du code pour les premiers tests. Dans le cas ci-dessous, nous chargeons uniquement les 300 premières lignes des différents fichiers.

```
authorhead = author.head(300)
keyword_head = keyword.head(300)
publication_head = publication.head(300)
```

Figure 71

Ensuite, il y a eu l'initialisation de certains graphes ; nous en avons deux exemples parmi d'autres ci-dessous. Ceux-ci renvoient des graphes de Dataframe avec pandas contenant une liste d'arêtes appelées "edgelist". Le dataframe doit contenir au moins deux colonnes des fichiers csv qui seront donc les noms de noeuds. Chaque ligne sera donc traitée comme une instance d'arête. La fonction from\_pandas\_edgelist prend en paramètre un dataframe et le nom des colonnes associées et effectue une itération sur les différentes valeurs. Le create\_using est utilisé pour le type de graphes que l'on souhaite avoir (orienté, non-orienté, multiple ...).

```
GraphAuthor = nx.from_pandas_edgelist(authorhead, 'name_author',
'nbr_publication',create_using=nx.DiGraph)

GraphPublication = nx.from_pandas_edgelist(publication_head, 'id_publication',
'date_pub','article_title',create_using=nx.DiGraph)
```

Figure 72

Enfin, nous avons réalisé dans la première phase de tests des fonctions permettant d'afficher les graphes. Dans le cas ci-dessous (Figure 1), c'est un graphe permettant d'afficher les valeurs des auteurs et leur nombre de publications. Toujours grâce à la librairie Networkx, la fonction permet de construire le graphe en y ajoutant plusieurs paramètres qui sont modifiables pour la visualisation.

La fonction automatisée de random\_layout() permet lors de la génération du graphe de donner ce caractère aléatoire de l'affichage des noeuds et arêtes. Elle prend en paramètre le graphe que l'on souhaite afficher. Ensuite, nous avons fait une boucle sur les noeuds pour leur spécifier des attributs afin de les différencier.

Ici, nous avons choisi dans un premier temps d'attribuer aux différentes valeurs de noeuds une couleur.

Pour la suite du code on a utilisé une autre librairie qui est la suivante :



Figure 73

Matplotlib est une bibliothèque complète pour créer des visualisations statiques, animées et interactives en python.

- Le plt.figure permet de générer la taille du graphe.
- Le nx.draw permet de dessiner le graphe avec le passage de plusieurs paramètres.
- Le plt.show() permet de retourner et d'afficher le graphe.

```
def afficher_graph_acteurs_publications():
    pos_author = nx.random_layout(GraphAuthor)
    colors = []
    for node in GraphAuthor:
        if node in authorhead["name_author"].values:
            colors.append("green")
        if node in authorhead["nbr_publication"].values:
            colors.append("blue")
    plt.figure(figsize=(20,20))
    nx.draw(GraphAuthor, pos_author, node_size=1000,
node_color=colors,linewidths=0.15,font_size=10, font_weight='bold', with_labels=True)
    plt.show()
|
afficher_graph_acteurs_publications()
```

Figure 74

Après cela on exécute la fonction ci-dessus pour afficher dans une fenêtre le graphe statique. Ainsi, nous avons notre premier graphe ci-dessous.

Nous avons donc la représentation du nombre de publication en jaune et le nom des auteurs en vert.

## 5.5.2 Visualisation en fonction de l'interface

**5.5.2.1 Une première représentation** Par la suite, nous avons décidé de créer des graphes en fonction des requêtes et de l'interface. Il y a donc deux librairies qui nous ont permis de réaliser ces nouveaux graphes.

Pour afficher une première représentation de graphes afin de mettre en exergue les valeurs de ceux-ci, nous avons utilisé une première librairie pour la création de graphe : Bokeh.



Figure 75

Sur le fond, la manière de créer le graphe est assez similaire à la deuxième représentation développée par la suite. En revanche, la représentation graphique est bien différente. Elle se compose de 4 étapes :

```
# Valeurs des noeuds secondaires
# Pour chaque list dans la list
for i in range (len(listRequest)):
    # Pour chaque élément dans la liste de la liste
    for j in range (len(listRequest[i])):
        if att[j] == "date_pub":
            listDatePub.append(listRequest[i][j])
        elif att[j] == "keyword":
            listNameKeyword.append(listRequest[i][j])
        elif att[j] == "year":
            listNameYear.append(listRequest[i][j])
        elif att[j] == "name_author":
            listNameAuthor.append(listRequest[i][j])
        elif att[j] == "article_title":
            listArticleTitle.append(listRequest[i][j])
        elif att[j] == "name_venue":
            listNameVenue.append(listRequest[i][j])
        elif att[j] == "type_venue":
            listTypeVenue.append(listRequest[i][j])
        elif att[j] == "categorie":
            listCategorie.append(listRequest[i][j])
```

**Figure 76**

```
def mainFunction(name, typename, att, typeAtt):
```

**Figure 77**

**1ère étape : Saisi des données** Une fois que l'utilisateur clique sur le bouton de validation, cela crée la requête et appelle la méthode principale qui est mainFunction. Celle-ci va s'occuper de la création du graphe, notamment en fonction du résultat de la requête et des différents éléments saisis et cochés.

Après avoir obtenu les différentes données via la requête liée à l'interface, il s'agit de séparer ces données de manières distinctes afin de pouvoir les afficher correctement. Ces données représentent donc celles qui apparaîtront sur les noeuds secondaires puisqu'ils sont les résultats de la requête.

**2ème étape : Création du graphe** On peut ensuite créer le graphe :

```
#### Crédit du graphe
G = nx.Graph()
```

**Figure 78**

```
creerNoeudsGraphe(G, name[0], listArticleTitle)
```

**Figure 79**

La fonction creerNoeudsGraphe prend les paramètres suivants :

- G représente le graphe vide
- name[0] est le noeud principal qui dépend du nom saisi dans les champs de texte de l'interface.
- listArticleTitle est la liste des noeuds secondaires qui dépendent des options cochées sur l'interface.

```
# Création du graphe en commençant par le noeud principale puis avec les noeuds secondaires
def creerNoeudsGraphe(G, noeudCentral, noeudsSecondaires):
    # Création du noeud principale
    G.add_node(noeudCentral)

    # Création des noeuds secondaires
    fin = len(noeudsSecondaires)-1
    for arg in noeudsSecondaires:
        G.add_node(arg)
        if arg != noeudsSecondaires[fin]: # Permet d'éviter d'avoir une arête sans noeud
            G.add_edge(noeudCentral,arg)

    nx.draw(G, with_labels=True)
    plt.show() # display
```

**Figure 80**

Ainsi, nous créons d'abord le noyau central du graphe puis nous y ajoutons les différents noeuds secondaires. Après l'appel de cette fonction, le graphe est donc créé de manière très primitive, c'est-à-dire qu'il ne contient pas encore toutes les données ni de représentation graphique dynamique.

**3ème étape : Mise en forme des données dans le graphe** Nous commençons par y ajouter les données :

```
#### Mise en place des données dans les différents noeuds
graph_renderer = from_networkx(G, nx.spring_layout, scale=1, center=(0, 0))
TOOLTIPS = []
# Si les listes sont remplies, ça veut dire qu'il y a des valeurs
# Dans ce cas, on donne les données au graphe pour pouvoir les afficher au survol du noeud
if len(listDatePub) > 0:
    TOOLTIPS.append(("Date Publication", "@DatePub"))
    graph_renderer.node_renderer.data_source.add(listDatePub, 'DatePub')
```

**Figure 81**

Pour chaque attribut, on cherche à savoir si la case est cochée, c'est-à-dire si des données sont à afficher dans le graphe. On crée une liste TOOLTIPS qui permettra d'afficher d'un point de vue visuel les données lors du survol du noeud avec le curseur. En effet, si des données venant de la requête sont trouvées (dans l'exemple ci-dessus si on a des dates de publications dans le résultat de la requête), on ajoute cet élément à la liste. Cet élément se compose de deux parties : le nom qui sera affiché lors du survol du noeud ("Date Publication") et sa valeur ("@DatePub").

La dernière ligne permet d'ajouter d'un point de vue technique les données. Le @ permet ainsi de faire le lien entre la donnée stockée et l'affichage de celle-ci.

Les données de chaque attribut sont mises dans des listes et ne distinguent pas les données du noeud principal de celles des noeuds secondaires sur le plan visuel. C'est pour cela qu'il faut insérer les valeurs du noeud central en début de liste, afin qu'elles créent d'abord le noeud central.

Par exemple, si on affiche un auteur et ses publications (le titre, la date et la catégorie), il faudrait pouvoir afficher les données de l'auteur (son nom et le nombre de publications) sur le noeud central.

```
listIdPublications.insert(0, "Nombre : " + str(get_nb_publication_author(name[0])))
listNamePublications.insert(0, '---')
listDatePublications.insert(0, '---')
listCategPublications.insert(0, '---')
```

**Figure 82**

**4ème étape : Affichage dynamique du graphe** Ensuite, il faut créer le visuel graphique, en lui donnant un titre, les données ci-dessus et les outils.

```
# Création graphique du graphe
plot = Plot(plot_width=400, plot_height=400, x_range=Range1d(-1.1, 1.1), y_range=Range1d(-1.1, 1.1))
plot.title.text = "Graphe interactif"

# Affichage des différents éléments au survol d'un noeud
node_hover_tool = HoverTool(tooltips=TOOLTIPS)

# Outils qui apparaissent à côté du graphe pour se déplacer dessus, le sauvegarder, etc.
plot.add_tools(node_hover_tool, PanTool(), WheelZoomTool(), BoxZoomTool(), ResetTool(), SaveTool())
```

**Figure 83**

Les outils sont affichés à droite du graphe et permettent de naviguer dedans (en se déplaçant, en zoomant) ou de le sauvegarder en format image. C'est notamment via l'outil "HoverTool" que les données de la requête apparaissent lors du survol du noeud. Ces outils permettent ainsi d'obtenir du dynamisme et de l'interactivité.

Une fois la mise en place des outils terminée, il faut créer visuellement les noeuds et les arêtes.

```
# Affichage graphique des noeuds et des arêtes
graph_renderer.node_renderer.glyph = Circle(size=15, fill_color=Spectral4[0])
graph_renderer.node_renderer.hover_glyph = Circle(size=21, fill_color='red')
graph_renderer.edge_renderer.glyph = MultiLine(line_color='black', line_width=1)
graph_renderer.edge_renderer.hover_glyph = MultiLine(line_color='green', line_width=5)
graph_renderer.selection_policy = EdgesAndLinkedNodes()
graph_renderer.inspection_policy = NodesAndLinkedEdges()
plot.renderers.append(graph_renderer)
```

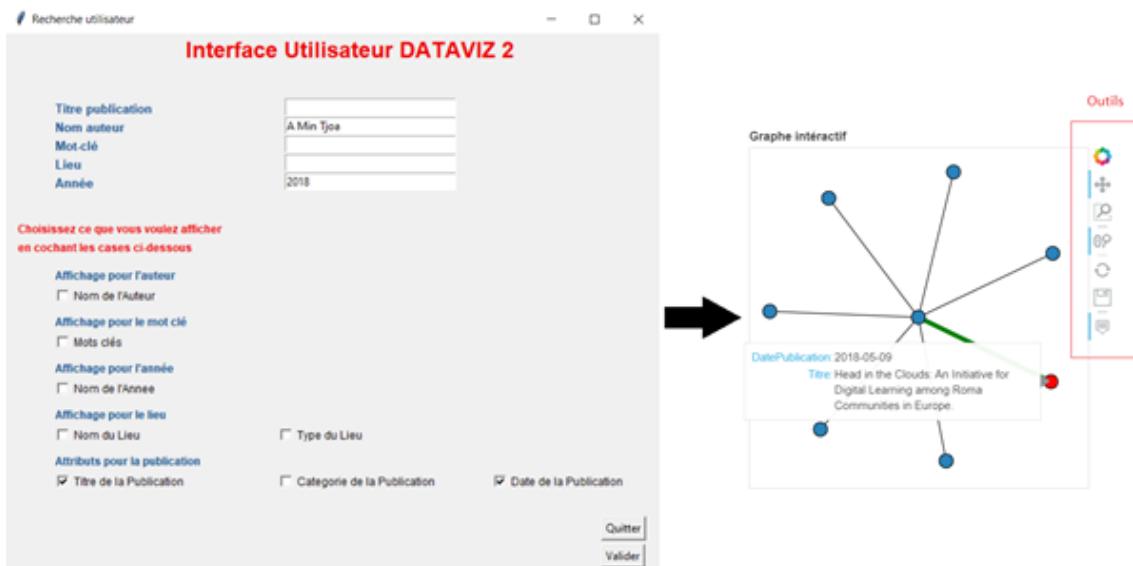
**Figure 84**

Avec cela, il est possible de mettre en valeur le noeud et les arêtes qui lui sont associées lors du survol du noeud avec le curseur. Ceci ajoute alors davantage de lisibilité et d'interactivité.

Enfin, on affiche le graphe dans une fenêtre de navigateur et l'utilisateur peut enfin voir et interagir avec le graphe issu de sa requête.

```
# On affiche le graphe dans une page HTML
output_file("interactive_graphs.html")
show(plot)
```

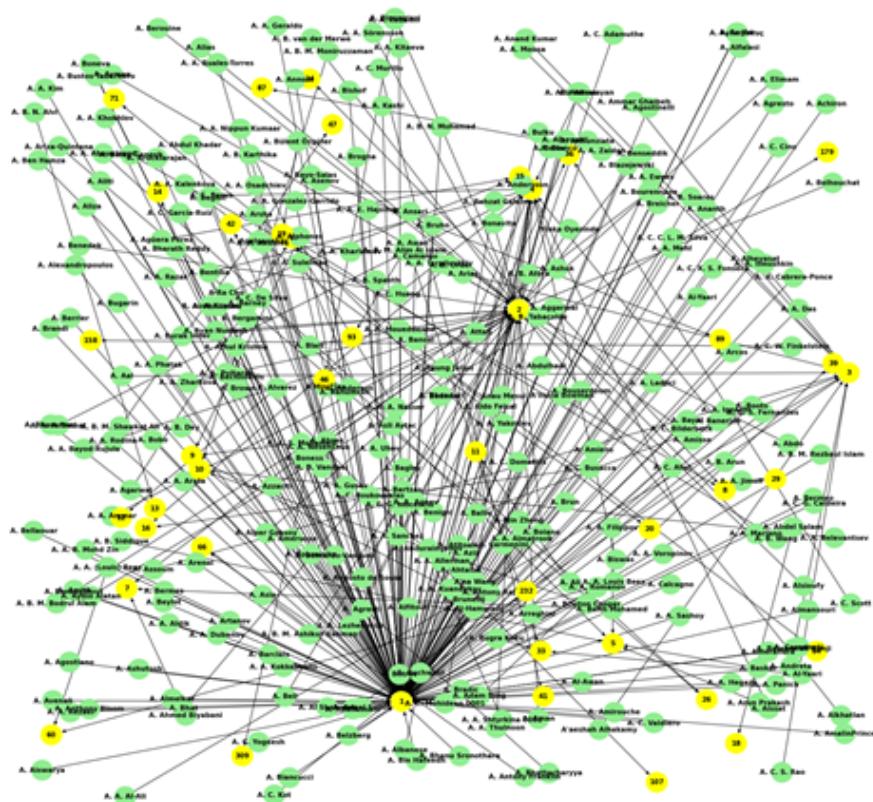
**Figure 85**



**Figure 86.** De l'interface utilisateur au graphe final

- > Lien vers le site de la librairie Bokeh
- > Se déplacer dans le graphe
- > Zoomer sur un endroit sélectionné
- > Zoomer avec la molette de la souris
- > Remettre le graphe dans sa position initiale
- > Sauvegarder et télécharger le graphe en format .png
- > Afficher les informations du noeud au survol avec le curseur

**Figure 87.** Utilisation des outils du graphe



**Figure 88.** Nombre de publication en fonction de l'auteur

**5.5.2.2 Une deuxième représentation possible** La deuxième librairie est la suivante :



**Figure 89**

Plotly est une librairie qui permet de développer grâce à ses outils des analyses et des visualisations de données en ligne avec Python. Grâce à cette librairie nous avons pu mettre en évidence un programme composé de deux fonctions.

Ainsi, la fonction `add_edge` permettant d'ajouter une arête dynamiquement en fonction du graphe construit, prend en paramètre les coordonnées `x` et `y` ainsi que l'utilisation de la spécificité de `plotly` avec `graph_objects`.

Le type de tracé permet avec `scatter` d'englober les graphes avec des données visuelles sous forme de dispersion de points qui sont définis en fonction de `x` et `y`. On peut donc y ajouter du texte et des paramètres pour dessiner des bulles d'infos sur chaque noeud qui va être lié par une arête spécifique.

```
1 import plotly.graph_objects as go
```

```
def add_edge(x, y, width):
    edge = go.Scatter(x=x,
                       y=y,
                       line=dict(width=width,
                                 color='darkgrey'),
                       hoverinfo='text',
                       mode='lines')
    return edge
```

Figure 90

Cette fonction sera importante pour la suite de l'affichage du graphe avec la fonction de requête.

Ensuite, nous avons donc une fonction principale pour la construction du graphe.

```
def afficher_Graph(recherche,dataset):
    #création du graphe
    Graph = nx.Graph()

    #ajout du noeud central
    center_node = str(recherche)
    Graph.add_node(center_node, size = 15)

    #ajout des noeuds du niveau 1
    nbr_nodes = len(dataset)-1      #nombre de noeuds
    nodes_title = [nbr for nbr in range(nbr_nodes)]
    node_w = 6                      #taille des noeuds
    for i in range(len(nodes_title)):
        Graph.add_node(nodes_title[i], size = node_w)
        Graph.add_edge(center_node, nodes_title[i], width = node_w)
```

Figure 91

Elle s'appelle afficher\_Graph et prend en paramètre une recherche qui est une valeur unique pouvant être un auteur, une publication, une venue et également un dataset ou dataframe qui est en fait le retour de la fonction de la requête que l'on veut afficher (voir la partie Création des requêtes).

La création du graphe se fait avec la fonction nx.Graph() de librairie networkx pour initialiser et créer le graphe pour le moment vide.

Après avoir parlé lors de multiples réunions, nous avons décidé que le graphe ait un noeud central qui est la valeur unique.

On utilise la fonction add\_node qui permet d'ajouter des noeuds au graphe. On a donc ajouté le noeud principal. Ensuite, nous ajoutons par une liste de longueur notre dataset qui va définir le nombre de noeuds liés au noeud central. On boucle sur la liste des noeuds et on les ajoute un par un jusqu'au dernier avec une taille particulière.

```

pos = nx.spring_layout(Graph)

edge_trace = []

#tracer les éléments du graphe
for edge in Graph.edges():
    x0, y0 = pos[edge[0]]
    x1, y1 = pos[edge[1]]
    trace = add_edge(
        [x0, x1, None],
        [y0, y1, None],
        width = 2
    )

    edge_trace.append(trace)

# ajouter les caractéristiques des éléments du graphe
node_trace = go.Scatter(x      = [],
                        y      = [],
                        text   = [],
                        textposition = "top center",
                        textfont_size = 15,
                        hoverinfo = 'text',
                        mode    = 'markers',
                        marker  = dict(color = [],
                                      size= [],
                                      line=dict(color='black', width=1
                                              )
                                         )
                        )

```

**Figure 92**

On ajoute et on trace les arêtes par la suite en définissant x et y par la position dans le graphe des arêtes.

A la suite de cela, on ajoute les informations et les caractéristiques des noeuds au moyen du scatter qui va renvoyer une liste avec des variables de la librairie de plotly.

```

# Boucle sur les noeuds pour jouer sur leurs caractéristiques.
for node in Graph.nodes():
    i=i+1
    try:
        text = str(dataset.iloc[i,:].values)
    except:
        text="txt"
    hover_text = ""
    # Condition pour ajouter des paramètres en fonction du type de noeud: central, autres
    if(node == center_node):
        hover_text = tuple(["{}".format(center_node)])
        color = tuple(['blue'])
        size = tuple([2 * Graph.nodes[node]['size']])
    else:
        if(not(dataset.iloc[i,0]==firstElem)):
            color = tuple([random.choice(colors)])
            firstElem = dataset.iloc[i,0]
        hover_text = tuple([text])
        size = tuple([2 * Graph.nodes[node]['size']])

```

**Figure 93**

La boucle suivante permet de gérer les noeuds du graphe grâce à l'indexation des valeurs du dataframe. On renvoie donc un tableau de valeurs qui seront les différents textes composant les valeurs des noeuds.

Ensuite, on ajoute grâce à la condition suivante des paramètres pour différencier nos types de noeuds à savoir le noeud central et les noeuds annexes au moyen de tuple pour effectuer une itération sur chaque élément de la liste (couleurs, taille, texte en hover).

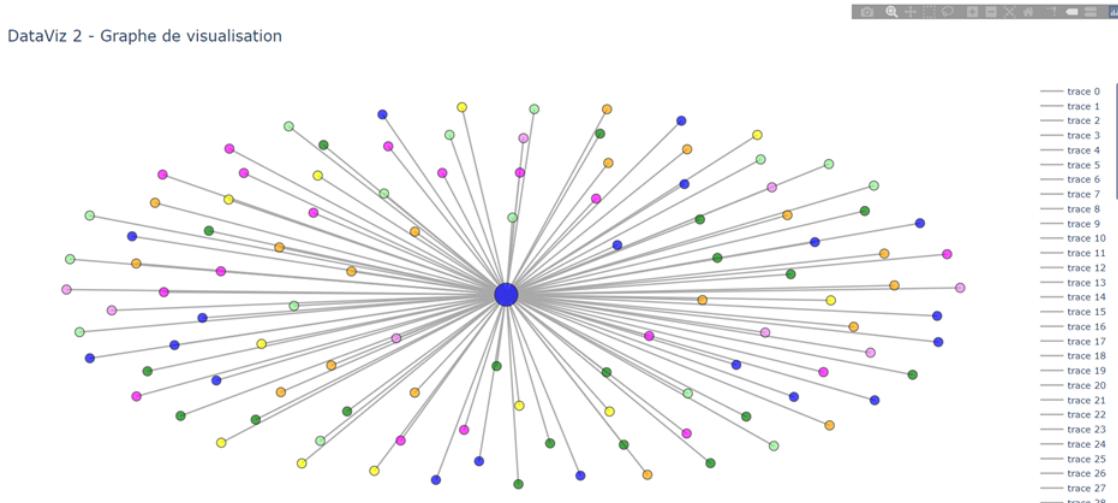
```
# création de la figure
fig = go.Figure(layout = layout)
fig.update_layout(
    title_text="DataViz 2 - Graphe de visualisation",
    titlefont_size=20,
    hoverlabel=dict(
        font_size=13,
        font_family="Calibri"
    )
)
#ajout des éléments (noeuds et arêtes) dans la figure et affichage du graphe
for trace in edge_trace:
    fig.add_trace(trace)
fig.add_trace(node_trace)
plot(fig)
fig.show()
```

**Figure 94**

A la fin de la fonction, nous avons une partie du code qui permet de créer la figure grâce à la librairie plotly pour pouvoir afficher le graphe dans une fenêtre lors du lancement de la fonction.

On donne plusieurs attributs dans la figure comme le titre, la forme du texte dans les différentes bulles.

A la fin, on boucle sur les noeuds et les arêtes et on ajoute graphiquement grâce à Matplotlib l'ensemble des éléments. Enfin grâce au fig.show(), on affiche et lance la fonction de graphe pour l'afficher dans une autre fenêtre.



**Figure 95.** Résultat du graphe en visualisation

## 5.6 Utilisation de l'application

Pour lancer le programme permettant de créer des graphes, il est possible de le faire de deux façons, dont la première qui est plus simple que la seconde.

### 5.6.1 Exécutable

Un exécutable est un fichier contenant un programme exécutable directement par le processeur. Nous souhaitions créer un exécutable de notre programme afin de pouvoir le lancer directement sans passer par un environnement Python.

Pour cela, nous avons utilisé la bibliothèque Auto Py To Exe[1] qui permet de réaliser facilement cette opération. Pour cela, il faut d'abord installer AutoPyToExe dans Anaconda, puis le lancer.

Code pour installer auto-py-to-exe, dans le terminal conda :

```
pip install auto-py-to-exe
auto-py-to-exe
```

La création de l'exécutable est alors réalisée, en prenant bien toutes les bibliothèques nécessaires à son fonctionnement.

Malheureusement, même si une fenêtre s'ouvre avec notre interface, le programme .exe généré n'arrive pas à modifier la page web pour afficher le graphe, rendant cet exécutable inutile.

### 5.6.2 Distribution Python

Cette méthode est une alternative plus longue et complexe que l'exécutable. Elle peut être utile pour les personnes voulant regarder le code ou qui savent utiliser des éditeurs de code. Cela consiste à installer une distribution Python puis à exécuter le code du programme directement depuis un logiciel. Pour cela, il existe une façon parmi d'autres en suivant les étapes suivantes :

- Installer une distribution Python tel qu'Anaconda[8].
- Installer le logiciel Spyder de la distribution Anaconda (sur Anaconda Navigator, il est possible de télécharger Spyder en cliquant sur le bouton "install").
- Dans l'onglet Environnements, installer si ce n'est pas déjà fait, les librairies *pandas*, *tkinter*, *networkx*, *matplotlib* et *bokeh*. Pour cela, il faut sélectionner "all" dans la liste déroulante puis écrire le nom des librairies dans la barre de recherche. Il faudra alors cliquer sur les librairies puis sur le bouton "apply" si la checkbox n'est pas remplie.
- Télécharger le dossier du projet. Dans celui-ci, il y a un fichier python (.py) contenant le code du projet et 9 fichiers CSV qui contiennent les données.
- Lancer Spyder puis ouvrir le fichier ayant pour extension .py.
- Changer le chemin d'accès de chaque ligne en fonction de l'emplacement du dossier du projet téléchargé.

```
#importation fichiers CSV
author = pd.read_csv("D:/Travail/Master 1/Projet intégré/DATAVIZ_2/author.csv", engine='python', error_bad_lines=False)
publication = pd.read_csv("D:/Travail/Master 1/Projet intégré/DATAVIZ_2/publication.csv", engine='python', error_bad_lines=False)
publication_author = pd.read_csv("D:/Travail/Master 1/Projet intégré/DATAVIZ_2/publication_author.csv", engine='python', error_bad_lines=False)
keyword = pd.read_csv("D:/Travail/Master 1/Projet intégré/DATAVIZ_2/keyword.csv", engine='python', error_bad_lines=False)
publication_keywords = pd.read_csv("D:/Travail/Master 1/Projet intégré/DATAVIZ_2/publication_keywords.csv", engine='python', error_bad_lines=False)
publication_venue = pd.read_csv("D:/Travail/Master 1/Projet intégré/DATAVIZ_2/publication_venue.csv", engine='python', error_bad_lines=False)
publication_year = pd.read_csv("D:/Travail/Master 1/Projet intégré/DATAVIZ_2/publication_year.csv", engine='python', error_bad_lines=False)
venue = pd.read_csv("D:/Travail/Master 1/Projet intégré/DATAVIZ_2/venue.csv", engine='python', error_bad_lines=False)
year = pd.read_csv("D:/Travail/Master 1/Projet intégré/DATAVIZ_2/year.csv", engine='python', error_bad_lines=False)
```

Figure 96

- Exécuter tout le code via le bouton vert dans la barre d'outils en haut.



Figure 97

- Une interface va apparaître et l'utilisateur pourra alors saisir un champ de texte et cocher les checkbox qu'il souhaite afin de constituer une requête, ce qui aura pour finalité la construction puis l'affichage du graphe dans une fenêtre de navigateur web.

## 5.7 Conclusion

Notre sujet portait sur la visualisation des données DBLP par le biais de graphes illustrant les liens entre les différentes entités. Ainsi, l'objectif de ce projet était de proposer une visualisation pertinente des données en utilisant une interface de requêtage adaptée pour l'utilisation du client.

Afin de bien gérer notre projet, nous avons appliqué la méthode Scrum, nous permettant ainsi d'améliorer la productivité de notre équipe. En effet, la méthode d'itération dans nos sprints a rendu la création de l'outil plus structurée. L'idée était de pouvoir proposer une amélioration du produit au client à chaque fin de sprint en respectant un objectif à atteindre.

La méthode SCRUM nous pousse à nous mettre dans la peau du client et de ce fait, à nous imposer certaines contraintes. En effet, le but était de concevoir un produit efficace mais avec une prise en main facile et une compréhension rapide des informations affichées. De ce qui est de la prise en main de l'interface, les différentes barres de recherche, justifiées dans ce rapport, illustrent notre volonté de rentrer des informations précises quant aux attentes du client. D'autre part, la compréhension des graphes est facile grâce à un affichage sobre des informations.

Cependant, nous pouvons relever quelques perspectives d'amélioration de notre application. En effet, nous aurions voulu intégrer l'interface et l'affichage du graphe sur une fenêtre unique plutôt que sur deux fenêtres distinctes. Nous aurions également aimé améliorer le visuel de l'interface en ne proposant plus qu'une seule barre de recherche unique pour n'importe quel type d'attribut demandé, ou en proposant le choix du graphe à afficher. De plus, si la somme d'informations demandée est trop importante la compréhension des graphes peut s'avérer compliquée.

## 6 GROUPE ANALYSE

### 6.1 Noms et prénoms des participants

- **Florian Gomez**
- **Pierre Le Galèze**
- **Noé Lebreton**
- **Inès Kara**
- **Livia Giusti** (Scrum Master)
- **Clément Malvy** (Product Owner)

### 6.2 Introduction

Dans le cadre des cours de Gestion de projet et de Projet intégré, nous avons décidé de nous occuper de l'analyse des données DBLP. Après avoir longuement regardé la structure de ces données, nous avons défini notre objectif de recherche : mettre en évidence les communautés d'auteurs qui publient ensemble et trouver leur thème de recherche à partir des titres de leurs publications.

Dans un premier temps, nous allons vous expliquer comment nous avons mis en place la méthode SCRUM puis dans un second temps, nous allons vous détailler les méthodes que nous avons utilisées pour atteindre notre objectif.

### 6.3 Gestion de Projet

#### 6.3.1 Introduction

Dans le cadre de ce projet, nous avons mis en pratique une approche Agile et plus particulièrement l'approche SCRUM pour nous donner un cadre de travail lors de la conception notre application.

La méthode SCRUM met en relation trois éléments fondateurs : les besoins du client, le fonctionnement de l'équipe de développement et la réalisation du projet.

#### 6.3.2 Besoin du client

Durant le sprint 0, nous avons recueilli les besoins généraux du client qui étaient de faire une analyse des données DBLP. Ce besoin étant au départ très vague, nous l'avons affiné après avoir étudié les données. Nous avons convenu avec nos clients que nous nous focaliserions sur l'analyse des communautés d'auteurs-co-auteurs, et que nous chercherions à trouver à partir de leurs publications leurs domaines de recherches.

Nous avons ensuite listé plus finement les besoins de nos utilisateurs en écrivant nos User Story. Ces User Story décrivent les fonctionnalités à implémenter dans notre application et forment le Product Backlog. Nous les avons par la suite ordonnés selon leur priorité d'implémentation. Chaque item ou regroupement d'items ont ensuite fait l'objet d'un sprint. Par exemple, nous avons défini que l'utilisateur devait pouvoir visualiser clairement les données sous forme de graphe. Cette fonctionnalité a été l'objectif de notre premier sprint.

De plus, tout au long du projet, au cours de nos réunions hebdomadaires, nous avons affiné les besoins de nos clients et intégré les différents ajustements dans les étapes de réalisation du projet afin que celui-ci soit au plus proche de leur attente. Nous avons donc ajouter au fur et à mesure de nouveaux User Story comme donner la possibilité à notre client de connaître la liste des publications d'un auteur lorsque celui-ci est sélectionné.

L'une des valeurs de la méthode SCRUM est la collaboration avec le client et c'est pour cela que nous avons fait très attention de prendre en considération toutes les remarques de nos deux clients. Une autre valeur importante est de prendre en compte les individus et leur interaction, ce qui nous amène au point suivant : le fonctionnement de l'équipe.

### 6.3.3 Fonctionnement de l'équipe

L'approche SCRUM définit trois rôles : un Product Owner, un Scrum Master et une équipe de développement.

L'équipe de développement comprend tous les membres du groupe, c'est-à-dire Livia Giusti, Florian Gomez, Ines Kara, Pierre Le Galèze, Noé Lebreton, et Clément Malvy. Ayant des parcours scolaires très différents et donc des connaissances très variées, la diversité de notre équipe nous a permis de tous trouver rapidement notre place au sein du groupe.

En plus de faire partie de l'équipe de développement, deux membres avaient des rôles spécifiques.

Le rôle du Product Owner a été attribué à Clément Malvy. Il a donc été le représentant du client. Afin d'avoir une vision globale du projet, il a fait le lien entre notre groupe analyse et les autres groupes du TD.

Avec l'aide du Scrum Master, Livia Giusti, il a créé un ensemble d'outils permettant à l'équipe de communiquer et d'avoir un accès au Product Backlog. Pour faire cela, il a été décidé à l'unanimité d'utiliser la plateforme Trello pour créer, maintenir le Product Backlog et donner la possibilité à tous les membres de l'équipe de le voir et le modifier. Cette plateforme nous a également permis de partager nos fichiers, nos liens de documentations, ... et lorsque ceux-ci étaient trop volumineux nous avons complété le Trello avec un Google Drive. En ce qui concerne la communication globale du TD nous avons créé un salon Discord commun pour faciliter les échanges entre les différents sous-groupes du TD, dans lequel chaque groupe avait également un espace qui lui était réservé. Notre communication interne s'est faite, quant à elle, principalement à l'écrit sur Messenger. Nous avons volontairement multiplié les plateformes de communication pour pallier aux problèmes de connexion internet de certains membres de notre groupe. La communication avec nos deux clients s'est déroulée sur leurs plateformes personnelles Zoom et Webex.

En plus d'aider le Product Owner dans la mise en place des outils de communication, le Scrum Master a veillé à ce que toutes les réunions de la méthode SCRUM aient bien lieu.

Les mélées se déroulaient durant les dix minutes précédant les cours de Gestion de projet et de Projet intégré. Ces réunions bi-hebdomadaires servaient à faire le point sur l'avancement du travail de chacun, lister les problèmes rencontrés pour chercher ensemble comment les résoudre et décider des tâches à faire. C'est également un moment où le Scrum Master reporte le temps de travail de chaque membre du groupe dans le Burndown Chart pour visualiser la progression générale du projet.

Lors de la réunion de revue du sprint, le travail étant jugé par tous comme terminé, la production faite au sein d'un sprint devenait un incrément. Cet incrément s'ajoutait aux productions précédentes. L'application finale est la résultante de toutes ces incrémentations.

A la suite de la première réunion, nous avons enchaîné sur la rétrospective du sprint afin d'optimiser l'organisation du groupe. Par exemple, c'est grâce à cette réunion qu'après le sprint 1, nous avons décidé de scinder notre groupe en deux sous-groupes pour faciliter la communication et le travail en équipe.

Directement après ces deux réunions, nous avons fait la réunion de planification du sprint suivant. C'est-à-dire que nous définissions le travail à réaliser durant le sprint. Nous écrivions le Sprint Backlog.

Cet ensemble de trois réunions (la revue du sprint, la rétrospective et la planification du nouveau sprint) s'est effectué certains mardis en début d'après-midi, afin de nous rencontrer sans être pressés par le temps.

C'est avec cette organisation que nous avons conçu notre projet. Dans la partie suivante, les artefacts produits tout au long de notre travail vont vous être présentés.

### 6.3.4 Réalisation du projet

**6.3.4.1 Product Backlog** Au début du projet le Product Backlog était une liste d'User Story ordonnée par ordre de priorité d'implémentation. Au fur et à mesure du projet, ces User Story ont été transformées en sprint. Voici l'état du Product Backlog à l'heure où j'écris ces lignes.

Sprint	User Story	Status	Progress	Validation
Sprint 4 : Préparation des livrables - 2/2 séances (du 30 novembre au 6 décembre)	Intégration des quatre projets en un logiciel	En cours	1/2	0/2
	Rédaction du compte-rendu	En cours	2/12	0/2
	Préparation soutenance	A faire	0/2	0/0
Sprint 3 : Traitement des vraies données et interface - 2/2 séances (23 novembre au 29 novembre)	Modifier la data donnée par le groupe intégration	Validé	5/5	2/2
	Créer l'interface	Validé	4/4	4/4
	Faire tourner les méthodes sur les données finales	Validé	5/5	5/5
Sprint 2 : Implémentation des méthodes - 5/5 séances (du 2 novembre au 22 novembre)	Classification des mots clés à l'aide de graphes	Validé	4/4	1/1
	LDA	Validé	6/6	11/11
	Analyse des communautés d'auteurs - co-auteurs	Validé	6/6	6/6
Sprint 1 : Préparation des données et choix des méthodes - 3/3 séances (du 15 octobre au 1 novembre)	Préparation des données	Validé	5/5	3/3
	Choix des méthodologies et documentations	Validé	3/3	0/0
	+ Ajouter une autre carte			
Sprint 0 : 1/1 séance (14 octobre)	Intégration des données dans Python	Validé	10/10	1/1
	Question de recherche : Trouver les communautés d'auteurs qui publient ensemble sur les mêmes thèmes	Validé	0/0	0/0
	+ Ajouter une autre carte			

Figure 98. Product Backlog

Le Product Backlog est maintenant constitué par un ensemble de Sprint Backlog. Il y a en tout cinq Sprint Backlog. Le premier, le sprint 0, était un sprint de démarrage où nous

avons défini l'objectif de notre projet. Le dernier, le sprint 4, a pour objectif la préparation de tous les livrables.

Les sprints terminés sont étiquetés en vert par l'intitulé « Validé ». Dans le sprint en cours, ici le sprint 4, chaque sous-partie du sprint est étiquetée selon sa progression : en rouge si la sous-partie n'a pas encore été commencée ( A faire ), en orange si cette étape a été commencée mais qu'elle n'est pas encore terminée ( En cours ) et enfin en vert lorsque le travail est terminé. L'utilisation de ces étiquettes colorées permet de visualiser instantanément la progression du projet.

Par exemple, nous sommes actuellement dans le sprint 4, l'intégration des quatre projets pour ne former qu'une seule application a commencé. La moitié du travail a déjà été effectuée. En ce qui concerne la rédaction du rapport huit étapes ont été terminé, quatre sont encore en cours. Et la préparation de la soutenance n'a pas encore été entamée.

**6.3.4.2 Sprint Backlog** A chaque nouveau sprint, lors de sa réunion de planification, un ensemble d'User Story était choisi pour créer l'objectif du sprint. C'était également un moment où nous prenions le temps de les affiner, de les détailler pour prendre en compte un maximum d'aspects à implémenter. Les User Story sélectionnés étaient ensuite regroupés en plusieurs sous-groupes afin de former les différents axes de travail. Les axes de travail sont ensuite répartis entre les différents membres du groupe ou dans le temps en fonction des sprints.

De plus chaque Sprint Backlog est également défini par le temps nécessaire à sa réalisation. Ce temps est calculé de deux manières. La première compte le nombre de séances de cours que nous voulons y consacrer. La deuxième est la période de l'année durant laquelle le sprint va se dérouler. Nous n'avons volontairement pas fixé un nombre d'heures de travail que ce soit pour les sprints ou les étapes elles-mêmes. Ainsi chaque membre de l'équipe pouvait y consacrer le temps dont il avait besoin pour se former, comprendre, et appliquer les nouvelles notions afin de faire sa partie. Cette façon de procéder nous paraissait plus juste. Nous n'apprenons pas tous au même rythme et nous ne voulions pas pénaliser certains éléments de l'équipe et les mettre inutilement en difficulté. Etant très motivés, nous avions également envie de découvrir différentes méthodes d'analyses et nous nous sommes donc laissés la possibilité de le faire. Avec une restriction du temps de travail, nous n'aurions pas pu le faire. Dans un cadre professionnel nous nous serions fixé des temps de travail mais étant encore en formation nous avons pris la liberté d'adapter la méthode SCRUM à notre cadre particulier. Notre objectif était donc de respecter les délais décidés en équipe pour chacun de nos sprints. Mais nous avons quand même pris soin de calculer les temps de travail de chacun pour compléter le Burndown Chart.



**Figure 99.** Sprint Backlog du sprint 2 et du sprint 4

Par exemple, nous avons consacré cinq séances de cours au sprint 2, deux séances de Projet Intégré et trois de Gestion de projet. Et nous avons fixé une période de trois semaines pour effectuer le travail prévu. Les étapes du sprint 2 (la classification des mots clés à l'aide de graphe, la LDA et l'analyse des communautés d'auteurs – co-auteurs) ont été réalisés en parallèle. En revanche, nous avons décidé de nous occuper de la partie traitant de la préparation de la soutenance dans le sprint 4 après avoir terminé les deux autres étapes de ce sprint, c'est-à-dire l'intégration des quatre projets et la rédaction du rapport. Sprint pour lequel nous avons dédié deux séances de cours et une semaine.

Chaque partie du Sprint Backlog est composée d'un titre exprimant l'objectif de l'étape, d'une étiquette indiquant l'état d'avancement de la partie, d'une description dans laquelle les User Story sont indiqués, de pièces jointes contenant les fichiers exécutables et les fichiers créés et d'une Checklist énumérant toutes les étapes d'implémentation. L'utilisation de ces Checklist nous a permis de connaître l'évolution du travail de chacun entre deux Mélées bi-hebdomadaires, sans avoir à faire de nouvelles réunions.

The screenshot shows a detailed view of a project item titled "Classification des mots clés à l'aide de graphes".

- ÉTIQUETTES:** Validé
- AJOUTER À LA CARTE:** Membres, Étiquettes, Checklist, Date limite, Pièce jointe, Image de couvert...
- Description:** User story:
  - L'utilisateur doit pouvoir voir le graphe des mots clefs (uniquement pour un échantillon)
  - L'utilisateur doit pouvoir voir le graphe des mots clefs avec différents filtres (toujours sur le même échantillon)
  - Sur un graphe, l'utilisateur peut observer les voisins du mots clefs en question, c'est à dire les différentes connexions.
- Pièces jointes:** GRAPHE\_MOTS\_DYNAMIQUE.py (Ajouté mercredi dernier à 13:21)
- ACTIONS:** Déplacer, Copier, Créer un modèle, Suivre, Archiver, Partager
- Checklist:**
  - Classification des mots clés Cacher les éléments complétés Supprimer
  - Crédit à Business Class pour obtenir un nombre illimité de Power-ups par tableau.
  - Mettre à niveau l'équipe
- Checklist détaillée:**
  - Crédit à Business Class pour obtenir un nombre illimité de Power-ups par tableau.
  - Mettre à niveau l'équipe

**Figure 100.** Détail de la partie « Classification des mots clés à l'aide de graphes » du sprint 2

Pour cette partie du sprint 2, portant sur la classification des mots clés à l'aide de graphes, nous avons finement détaillé notre User Story de départ en trois User Story beaucoup plus précis. Puis nous avons listé les quatre étapes de réflexion et d'implémentation nécessaires à la réalisation de l'objectif. A chaque fois qu'une étape était terminée, nous la cochions. Lorsque toutes les étapes étaient effectuées la Checklist devenait verte, cela signifiait que cette partie était prête à devenir un incrément.

Numéro de semaine	0	S1	S2	S3	S4	S5	S6	S7	S8	Total
Date		14/10 - 18/10	19/10 - 25/10	26/10 - 1/11	2/11 - 8/11	9/11 - 15/11	16/11 - 22/11	23/11 - 29/11	30/11 - 6/12	
Sprint 0	Temps prévu	105								105
Sprint 0	Temps effectué	105								105
Sprint 1	Temps prévu	105	210	0						315
Sprint 1	Temps effectué	105	210	100						415
Sprint 2	Temps prévu				210	105	210			525
Sprint 2	Temps effectué				305	205	300			810
Sprint 3	Temps prévu							210		210
Sprint 3	Temps effectué							330	125	455
Sprint 4	Temps prévu								210	210
Sprint 4	Temps effectué								485	485
Par semaine	Temps prévu	210	210	0	210	105	210	210	210	
Par semaine	Temps effectué	210	210	100	305	205	300	330	610	
Total	Temps prévu	1365	1155	945	945	735	630	420	210	0
Total	Temps effectué	1365	1155	945	845	540	335	35	-295	-905

**Figure 101.** Tableau récapitulant le temps de travail de chaque sprint : le temps prévu est le temps passé en classe et le temps effectué est une moyenne en minutes du temps réellement passé par le groupe sur le projet

**6.3.4.3 Burndown chart** Le tableau ci-dessus représente le temps en minutes consacré à chacun des sprints. Comme expliqué précédemment, chaque sprint est également défini par une période de temps calculée en nombre de semaines de travail.

Nous avons différencié deux temps. Le temps prévu correspond au nombre de minutes passées en cours. Un cours dure 1h45 donc converti en minutes cela fait 105 minutes. Si la semaine comprend deux cours, soit deux fois 105 minutes, le temps prévu est égal à 210 minutes. Le temps effectué est quant à lui une moyenne du temps que chaque membre du groupe a passé sur le projet. Ce temps moyen est calculé de la manière suivante : le temps prévu auquel on ajoute la somme du temps de travail de chaque membre divisé par six pour avoir le temps moyen et multiplié par soixante pour convertir ce temps en minutes.

Pour que ce calcul soit plus clair, voici le calcul du temps effectué lors de la quatrième semaine du projet :  $= 2 * 105 + (1 + 2 + 1 + 2 + 1 + 2,5) / 6 * 60$ . Nous avons eu cette semaine-là deux cours de 105 minutes (l'un en Projet Intégré, l'autre en Gestion de projet). Les membres de l'équipe ont travaillé individuellement 1h, 2h, 1h, 2h, 1h et 2h30. Nous sommes six personnes dans le groupe, donc pour avoir une moyenne du temps travaillé en dehors des cours, cette somme a été divisé par six puis multiplié par soixante pour être convertie en minutes. Ce qui donne finalement une moyenne de travail pour la semaine 4 de 305 minutes.

Un total du temps prévu et du temps effectué a été calculé pour chaque sprint.

Le premier cours de Projet Intégré a été à lui seul notre sprint 0, qui était dédié à la préparation du projet. Pour ce sprint le temps prévu et effectué est identique (105 minutes).

Le sprint 1 s'est déroulé sur les trois premières semaines du projet, du 14 octobre au 1 novembre. Pour rappel, ce sprint était consacré à la préparation des données et aux choix des méthodes d'analyse. Durant ces trois semaines, nous avons eu trois autres cours. Donc le total du temps prévu est de 315 minutes et celui du temps effectué est de 415 minutes car nous avons continué à travailler sur le projet pendant notre semaine de vacances.

C'est d'ailleurs à partir de cette semaine de vacances (qui correspond également au début du confinement) que notre équipe s'est mise à travailler en dehors des heures de cours. Comme le montre la différence entre le temps total prévu et effectué par sprint (dernière colonne du tableau), plus le projet avançait et plus nous travaillions à la maison. En effet lors du sprint 2, nous avions travaillé 285 minutes (=810-525), c'est-à-dire 4h45, en plus des heures de cours en trois semaines. Donc en moyenne par semaine nous avons travaillé 95 minutes soit environ 1h30. Et lors du dernier sprint qui ne dure qu'une semaine, nous avons

travaillé 275 minutes (=485-210) en plus des cours, ce qui correspond approximativement à 4h30.

Mais comme expliqué précédemment, notre objectif n'était pas de respecter scrupuleusement le temps prévu mais de se conformer au deadline que nous nous fixions. Nous avons réussi à respecter les délais, sauf pour le sprint 3 où nous l'avons dépassé de quelques jours car nous n'avions pas prévu de faire autant de nettoyage des données finales. A nos yeux, nous avons quasiment toujours atteint nos objectifs temporels. Et nous avons pu, comme nous le souhaitions, nous auto-former sur différents sujets, ce qui explique le temps passé à travailler en dehors des cours.

Pour créer le Burndown Chart, nous avons commencé par synthétiser le temps prévu et effectué par semaine dans le tableau ci-dessous.

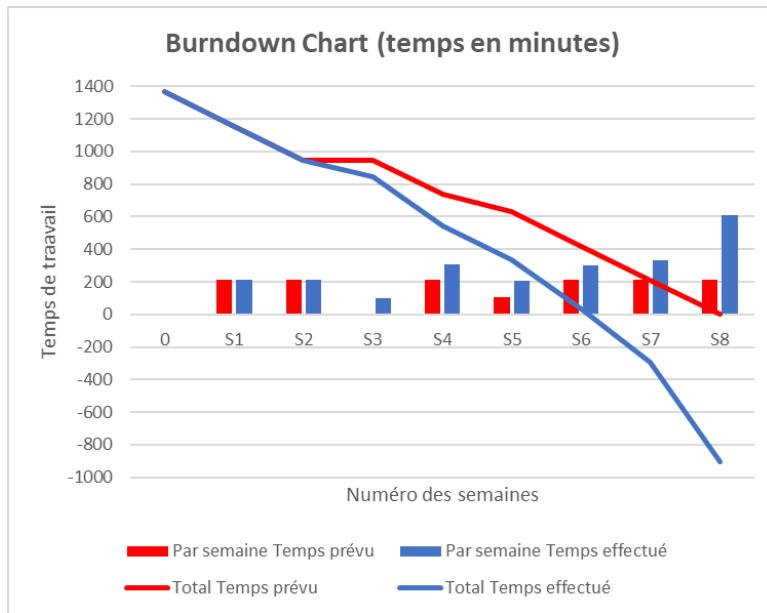
Numéro de semaine	0	S1	S2	S3	S4	S5	S6	S7	S8
Date		14/10 - 18/10	19/10 - 25/10	26/10 - 1/11	2/11 - 8/11	9/11 - 15/11	16/11 - 22/11	23/11 - 29/11	30/11 - 6/12
Par semaine	Temps prévu	210	210	0	210	105	210	210	210
	Temps effectué	210	210	100	305	205	300	330	610
Total	Temps prévu	1365	1155	945	945	735	630	420	210
	Temps effectué	1365	1155	945	845	540	335	35	-295
									-905

**Figure 102.** Tableau de synthèse du temps prévu et effectué sur le projet

Les deux premières lignes sont calculées en faisant la somme des temps prévus de la semaine et la somme des temps effectués.

Par exemple, le temps prévu de la semaine 8 est égal au temps prévu du sprint 4 (210 minutes) alors que le temps effectué de la semaine 8 est la somme du temps effectué sur le sprint 3 et 4 (125+485=610).

Les deux dernières lignes, nommée total, sont calculées de la manière suivante. Leur point de départ est la somme du temps de travail réalisé durant les treize séances de cours en minutes ( $1365=13 \times 105$ ). Et chaque semaine, le temps réalisé est soustrait du temps total de la semaine précédente. La ligne de total du temps prévu, nous permettait donc de voir le temps qu'il nous restait à travailler durant les séances de cours. Alors que la ligne du total du temps effectué permet de connaître la somme totale du temps travaillé à la maison sur le projet. Ainsi au total nous avons travaillé 905 minutes en plus du temps de cours, soit 15h, qui correspond à environ 8,5 séances supplémentaires.



**Figure 103.** Burndown Chart du projet créé à partir du tableau de synthèse du temps prévu et effectué

Le Burndown Chart montre graphiquement cette différence entre le temps prévu et le temps effectué. Les deux courbes correspondent aux deux dernières lignes du tableau de synthèses, c'est-à-dire au temps total prévu et effectué. Et les différents petits histogrammes se réfèrent aux deux premières lignes, donc chaque histogramme représente le temps prévu ou effectué de la semaine.

### 6.3.5 Conclusion

Ce projet nous a donc permis de mettre en pratique la méthode SCRUM. Nous avons découvert grâce à elle une nouvelle façon de travailler, de s'organiser et surtout une nouvelle façon d'appréhender un projet.

En suivant la logique de cette méthode, nous avons appris à nous poser régulièrement pour réfléchir ensemble aux besoins du client, afin de diriger la suite de nos recherches et notre travail dans le seul but de s'approcher au plus près des attentes du client.

Nous avons également compris l'importance de se fixer dès le départ des objectifs très précis lors des réunions de planification des sprints, pour être plus efficaces dans nos recherches et dans nos implémentations. Plus l'objectif est précis, moins il est possible de divaguer et de perdre du temps.

Nous avons aussi découvert la nécessité de faire des réunions régulières afin de régler tous ensemble les problèmes dès leur apparition : la mêlée bi-hebdomadaire permettaient de régler les difficultés techniques et les rétrospectives des sprints servaient à résoudre les désagréments organisationnels.

De plus, cette approche étant très visuelle grâce au Product Backlog (aux Sprint Backlog) et au Burndown Chart, permet de suivre l'avancée du projet facilement. Les réunions sont plus efficaces puisque la majeure partie des éléments sont synthétisés progressivement dans le Product Backlog. Les mêlées bi-hebdomadaires sont centrées uniquement sur l'essentiel : l'avancement réel du travail.

N'ayant pas de hiérarchie, cette méthode permet à chacun de s'épanouir, d'apporter ses compétences et ses idées. La coopération entre les différents membres est ainsi favorisée.

Ce projet a donc été l'occasion de tester pour la première fois la méthode SCRUM et de comprendre concrètement son impact sur notre manière de travailler.

## 6.4 Projet Intégré

### 6.4.1 Introduction

Notre mission est d'analyser des données de DBLP, pour ce faire nous avons regardé le type de données qui nous ont été fourni au début du projet. En analysant les données, nous nous sommes intéressés aux différents liens qu'il pouvait y avoir entre les auteurs. Nous sommes donc parties sur une analyse des relations d'auteur co-auteurs en fonction des articles qu'ils avaient publiés sur DBLP.

Au travers de notre analyse nous voulions savoir s'il existait des auteurs qui publiait exclusivement tout seul. S'il existait des auteurs qui travaillaient qu'avec un groupe d'individu de façon exclusif. Ou au contraire si certains auteurs travaillaient avec de nombreux groupes d'auteurs différents. Nous voulions également déduire du titre des publications des auteurs leur thème de recherche.

### 6.4.2 Nettoyage des données

**6.4.2.1 Choix de l'échantillon** Pour répondre à nos objectifs nous avons dû préparer nos données afin qu'elles correspondent à nos attentes. C'est-à-dire que nous avons sélectionné les publications complètes, composées d'un titre en anglais.

**6.4.2.2 Packages utilisés** Pandas est une bibliothèque permettant l'analyse des données notamment les dataframes dans notre cas. Le package nous a permis de faire l'ouverture des fichiers Csv, les jointures entre plusieurs tables mais aussi l'enregistrement en Csv.

Enchant, est un package très complet, il sert notamment à la correction orthographique, nous nous en sommes servi afin de détecter si les titres d'articles sont en anglais à l'aide de la librairie SpellChecker.

Nltk, est une bibliothèque permettant le traitement de texte pour la classification et la tokenisation des mots dans notre cas nous nous en sommes servi pour sa librairie de stopword (mot de liaison sans intérêt pour l'analyse).

Gensim est une librairie pour le traitement du langage naturel, nous l'avons utilisé pour la tokenisation de nos stopword.

**6.4.2.3 Création de l'échantillon** Pour créer notre échantillon de données, nous avons importé le fichier publication fourni par le groupe interface, sur notre demande ils ont ajouté une colonne is\_complete (booléen) qui nous permet de savoir si la publication est complète ou non. Nous avons donc filtré les publications afin d'obtenir uniquement celles complètes. Après avoir étudié nos données, nous nous sommes aperçus que ce fichier contenait plusieurs langues. C'est pourquoi nous avons décidé d'appliquer une fonction du package enchant, pour pouvoir récupérer uniquement les publications anglaises. Suite à cela nous avons remarqué qu'il y avait une catégorie d'article qui n'était pas réellement composée de titres d'article. Leur titre contenait seulement "home page". Tous les articles appartenaient à la catégorie "www", nous avons également fait le choix de les exclure.

Avant la mise en place de nos filtres il y avait 7 914 201 publications, après l'application de nos premiers filtres (publications complètes et en excluant les catégories « www » pour les titres « Home Page ») nous avons obtenu 636 648 publications. À la suite de la détection de l'anglais, il nous restait seulement 349 331 publications, cela nous paraît assez étonnant qu'il n'y ait que 55% de ces publications entières en anglais. Nous avons fait quelques tests afin de

pouvoir regarder les publications considérées comme non anglais et il s'avère que certaines sont de faux négatifs. Cela peut s'expliquer par la fonction de détection du langage anglais qui exclut la publication dès qu'un mot dans son titre n'est pas considéré comme anglais. Nous avons remarqué que certaines publications étaient à la fois écrites en anglais et dans une autre langue, celles-ci sont malheureusement exclues.

Nous avons ensuite fait la jointure des tables afin que nos filtres soient aussi appliqués sur l'intégralité de nos tables à l'aide de la fonction merge du package pandas.

Pour la table keywords nous avons décidé de la recréer entièrement. Nous avons lister les keywords présent dans la table publications\_keyword\_filtre et nous avons compté le nombre d'occurrence grâce à la fonction value\_counts(). Nous avons fait de même pour la table author mais à partir de la table publication\_author\_filtre.

Afin d'avoir une table publications\_keyword\_filtre contenant que des mots intéressant pour l'analyse, nous avons retiré les StopWords. Pour cela le package Nltk nous a permis d'avoir accès à une petite bibliothèque de StopWords et Gensim a été utile pour la tokenisation de StopWord. Mais pour des raisons de temps d'exécution, nous avons pris la décision de nous concentrer sur un échantillon de mot clés. Pour cela nous avons sélectionné de manière aléatoire un échantillon de publication. Nous n'avons pas pris les publications dans l'ordre du fichier car cela aurait biaisé la sélection, puisque les publications sont ordonnées selon leur catégorie. Une fois nos publications sélectionnées nous faisons une jointure entre l'id de la publication et la table publication keywords afin de récupérer l'intégralité des mots-clés de chaque publication sélectionnée. Nous nous sommes concentrés sur 20 000 publications. Avec ces 20 000 publications nous obtenons environ 183 000 mots-clés. Une fois les StopWords retirés, il restait 141 000 mots-clés.

C'est sur cet ensemble de fichiers que toutes les analyses ont été faites.

### 6.4.3 Analyse des communautés d'auteurs – co-auteurs

**6.4.3.1 Introduction** Pour analyser nos communautés d'auteurs nous avons décidé d'utiliser la théorie des graphes. Cette méthode est très pratique pour représenter sous forme de graphe des relations entre des individus. Nous obtenons un réseau d'individu, où les sommets sont nos individus et les arcs représentent les liens, ici les publications, entre deux sommets de notre graphe.

Il existe de nombreuses méthodes, en théorie des graphes, pour faire de la détection de centralité, puisqu'il existe de nombreuses notions de centralité. Nous avons donc décidé de partir sur trois méthodes de détection de centralité, car elle nous permettait de répondre de façon efficace aux questions que nous nous sommes posés.

Dans un premier temps nous avons cherché la centralité des degrés. Elle consiste à trouvé le/les sommets qui ont le plus de relation directe avec d'autres sommets. Nous avons donc chercher le degré (c'est-à-dire le nombre d'arcs que possède un sommet) maximal du graphe.

Dans un premier temps nous avons cherché la centralité des degrés. Elle consiste à trouvé le/les sommets qui ont le plus de relation directe avec d'autres sommets. Nous avons donc chercher le degré (c'est-à-dire le nombre d'arcs que possède un sommet) maximal du graphe.

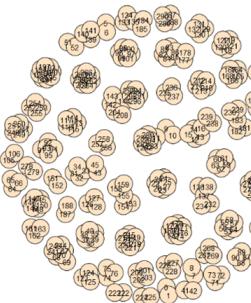
Dans un premier temps nous avons cherché la centralité des degrés. Elle consiste à trouvé le/les sommets qui ont le plus de relation directe avec d'autres sommets. Nous avons donc chercher le degré (c'est-à-dire le nombre d'arcs que possède un sommet) maximal du graphe.

Dans un deuxième temps nous nous sommes intéressés aux communautés que pouvaient former les auteurs. Nous avons choisi de créer nos communautés à partir de l'algorithme de partition Louvain (2008, Vincent Blondel).

Cet algorithme nous permet de détecter nos communautés en effectuant un partitionnement d'un réseau en optimisant la modularité. La modularité est une valeur comprise entre -1 et 1, qui est attribuée à l'arrête entre deux sommets afin de définir si un sommet est plus proche de tel ou tel groupe. Nous avons choisi d'utilisé cette algorithme car il est pratique sur de gros réseaux comme le nôtre. En utilisant l'algorithme de Louvain nous avons implicitement choisi de ne pas traiter les communautés chevauchantes, donc un auteur ne peut appartenir qu'à un seul groupe.

**6.4.3.2 Transformation de nos données** Dans un premier temps, pour créer notre graphe nous avons transformé nos données en matrice d'adjacence, pour pouvoir faire le lien entre nos auteurs. Pour cela nous nous sommes servis des fichiers auteurs\_publication\_filtre.csv, auteur\_filtre.csv et publication\_filtre.csv. Nous avons construit notre matrice de la manière suivante : Nous parcourons chaque publication, puis récupéreront la liste des auteurs qui les ont écrites. Nous avons créé une matrice carrée composé de 0 et de 1. Un 1 entre deux auteurs signifie qu'ils ont déjà écrit un article ensemble. Cette matrice n'est pas pondérée, donc on ne prend pas en compte le fait qu'un auteur écrive beaucoup avec un autre auteur. A la fin nous obtenons la matrice d'adjacence des auteurs – co-auteurs.

Comme notre matrice était très grosse du fait qu'il y avait beaucoup d'auteur, nous avons décidé de l'afficher sous forme de graphe, pour nous aider à mieux représenter les relations entre les auteurs. Pour cela nous avons utilisé la librairie igraph sous python qui nous permet d'afficher des graphes à partir d'une matrice d'adjacence. Nous avons été très déçus du résultat car ce n'est pas lisible. On ne voyait pas les arcs entre nos sommets ce qui n'améliorait pas notre compréhension comme nous l'avions espéré.



**Figure 104.** Graphe statique sur 100 publications

Au vu de ce problème, nous avons décidé de changer de librairie. Nous avons donc cherché une librairie qui nous permettait d'afficher nos graphes de façon dynamique, pour pouvoir zoomer sur les liens entre nos auteurs et donc analyser des gros réseaux.

**6.4.3.3 Création du graphe dynamique** Nous avons donc trouvé le package pyvis, qui nous permet de créer des fenêtres, afin d'afficher nos graphes de façon dynamique. Cela a rendu la lecture du graphe interactif car nous avons la possibilité de zoomer et de déplacer nos sommets, et donc cela nous a permis de mieux visualiser les relations entre nos auteurs. Toutefois, pour utiliser pyvis nous avons dû créer de nouveaux fichiers .csv qui étaient plus conformes au type de fichier que peut exploiter pyvis.

Nous avons, donc, créé un fichier .csv que nous avons appelé edge, qui nous a permis pour chaque auteur de lui recenser ces liens avec les autres auteurs en fonction des articles

qu'ils ont publié. Cette fois, les liens entre deux auteurs sont pondérés par le nombre de publications qu'ils ont écrit ensemble. Pour cela nous sommes parties des fichiers publication\_auteur\_filtre.csv, auteur\_filtre.csv et publication\_filtre.csv.

Principe de notre code :

- On applique, un premier filtre qui nous permet d'enlever les publications qui n'ont aucun auteur, c'est-à-dire que nb\_authors = 0 dans le fichier publication qui nous a été fournis par le groupe interface. Ensuite nous nous sommes assurées de ne récupérer que les publications qui ont plus d'un auteur, puisque nous voulons afficher les liens entre nos auteurs, il faut donc qu'il y ait un minimum de deux auteurs pour une même publication. Nous nous sommes rendu compte qu'aucun de nos articles n'avait été écrit par un seul auteur.
- Pour chaque publication, on récupère sa liste d'auteurs, que l'on trie par ordre alphabétique. Lors de la première exécution, nous nous sommes aperçus que pour certains articles, la liste de ses auteurs n'avait pas été définie correctement. Par exemple, pour une publication qui a été écrite par dix auteurs nous ne connaissons pas la liste de ces auteurs, ce qui est problématique pour le reste du code. Nous avons donc dû enlever les publications dont la liste des auteurs n'étaient pas connues.
- On récupère ensuite le premier auteur de la liste, puis on boucle sur chacun de ses co-auteurs, on vérifie que ce couple n'existe pas déjà, s'il existe on lui incrémente leur nombre de lien d'un, sinon on crée leur relation.
- A la fin on enregistre le DataFrame dans un fichier edge.csv pour pouvoir l'utiliser pour la création de notre graphe.

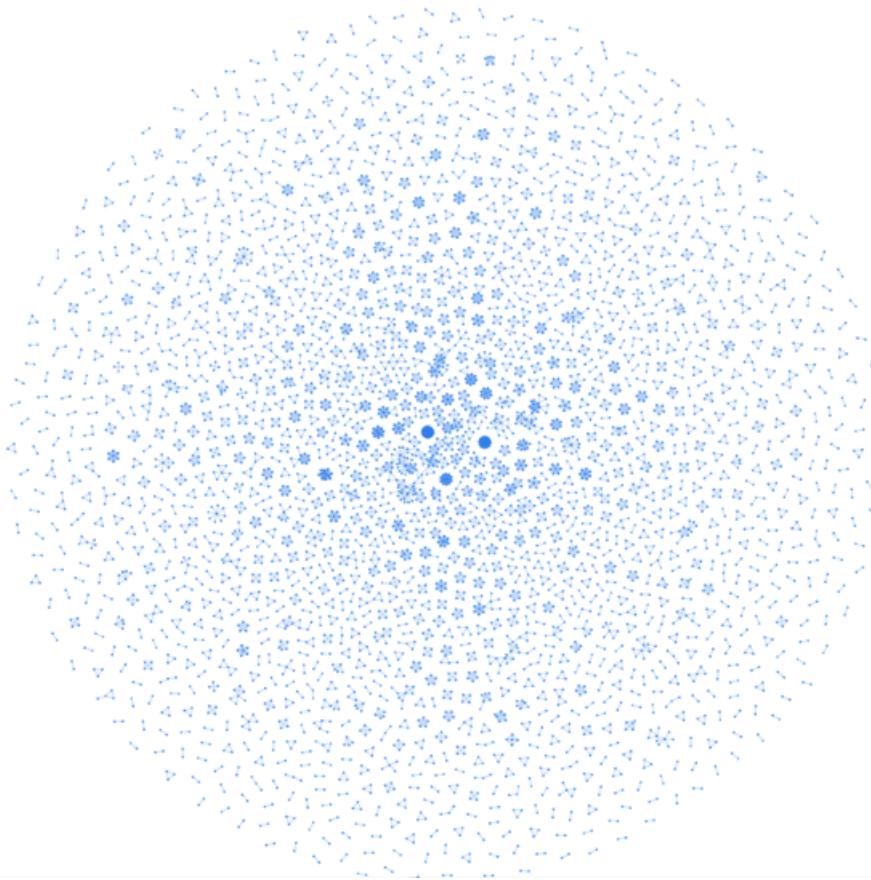
Nous avions également écrit le code pour créer un fichier node qui nous aurait servi si certains auteurs avaient écrit des publications seuls, afin de leur faire apparaître sur notre graphe. Mais dans nos données finales ce cas n'existaient plus donc nous n'avons finalement pas utilisés cet algorithme.

Nous avons fait notre analyse sur un échantillon de 7 000 publications sur 349 331, car le graphe généré pour la totalité des publications est beaucoup trop lourd pour nos ordinateurs. Nous avions précédemment générer un graphe avec 16 000 publications, puis 10 000, mais même constat, nous avions des graphes trop lourd. Ils n'étaient plus vraiment dynamique à cause du temps de chargement trop long entre chaque manipulation (plus de cinq minutes pour se déplacer que de quelques millimètres).

Nous avons recréé un fichier edge.csv à partir de notre échantillon. Nous sommes arrivés à un total de 2 613 publications exploitables, car 1 468 publications n'avaient aucun auteur et 2 219 publications avaient une liste inconnue alors qu'il y avait un nombre d'auteurs supérieur à 1.

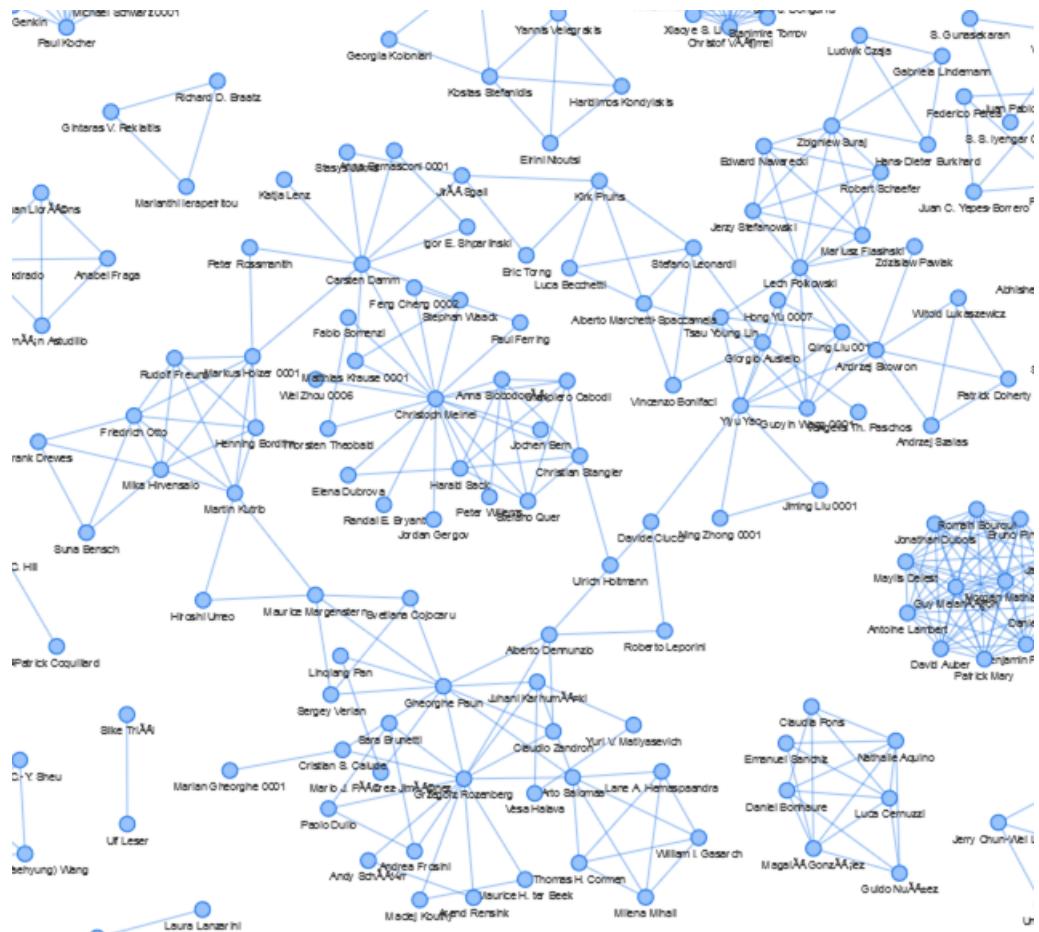
Nous avons pu afficher notre réseau dans la fenêtre du graphe dynamique.

Cette première visualisation nous a permis de voir que notre graphe était partiellement connexe, et que la majorité des auteurs travaillent avec de petits groupes sans avoir de liens avec une autre structure.



**Figure 105.** Graphe des communautés auteurs – co-auteurs

En agrandissant notre graphe nous avons trouvé un sous-graphe qui avait de nombreuse connexion entre les auteurs, sans qu'ils ne forme une clique, c'est-à-dire qu'il n'avait pas tous des liens. Nous soupçonnons que nos auteurs centraux se trouveront dans ce sous graphe.



**Figure 106.** Graphe des communautés auteurs – co-auteurs en faisant un zoom sur la communauté la plus grosse

Lorsque nous avons voulu analyser plus en profondeur notre graphe, nous nous sommes rendu compte que la librairie pyvis n'était conçue que pour l'affichage d'un graphe, nous avons donc dû trouver une librairie qui nous permettait de travailler sur des graphes mais qui était compatible avec pyvis pour que nous puissions garder cet affichage interactif. Nous sommes donc parties sur la librairie Networkx, qui nous permet d'analyser notre graphe. Puis nous sommes passé à l'analyse de notre graphe.

#### 6.4.3.4 Détection des auteurs centraux

**Centralité de degré** La première centralité que nous avons cherchée est la centralité de degrés.

Cette méthode consiste à récupérer les degrés des sommets. Le degré d'un nœud est égal au nombre d'arrêtes que possède ce nœud, comme nous ne travaillons pas sur un graphe orienté, il n'y a pas de différence entre les arrêtes qui arrivent sur le nœud et les arrêtes qui en sortent.

Après avoir calculé le degré de chaque sommet, la méthode sélectionne les auteurs qui possèdent le degré maximal. Cela signifie que ces auteurs sont en contact avec le plus grand nombre d'auteurs.

La commande que nous avons utilisée est `degree_centrality`. Elle prend en entrée le graphe d'auteur – co-auteur. Et on récupère un dictionnaire, qui associe à chaque sommet (c'est-à-dire à chaque auteur) leur degré qui a été normalisé.

Pour trouver les auteurs qui ont le degré maximal nous avons d'abord récupéré la valeur du degré maximal. Dans notre échantillon le degré maximal est de 0,0032. Puis nous avons parcouru tous nos sommets et nous avons stocké les individus qui avaient ce degré maximal.



**Figure 107.** Graphe des communautés auteurs – co-auteurs, en faisant un zoom sur l'auteur ayant le degré de centralité maximal

Dans notre échantillon, il n'y a qu'un seul auteur, Christoph Meinel (c'est le gros point bleu), qui possède le degré de centralité maximal.

**Centralité de proximité** Nous nous sommes ensuite intéressés à la centralité de proximité. Nous voulions savoir quels étaient les individus qui pouvait interagir facilement et rapidement avec une majorité des sommets accessible pour lui. Formellement, ont défini la proximité comme l'inverse de la somme des distances parcouru entre des sommets u et v.

$$C_c(v) = \frac{1}{\sum_{u \in V \setminus \{v\}} d_G(u,v)}$$

avec  $d_G(u,v)$  la distance entre les sommets  $u$  et  $v$ .

**Table 6.** Formule de la proximité 1

La commande que nous avons utilisée est `clonness_centrality`. Elle prend en entrée notre graphe. Et on récupère un dictionnaire, qui associe à chaque sommet (qui sont les auteurs) leur degré de proximité qui a été normalisé en fonction du nombre de nœuds dans la partie connectée du graphe qui contient le nœud.


**Figure 108.** Graphe des communautés auteurs – co-auteurs, en faisant un zoom sur l'auteur le degré de centralité de proximité maximal

Pour détecter nos auteurs qui ont une centralité de proximité maximale, on trouve la valeur maximale de proximité, qui est dans notre échantillon de 0,0039. Puis on récupère les individus qui sont égaux à cette valeur.

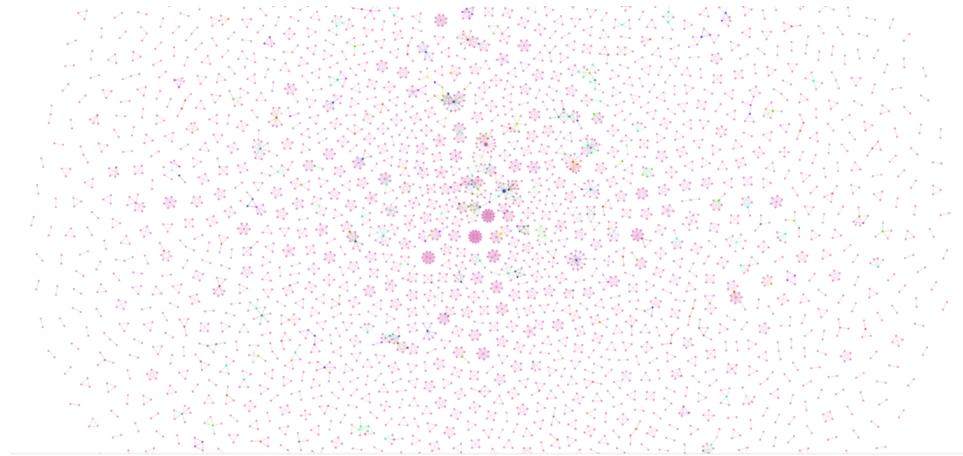
Nous nous sommes aperçus que, Gheorghe Paun, était le seul individu qui possédait cette valeur de degré de proximité. Comme on peut le voir sur le graphe, il est en contact de façon directe avec plein d'individus qui sont en contact avec d'autres sous-groupes ce qui explique le fait que ce soit lui qui soit l'individu avec le degré de proximité le plus élevé.

**Centralité d'intermédiarité** Enfin, la dernière méthode que nous avons testée est la centralité d'intermédiarité. Elle détecte les auteurs qui permettent de faire circuler l'information entre deux autres auteurs. En effet le degré d'intermédiarité calcule le nombre de fois qu'un sommet est apparu lorsque l'on trace les plus courts chemins de toutes les paires de sommets possibles. Nous cherchons donc les sommets qui permettent de faire le lien rapidement entre deux autres sommets du graphe. Cette notion est beaucoup étudiée par les sociologues pour comprendre les dynamiques de groupe. Dans notre cas ces individus permettent de faire le pont entre deux groupes de chercheurs qui n'ont aucun lien direct.

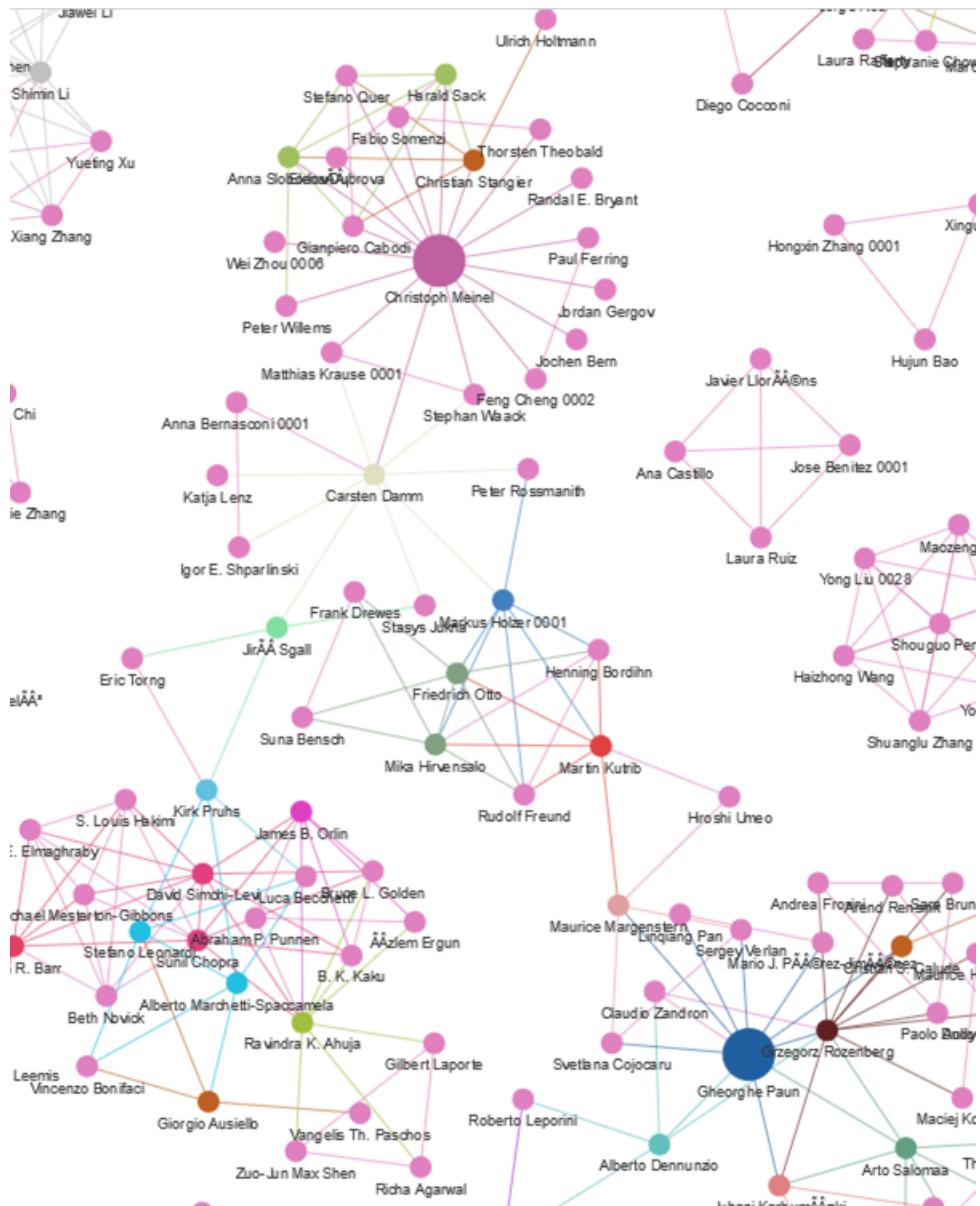
La commande que nous avons utilisée est `betweenness_centrality`. Elle prend en entrée notre graphe. Et on récupère un dictionnaire, qui associe à chaque sommets (qui sont les identifiants des auteurs) leur degré d'intermédiarité qui a été normalisé.

Nous avons ensuite récupéré le degré maximal d'intermédiarité, qui est de 0,00014. Puis nous l'avons comparé avec tous les degrés des sommets afin de récupérer les sommets ayant ce degré d'intermédiarité maximal. Cet auteur est Gheorghe Paun, c'est également lui qui a le degré de proximité maximal. Donc ce chercheur est un point de passage important.

Afin de voir tous les auteurs ponts du graphe, nous avons décidé de colorer les sommets selon leur degré de centralité d'intermédiarité. Pour chaque degré de centralité, la couleur est générée aléatoirement. Nous avons également afficher la valeur des degrés lorsqu'un sommet est sélectionné.



**Figure 109.** Graphe des communautés auteurs – co-auteurs avec les sommets colorés selon le degré de centralité d'intermédiarité



**Figure 110.** Graphe des communautés auteurs – co-auteurs avec les sommets colorés selon le degré de centralité d'intermédiarité en faisant un zoom sur les deux auteurs centraux

En zoomant, nous pouvons voir que les principaux auteurs ponts se trouvent dans le plus grand sous-graphe. Nous nous sommes aperçus que les auteurs qui avaient un degré d'intermédiarité élevé servait de chemin entre nos deux auteurs centraux.

**Détection des communautés** Après avoir trouvé nos auteurs centraux nous avons cherché à partitionner le graphe en communauté d'auteurs.

Nous nous sommes intéressés qu'aux communautés non chevauchante c'est-à-dire que nos auteurs ne pouvaient appartenir qu'à un seul et unique groupe. On découvre des cliques d'auteurs, c'est-à-dire des sous-graphes très connexes, dont ces auteurs ont des liens directs.

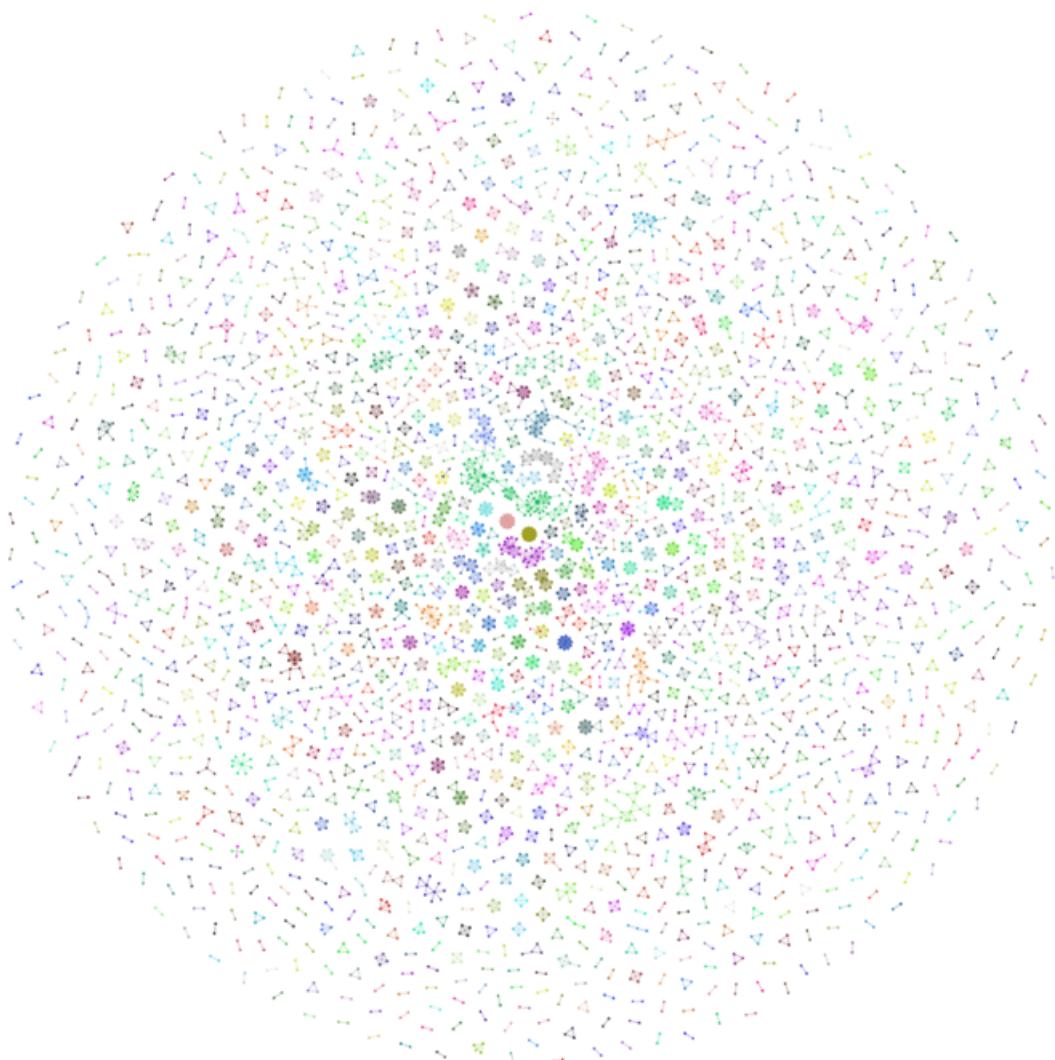
Pour cela nous avons utilisé l'algorithme de Louvain (2008, Vincent Blondel), qui est un algorithme qui est souvent utilisé pour faire de la détection de communauté dans des grands

graphes.

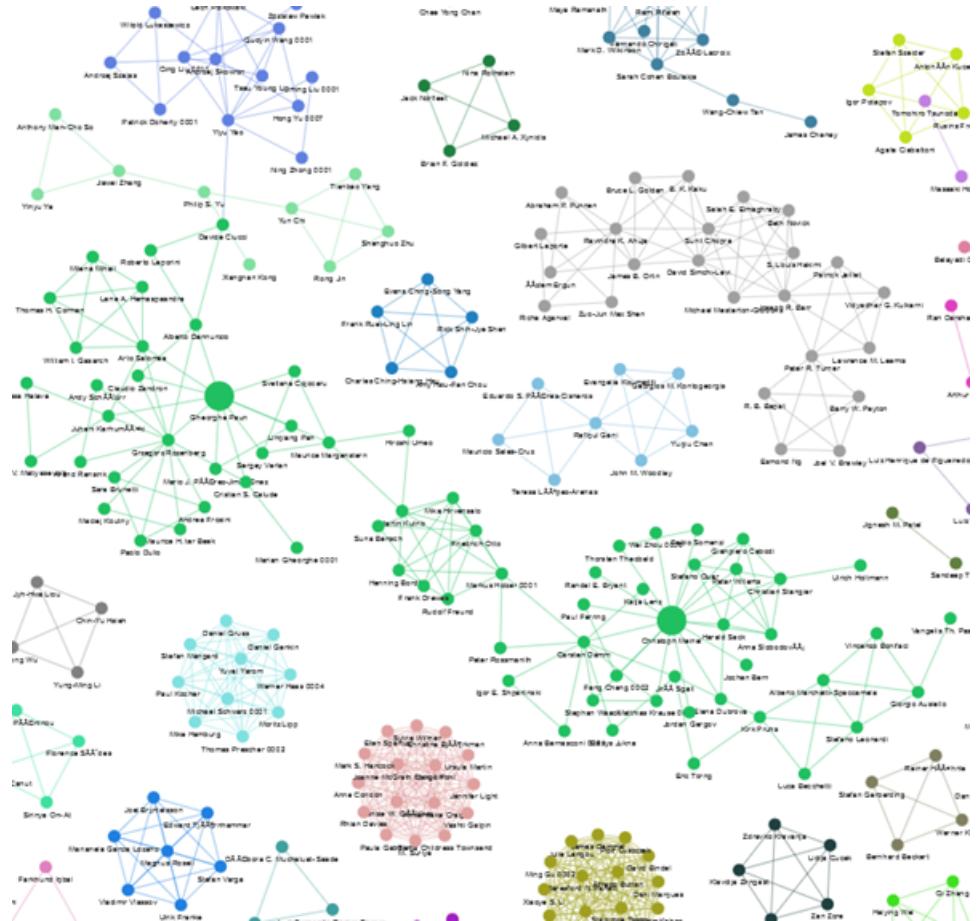
Il va calculer la modularité de chaque sommet, puis il va regrouper les sommets qui ont une modularité forte (c'est-à-dire qui sont proche et qui sont souvent utilisé pour des plus court chemin) puis il va les rassembler sous un sommet unique, ce qui va créer un nouveau graphe. Il va se rappeler sur ce nouveau graphe. Il va continuer ainsi jusqu'à que tous les individus ai été traité, c'est-à-dire qu'il n'y a plus d'individu à rassembler. Cet algorithme a été optimisé, car il va d'abord s'occupé des petits groupes et finir par les plus gros sur un graphe plus petit.

Nous avons utilisé la librairie community et la fonction best\_partition qui utilise l'algorithme de Louvain. Elle prend en entrée notre graphe. Et on récupère un dictionnaire, qui associe à chaque sommets (qui sont les identifiants des auteurs) une valeur comprise entre 0 et le nombre de partition trouver. Cette valeur correspond au groupe auquel l'auteur appartient.

Nous avons ensuite coloré notre graphe selon les différentes partitions. Pour chaque partition, la couleur est générée aléatoirement.



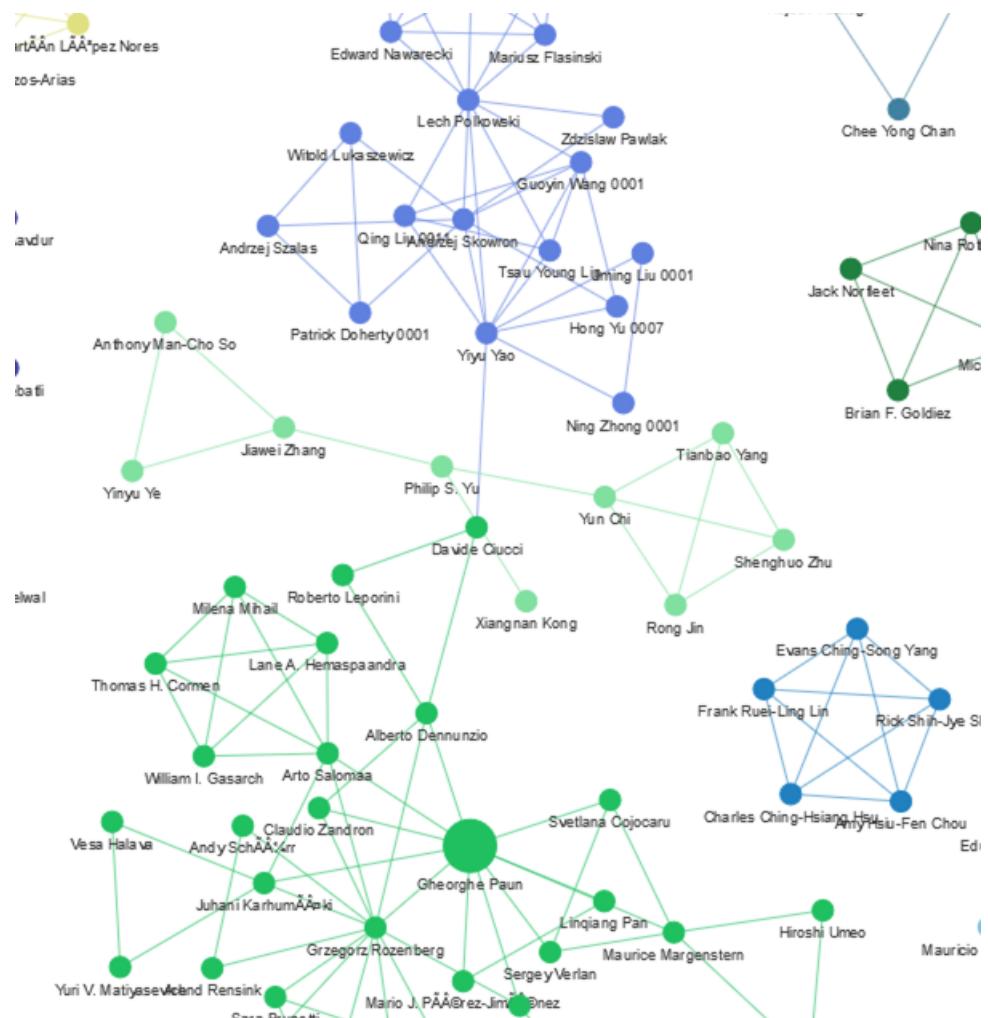
**Figure 111.** Graphe des communautés auteurs – co-auteurs avec les sommets colorés selon leur partition



**Figure 112.** Graphe des communautés auteurs – co-auteurs avec les sommets colorés selon leur partition en faisant un zoom sur la plus grande partition

En agrandissant, nous nous sommes aperçus que nos auteurs centraux faisait bien partie de notre plus grande communauté qui n'est autre que notre sous-graphe.

De plus, la plupart des auteurs travaillent uniquement avec des individus appartenant à leur communauté. Seul un individu, Davide Ciucci, travaillent avec deux auteurs (Philip S. Yu et Yiyu Yao) qui appartiennent à des communautés différentes.



**Figure 113.** Graphe des communautés auteurs – co-auteurs avec les sommets colorés selon leur partition en faisant un zoom sur Davide Ciucci

**Conclusion** En conclusion, les auteurs ont tendance à travailler avec des petits groupes d'auteurs. Cependant, il existe des auteurs qui travaillent avec pleins auteurs qui ne collaborent pas forcément tous ensemble. Nous avons même découvert un auteur qui travaille avec deux individus qui sont dans deux communautés différentes de la sienne.

La détection des auteurs centraux ont mis en avant deux auteurs : Christoph Meinel qui est l'auteur ayant la plus forte centralité de degré et Gheorghe Paun qui est à la fois l'auteur ayant la plus forte centralité de proximité et la plus forte centralité d'intermédiairité.

#### 6.4.4 Créeation de thème à partir des titres des publications

##### 6.4.4.1 LDA

**Choix du modèle** Le modèle choisi est le Latent Dirichlet Allocation (LDA). En plus d'être un modèle conseillé dans l'analyse de document texte, l'avantage du LDA est qu'il considère qu'un document est une composition de topics : un document A peut-être composé de 30 % de topic 1 et de 70 % de topic 2, tandis qu'un document B peut être composé de 80 % de topic 1 et de 20% de topic 2.

Un autre avantage est que contrairement au modèle du KMeans, le LDA autorise une « superposition » des topics d'un document à l'autre, pour une catégorisation plus souple, et

donc plus précise.

La visualisation est quant à elle très simple et ne requiert que quelques lignes de code pour un interface interactif sous html.

## Nettoyage des données

```
#keyword = pd.read_csv("/Volumes/PierreLGZ/Mes documents/M1/Projet/Final/publication_keywords_filtre.csv",encoding='latin-1')
keyword["keyword"] = keyword["keyword"].astype(str)

#Regroupe les mots clés avec le même identifiant en une phrase séparé par une espace
keyword_conf = pd.DataFrame(keyword.groupby('id_publication')[['keyword']].agg(lambda col: ' '.join(col)))

#Transforme en liste
data = keyword_conf.values.tolist()
```

**Figure 114.** Nettoyage des données

Interprétation du code : Étant donné que l'on a une ligne pour chaque mots clé de chaque publication, on va regrouper tous les mots clés en une phrase en les séparant par un espace.

On obtient donc la base ci-dessous :

Index	Type	Size	Value
21	list	1	['SCALP A ASPECTS FOR INFORMATION LANGUAGE PATTERNS REACTIVE SYSTEMS']
22	list	1	['ENGINE IMPLEMENTING OODB SEARCH A AN USING']
23	list	1	['CLASS DIAGRAMS EARLY MEASURES UML FOR']
24	list	1	['COMPARAISON ALGORITHMES CLASSES CONSTRUCTION D DE HIÉRARCHIES']
25	list	1	['DATA DOMAIN GENERATIVE INFORMATION ORIENTED REUSE SIGNIFICANCE SOFTWARE']
26	list	1	['100 JAVA MANAGER PERSISTENT STORM STORAGE A']
27	list	1	['ADAPTERS CORBA DESIGN OBJECTS PATTERNS SYNCHRONIZATION FOR OF']
28	list	1	['DESIGN A BASED CASE FOR MODELLING PATTERNS REASONNING TOOL']
29	list	1	['A APPLICATIONS DISTRIBUTED QUALITY SERVICE FOR IN MANAGING OF PATTERN']
30	list	1	['A FRAMEWORK JAVA MANAGEMENT OMS OBJECT PERSISTENT']

**Figure 115.** Nettoyage des données 2

**Tokenization** La tokenisation est une façon de séparer un morceau de texte en morceaux plus petits appelés tokens. On peut donc les classer en trois grandes catégories : les mots, les caractères et les sous-mots.

Prenons par exemple la phrase : « Vive la data science ».

La façon la plus courante de former des jetons est basée sur l'espace. En prenant l'espace comme délimiteur, la symbolisation de la phrase se traduit par 4 jetons : « Vive », « la », « data » et « science ». Un autre exemple mais pour des tokens caractères et sous-mots avec "smarter" :

- Caractères : s-m-a-r-t-e-r
- Sous-mots-clés : smart-er

Le but de la tokenisation est d'obtenir un vocabulaire correspondant à l'ensemble des tokens uniques dans le corpus (ensemble des documents).

Code tokenisation :

```
#Permet de tokeniser la phrase : separer chaque mot de chaque phrase
def token(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence), deacc=True))
    #deacc=True enlève ponctuation
    #simple_preprocess : ne prend pas en compte les mots avec une longueur inférieure à 2

data_words = list(token(data))
```

**Figure 116.** code tokenisation

Interprétation du code : on met chaque mot des publications dans une liste avec un élément de la liste correspondant à un seul mot (contrairement à avant où on avait un élément de la liste qui correspondait à tous les mots de la publication)

**Stop-words** Quel que soit la langue, certains mots apparaissent plus fréquemment que d'autre, en anglais, on les appelle les « stop-words ». Un stop-word est un mot couramment utilisé (tel que "le", "la", "les") qu'un moteur de recherche a été programmé pour ignorer car n'apportant pas d'informations intéressantes.

Afin de les supprimer, on peut utiliser la bibliothèque NLTK (Natural Language Toolkit) en python qui possède une liste de mots d'arrêt stockée dans 16 langues différentes.

On peut voir un extrait de son contenu ci-dessous :

Ind	Type	Size	Value
0	str	1	i
1	str	1	me
2	str	1	my
3	str	1	myself
4	str	1	we
5	str	1	our
6	str	1	ours
7	str	1	ourselves
8	str	1	you
9	str	1	you're
10	str	1	you've
11	str	1	you'll

**Figure 117**

Code stop-words :

```
#Fonction pour enlever les stopwords
def stopword(texts):
    stop_words = stopwords.words('english')
    stop_words.extend(['from', 'subject', 're', 'edu', 'use'])
    return [[word for word in doc if word not in stop_words] for doc in texts]
```

**Figure 118**

Interprétation du code : On parcourt les listes de mots des publications et on compare avec le dictionnaire « stopword » du package nltk. Si le mot en fait partie, on l'enlève.

**Lemmatisation** La lemmatisation est un processus de transformation consistant à regrouper les différentes formes « conjuguées » d'un mot (ex : corpora = corpus / better = good). Ce principe de transformation ressemble fortement au stemming mais apporte du contexte au mot.

On peut utiliser la bibliothèque spacy, avec le modèle « en\_core\_web », qui est un modèle à base de réseau neuronal convolutif, permettant de labéliser chaque mot en leur catégorie grammaticale et de récupérer leur forme canonique. La taille du modèle varie et est indiquée à la fin du nom du modèle : small (\_sm) / medium (\_md) / large (\_lg).

Code lemmatisation :

```
#Fonction de lemmatisation
def lemmatisation(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    """https://spacy.io/api/annotation"""
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags])
    return texts_out
```

Figure 119

**Création du dictionnaire et du corpus** Les deux principales entrées pour le model LDA sont le dictionnaire de donnée et le corpus.

Le dictionnaire correspond tout simplement au regroupement des mots en un seul fichier. Chaque mot a un identifiant qui lui est spécifique. Le corpus s'appuie sur le dictionnaire, pour chaque publication, il va indiquer l'identifiant dans le dictionnaire de chaque mot présent dans la publication mais aussi combien de fois ce mot apparaît dans la publication soit mot\_id soit mot\_fréquence

Par exemple : pour index = 1, on a le mot d'identifiant n°1 qui n'est présent qu'une fois dans la publication, même chose pour le mot d'identifiant n°2.

Index	Type	Size	Value
0	list	1	[(0, 1)]
1	list	2	[(1, 1), (2, 1)]
2	list	1	[(3, 1)]
3	list	3	[(4, 1), (5, 1), (6, 1)]
4	list	3	[(7, 1), (8, 1), (9, 1)]
5	list	2	[(7, 1), (10, 1)]
6	list	5	[(11, 1), (12, 1), (13, 1), (14, 1), (15, 1)]
7	list	4	[(16, 1), (17, 1), (18, 1), (19, 1)]
8	list	0	[]
9	list	5	[(20, 1), (21, 1), (22, 1), (23, 1), (24, 1)]
10	list	4	[(12, 1), (13, 1), (25, 1), (26, 1)]

Figure 120

## Code nettoyage

```
#Créer le dictionnaire avec tout le vocabulaire (les données n'apparaissant qu'une seule fois)
id2word = corpora.Dictionary(data_lemmatized)

#Pour pouvoir créer le corpus
texts = data_lemmatized

#Fréquence des termes dans un document
corpus = [id2word.doc2bow(text) for text in texts]
```

Figure 121

**Principe du LDA** Le modèle Latent Dirichlet Allocation (LDA) est un modèle probabiliste génératif qui permet de décrire des collections de documents de texte ou d'autres types de données discrètes.

LDA fait partie d'une catégorie de modèles appelés "topic model", qui cherchent à découvrir des structures thématiques cachées dans des vastes archives de documents. Ceci permet d'obtenir des méthodes efficaces pour le traitement et l'organisation des documents de ces archives : organisation automatique des documents par topics, recherche, compréhension et analyse du texte, ou même résumer des textes.

Aujourd'hui, ce genre de méthode s'utilise fréquemment, notamment en fouille de données et en traitement automatique des langues.

Les trois principaux inputs du LDA sont : un nombre K de topics que l'on aura choisi au préalable, un dictionnaire de mots (mots présents dans tout) et un corpus de fréquence (la fréquence de chaque mot dans le document).

**Initialisation** Ensuite, on attribue un topic à chaque mot de chaque document (principe de topic modeling), selon une distribution de Dirichlet sur un ensemble de K topics (soit de manière aléatoire).

$\Theta_i \sim \text{Dir}(\alpha)$ , avec  $i \in 1, \dots, M$  et  $\text{Dir}(\alpha)$  est une distribution de Dirichlet avec un paramètre symétrique  $\alpha < 1$ .

Ce premier topic model n'est pas très intéressant du point de vue analyse car généré aléatoirement mais il permet de poser les bases pour la suite.

**Déroulement** Une fois que l'on a notre topic model de base, on cherche ensuite à améliorer celui-ci en prenant chaque mot et en mettant à jour son topic le plus représentatif. Ce nouveau topic est celui qui aurait la plus forte probabilité de générer ce mot dans ce document. On fait donc l'hypothèse que tous les thèmes sont corrects, sauf pour le mot en question.

On a donc : pour chaque mot (w) de chaque document (d), on calcule deux probabilités pour chaque topic (t) :

- $p(t | d)$  : la probabilité que le document d soit assigné au topic t
- $p(w | t)$  : la probabilité que le topic t soit assigné au mot w

Le nouveau topic correspond alors à celui avec la probabilité  $p(t | d) * p(w | t)$  soit la probabilité que le topic t génère le mot w dans le document d.

On répète l'étape d'apprentissage jusqu'à ce que les assignations ne bougent plus. On obtient alors les thèmes de chaque document et ainsi que les mots associés.

**Package utilisé** Gensim (Generate Similar) est une bibliothèque de traitement du langage naturel (NLP) open source populaire, utilisée pour la modélisation de topics non supervisée. On peut l'utiliser notamment dans :

- La création de documents ou de vecteurs de mots
- L'identification de topics

- La comparaison de documents (récupération de documents sémantiquement similaires)
- L'analyse de documents en texte brut pour la structure sémantique

Gensim est conçu pour gérer de grandes collections de texte en utilisant le streaming de données ainsi que des algorithmes en ligne incrémentiels (qui apprend à partir de données reçues au fur et à mesure). Il est de nature évolutive, car il n'est pas nécessaire que tout le corpus d'entrée réside entièrement dans la mémoire vive (RAM) à un moment donné. En d'autres termes, tous ses algorithmes sont indépendants de la mémoire par rapport à la taille du corpus.

Gensim est licencié sous la licence GNU LGPL approuvée par OSI qui lui permet d'être utilisé gratuitement. De plus, toutes les modifications apportées à Gensim sont à leur tour open-source et bénéficient également du soutien de la communauté.

**Cohérence** Il existe différents types de cohérence dans gensim chacun ayant son calcul propre, mais de manière générale, elle mesure la distance relative entre deux mots d'un même topic. Pour notre projet, nous nous servirons principalement de la cohérence CV, qui est la cohérence la plus précise sous gensim, ainsi que de la cohérence UMass afin de vérifier nos résultats.

**Cohérence CV** La cohérence CV combine la mesure de confirmation indirect et la fenêtre booléenne coulissante (ou en anglais, the boolean sliding window).

**Mesure de confirmation indirect** La mesure de confirmation indirecte calcule la similarité des mots dans  $W'$  et  $W^*$  par rapport aux confirmations directes de tous les mots. On représente les ensembles de mots  $W'$  et  $W^*$  comme des vecteurs de la taille totale de l'ensemble de mots  $W$ . Ce vecteur peut être calculé par rapport à toute mesure de confirmation directe  $m$ . Sa formule est :

$$\tilde{m}_{sim(m,\gamma)}(W', W^*) = ssim(\vec{v}_{m,\gamma}(W'), \vec{v}_{m,\gamma}(W^*)),$$

Figure 122

ou :

- ssim représente le cosinus

$$\vec{v}_{m,\gamma}(W') = \left\{ \sum_{w_i \in W'} m(w_i, w_j)^\gamma \right\}_{j=1, \dots, |W|}$$

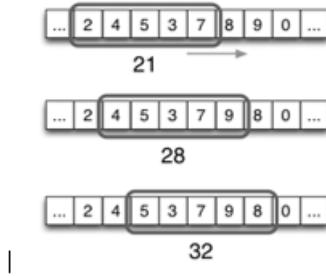
Figure 123

- Y prenant des valeurs dans une plage 1,3

Exemple : Deux marques de montres concurrentes, étant rarement mentionnés dans le corpus, ont une confirmation fortement corrélée à d'autres mots telle que « aiguille » ou « suisse ». Donc la mise en relation de ces deux marques, est ce qu'on appelle une mesure de confirmation indirecte.

**Fenêtre coulissante booléenne** La fenêtre coulissante booléenne détermine le nombre de mots à l'aide d'une fenêtre coulissante. La fenêtre se déplace sur les documents à raison d'un token de mot par étape. Chaque étape définit un nouveau document virtuel en copiant le contenu de la fenêtre. Un document booléen est appliqué à ces documents virtuels pour calculer les probabilités de mots dans cette fenêtre.

Exemple :



**Figure 124**

**Interprétation de la cohérence CV** De manière générale, la cohérence d'un modèle est comprise entre 0 et 1. Avec la cohérence CV, on aimerait obtenir un résultat le plus proche possible de 1, néanmoins, il faut faire attention car obtenir un résultat de 0,9 ou 1 est impossible à moins que tous les mots du corpus ne soient identiques ou très liés (United et States retourneraient un résultat de 0,96 ou bien deux mots identiques, un résultat de 1).

On notera le résultat de la cohérence CV de la manière suivante :

- 0,3 est mauvais
- 0,4 est faible
- 0,55 est pas mal
- 0,65 est bien (objectif)
- 0,7 est excellent
- 0,8 est improbable
- 0,9 est très certainement faux

### Cohérence UMass

#### Mesure de la cohérence UMass

La formule de la mesure de la cohérence UMass est

$$C_{\text{UMass}} = \frac{2}{N \cdot (N - 1)} \sum_{i=2}^N \sum_{j=1}^{i-1} \log \frac{P(w_i, w_j) + \epsilon}{P(w_j)}$$

**Figure 125**

où :

- $P(w_i, w_j)$  représente le nombre de documents dans lesquels les mots  $w_i$  et  $w_j$  apparaissent ensemble. On ajoute un petit quelque chose afin d'éviter de prendre un log de 0 si deux mots n'apparaissent pas ensemble.
- $P(w_j)$  est le nombre de documents dans lesquels apparaît le mot  $w_j$ .
- $N$  représente la quantité de top-words (mots les plus présents d'un topic). Par défaut avec gensim, on a  $N = 20$ .

Par exemple, pour les mots [sport, game, ball, team], on calculerait la cohérence UMass de cette manière :

$$\begin{aligned} C_{\text{UMass}} &= \frac{1}{6} (\log(P(\text{sport}|\text{game})) + \log(P(\text{ball}|\text{game})) \\ &\quad + \log(P(\text{ball}|\text{sport})) + \log(P(\text{team}|\text{game})) \\ &\quad + \log(P(\text{team}|\text{sport})) + \log(P(\text{team}|\text{ball}))) \end{aligned}$$

Avec  $\frac{1}{6} = \frac{2}{4 * (4-1)}$

**Figure 126**

**Interprétation de la cohérence UMass** De manière générale, plus la valeur de la cohérence UMass se rapproche de 0, mieux c'est.

Nous n'allons pas aller plus loin dans l'interprétation car la cohérence UMass ne nous sert qu'à valider nos choix de paramètre pour notre modèle.

**Alpha et éta** Alpha et éta sont deux hyperparamètres du modèle LDA permettant de manipuler la distribution des poids : alpha contrôle la distribution préalable des poids des topics dans chaque document, tandis qu'éta contrôle la distribution préalable des poids des mots dans chaque topic. Ils sont tous deux considérés comme des paramètres de lissage lorsque nous calculons dans quelle mesure chaque document "aime" un topic (dans le cas de alpha) ou dans quelle mesure chaque topic "aime" un mot (dans le cas de gamma).

Pour observer de plus près l'effet de la manipulation de ces hyperparamètres, on peut prendre par exemple diverses valeurs de alpha et éta : 10 (élevé), 0.1 (normal) et 0.01 (faible). Lorsque le paramètre alpha est faible, la majeure partie du poids dans la distribution des topics pour cet article va à un seul topic, mais lorsqu'il est élevé, le poids est beaucoup plus uniformément réparti entre les topics.

Lorsque le paramètre éta est faible, cela se traduit par un poids plus élevé placé sur les mots du haut et un poids plus faible placé sur les mots du bas pour chaque topic. En revanche, un modèle ayant un éta élevé accorde moins de poids aux mots du haut et plus de poids aux mots du bas.

Bien sûr, comme pour tout modèle, il n'existe pas de choix parfait pour les hyperparamètres. Étant donné que les deux hyper-paramètres suivent une distribution de Dirichlet, on peut choisir toute valeur correspondant à une distribution Dirichlet (tout nombre réel positif et on peut aller aussi haut que l'on veut, même si 10 est déjà considéré comme élevé). En ce qui nous concerne, on a choisi un alpha et un éta de 0.1 et 1 respectivement, au vue de la performance de la cohérence du modèle (voir partie suivante : « Nombre de cluster »).

### Code LDA

```
n_topics = 50
# Construction du model avec le nombre de cluster décidé précédemment
lda_model = gensim.models.LdaModel(corpus=corpus,
                                    id2word=id2word,
                                    num_topics=n_topics,
                                    alpha = 0.1,
                                    eta = 1,
                                    random_state=100)
```

Figure 127

Interprétation du code : création du LDA avec les hyperparamètres alpha / éta de 0.1/1 et random\_state sur 100 afin de toujours avoir le même résultat.

**Nombre cluster** Comme indiqué précédemment, pour pouvoir utiliser le modèle de LDA, il va falloir définir au préalable un nombre K de topics. Pour être le plus précis possible, on va comparer plusieurs méthodes.

### Via la cohérence CV et la cohérence UMass Code cohérence

```

#Prend la cohérence la plus élevée des tests
def nombre_topic_coherence(dictionary, corpus, texts, limit, start, step):
    coherence_values_cv = []
    coherence_values_UMass = []
    model_list = []
    n = 1
    for num_topics in range(start, limit, step):
        debut_algo = time.process_time()
        maxi = 0
        for eta in [0.01,0.1,1,5,10]:
            for alpha in [0.01,0.1,1,5,10]:
                model = gensim.models.ldamodel.LdaModel(corpus=corpus, num_topics=num_topics, id2word=id2word, eta = eta, alpha = alpha, random_state=100)
                coherencemodel_cv = CoherenceModel(model=model, texts=texts, dictionary=dictionary, coherence='c_v')
                coherencemodel_UMass = CoherenceModel(model=model, texts=texts, dictionary=dictionary, coherence='u_mass')
                if maxi < coherencemodel_cv.get_coherence():
                    maxiCV = coherencemodel_cv.get_coherence()
                    maxiUM = coherencemodel_UMass.get_coherence()
                    maxiM = model
                coherence_values_cv.append(maxiCV)
                coherence_values_UMass.append(maxiUM)
                model_list.append(maxiM)
        fin_algo = time.process_time()
        print("Boucle exécutée en",round(fin_algo-debut_algo,2),"secondes \nIterations restantes :",
              round((limit-start)/step,0)-n)
        n = n + 1
    x = range(start, limit, step)
    plt.plot(x, coherence_values_cv)
    plt.xlabel("Num Topics")
    plt.ylabel("Coherence score title")
    plt.legend(("coherence_values"), loc='best')
    plt.show()

    plt.plot(x, coherence_values_UMass)
    plt.xlabel("Num Topics")
    plt.ylabel("U_mass score title")
    plt.legend(("U_mass"), loc='best')
    plt.show()

    return model_list, coherence_values_cv

```

Figure 128

Interprétation du code : On teste différents modèles de LDA ayant un nombre de K topics variant entre, par exemple, start = 10 et limit = 100 avec step = 10. Cela signifie que le code va tourner sur K = 10, puis K = 20 , ..., jusqu'à 100.

Pour chacun de ces modèles, on va calculer la cohérence CV et UMass par chaque couple éta/alpha donnant la cohérence la plus élevée pour un K topic donné. A la fin, on affiche les graphiques de l'évolution CV / UMass sur les différents K topics. Ceux-ci nous permettent de prendre une décision quant au nombre de topics le plus optimal.

**Prise de décision via les graphiques** Avec cette méthode, il suffit de calculer la cohérence CV et UMass pour chaque K clusters.

Pour la cohérence CV, on doit choisir le nombre K de topic ayant eu la cohérence la plus élevée en premier, soit ici, 50 topics.

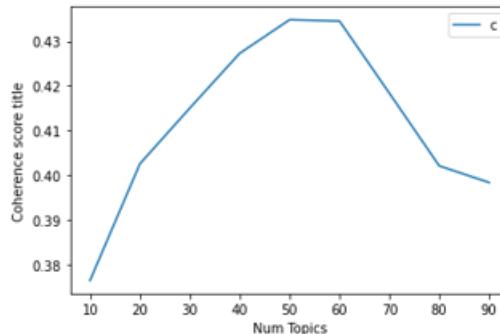


Figure 129

Pour la cohérence UMass, on vérifie si vers 50 topics, la cohérence UMass n'est pas trop faible. Ici, on remarque même qu'elle s'aplatis à partir de 30-40.

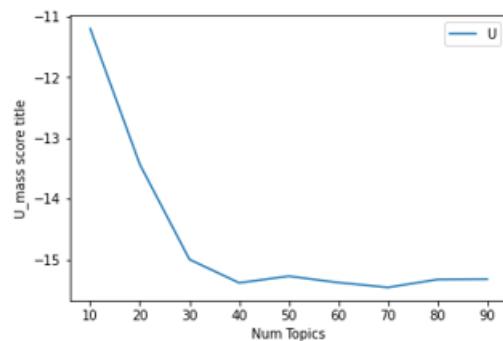


Figure 130

**Via HDP (Processus de Dirichlet Hiérarchique)** Le Processus Dirichlet Hiérarchique (HDP) est une extension de LDA, il partage donc le même principe mais avec comme particularité de pouvoir fournir un résultat sans connaître au préalable le nombre de topics voulu. Ce dernier est déterminé à partir des données via un processus de Dirichlet et peut être illimité.

#### Code HDP

```
#Modèle HDP, modèle sans paramètre, qui apprend des données
#La pour vérifier nos soupçons
Hdp_model = gensim.models.hdpmodel.HdpModel(corpus=corpus, id2word=id2word)
print(Hdp_model.print_topics(num_topics=500 , num_words=5))
```

Figure 131

Interprétation du code : comme on peut le voir, ce modèle a en entrée le même type de donnée que le LDA donc pas besoin de faire un autre nettoyage. Le num\_topic = 500 est juste présent pour pouvoir afficher tous les topics (et non les 20 premiers comme par défaut) et num\_words est là pour n'afficher que les 5 premiers mots du topic.

On obtient via ce code, le résultat suivant (seulement les 25 premiers topics sont visibles ici) :

```
((0, '0.009*network + 0.008*base + 0.007*system +
0.007*model + 0.007*use'), (1, '0.009*network + 0.008*base +
0.007*system + 0.007*model + 0.006*use'), (2,
'0.009*network + 0.008*base + 0.007*system + 0.007*model +
0.006*use'), (3, '0.009*network + 0.008*base + 0.007*system +
0.007*model + 0.006*use'), (4, '0.009*network +
0.008*base + 0.007*system + 0.007*model + 0.006*use'), (5,
'0.009*network + 0.008*base + 0.007*system + 0.007*model +
0.006*use'), (6, '0.009*network + 0.008*base + 0.007*system +
0.007*model + 0.006*use'), (7, '0.009*network +
0.008*base + 0.007*system + 0.007*model + 0.006*use'), (8,
'0.009*network + 0.008*base + 0.007*system + 0.007*model +
0.006*use'), (9, '0.009*network + 0.008*base + 0.007*system +
0.007*model + 0.006*use'), (10, '0.009*network +
0.008*base + 0.007*system + 0.007*model + 0.006*use'), (11,
'0.009*network + 0.008*base + 0.007*system + 0.007*model +
0.006*use'), (12, '0.009*network + 0.008*base +
0.007*system + 0.007*model + 0.006*use'), (13,
'0.009*network + 0.008*base + 0.007*system + 0.007*model +
0.006*use'), (14, '0.009*network + 0.008*base +
0.007*system + 0.007*model + 0.006*use'), (15,
'0.009*network + 0.008*base + 0.007*system + 0.007*model +
0.006*use'), (16, '0.009*network + 0.008*base +
0.007*system + 0.007*model + 0.006*use'), (17,
'0.009*network + 0.008*base + 0.007*system + 0.007*model +
0.006*use'), (18, '0.009*network + 0.008*base +
0.007*system + 0.007*model + 0.006*use'), (19,
'0.009*network + 0.008*base + 0.007*system + 0.007*model +
0.006*use'), (20, '0.009*network + 0.008*base +
0.007*system + 0.007*model + 0.006*use'), (21,
'0.009*network + 0.008*base + 0.007*system + 0.007*model +
0.006*use'), (22, '0.009*network + 0.008*base +
0.007*system + 0.006*model + 0.006*use'), (23,
'0.009*network + 0.008*base + 0.007*system + 0.007*model +
0.006*use'), (24, '0.009*network + 0.008*base +
0.007*system + 0.007*model + 0.006*use'), (25,
'0.009*network + 0.008*base + 0.007*system + 0.006*model +
0.006*use'))
```

**Figure 132**

Au final, les résultats du HDP ne sont pas probant, on a les mêmes contenus de topics, ce qui est dommage car cela aurait-été intéressant de comparer les résultats du LDA et du HDP.

**Via KMeans** Étant donné que KMeans ne prend pas en entrée les mêmes types de données que le LDA, c'est-à-dire un corpus de fréquence et un dictionnaire, on en profite pour utiliser une autre méthode de préparation de données : TF-IDF.

**Nettoyage des données avec TD IDF** La méthode TF-IDF (Term Frequency - Inverse Document Frequency) est une méthode de pondération particulièrement utilisée dans le domaine de la fouille de texte. Elle attribue un poids permettant d'évaluer l'importance d'un terme contenu dans un document, relativement à une collection ou un corpus. Le poids attribué augmente en fonction du nombre d'occurrences du mot dans le document mais également en fonction de la fréquence du mot dans le corpus.

La TF-IDF se fait en deux parties :

- La fréquence du terme représente le nombre d'occurrences d'un terme dans le document considéré (on parle de « fréquence » par abus de langage). On peut choisir cette fréquence brute pour exprimer la fréquence d'un terme.
- La fréquence inverse de document mesure de l'importance du terme dans l'ensemble du corpus. Dans le schéma TF-IDF, elle vise à donner un poids plus important aux termes les moins fréquents, considérés comme plus discriminants. Elle consiste à calculer le logarithme (en base 10 ou en base 2) de l'inverse de la proportion de documents du corpus qui contiennent le terme.

$$:\quad idf_i = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|}$$

**Figure 133**

où :

- $|D|$  : nombre total de documents dans le corpus ;
- $\{d_j : t_i \in d_j\}$  : nombre de documents où le terme apparaît (c'est-à-dire  $n_j \neq 0$ ).

Calcul de TF-IDF : Le poids final est le résultat de la multiplication de ces deux mesures.  
Exemple de TF-IDF :

Phrase n°1 : « The car is driven on the road »

Phrase n°2 : « The truck is driven on the highway »

Dans cet exemple, chaque phrase est un document séparé. On peut désormais calculer le TF-IDF pour les deux documents ci-dessus, qui représentent notre corpus :

Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	$\log(2/2) = 0$	0	0
Car	1/7	0	$\log(2/1) = 0.3$	0.043	0
Truck	0	1/7	$\log(2/1) = 0.3$	0	0.043
Is	1/7	1/7	$\log(2/2) = 0$	0	0
Driven	1/7	1/7	$\log(2/2) = 0$	0	0
On	1/7	1/7	$\log(2/2) = 0$	0	0
The	1/7	1/7	$\log(2/2) = 0$	0	0
Road	1/7	0	$\log(2/1) = 0.3$	0.043	0
Highway	0	1/7	$\log(2/1) = 0.3$	0	0.043

**Figure 134**

La dernière colonne TF\*IDF représente les poids associer pour chaque mot du corpus.

#### Code TF-IDF

```
#Methode TF_IDF : attribue poids à chaque mots
tfidf = TfidfVectorizer(
    max_features = 10000,
    stop_words = 'english'
)

tfidf.fit(keyword_conf["keyword"])
text_tfidf = tfidf.transform(keyword_conf["keyword"])
```

**Figure 135**

Interprétation du code : max\_feature correspond aux mots sur lesquels on va lancer le programme, soit ici les 10 000 mots les plus récurrent dans le corpus (corpus différent du LDA, correspond au fichier regroupant tous les documents).

**Principe de KMeans** En entrée, le modèle KMeans a besoin du corpus transformé par TF-IDF et d'un nombre K de clusters.

**Initialisation** On place les K centroids des K clusters au hasard dans le dataset.

**Déroulement** Chaque point du dataset est affecté au centroid le plus proche. Une fois que chaque point est relié à un centroid, ces derniers sont déplacés au milieu de leur groupe (ou cluster). Cette localisation est la moyenne des points du groupe. On continu de faire cela jusqu'à ce que les centroids ne bouge plus.

De manière plus visuelle, l'algorithme KMeans passe par ces trois étapes :

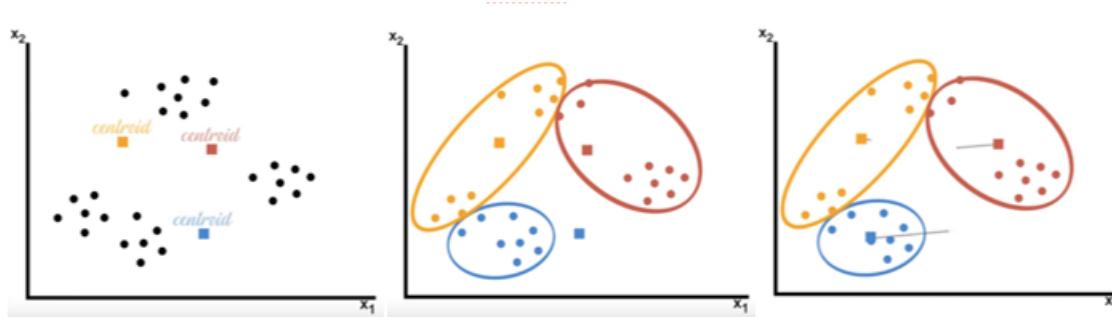


Figure 136

**Package : sklearn** Scikit-learn est une bibliothèque libre Python destinée à l'apprentissage automatique. Elle propose dans son framework de nombreuses bibliothèques d'algorithme à implémenter clé en main. Elle se veut facile à prendre en main et elle est conçue pour marcher avec d'autres bibliothèques tel que Numpy et Scipy.

**Inertie** L'inertie est un indicateur de compacité des classes : il mesure la dispersion à l'intérieur de chaque groupe (inertie intra-classes). Par conséquent, une faible inertie est visée pour le model. Bien entendu, la valeur de l'inertie diminue à mesure que le nombre de clusters augmente. Il y a donc un compromis à faire.

$$\sum_{k=1}^K \sum_{i=1}^{n_k} d^2(i, G_k)$$

Figure 137

où :

- $G_k$  est un centre de classes
- $d()$  est une mesure de distance caractérisant les proximités entre les individus. Par exemple, la distance euclidienne ou euclidienne pondérée par l'inverse de la variance. Il faut faire attention aux points aberrants.

Pour trouver le nombre K de clusters optimal avec l'inertie, on doit trouver le « point de coude » dans le graphique. Ce point de coude représente un ralentissement de la tendance baissière et une consolidation de l'inertie. De plus, après cela, le changement de la valeur de l'inertie n'est pas significatif. Soit, dans notre cas, on va choisir un nombre K de clusters entre 40 et 60, ce qui valide notre nombre de topics.

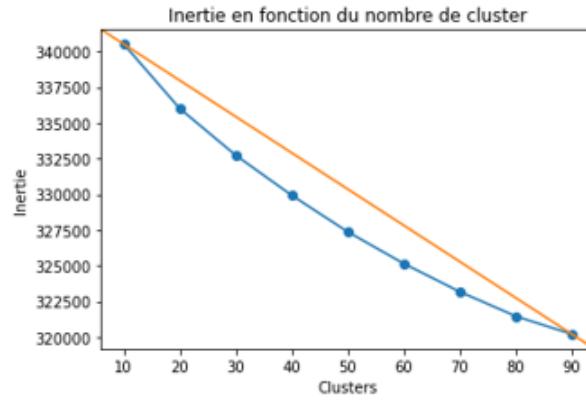


Figure 138

### Code KMeans

```
#Affiche un graphique nous montrant l'inertie en fonction du nombre de cluster
def cluster_optimal(data, max_k):
    iters = range(10, max_k, 10)
    inertie = []
    for k in iters:
        inertie.append(KMeans(n_clusters=k, random_state=20).fit(data).inertia_)
        print('Fit {} clusters'.format(k))

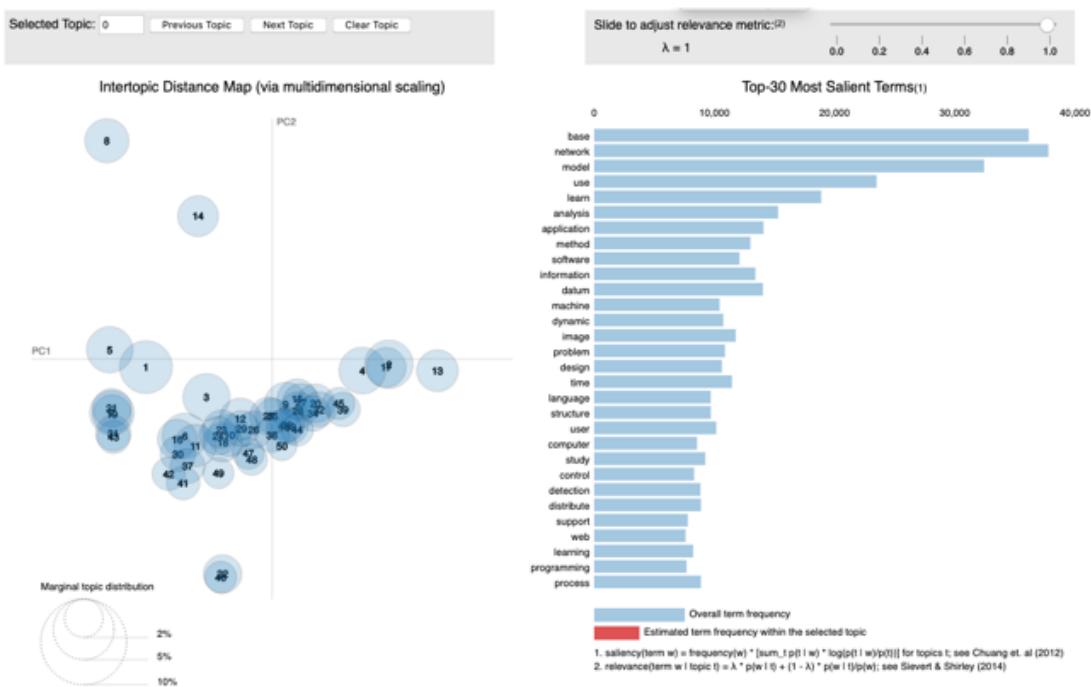
    f, ax = plt.subplots(1, 1)
    ax.plot(iters, inertie, marker='o')
    ax.plot([1, 90], [0, 1], transform=ax.transAxes)
    ax.set_xlabel('Clusters')
    ax.set_xticks(iters)
    ax.set_xticklabels(iters)
    ax.set_ylabel('Inertie')
    ax.set_title('Inertie en fonction du nombre de cluster')
```

Figure 139

Interprétation du code : On observe l'inertie du modèle pour un nombre de clusters compris entre 10 et max\_k. Une fois terminé, cela affiche le graphique ci-dessus.

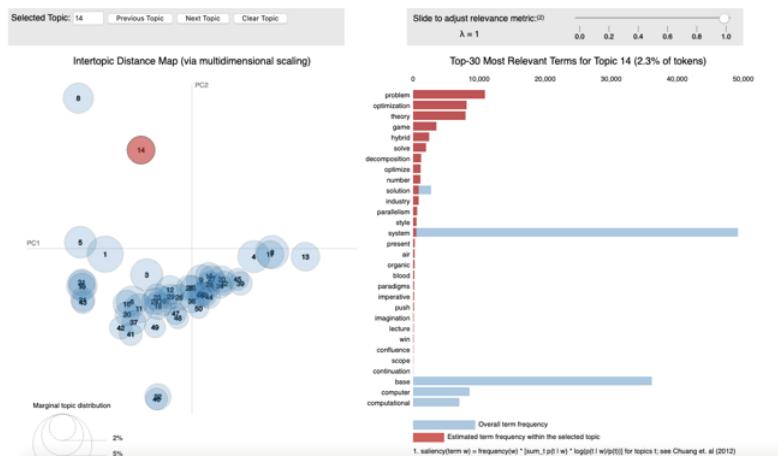
Key	Type	Size	Value
0	list	10	['base', 'intelligence', 'system', 'artificial', 'global', 'impact', '...']
1	list	10	['analysis', 'domain', 'practical', 'share', 'key', 'empirical', 'comp ...']
2	list	10	['model', 'process', 'estimation', 'business', 'variable', 'fundamenta ...']
3	list	10	['framework', 'resource', 'integration', 'allocation', 'library', 'con ...']
4	list	10	['structure', 'generation', 'text', 'formal', 'secure', 'effective', '...']
5	list	10	['knowledge', 'parallel', 'automatic', 'internet', 'order', 'heterogen ...']
6	list	10	['application', 'design', 'system', 'engineering', 'identification', '...']
7	list	10	['control', 'vision', 'cognitive', 'map', 'reasoning', 'measurement', ...']
8	list	10	['machine', 'planning', 'inference', 'shape', 'active', 'surface', 'li ...']
9	list	10	['detection', 'security', 'simulation', 'tool', 'drive', 'view', 'syst ...']
10	list	10	['orient', 'spatial', 'statistical', 'tolerant', 'health', 'translatio ...']

Figure 140. Liste des mots contenus dans les différents topics



**Figure 141.** Visualisation des résultats sous pyLDAvis

**Modèle Final** Comme on peut le voir sur la capture d'écran de pyLDAvis (package de visualisation de LDA), les topics sont assez regroupés, ce qui dénote d'une certaine proximité dans les thèmes abordés du corpus.



**Figure 142.** Visualisation des résultats sous pyLDAvis, lorsque l'on sélectionne le topic 14

On peut prendre par exemple le topic n°14. Le rouge représente l'occurrence des mots appartenant au topic et le bleu, au corpus. Le n°14 a comme principaux mots, « problem », « optimization » and « theory ». On remarque aussi que le mot « system » a peu d'occurrence dans le topic contrairement à dans le corpus.

Maintenant, si on se concentre sur les topics du milieu, on remarque que l'analyse en composante principale (PCA) montre une corrélation et ce qui inclue une certaine redondance entre les mots. Un nombre de topics inférieur serait requis sans prendre en compte la valeur de la cohérence.

## Automatisation des thèmes des topics

**Presentation Wordnet** WordNet est une base de données lexicale développée par le laboratoire des sciences cognitives de l'université de Princeton. Son but est de répertorier, classifier et mettre en relation de diverses manières le contenu sémantique et lexical de la langue anglaise. WordNet existe aussi dans d'autres langues mais de manière moins complet. WordNet est téléchargeable gratuitement et sa structure en fait un outil utile pour la linguistique informatique et le traitement du langage naturel.

**Synsets** Le système Wordnet se repose sur l'utilisation de « synset ». Ce sont des groupes de mots (nom / verbes / adjectifs / adverbes) étant des synonymes cognitifs, chacun exprimant un concept distinct.

Exemple : La version 1.7 de WordNet définit ainsi le nom commun anglais « car » à l'aide de cinq synsets :

1. car, auto, automobile, machine, motorcar – (4-wheeled motor vehicle; usually propelled by an internal combustion engine; he needs a car to get to work)
2. car, railcar, railway car, railroad car – (a wheeled vehicle adapted to the rails of railroad; three cars had jumped the rails)
3. car, gondola – (car suspended from an airship and carrying personnel and cargo and power plant)
4. car, elevator car – (where passengers ride up and down; the car was on the top floor)
5. cable car, car – (a conveyance for passengers or freight on a cable railway; they took a cable car to the top of the mountain)

Chaque synset contient un sens différent du mot car, décrit par une courte définition.

Mais les synsets permettent bien plus : ils peuvent représenter des concepts plus complexes, que l'on peut regrouper sous forme d'ontologies (système de catégories permettant de classifier les éléments d'un univers linguistique telles que les mots, les sens ou bien les concepts). Afin que cela se fasse de manière cohérente, on se sert des relations sémantiques.

Pour résumer, WordNet est composé de relations sémantiques permettant d'organiser le sens des mots (et donc les mots en eux-mêmes) en des systèmes de catégories facilement consultable. Il existe de nombreuses relations tel que l'hyperonymie, l'hyponymie, la synonymie ou encore l'antonymie mais pour notre projet, nous n'utiliserons que l'hyperonymie.

**Hyperonymies** Le principe d'hyperonymie est simple : à partir du sens le plus commun d'un mot, la relation d'hyperonymie définit un arbre de concepts de plus en plus généraux : Par exemple, si on étudie les hyperonymes du mot car, on obtient l'arbre suivant :

- ```
1. car, auto, automobile, machine, motorcar
2.   a. motor vehicle, automotive vehicle
3.     i. vehicle
4.       1. conveyance, transport
5.         a. instrumentality, instrumentation
6.           i. artifact, artefact
7.             1. object, physical object
8.               a. entity, something
```

Le dernier concept représenté par « entity, something » est le concept le plus général et abstrait possible.

Dans notre cas, afin d'obtenir les mots qui représentent le mieux de manière général chacun de nos topics, il suffit de compter les hyperonymes qui apparaissent le plus souvent.

### Code

```
publication_author = pd.read_csv("publication_author_filtre.csv")

#Pour comprendre comment ça marche
exemple = corpus[20] #document n°20
vector = lda_model[exemple]
print(vector)

#Préparation
data_pub = keyword_conf.copy()
data_pub["id_publication"] = data_pub.index
data_pub = data_pub.reset_index(drop=True)
data_pub["corpus"] = corpus

data = publication_author.merge(data_pub, how='right')
data.head()

data = data.dropna().reset_index(drop=True)
corpus2 = data["corpus"].values.tolist()

data_test = data.copy()
len(data_test)
l_topic_name = list(range(n_topics))
list_proba = []
for j in range(len(data_test)):
    list_proba_pub = [0 for i in range(n_topics)]
    vector = lda_model[corpus2[j]]
    for i in range(len(vector)):
        list_proba_pub[vector[i][0]] = vector[i][1]
    list_proba.append(list_proba_pub)

print(list_proba)

df = pd.DataFrame(list_proba, columns=l_topic_name)
df1 = data_test.join(df, how="outer")
df1 = pd.concat([data_test, df1], axis=1)

data_groupby_author = df1.groupby('id_author')[l_topic_name].mean()
data_groupby_author["num_topic"] = data_groupby_author.idxmax(axis=1)

#Pour enregistrer les données sous csv
#data_groupby_author.to_csv("Donnée_model_topic_plus_present.csv")
```

Figure 143

```
#On récupère les n_mot de chaque topic
n_mot = 10
mot_topic = {}
for j in range(0,n_topics):
    liste_mot1 = []
    for i in range (0,n_mot):
        liste_mot1.append(lda_model.show_topic(j,n_mot)[i][0])
    mot_topic[j]=liste_mot1

#On parcours les hyperonymes de chaque mots et on les récupère dans un dico
dico = {}
n=1
for topic in mot_topic.values():
    dictionnaire_hyp = {}
    for mot in topic:
        if mot in wn_lemmas:
            syn = wordnet.synsets(mot)[0]
            #pour les hyperonymes du mot
            theme = syn.hypernyms()
            for i in range(len(theme)):
                if (theme[i] in dictionnaire_hyp):
                    dictionnaire_hyp[theme[i]] += 1
                else:
                    dictionnaire_hyp[theme[i]] = 1
            #Pour les hyperonymes d'hyperonymes
            theme_hyp = theme[i].hypernyms()
            for hyp in range(len(theme_hyp)):
                if (theme_hyp[hyp] in dictionnaire_hyp):
                    dictionnaire_hyp[theme_hyp[hyp]] += 1
                else:
                    dictionnaire_hyp[theme_hyp[hyp]] = 1

    #On ne prend que les hyperonymes les plus présents
    dico[n],partition1,partition2 = max(dictionnaire_hyp.items(), key = lambda x: x[1])[0].name().partition(".")
    #partition1,partition2 inutile car juste la pour récup les autres sortie de fonction partition et que code marche
    n = n + 1
```

Figure 144

Interprétation du code : Tout d'abord, on va récupérer les thèmes les plus présents pour

chaque auteur (on fait une moyenne des probabilités d'appartenance à chaque topic et on choisit celle maximale). Ensuite, on récupère les 10 mots les plus récurrents de chaque topic. Puis, avec l'aide de la base de données « WordNet », on va compter combien de fois un hyperonyme d'un mot apparaît, pour plus de chance de succès, on va aussi regarder les hyperonymes d'hyperonymes. Au final, on ne garde que l'hyperonyme le plus présent pour chaque topic et le considère comme le thème du topic.

Etant donné que cette partie est en lien avec les graphes de communauté auteurs – co-auteurs, le LDA a été relancé mais que sur les 7000 premières publications, échantillon utilisé pour faire le graphe. Le nombre de topics a forcément lui aussi changé, on est passé sur 20 topics. Afin de déterminer le nombre de topic, on utilise la même méthode que précédemment soit avec la cohérence :

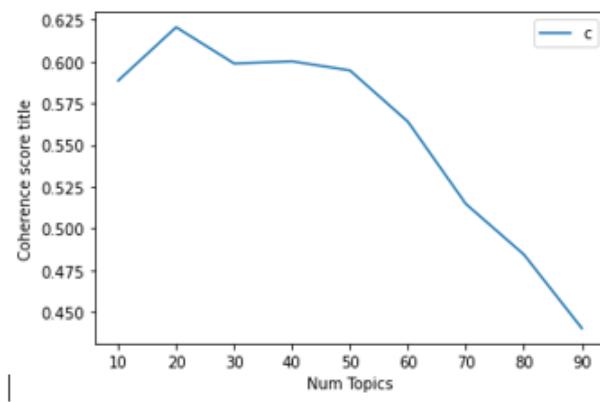


Figure 145

On obtient donc ce résultat, que l'on va associer à chaque auteur en fonction de son topic favoris :

```
{1: 'enterprise', 2: 'collection', 3: 'philosophy', 4: 'entity', 5: 'creating_by_mental Acts', 6: 'abstraction', 7: 'system',
8: 'improvement', 9: 'information', 10: 'device', 11: 'code', 12: 'information', 13: 'quality', 14: 'message', 15: 'safety',
16: 'act', 17: 'activity', 18: 'investigation', 19: 'instrumentality', 20: 'difficulty'}
```

Figure 146

Ces résultats sont enregistrés dans le fichier Donnee\_model\_nom\_cluster.csv.

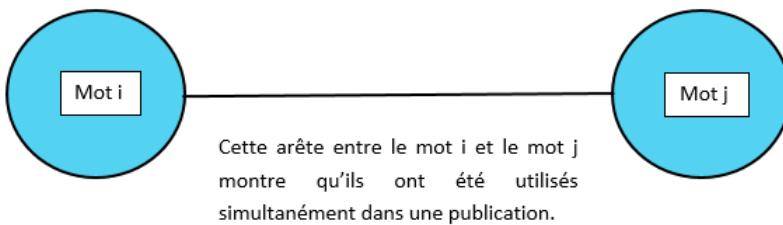
**Conclusion** Notre objectif était d'obtenir des topics à partir des mots clé des titres des publications via un LDA, notre objectif est atteint. On remarquera tout de même que malgré un choix des topics via la cohérence, la visualisation des topics de LDA est assez condensé. Il se pourrait qu'avec une plus grande expérience dans le monde du topic-modeling, on puisse paramétriser le LDA en fonction des poids observés dans les topics et donc amélioré l'indépendance des topics.

#### 6.4.4.2 Graphe

**Introduction** Dans cette deuxième sous-partie, notre objectif était le même que précédemment c'est-à-dire la création de groupe. Pour être davantage précis, nous voulions pouvoir attribuer des groupes de mots clés aux auteurs pour pouvoir faire des analyses fondées sur cette nouvelle information.

Ici l'objectif était de trouver des groupes de mots par le moyen de graphe, ces graphes ont été réalisés avec les fichiers keyword et publication\_keyword\_filtre\_random\_stopword.

Nous avons donc défini des sommets ainsi que des arêtes pour mettre en relation tous les mots clés. Comme vous pouvez le voir dans le schéma ci-dessous, l'ensemble des sommets correspond à la totalité des mots clés et on paramètre une arête en deux mots clés s'ils ont été utilisés dans une même publication.



**Figure 147.** Exemple de graphe

De plus nous avons insérer à cette partie une notion de pondération des arêtes permettant de mesurer l'importance des liaisons entre les sommets de nos graphes.

Pour réaliser notre objectif on a séparé le travail en deux parties, premièrement la conception d'un fichier Excel permettant d'obtenir les liaisons entre l'ensemble des mots clés, deuxièmement la conception des graphes.

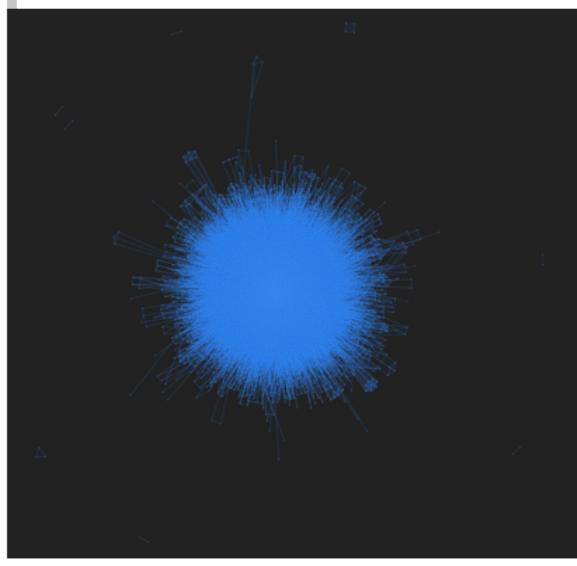
**Liaisons entre les mots clés** Étant donné que les fichiers originaux n'étaient pas adaptés à la création de graphe nous avons dû effectuer certaines manipulations au niveau des données pour obtenir quelque chose correspondant à nos besoins.

Premièrement, nous nous sommes orientés vers le package iGraph qui nécessitait une matrice d'adjacence en entrée pour les fonctions graphiques. À la suite de quelques tests, nous avons vite observé que ce package ne répondait pas à nos besoins, pour être plus précis nous avions besoin de graphes dynamiques ce qui n'étaient pas possible avec ce package.

C'est pour cela que nous nous sommes orientés vers un autre package nommé pyvis qui lui permet de créer des graphes dynamiques (possibilité de zoom, information sur les sommets,...). Le problème était que ce nouveau package n'accepte pas les matrices d'adjacence en données d'entrée. Pour pallier cela nous avons donc créer un fichier Excel contenant 3 variables (keyword1, keyword2 et nb\_lien) qui correspond donc au package.

Étant donné que la complexité temporelle de la création de ce fichier est grande nous avons décidé de travailler sur un échantillon de données relativement petit. Nous avons donc sélectionné les 20 000 premières lignes du fichier publication\_keyword\_filtre\_random\_stopword pour au final obtenir un fichier Excel avec 32 372 lignes ce qui représente le nombre total de liaisons entre les mots clés.

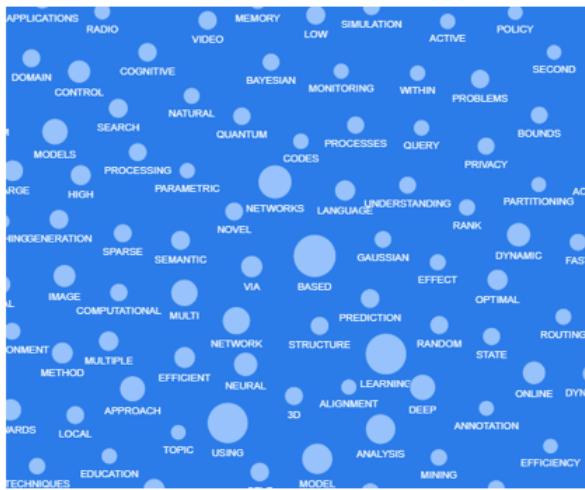
**Conception des graphes** Pour revenir à notre objectif, l'idée était d'obtenir des groupes de mots clés pour ensuite affecter des auteurs à ces groupes en fonction des mots clés qu'ils utilisent. Dans un premier temps nous avons voulu observer si avec nos données on pouvait constater des groupes plus ou moins distincts de mots clés. Pour cela on a représenté les liaisons entre les données par le biais d'un graphe :



**Figure 148.** Graphe mots clés sans zoom

Dans ce premier graphe, nous pouvons observer plusieurs phénomènes, premièrement on constate un énorme groupe de mots clés central. Puis on distingue également quelques petits groupes annexes composés de mots spéciaux comme par exemple le couple de mots [buys, retrospective, loser, troublesome]. Ce graphe permet également de voir le nombre important de liaisons entre la quasi-totalité des mots clés sélectionnés pour ce graphe.

Pour entrer plus en détails dans l'analyse de ce graphe, vous pouvez observer dans le graphe ci-dessous un zoom élevé sur le centre du graphe.



**Figure 149.** Graphe mots clés avec zoom sur le centre du graphe

Dans ce graphe, nous pouvons premièrement observer les principaux mots clés comme Learning, Based ou encore Using, ces mots sont ceux qui ont le plus de liaisons avec les autres mots. On peut donc conclure que ce sont les mots centraux de notre graphe. Deuxièmement, les arêtes entre les mots ne se distinguent plus, cela est dû à la masse très importante de liaisons entre les mots ce qui rend impossible la détection de groupes de mots clés. Pour rappel, nous travaillons ici sur un échantillon de taille fortement réduite ce qui laisse penser qu'avec la totalité des données ce phénomène sera amplifié.

A ce niveau-là de notre étude sur les mots clés, il est donc difficile de répondre à notre objectif. Pour essayer de pallier ce problème, nous avons décidé dans un premier temps d'observer plus précisément notre fichier edge contenant la totalité des liaisons entre les mots clés.

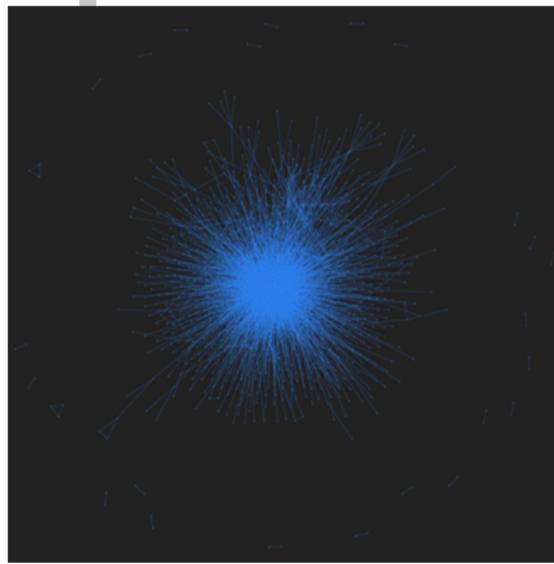
|          |                        |
|----------|------------------------|
| In [56]: | weights.value_counts() |
| Out[56]: |                        |
| 1.0      | 56639                  |
| 2.0      | 3556                   |
| 3.0      | 845                    |
| 4.0      | 316                    |
| 5.0      | 161                    |
| 6.0      | 59                     |
| 7.0      | 38                     |
| 8.0      | 21                     |
| 11.0     | 15                     |
| 10.0     | 12                     |
| 9.0      | 10                     |
| 13.0     | 8                      |
| 17.0     | 6                      |
| 12.0     | 5                      |
| 19.0     | 4                      |
| 14.0     | 4                      |
| 15.0     | 3                      |
| 16.0     | 3                      |
| 20.0     | 2                      |
| 30.0     | 2                      |
| 61.0     | 1                      |
| 25.0     | 1                      |
| 41.0     | 1                      |
| 21.0     | 1                      |
| 23.0     | 1                      |

Name: nb\_lien, dtype: int64

**Figure 150.** Répartition des pondérations des arcs du fichier edge

Comme vous pouvez le voir ci-dessus, sur 61 714 arêtes il y en a 56 639 qui ont un poids de 1 ce qui correspond à dire que 56 639 connexions entre 2 mots ont été observé uniquement dans une publication. Nous nous sommes donc posé la question : à partir de quel poids peut-on dire qu'une liaison entre 2 mots est pertinente ?

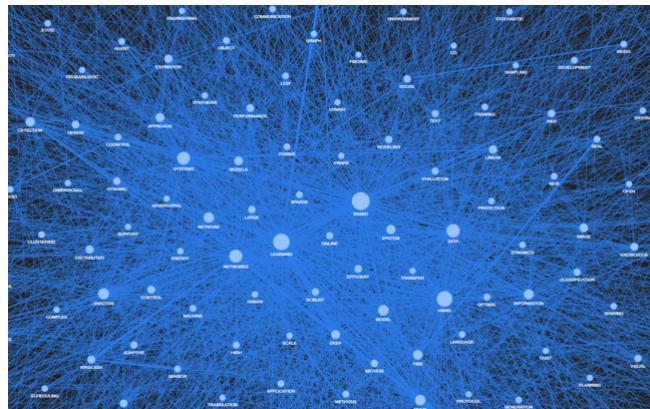
Pour répondre à cette question, nous avons donc choisis de filtrer les données selon leurs pondérations pour ensuite pouvoir générer de nouveaux graphes. Pour le deuxième graphe, on a donc choisi d'éliminer toutes les arêtes avec un poids de 1 ce qui représente plus de 90% des données.



**Figure 151.** Graphe mots clés sans les arêtes avec une pondération égale à 1(sans zoom)

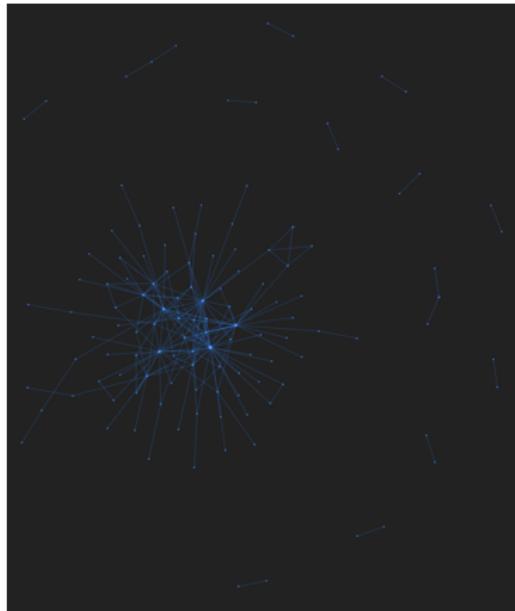
Dans le graphe ci-dessus qui est obtenu en filtrant les arêtes, on constate plus de petits « groupes » de mots, ils sont composés de 2 ou 3 mots ce qui n'est pas nécessaire pour les considérés comme des groupes à part entière dans le cadre de notre objectif principal. Globalement on peut voir que le graphe à l'air moins dense, ce qui est normal au vu de la

suppression d'environ 90% des arêtes. En faisant un zoom sur ce graphe, on arrive maintenant à distinguer les arêtes dont la largeur varie en fonction de la pondération, mais au-delà de ça, on remarque que notre premier problème qui est le nombre important de connexion entre les arêtes est toujours présente.



**Figure 152.** Graphe mots clés sans les arêtes ayant une pondération égale à 1 (avec zoom)

Pour aller encore plus loin dans la suppression d'arête, nous avons essayé d'enlever les liaisons ayant une pondération inférieure ou égale à 5 pour observer uniquement les liaisons conséquentes entre les mots clés.



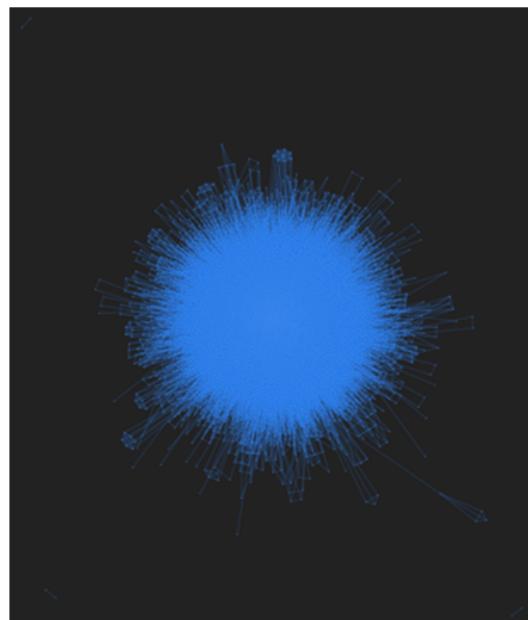
**Figure 153.** Graphe mots clés sans les arêtes avec une pondération inférieure ou égale à 5

Le graphe présent ci-dessus devient beaucoup moins dense, cela vient du fait que nous avons plus que 197 arêtes ce qui représente moins de 1% de nos données initiales. Avec ce graphe nous perdons beaucoup d'informations ce qui aurait posé un problème plus tard lorsque nous aurions élaborés les groupes de mots. De plus, nous ne constatons toujours pas de groupes de mots.

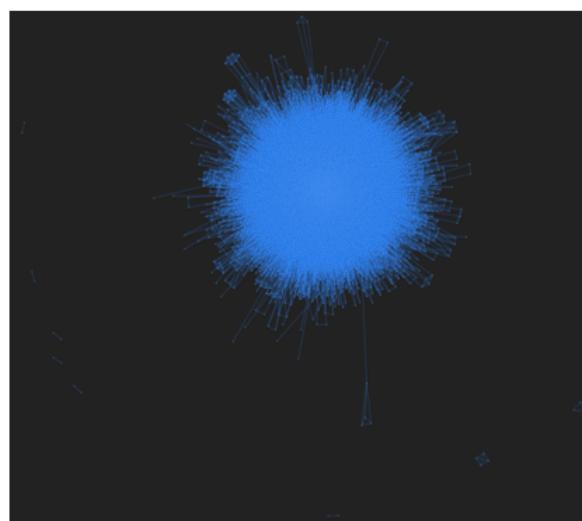
Comme vous avez pu le voir précédemment, lorsque nous avons filtré les arêtes nous

avons supprimé celles qui ont un faible poids. À la suite de nos résultats, non pertinents vis-à-vis de notre objectif, nous avons voulu essayer de faire l'inverse. C'est-à-dire de supprimer les arêtes avec un poids trop élevé pour essayer d'enlever les mots les plus forts du graphe. Nous avons donc fait 2 graphes, le premier en gardant uniquement les arêtes avec une pondération inférieure ou égale à 5 (Graphe 1) ce qui conserve 99% des données ; le deuxième en gardant uniquement les arêtes avec un poids de 1 (Graphe 2) ce qui donne un graphe avec plus de 90% des données.

Dans ces 2 graphes, nous supprimons peu de données, donc la densité des connexions n'est pas réduite, les résultats ressemblent au premier graphe avec toutes les données, donc malheureusement, cette méthode dans laquelle on enlève les mots fortement connectés ne répond pas à notre objectif.



**Figure 154.** Graphe 1 : Graphe des mots clés sans les arêtes avec un poids supérieur ou égale à 5



**Figure 155.** Graphe 2 : Graphe des mots clefs sans les arêtes avec un poids supérieur à 1

**Conclusion** A ce stade de notre étude sur les mots clés, il est donc impossible de répondre à notre objectif qui était de faire des groupes de mots via les graphs. Cette conclusion plutôt négative est dû à la masse importante de liaisons entre les mots qui rend compliqué l'obtention de groupe. Les résultats de notre seconde approche ne nous permettent pas de conclure sur notre objectif car en filtrant les arêtes selon leurs pondérations nous perdons une majeure partie des données ce qui pose un problème dans le cadre de notre objectif. La troisième approche n'est toujours pas pertinente car la densité des graphes reste toujours fortement élevée.

#### 6.4.5 Analyse des thèmes de chaque auteur

**6.4.5.1 Importation des données** Tout d'abord pour créer notre graphe avec les colorations selon les thèmes des auteurs, nous avons besoin du fichier Donnee\_model\_nom\_cluster.csv. Nous avons sélectionné seulement les colonnes si nous intéressaient, à savoir, l'identifiant des auteurs (id\_author), le numéro de leur topic le plus représentatif (num\_topic) et le nom du thème auxquelles ils appartiennent (theme\_topic).

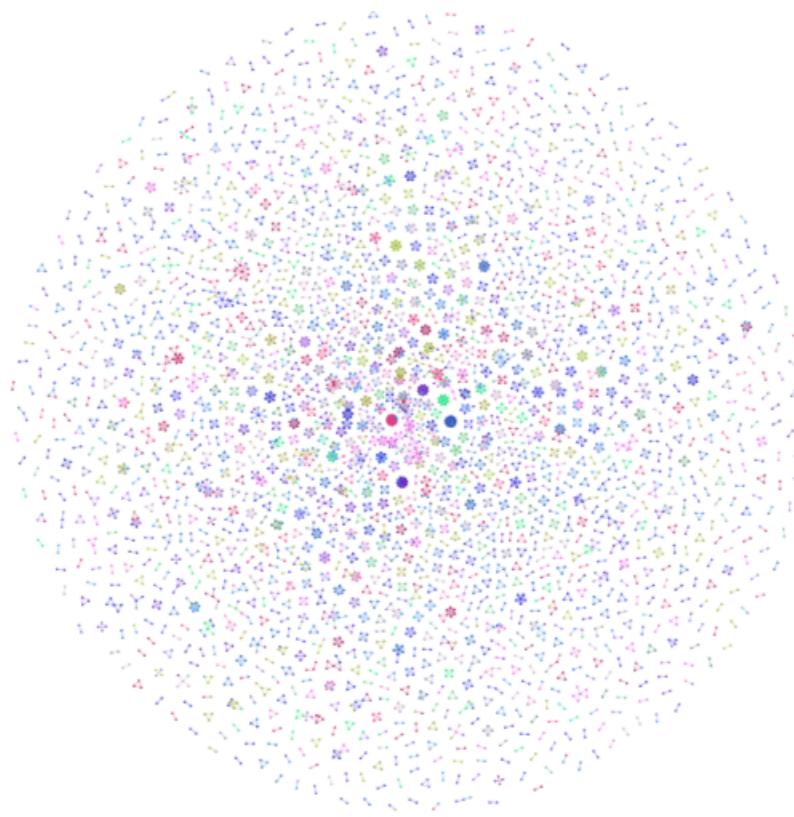
Nous avons aussi utilisé le fichier edge.csv qui contient les liens auteurs - co-auteurs en fonction des articles qu'ils ont publiés.

Et enfin le fichier publication\_chaque\_auteur.csv qui contient le titre de toutes les publications de chaque auteur.

**6.4.5.2 Graphe dynamique coloré selon les thèmes des auteurs** Une fois nos données importées, nous avons coloré les sommets de notre graphe, c'est-à-dire les auteurs, selon le numéro de leur topic le plus représentatif.

Enfin nous ajoutons aux sommets la labélisation de leur thème et la liste des titres des publications de l'auteur en question.

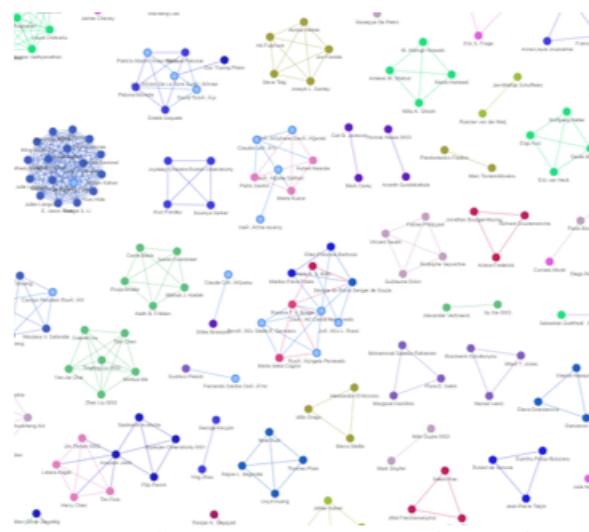
Voici le graphe des auteurs colorés en fonction de leurs thèmes :



**Figure 156.** Graphe des communautés auteurs – co-auteurs avec les sommets colorés selon leur thème

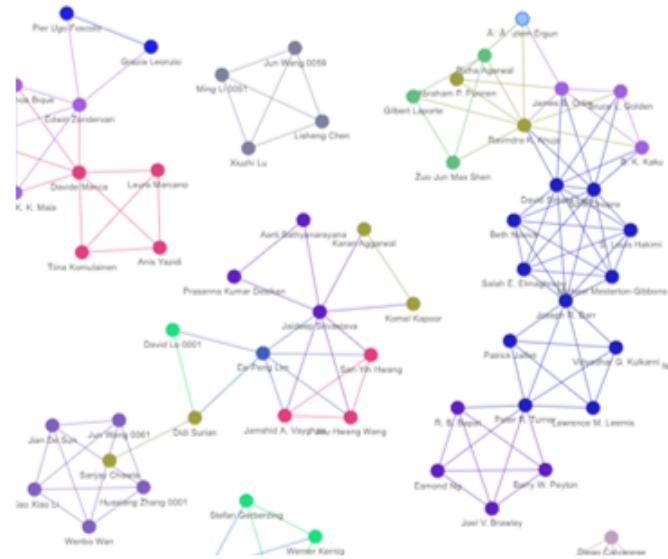
En s'approchant d'un peu plus près, il est possible d'observer des groupes d'auteurs qui ont travaillé ensemble, et qui ont pour la majorité des cas, on travaillé sur le même thème. Au total, il y a une vingtaine de thèmes différents.

On remarque que parfois certains auteurs n'ont pas de thème (couleur bleu clair des sommets), cela est dû à des problèmes lors de la jointure des données, mais le résultat est globalement satisfaisant.



**Figure 157.** Graphe des communautés auteurs – co-auteurs avec les sommets colorés selon leur thème avec un zoom sur une partie du graphe

Il est également possible de voir que certains groupes d'auteurs ont des thèmes différents. La coloration de leur thème permet de mettre en évidence les différents domaines de recherches des auteurs ponts.



**Figure 158.** Graphe des communautés auteurs – co-auteurs avec les sommets colorés selon leur thème avec un zoom sur une autre partie du graphe

#### 6.4.6 Conclusion

Nos objectifs d'analyse étaient de détecté des auteurs centraux ainsi que leurs communautés pour pouvoir étudier les équipes qui pouvaient se former et étudier les différentes relations qu'il peut y avoir entre nos auteurs. Nous avons également tenu à trouver les thèmes de prédilection des auteurs en fonction des titres de leurs publications.

En conclusion, nous avons réussi à trouver nos auteurs centraux et nos communautés d'auteurs. Nous avons également réussi à répondre à nos interrogations consternant l'organisation de travail de nos auteurs.

De plus, nous avons également put connaitre leur thème de recherche. Toutefois, la labélisation des topics (donc des thèmes) pourrait être améliorée.

## 7 Intégration

### 7.1 Gestion de Projet

Comme la librairie utilisée par le groupe Interface est différente des librairies que les autres groupes ont utilisé, nous nous sommes tous mis d'accord pour que chaque groupe décide de son interface, mais que ce soit le groupe Interface qui les implémente.

En effet, s'étant formée à cette librairie durant toute la durée du projet contrairement aux membres des autres groupes, cela semblait être un gain de temps et d'énergie de procéder ainsi.

Dans une logique similaire, comme c'est le groupe Interface qui a développé le code du produit final, il l'a organisé à sa façon, et était à l'aise avec la structure du code et sa répartition en différents fichiers.

On a donc décidé avec les autres groupes que chacun devrait préparer son code, et que ce serait au groupe Interface de lier le code à l'interface.

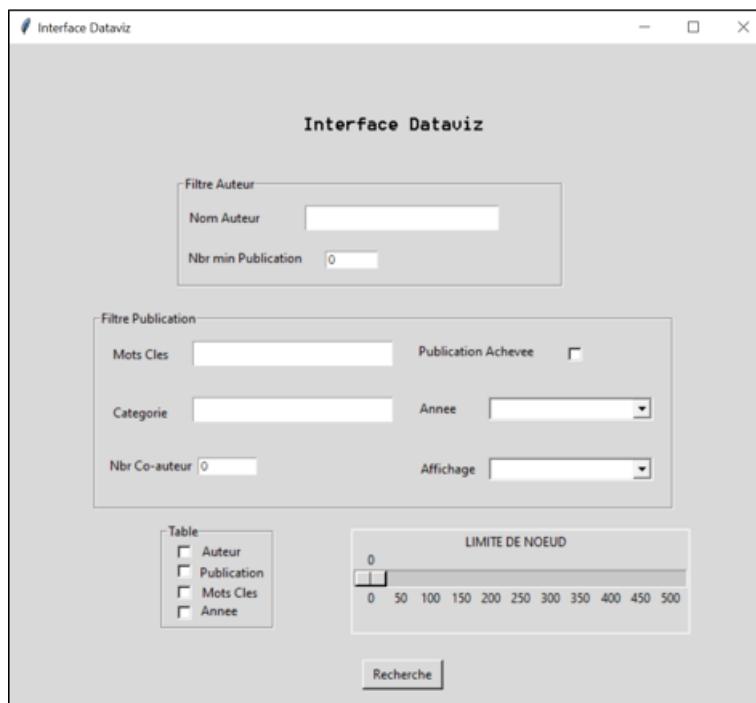
C'est pour ces raisons qu'il n'y a pas eu de gestion de projet à part entière dédiée à l'intégration, mais que cela s'est organisé au sein de chaque groupe, en influençant juste la façon de faire de chacun sans aller jusqu'à ajouter de nouvelles tâches.

### 7.2 Projet Intégré

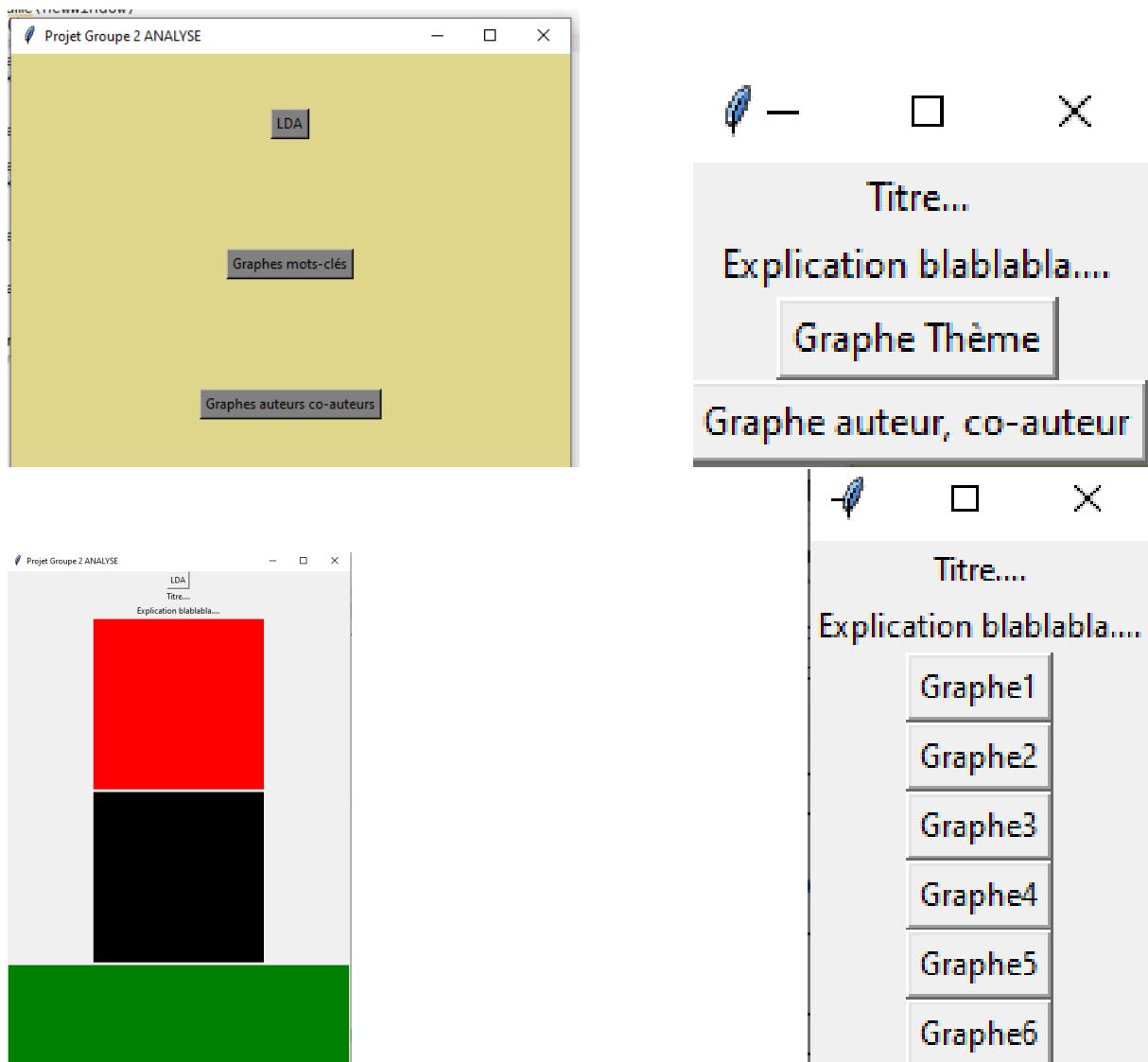
#### 7.2.1 Intégration de l'interface

Comme expliqué précédemment, chaque groupe a imaginé son interface, et a fait ses maquettes de son côté. Cela s'est fait assez naturellement, puisque pour tester les fonctionnalités développées par chacun, une interface était nécessaire.

Chaque groupe a donc développé son interface pour les tests en interne, puis a envoyé des captures d'écran des différentes fenêtres au groupe Interface afin qu'il puisse les développer dans la librairie utilisée pour le produit final.



**Figure 160.** Maquette du groupe Data Visualisation 1



**Figure 159.** Maquettes du groupe Analyse

### 7.2.2 Intégration des fonctionnalités

Comme également expliqué précédemment, une fois que chaque groupe a développé ses outils, il a préparé son code afin qu'il soit exécutable facilement via l'appel à une ou plusieurs fonctions.

Ensuite, le groupe Interface a récupéré ces fonctions, et a lié les boutons correspondants de l'interface sur les outils de chaque groupe, afin que le produit final permette de lancer les analyses et les visualisation de chaque groupe.

## Références

- [1] Documentation auto-py-to-exe. url : <https://pypi.org/project/auto-py-to-exe>.
- [2] Documentation GZIP. url : <https://docs.python.org/3/library/gzip.html>.
- [3] Documentation JSON. url : <https://docs.python.org/fr/3/library/json.html>.
- [4] Documentation LXML. url : <https://lxml.de>.
- [5] Documentation TKinter. url : <https://docs.python.org/fr/3/library/tkinter.html>.
- [6] Documentation URLLIB. url : <https://docs.python.org/fr/3/library/urllib.html>.
- [7] Documentation XMLTODICT. url : <https://github.com/martinblech/xmltodict>.
- [8] Lien de téléchargement d'Anaconda. url : <https://www.anaconda.com/products/individual>.
- [9] palette de couleur. url : <https://colors.muz.li/palette/101f43/dde3f2/88294d/8f94a4/c09a6f>.
- [10] Site internet dblp. url : [www dblp org](http://www dblp org).
- [11] Site internet MockFlow. url : <https://www.mockflow.com>.
- [12] Site internet QT. url : <https://www.qt.io>.
- [13] Site internet Trello. url : <https://trello.com>.
- [14] Wave PNG. url : <https://s.muz.li/ODM2MGYyMzQz>.