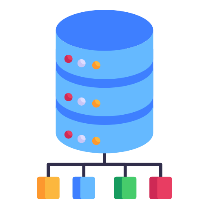
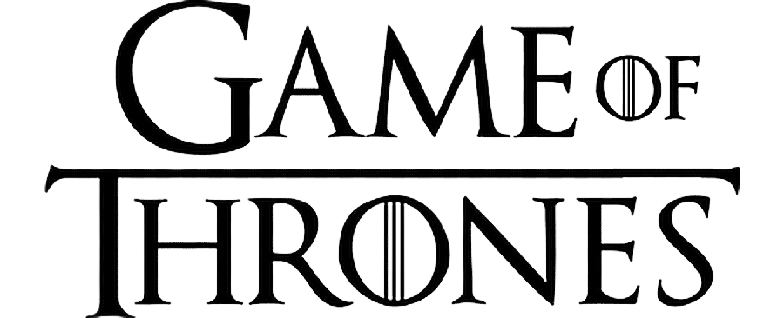


Université Chouaib Doukkali

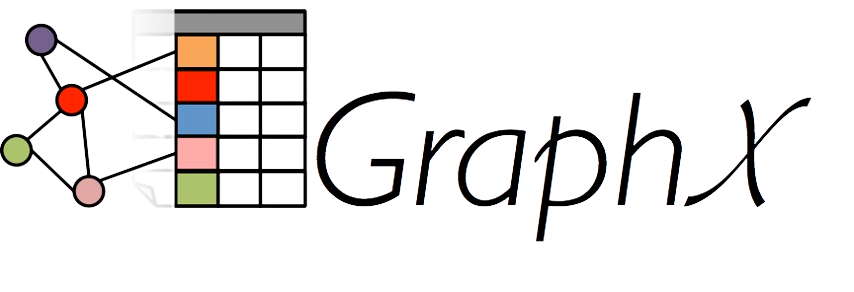
Ecole Nationale des Sciences Appliquées d’El Jadida

Département Télécommunications, Réseaux et Informatique





Analyse du réseau de Game of Thrones en utilisant Apache GraphX, Neo4j et Spark ML



Réalisé Par :

**Elhadi Refki**

**Reda Makaoui**

**Oussama Ouhayou**

Encadré Par :

**Fahd Kalloubi**

# Table des Figures

[Figure 1: Dataset Kaggle 5](#_Toc154161897)

[Figure 2: Architecture du projet 6](#_Toc154161898)

[Figure 3 docker-compose.yml file 8](#_Toc154161899)

[Figure 4 le cluster docker 8](#_Toc154161900)

[Figure 5 version de spark dans zeppelin 8](#_Toc154161901)

[Figure 6 fichier JAR de connexion de neo4j et spark 9](#_Toc154161902)

[Figure 7 importation du jar dans zeppelin 9](#_Toc154161903)

[Figure 8 configuration du JAR 10](#_Toc154161904)

[Figure 9 les notebook du Projet 10](#_Toc154161905)

[Figure 10 code de concatenation des fichier CSV 11](#_Toc154161906)

[Figure 11 stockage dans un nouveau fichier result.csv 12](#_Toc154161907)

[Figure 12 insertion des données 12](#_Toc154161908)

[Figure 13 visualisation 13](#_Toc154161909)

[Figure 14 configuration de l'interpréteur Neo4j 13](file:///C:\Users\reda\Downloads\GuideProjet.docx#_Toc154161910)

[Figure 15 connecter l'interpreteur avec le contenuer neo4j 14](file:///C:\Users\reda\Downloads\GuideProjet.docx#_Toc154161911)

[Figure 16 visualisation dans zeppelin 14](#_Toc154161912)

[Figure 17 lire des données de neo4j dans zeppelin pour les manipuler 15](#_Toc154161913)

[Figure 18 créer un graph à partir des données 15](#_Toc154161914)

[Figure 19 visualiser les données en utilisant les CYPHER query 16](#_Toc154161915)

[Figure 20 visualiser les données en utilisant SPARK 16](#_Toc154161916)

[Figure 21 les nœuds du premier livre 17](#_Toc154161917)

[Figure 22 les nœuds du 2eme livre 17](#_Toc154161918)

[Figure 23 les nœuds du 3eme livre 17](#_Toc154161919)

[Figure 24 les nœuds du 4eme livre 18](#_Toc154161920)

[Figure 25 les nœuds du dernier livre 18](#_Toc154161921)

[Figure 26 les personnages qui sont dans tous les livres 19](#_Toc154161922)

[Figure 27 la personne la plus présente dans la série 19](#_Toc154161923)

[Figure 28 ajouter un noeud 20](#_Toc154161924)

[Figure 29 verifier l'ajout 20](#_Toc154161925)

[Figure 30 modification des données 21](#_Toc154161926)

[Figure 31 vérification de la modification 21](#_Toc154161927)

[Figure 32 la suppression des données 22](#_Toc154161928)

[Figure 33 Récupération des données 22](#_Toc154161929)

[Figure 34 créer un Graphe à l'aide de graphX pour les données récupérés 23](#_Toc154161930)

[Figure 35 Affichage des (vertices et Edges) 23](#_Toc154161931)

[Figure 36 des statistiques sur le graph 24](#_Toc154161932)

[Figure 37 afficher les degrées des noeuds 24](#_Toc154161933)

[Figure 38 PageRank Algorithm 25](#_Toc154161934)

[Figure 39 resultat de l'algorithme 25](#_Toc154161935)

[Figure 40 Connected Components Algorithm 26](#_Toc154161936)

[Figure 41 label propagation algorithme 27](#_Toc154161937)

[Figure 42 triangle count algorithme 28](#_Toc154161938)

[Figure 43 strongly connected algorithme 29](#_Toc154161939)

[Figure 44 resultat de l'algorithme 29](#_Toc154161940)

[Figure 45 Page Rank algorithme 29](#_Toc154161941)

[Figure 46 connected algorithme 30](#_Toc154161942)

[Figure 47 Label Propagation Algorithm 30](#_Toc154161943)

[Figure 48 Triangle Count 31](#_Toc154161944)

[Figure 49 strongly Connected algorithme 31](#_Toc154161945)

[Figure 50 Dashboard 32](#_Toc154161946)

[Figure 51 récupération des données 33](#_Toc154161947)

[Figure 52 affichage des donées 33](#_Toc154161948)

[Figure 53 resultat 35](#_Toc154161949)

[Figure 54 Prediction du Degree\_Centrality 36](#_Toc154161950)

[Figure 55 resultat 36](#_Toc154161951)

[Figure 56 les métriques du model 36](#_Toc154161952)

[Figure 57 Prediction du PageRank 37](#_Toc154161953)

[Figure 58 les metriques du model 37](#_Toc154161954)

[Figure 59 Prediction du Betweenness\_Centrality 37](#_Toc154161955)

[Figure 60 resultat 38](#_Toc154161956)

[Figure 61 les metiriques du model 38](#_Toc154161957)

[Figure 62 Prediction du Weighted\_Degree 38](#_Toc154161958)

[Figure 63 resultat 38](#_Toc154161959)

[Figure 64 les métriques 38](#_Toc154161960)

[Figure 65 le clustring avec Kmeans 39](#_Toc154161961)

[Figure 66 affichage des clusters 39](#_Toc154161962)

**Table des matières**

[2 Introduction 4](#_Toc154161964)

[3 Objectif : 4](#_Toc154161965)

[4 Étapes du Projet : 4](#_Toc154161966)

[5 Dataset: 5](#_Toc154161967)

[6 Architecture : 6](#_Toc154161969)

[7 L’image Docker : 7](#_Toc154161970)

[8 Importation des JARs : 8](#_Toc154161971)

[9 Manipulation & Insertion des données: 10](#_Toc154161972)

[9.1 Récupération et Manipulation des données 11](#_Toc154161974)

[9.2 Insertion des données 12](#_Toc154161975)

[9.3 Vérification dans Neo4j 13](#_Toc154161976)

[10 CRUD : 13](#_Toc154161977)

[10.1 Récupération d’après Neo4j 13](#_Toc154161978)

[10.2 Exemples de filtrages 16](#_Toc154161979)

[10.3 ADD 20](#_Toc154161980)

[10.4 EDIT 21](#_Toc154161981)

[10.5 DELETE 22](#_Toc154161982)

[11 GraphX : 22](#_Toc154161983)

[11.1 Récupération des données et création du graph 22](#_Toc154161984)

[11.2 PageRank Algorithm 25](#_Toc154161985)

[11.3 Connected Components Algorithm 26](#_Toc154161986)

[11.4 Label Propagation Algorithm 26](#_Toc154161987)

[11.5 Triangle Count Algorithm 27](#_Toc154161988)

[11.6 Strongly Connected Algorithm 28](#_Toc154161989)

[12 Visualisation des résultats GraphX via tableau : 29](#_Toc154161990)

[13 Spark ML : 32](#_Toc154161991)

[13.1 Récupération des données 32](#_Toc154161992)

[13.2 Enrichir le dataset 33](#_Toc154161993)

[13.3 Application de quelques algorithms de prédiction : 36](#_Toc154161994)

[a. Prediction du Degree\_Centrality : 36](#_Toc154161995)

[b. Prediction du PageRank : 37](#_Toc154161996)

[c. Prediction du Betweenness\_Centrality 37](#_Toc154161997)

[d. Prediction du Weighted\_Degree 38](#_Toc154161998)

[13.4 : Clustring du Weight (KMeans) 39](#_Toc154161999)

# Introduction

Ce projet s'attache à décortiquer le réseau complexe de Game of Thrones en utilisant des outils technologiques avancés tels qu'Apache GraphX, Neo4j, et Spark ML. En plongeant dans les relations entre les personnages, les maisons nobles, et les événements clés de la série, nous cherchons à élucider les dynamiques politiques et sociales qui façonnent l'intrigue. En combinant la puissance des graphes, des bases de données orientées graphe, et du machine learning, cette analyse promet une exploration approfondie du tissu relationnel unique de l'univers **de Game of Thrones**.

# Objectif :

Concevoir et mettre en œuvre une solution complète d'analyse de réseau social basée sur le dataset Game of Thrones en utilisant les technologies Spark GraphX, Neo4j, Apache Zepplin, Spark ML, et en intégrant des visualisations dans un tableau de bord personnalisable. L'objectif est de comprendre les relations entre les personnages de Game of Thrones, d'effectuer des analyses exploratoires, d'appliquer des algorithmes de graphes, et d'appliquer des modèles d'apprentissage automatique pour tirer des insights intéressants du dataset.

# Étapes du Projet :

**Lire le Contenu du Dataset :**

* Télécharger le dataset Game of Thrones depuis le lien Kaggle fourni.
* Explorer le dataset pour comprendre sa structure et son contenu.

**Connecter Spark GraphX avec Neo4j :**

* Suivre la documentation de Neo4j pour intégrer Spark GraphX avec Neo4j.

**Importer le Dataset sur Neo4j :**

* Utiliser Cypher Query Language pour importer les données du dataset Game of Thrones dans Neo4j.

**Analyse Exploratoire avec Apache Zepplin et GraphX :**

* Lire les données depuis Neo4j avec Apache Zepplin et GraphX.
* Effectuer une analyse exploratoire du réseau social (e.g., degré de centralité, communautés, etc.).

**Exécuter 5 Algorithmes de Graphes avec GraphX :**

* Choisir cinq algorithmes de graphes (par exemple, PageRank, Label Propagation, etc.).
* Appliquer ces algorithmes sur le réseau social Game of Thrones.
* Visualiser les résultats obtenus.

**Créer un Dashboard Personnalisable :**

* Utiliser un outil de tableau de bord (par exemple, Tableau, Power BI) pour créer un tableau de bord interactif.
* Intégrer des visualisations basées sur les analyses exploratoires et les résultats des algorithmes de graphes.

**Appliquer des Algorithmes ML avec Spark ML :**

* Utiliser Spark ML pour appliquer des algorithmes d'apprentissage automatique sur les données Game of Thrones (par exemple, classification des personnages en fonction de certains attributs).

# Dataset:

# 

Figure 1: Dataset Kaggle

<https://www.kaggle.com/code/mmmarchetti/game-of-thrones-network-analysis/input>

Cette dataset est utilisé pour réaliser une analyse de réseau sur les personnages de la série de livres "Game of Thrones" de George R.R. Martin. Le jeu de données représente un réseau ou un graphe où les personnages sont des nœuds, et les connexions entre eux (arêtes) représentent les co-occurrences dans un certain contexte dans le texte.

L'analyse de cette dataset permet plusieurs investigations :

* **Relations entre personnages :** Elle permet de comprendre les relations ou interactions entre les personnages en se basant sur leurs co-occurrences. Par exemple, des personnages souvent mentionnés ensemble peuvent signifier des relations, des alliances ou des conflits.
* **Importance des personnages :** En appliquant diverses mesures d'analyse de réseau telles que la centralité degré, la centralité d'intermédiarité et le PageRank, cela évalue l'importance ou la centralité des personnages dans le récit. Cela peut montrer qui a le plus d'influence, qui relie différentes parties de l'histoire ou joue un rôle crucial.
* **Évolution des personnages :** Suivre les changements dans l'importance des personnages à travers différents livres offre des perspectives sur les arcs narratifs des personnages, le développement de l'intrigue ou des changements significatifs dans la focalisation narrative.
* **Analyse comparative** : Comparer la centralité des personnages à travers les livres peut révéler comment leurs rôles et leur importance évoluent avec le temps ou en raison de certains événements dans l'histoire.

Voici un aperçu de ce que représente chaque caractéristique :

* **Source :** Cette caractéristique représente le nœud source dans le réseau, indiquant un personnage impliqué dans une interaction ou une co-occurrence.
* **Target :** Elle indique le nœud cible dans le réseau, représentant un autre personnage impliqué dans la même interaction ou co-occurrence que le personnage source.
* **Type :** Cette caractéristique spécifie le type de relation ou d'arête entre les nœuds source et cible. Dans cette analyse, il est mentionné que toutes les arêtes sont "Non dirigées", indiquant une relation non directionnelle entre les personnages.
* **Weight :** L'attribut de poids représente la force ou la fréquence des interactions ou co-occurrences entre les personnages source et cible. Une valeur de poids plus élevée signifie une relation plus forte ou des co-occurrences plus fréquentes.
* **Book :** Cette caractéristique représente le numéro du livre ou le volume dans la série "Game of Thrones" où ces interactions ou co-occurrences entre les personnages ont été observées.

# Architecture :

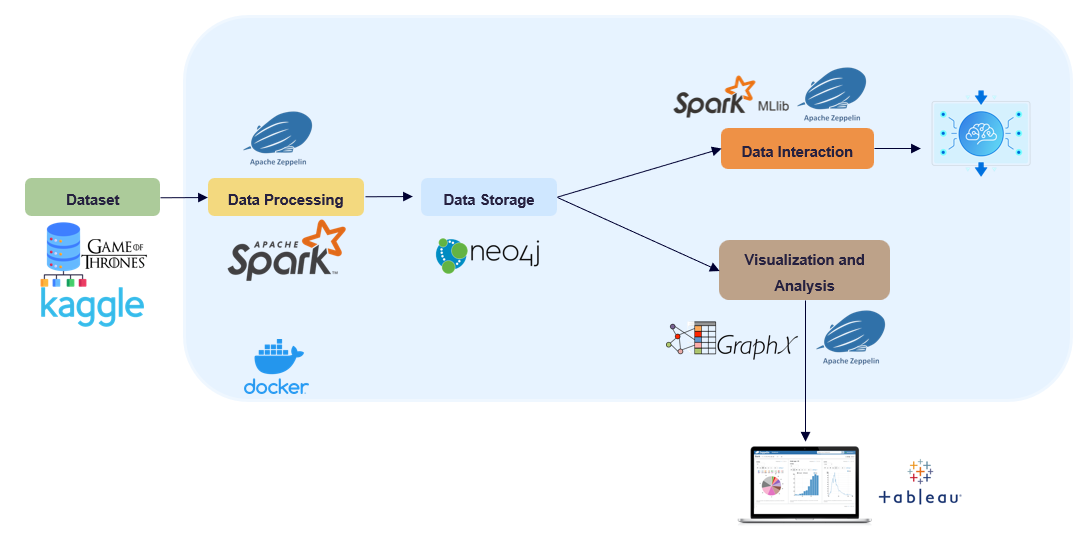


Figure 2: Architecture du projet

* **Collecte de Données depuis Kaggle :**

Les données sources sont collectées depuis Kaggle, où des ensembles de données liés à Game of Thrones sont disponibles. Ces données peuvent inclure des informations sur les personnages, les maisons, les relations, etc.

* **Chargement des Données avec Spark dans Zeppelin :**

Utilisation d'Apache Spark pour le chargement et la manipulation des données. Spark permet de traiter de grands ensembles de données de manière distribuée et efficace. Les données sont chargées dans Apache Zeppelin, un environnement de notebook interactif.

* **Traitement des Données dans Zeppelin :**

Les données provenant de différents fichiers CSV sont traitées dans Apache Zeppelin. Les étapes de traitement incluent la fusion des fichiers pour rassembler les informations de manière cohérente, le nettoyage des données, et la transformation si nécessaire.

* **Stockage Direct dans Neo4j :**

Les données traitées sont directement stockées dans Neo4j, une base de données orientée graphe. Neo4j est idéal pour représenter et stocker des relations complexes, ce qui est essentiel pour modéliser le réseau de Game of Thrones.

* **Visualisation et Analyse avec GraphX :**

Utilisation d'Apache GraphX, un module de traitement de graphes de Spark, pour visualiser et analyser le réseau complexe de Game of Thrones. GraphX permet de déduire des motifs, des clusters et d'explorer les relations entre les différents éléments du réseau.

* **Affichage des Résultats d'Algorithmes dans Tableau :**

Les résultats obtenus à partir de l'analyse avec GraphX peuvent être visualisés de manière plus conviviale en utilisant Tableau. Cette étape permet de présenter les insights gagnés à un public plus large de manière intuitive.

* **Application de Modèles de Machine Learning à partir de Neo4j :**

À partir de la base de données Neo4j, des modèles de machine learning peuvent être appliqués pour tirer des conclusions prédictives. Cela pourrait inclure des prédictions basées sur les relations entre les personnages, les maisons, ou d'autres entités.

# L’image Docker :

Cet image de configuration Docker Compose définit un ensemble de services pour Apache Zeppelin (0.10.0), Spark (3.1.2), et Neo4j (5) au sein d'un réseau Docker commun. Zeppelin est configuré pour se connecter au Spark Master, et un connecteur Neo4j-Spark est ajouté au répertoire des notebooks. Les services Spark et Neo4j sont également configurés avec les ports nécessaires pour l'interface utilisateur et la communication entre les conteneurs. Le réseau Docker personnalisé facilite la communication entre les services, tandis que les volumes Docker assurent la persistance des données de Neo4j et offrent une flexibilité pour le stockage des notebooks Zeppelin. En résumé, ce fichier permet de déployer un environnement intégré pour l'analyse de données avec Apache Zeppelin, Spark, et Neo4j.

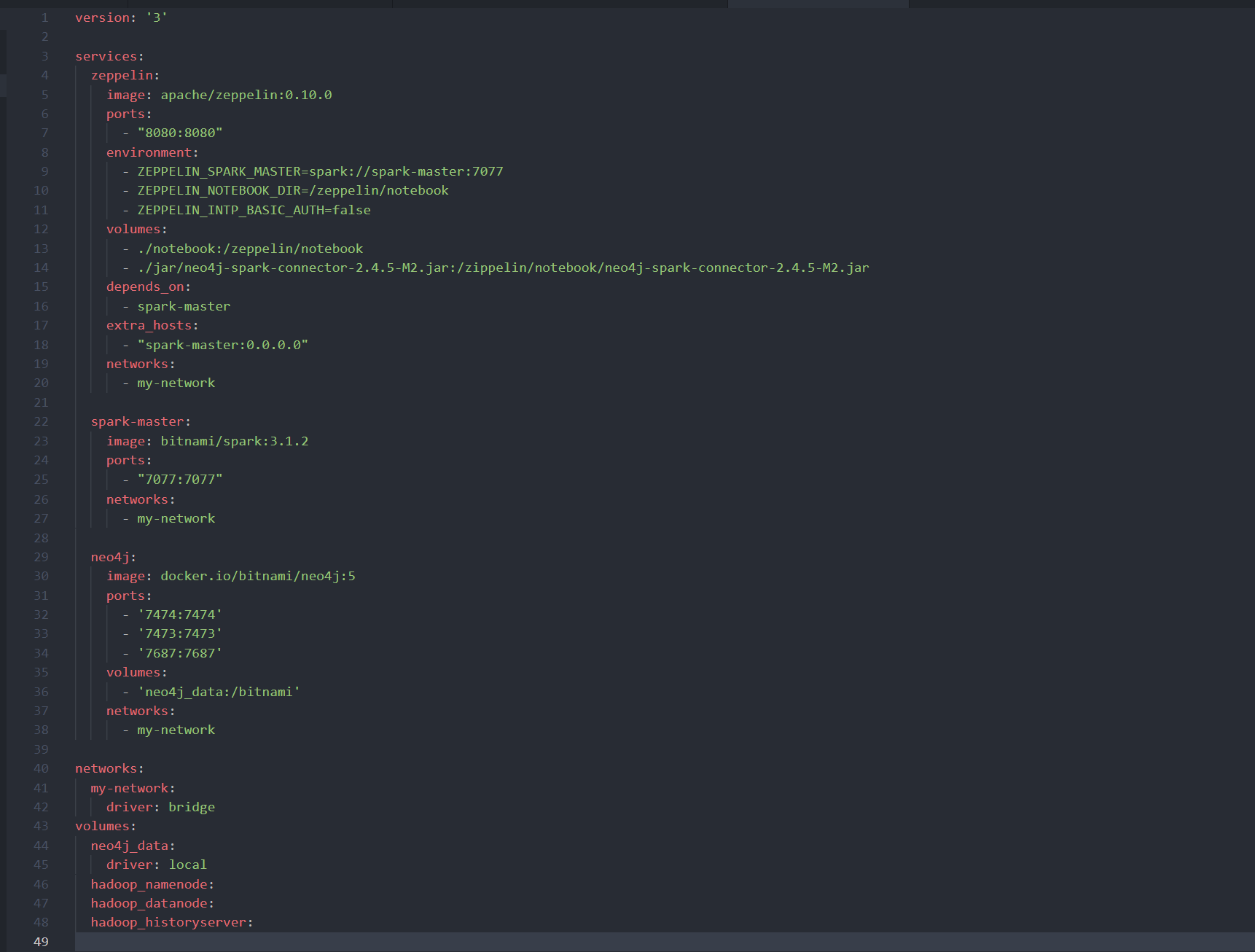


Figure 3 docker-compose.yml file

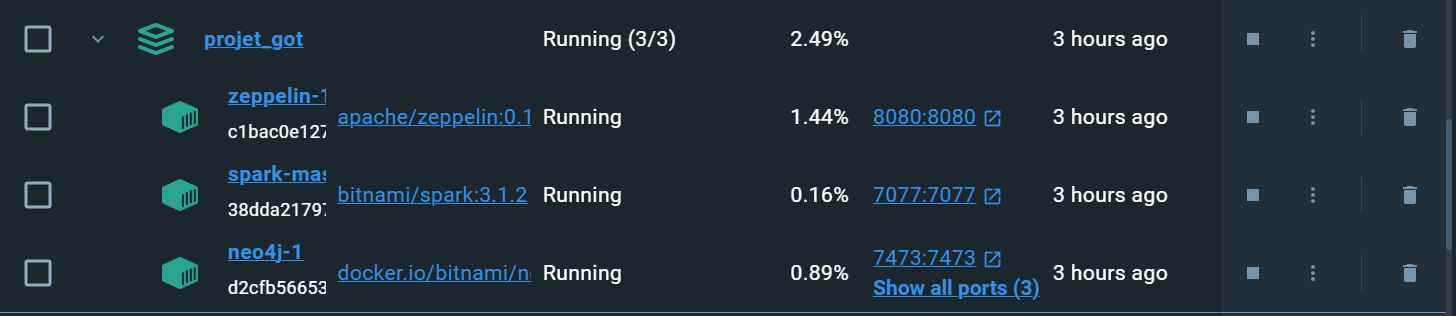


Figure 4 le cluster docker

# Importation des JARs :

Pour établir une connexion entre Neo4j et Spark dans Zeppelin, assurez-vous tout d'abord de la version de Spark installée dans Zeppelin. Vous pouvez vérifier cela en exécutant la commande suivante dans un paragraphe Zeppelin :

%spark

spark.version



Figure 5 version de spark dans zeppelin

Pour utiliser le connecteur Neo4j-Spark, vous devrez télécharger le JAR correspondant à votre environnement. Voici comment vous pouvez télécharger le JAR depuis la page des releases GitHub :

<https://github.com/neo4j-contrib/neo4j-spark-connector/releases?page=3>

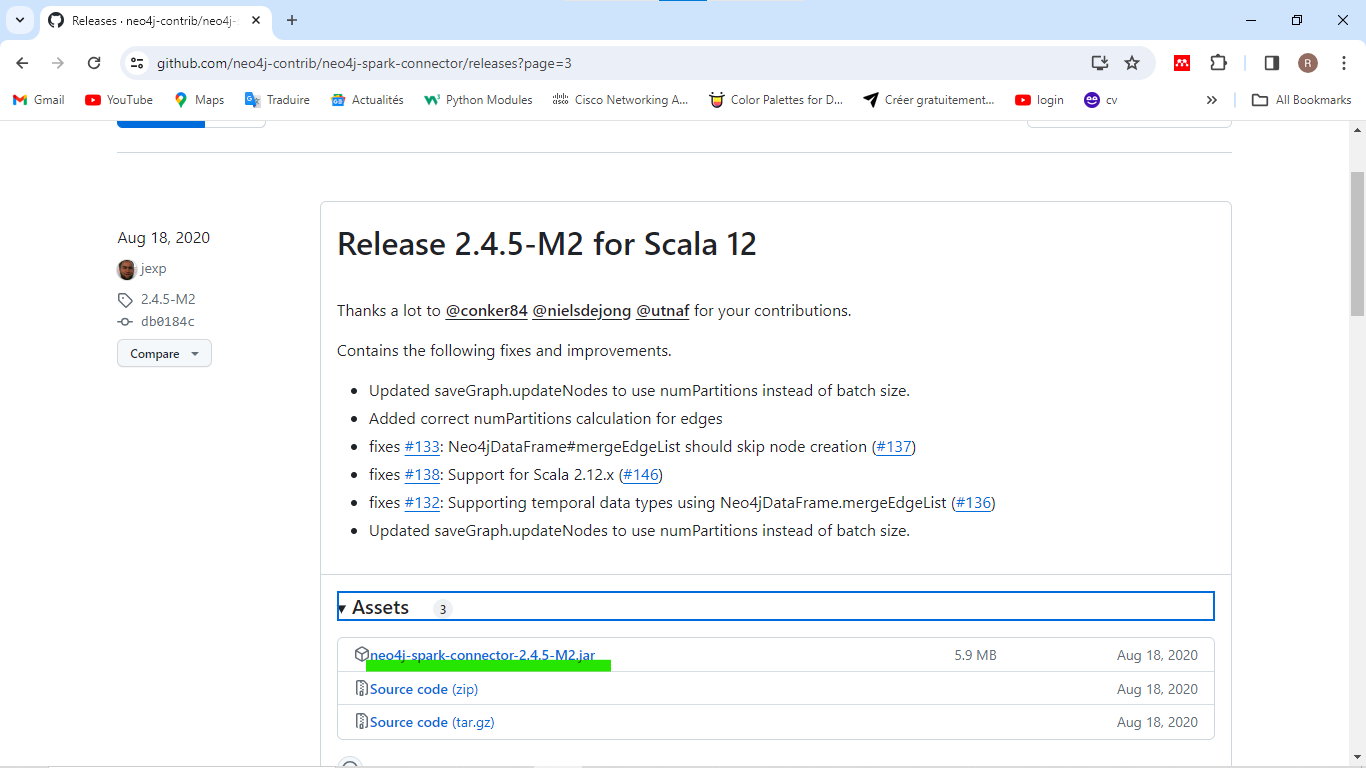


Figure 6 fichier JAR de connexion de neo4j et spark

Si vous souhaitez copier un fichier JAR depuis votre système local vers un conteneur Docker Zeppelin, vous pouvez utiliser la commande docker cp. Assurez-vous que le conteneur Zeppelin est en cours d'exécution.

**docker cp chemin\_local\_du\_jar container\_id:/opt/zeppelin/interpreter/spark/**

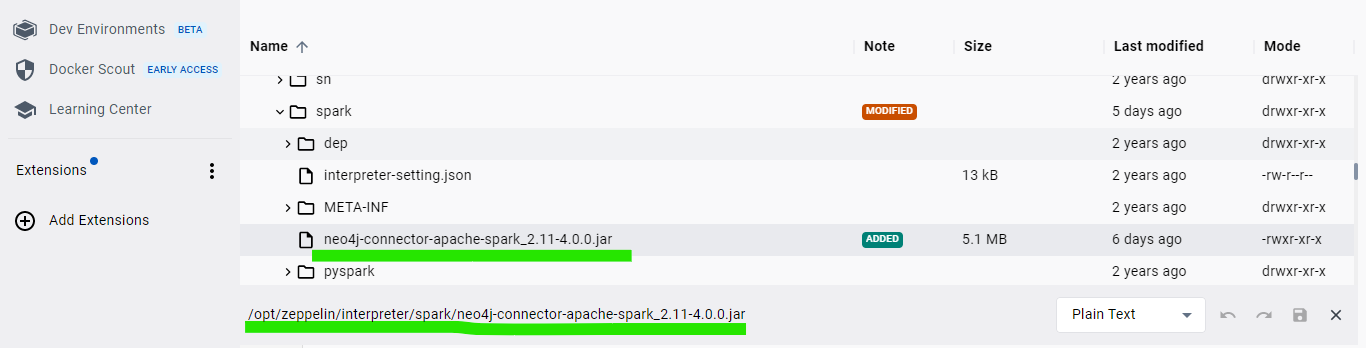


Figure 7 importation du jar dans zeppelin

On passe au chemin de configuration de Zeppelin et on ajoute la configuration du connecteur :

cd /opt/zeppelin/conf/

cat zeppelin-env.sh.template

On passe à ce chemin et on ajoute cette ligne :

# export ZEPPELIN\_INTP\_CLASSPATH\_REMOTE=/opt/zeppelin/interpreter/spark/neo4j-connector-apache-spark\_2.11-4.0.0.jar

Le JAR file sera present dans le repository de GitHub.



Figure 8 configuration du JAR

# Manipulation & Insertion des données:

* Vous trouverez tous les notebooks nécessaires à utiliser dans ce dossier

# 

Figure 9 les notebook du Projet

* Copiez les fichiers CSV vers le conteneur :

**docker cp chemin\_local\_des\_fichiers\_csv container\_id:/chemin\_dans\_conteneur/**

* Accédez au conteneur Zeppelin en utilisant exec :

**docker exec -it zeeplin\_container\_id bash**

* Déplacez les fichiers CSV vers le répertoire approprié à l'intérieur du conteneur :

**mv /chemin\_dans\_conteneur/fichiers\_csv/\* /zeppelin/notebook/**

## Récupération et Manipulation des données

* Puis concaténer l'ensemble des données (datasets) provenant des cinq livres et faire des manipulations :

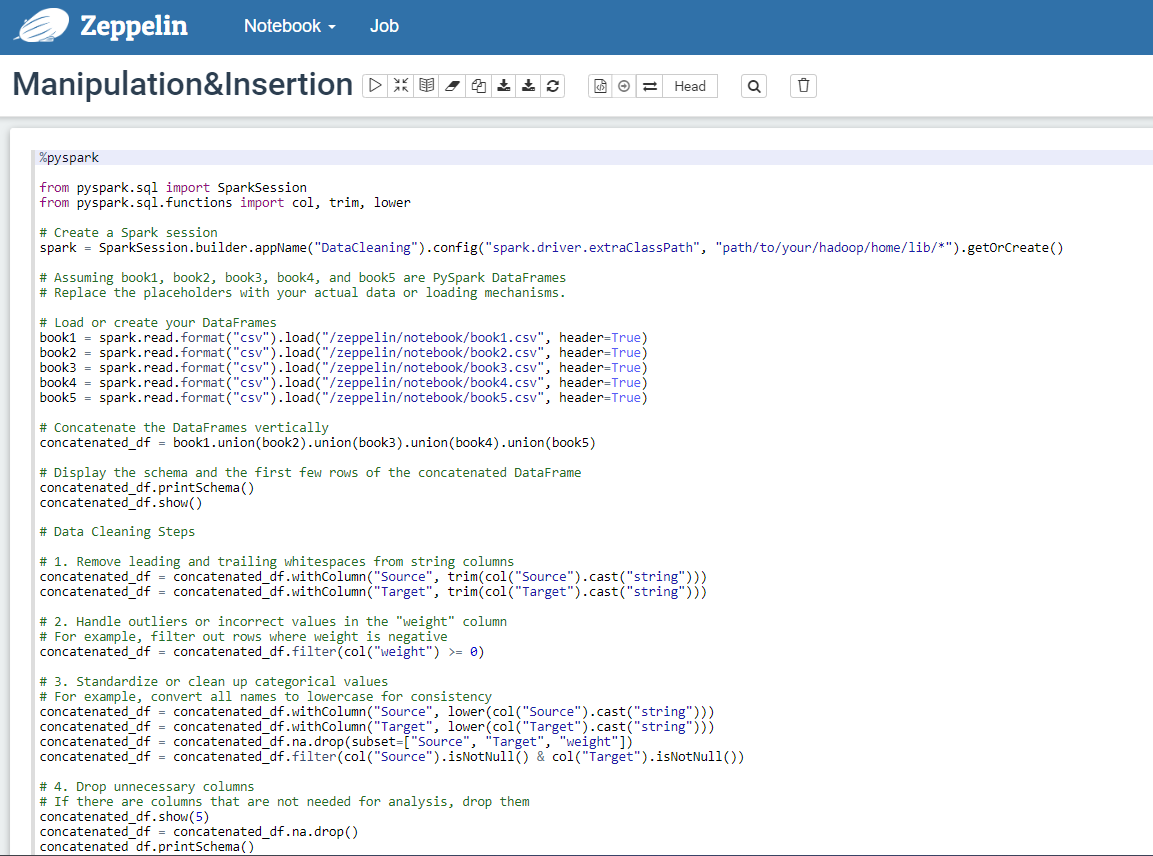


Figure 10 code de concatenation des fichier CSV

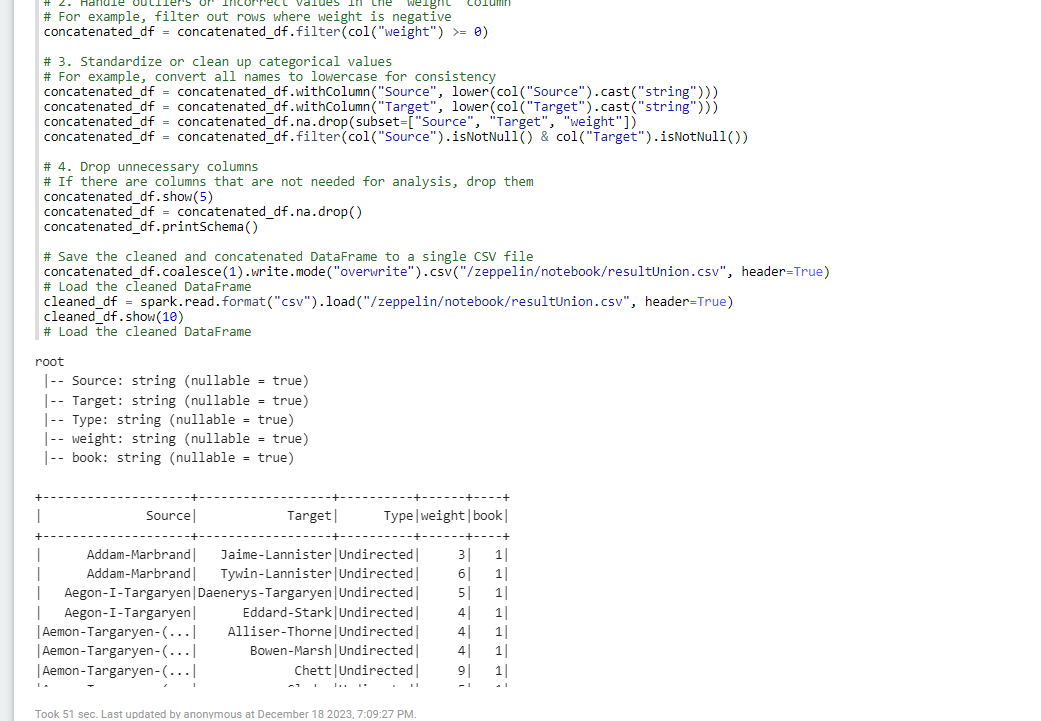


Figure 11 stockage dans un nouveau fichier result.csv

## Insertion des données

* Insertion de données dans Neo4j avec Spark :

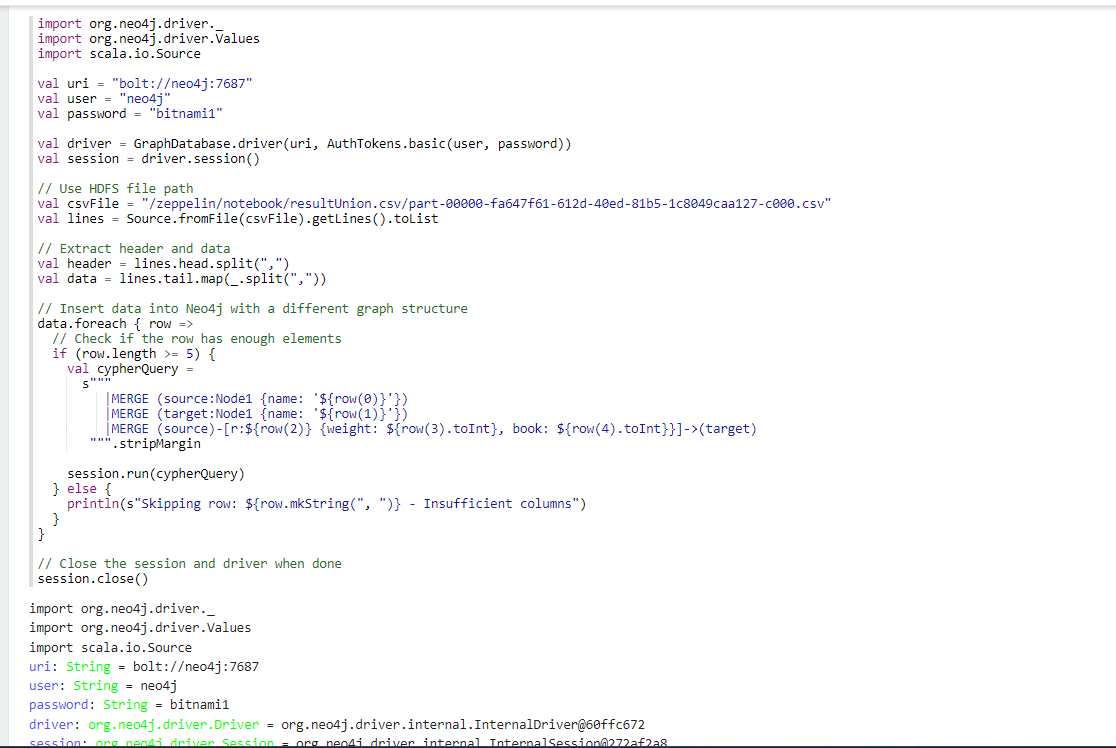


Figure 12 insertion des données

## Vérification dans Neo4j

* On peut visualiser les résultats stockés dans Neo4j en accédant à l'adresse

http://localhost:7474/browser/

* Visualiser vos données dans le conteneur neo4j

MATCH (n:Node1) return n;

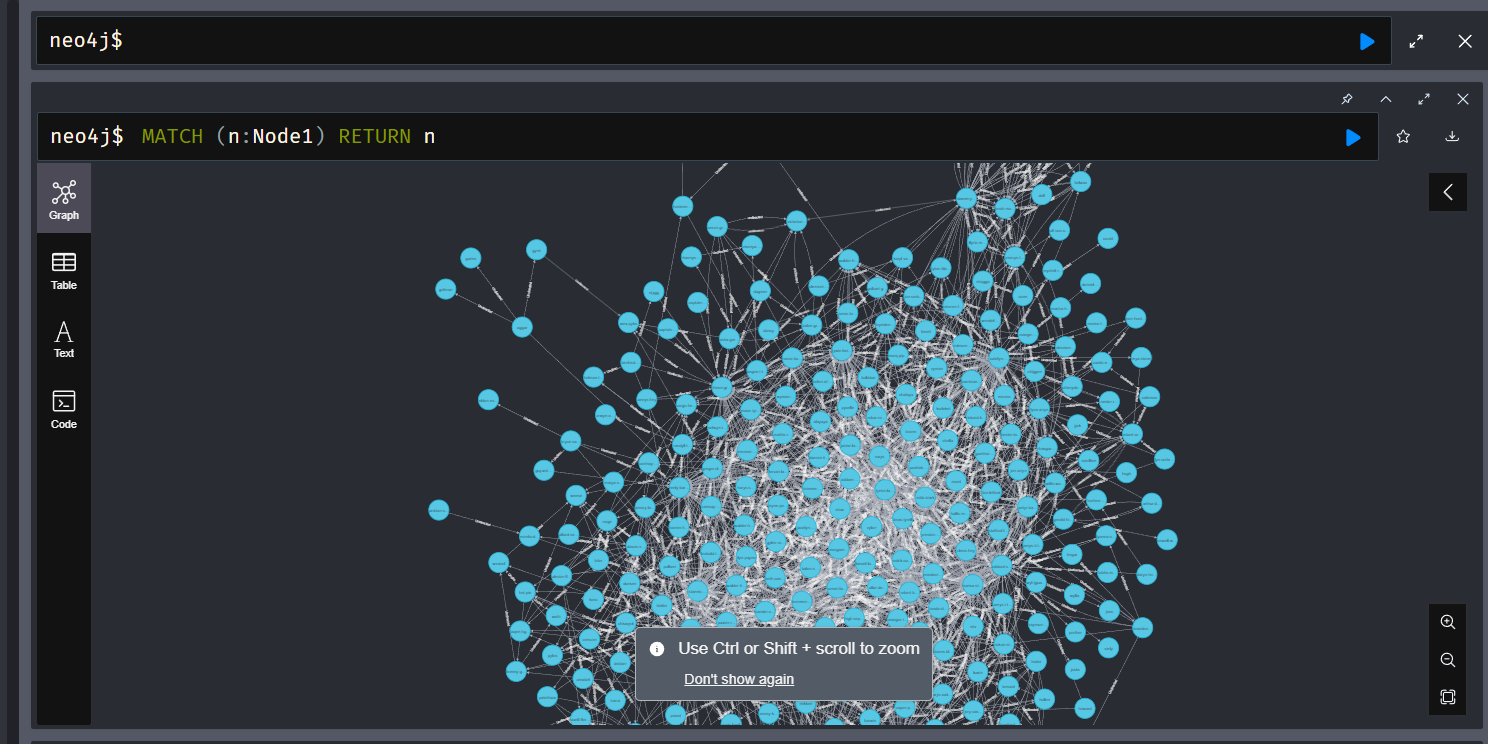


Figure 13 visualisation

# CRUD :

## Récupération d’après Neo4j

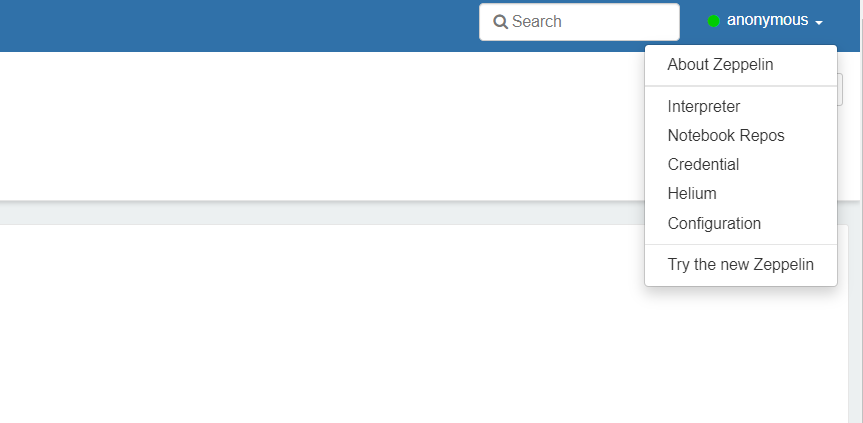
* Exécution de commande Neo4j dans zeppelin :

Figure 14 configuration de l'interpréteur Neo4j

* Premièrement si vous n’avez pas Neo4j comme interpréteur dans Zeppelin il faut l’ajouter, et la configurer

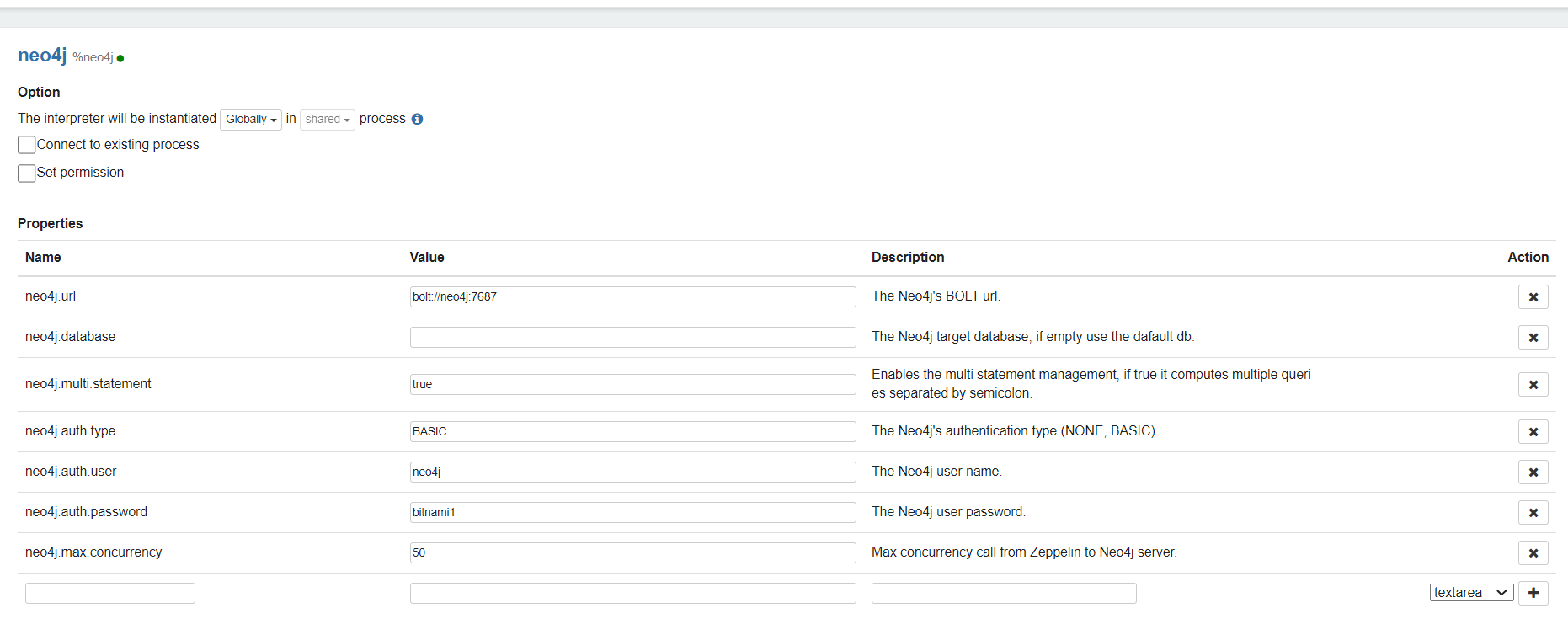


Figure 15 connecter l'interpreteur avec le contenuer neo4j

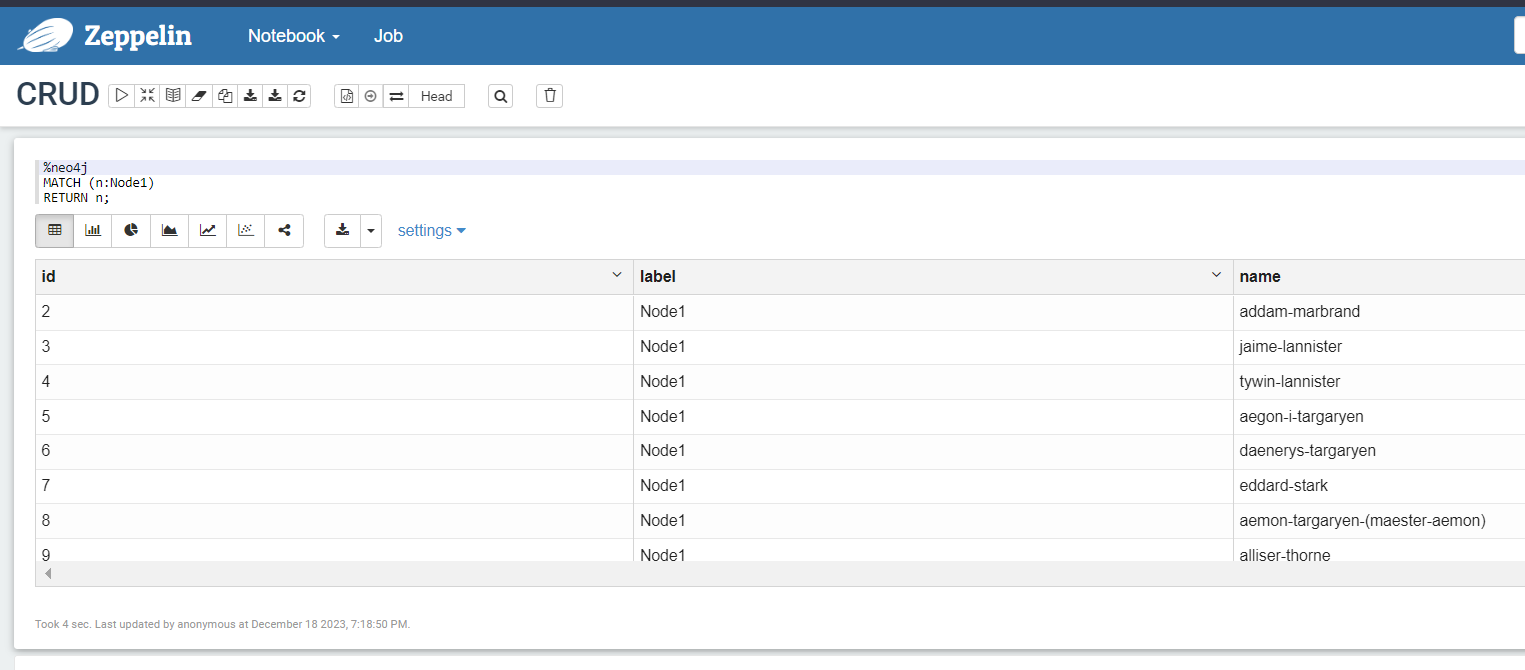


Figure 16 visualisation dans zeppelin

En utilisant les requetés CYPHER dans zeppelin avec l’interpreteur neo4j.

%neo4j

* Récupération des Nodes d’après neo4j :



Figure 17 lire des données de neo4j dans zeppelin pour les manipuler

* Création d’un graph :

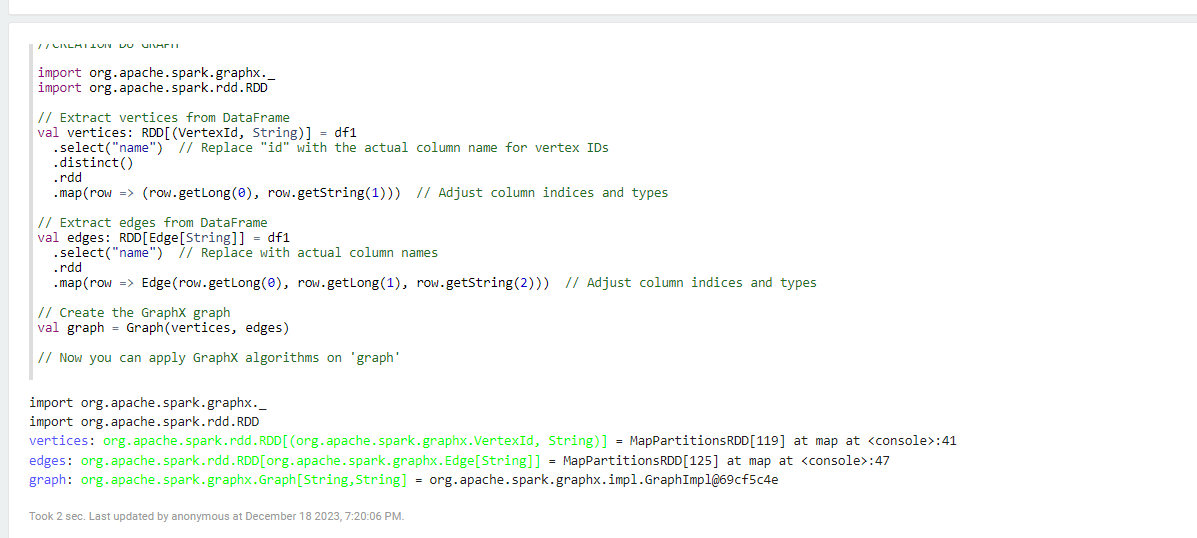


Figure 18 créer un graph à partir des données

* Récupération des données d’après neo4j :

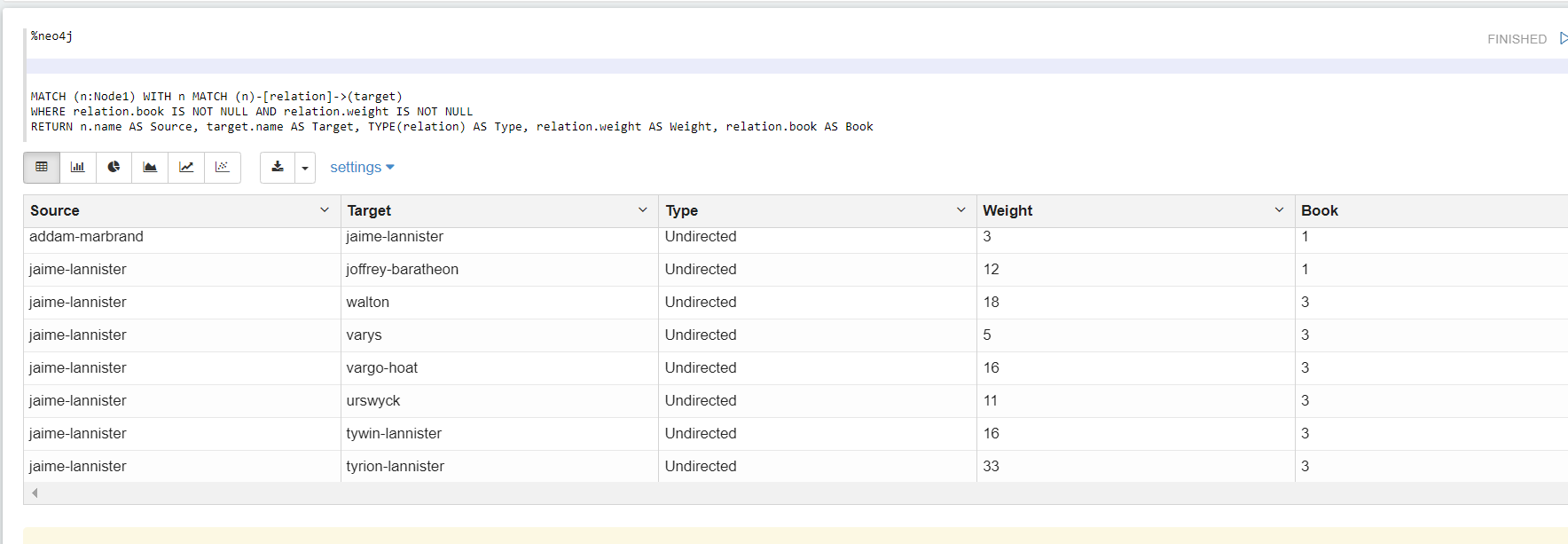


Figure 19 visualiser les données en utilisant les CYPHER query

On rapporte ici les différents nœud et les relation entre eux.



Figure 20 visualiser les données en utilisant SPARK

## Exemples de filtrages

* Récupération des donnes du Book1:



Figure 21 les nœuds du premier livre

* Récupération des donnes du Book2:



Figure 22 les nœuds du 2eme livre

* Récupération des donnes du Book3:

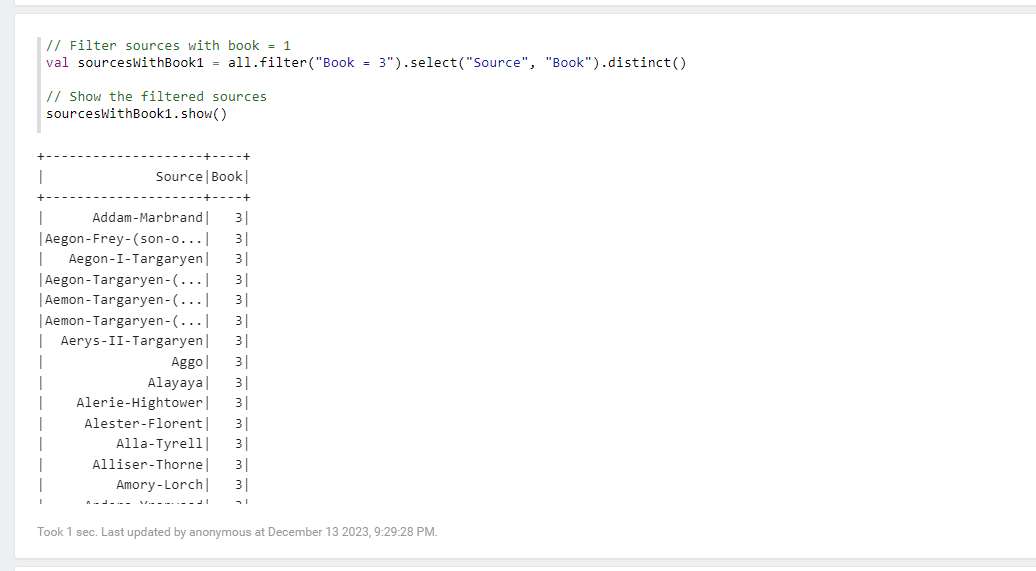


Figure 23 les nœuds du 3eme livre

* Récupération des donnes du Book4:

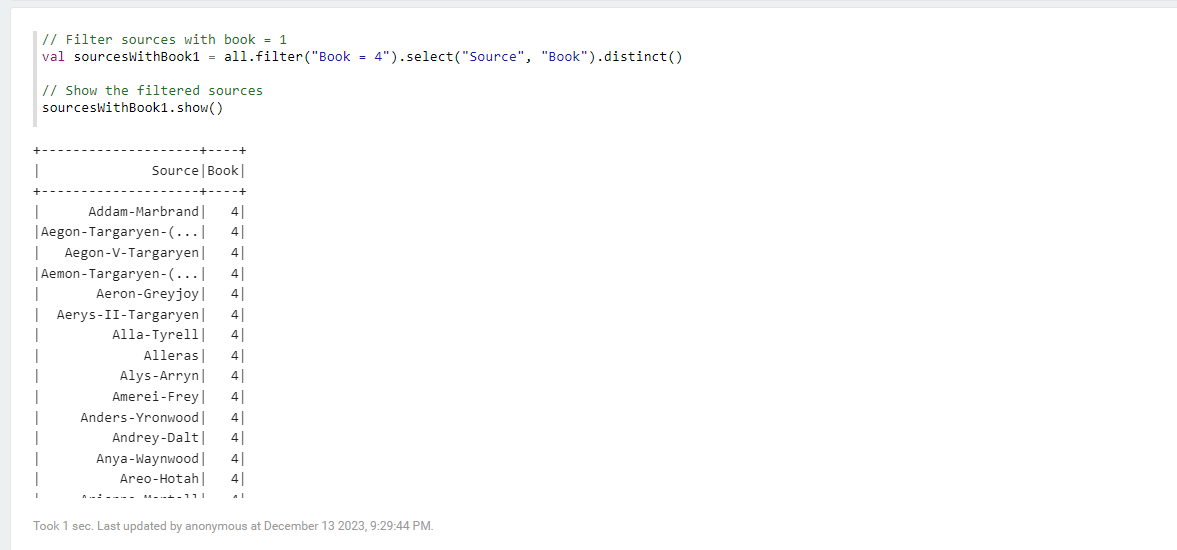


Figure 24 les nœuds du 4eme livre

* Récupération des donnes du Book5:



Figure 25 les nœuds du dernier livre

* Les acteurs qui existent dans tous les livres :

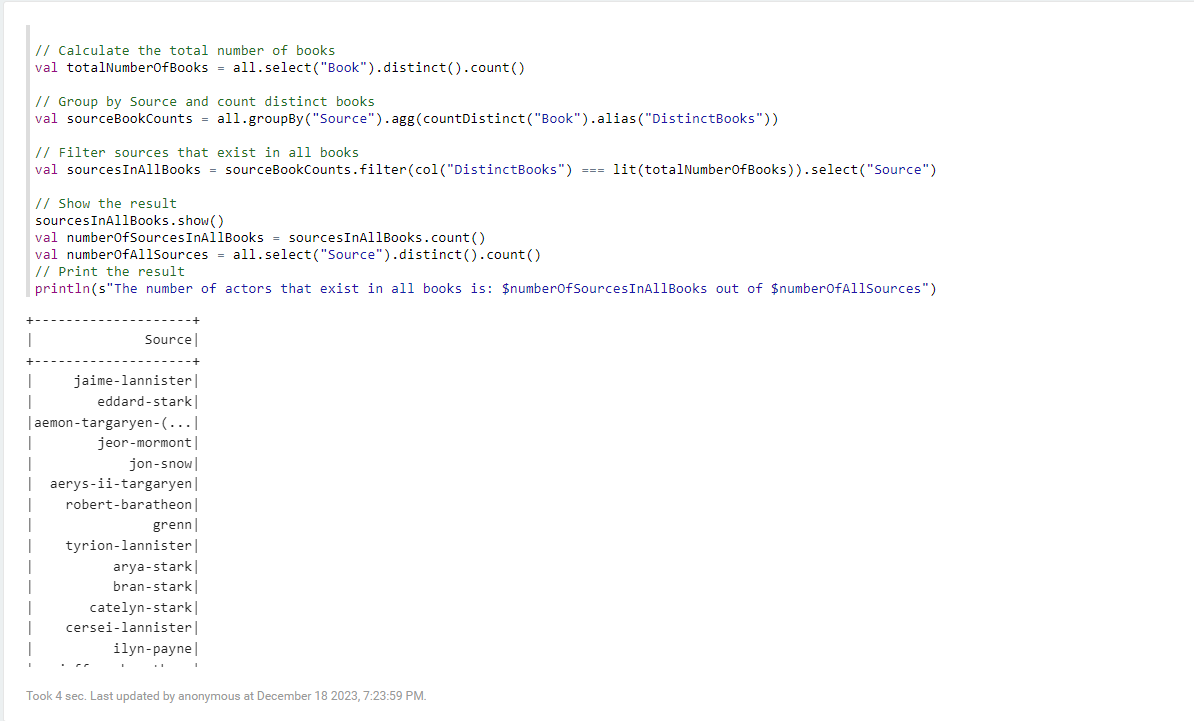


Figure 26 les personnages qui sont dans tous les livres

* La personne la plus présente dans la base de données :

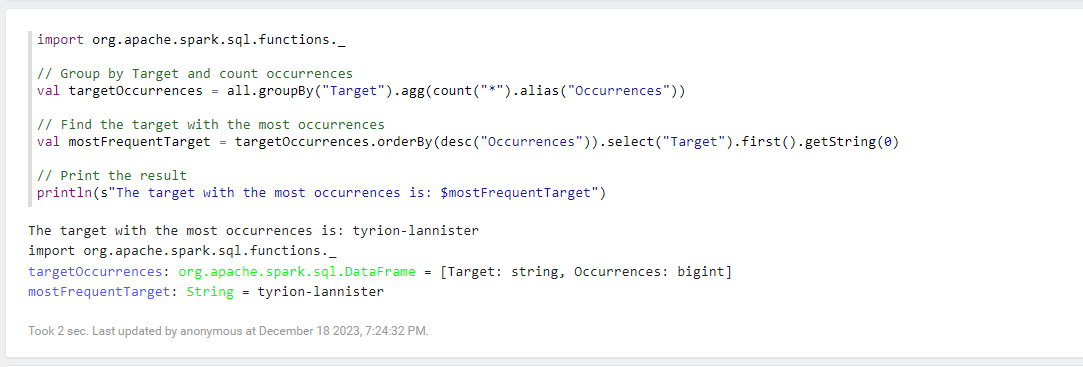


Figure 27 la personne la plus présente dans la série

## ADD

* Ajoute des données:



Figure 28 ajouter un noeud

* Vérification:

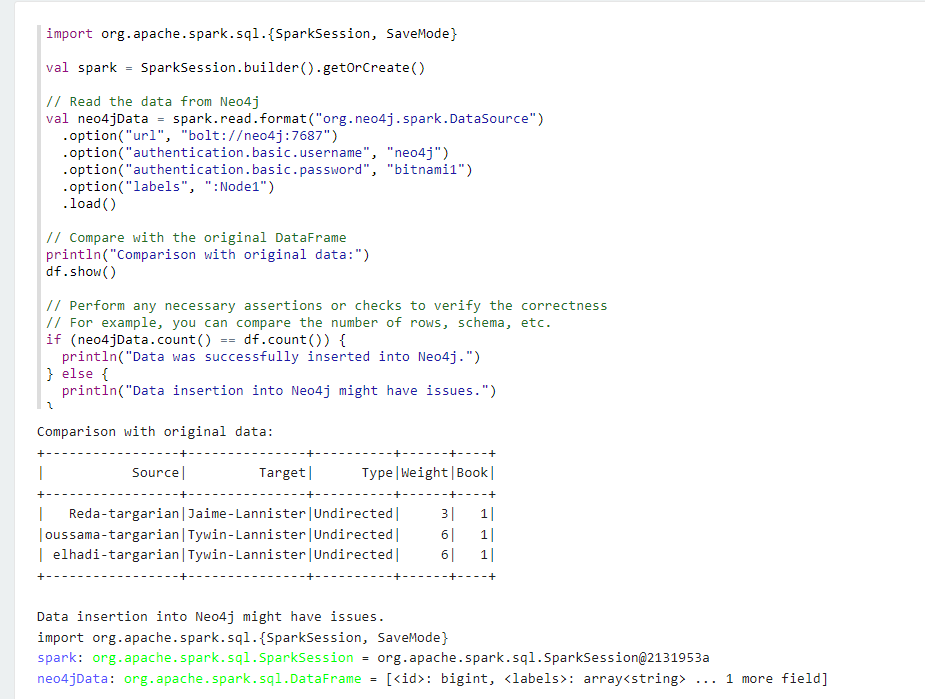


Figure 29 verifier l'ajout

## EDIT

* Edit:

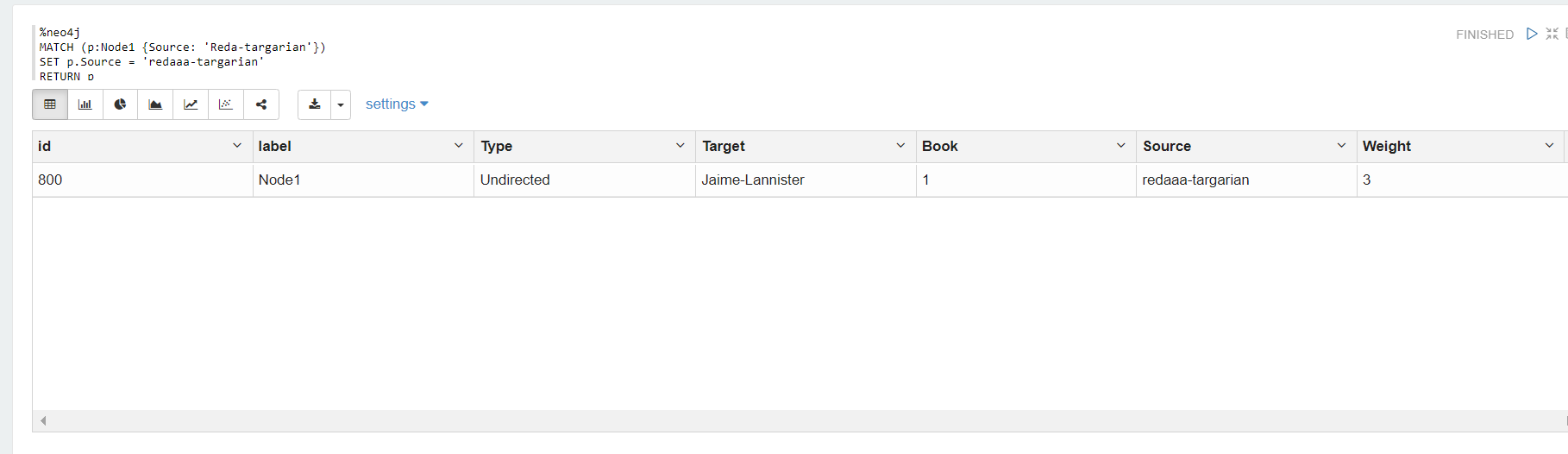


Figure 30 modification des données

* vérification :

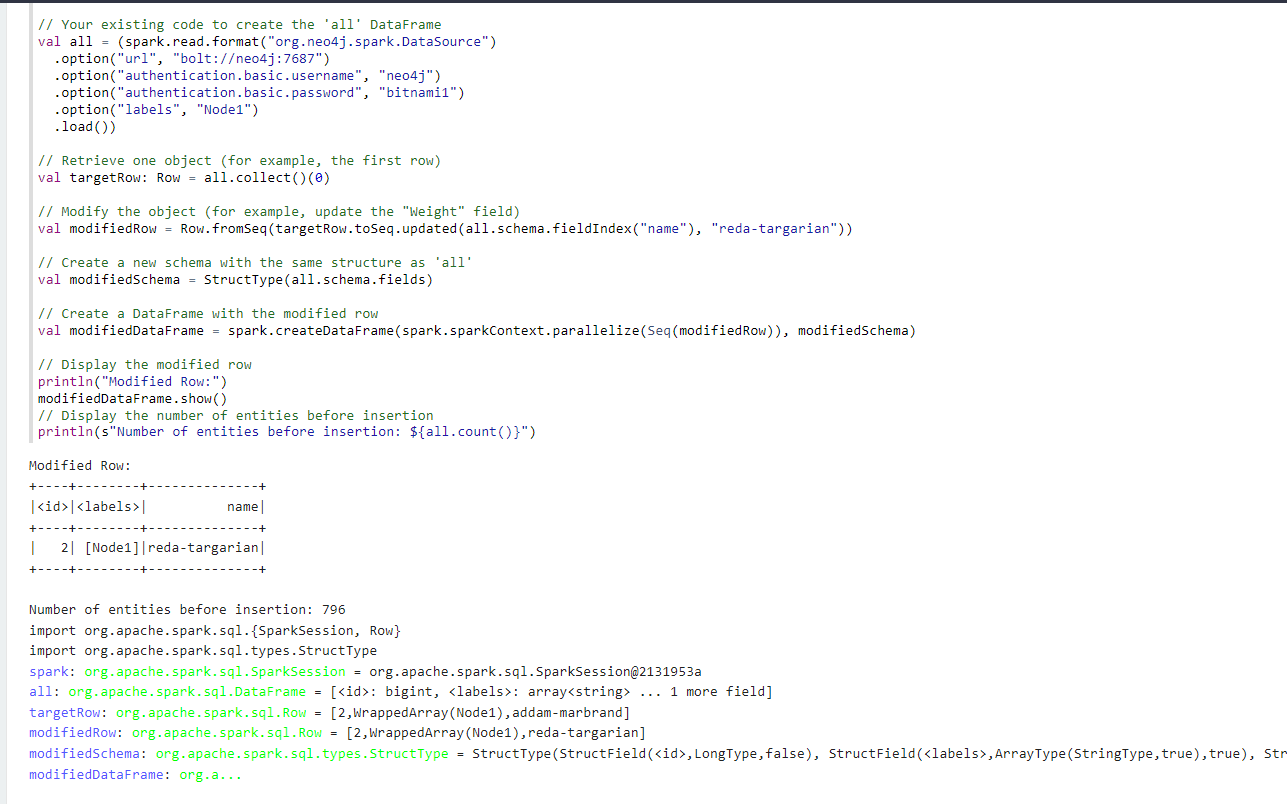


Figure 31 vérification de la modification

## DELETE

* Delete:

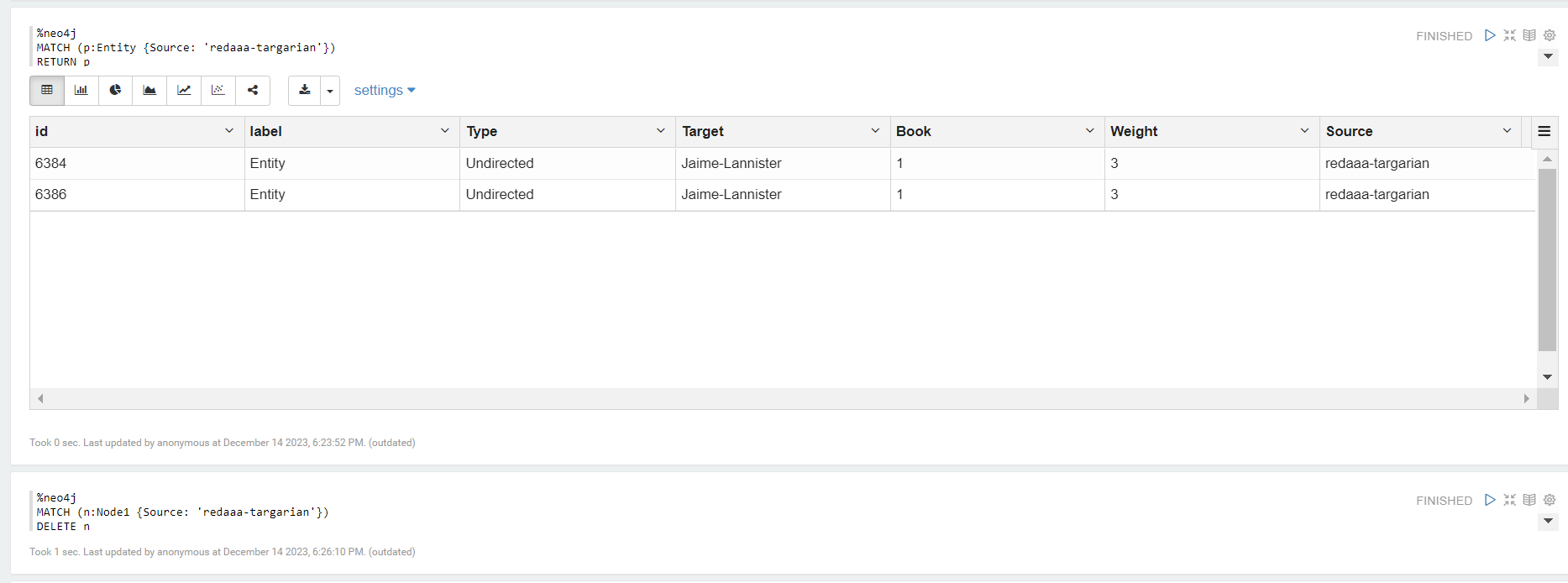


Figure 32 la suppression des données

# GraphX :

## Récupération des données et création du graph

* Récupération des données :

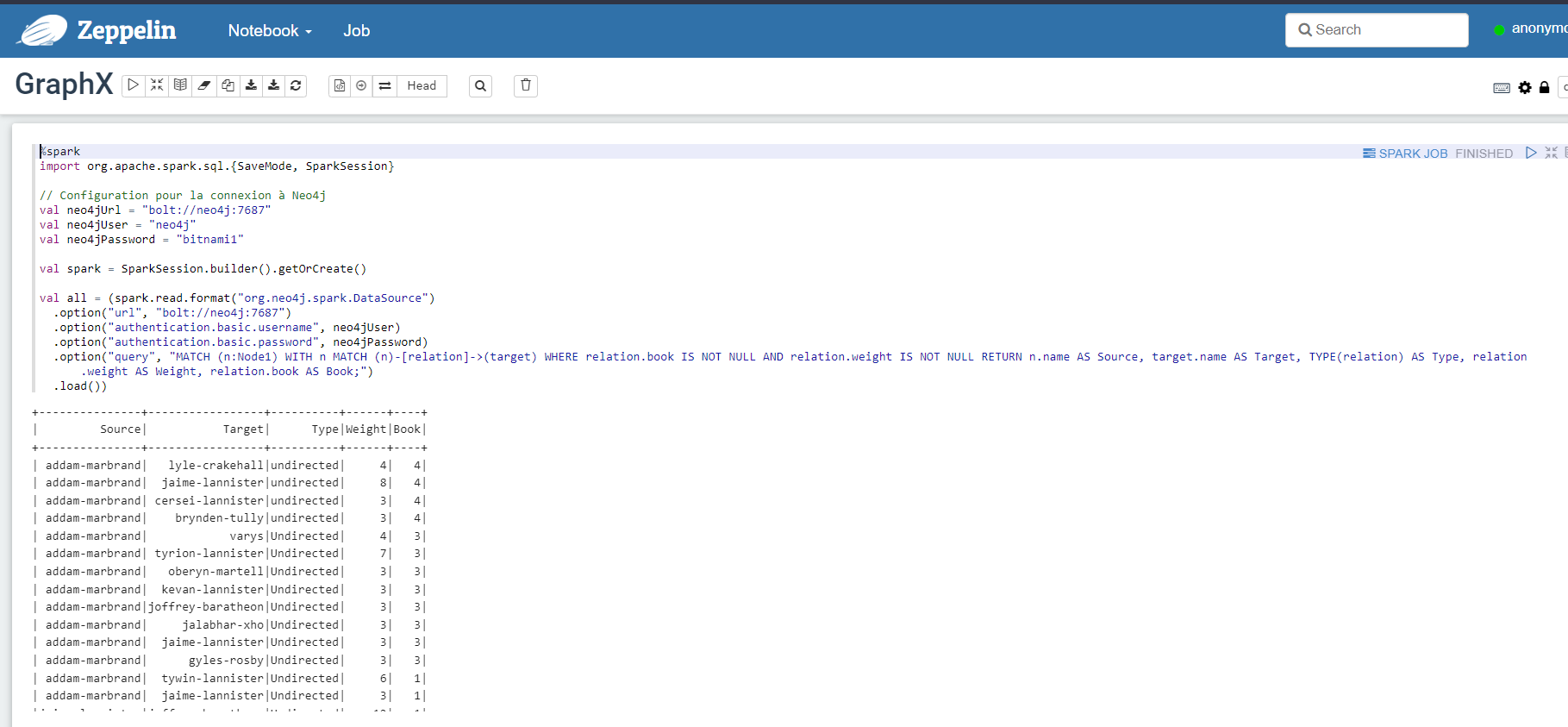


Figure 33 Récupération des données

* Création du Graph en utilisant GraphX :

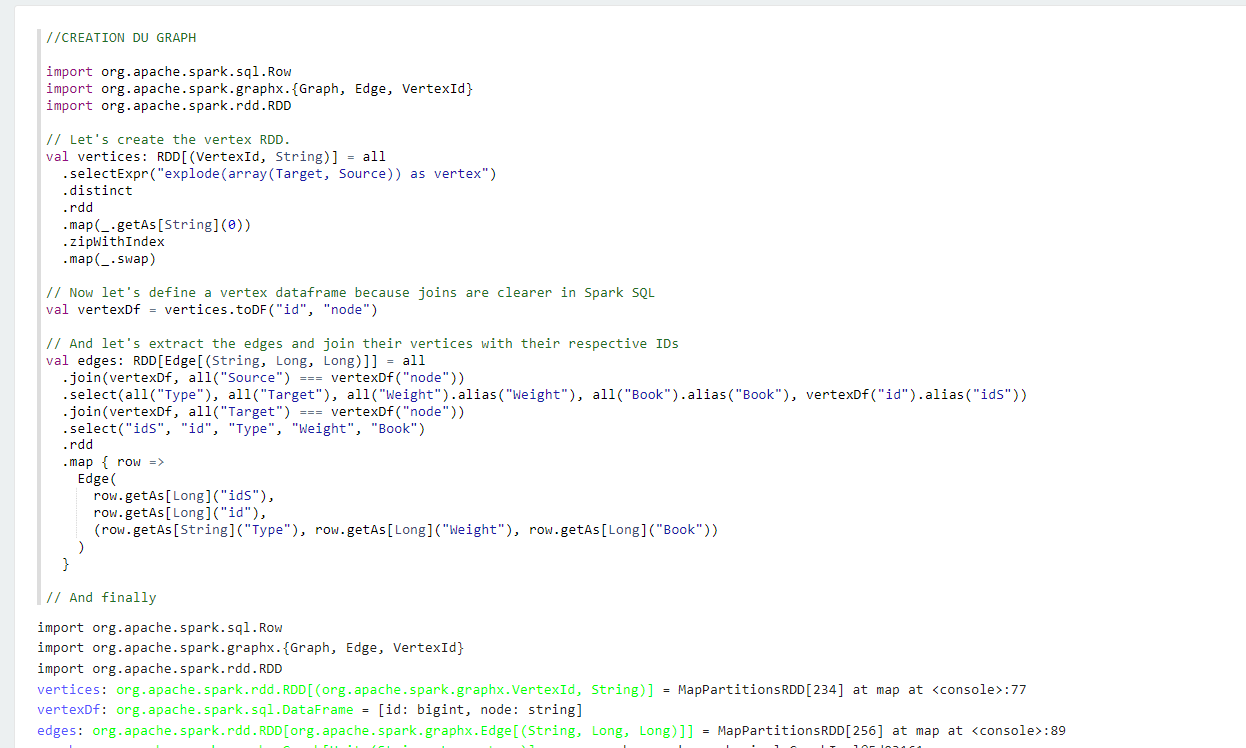


Figure 34 créer un Graphe à l'aide de graphX pour les données récupérés

* Affichage des nodes et relations (vertices et Edges):



Figure 35 Affichage des (vertices et Edges)

* Analyse statistique du Graph en utilisant GraphX :

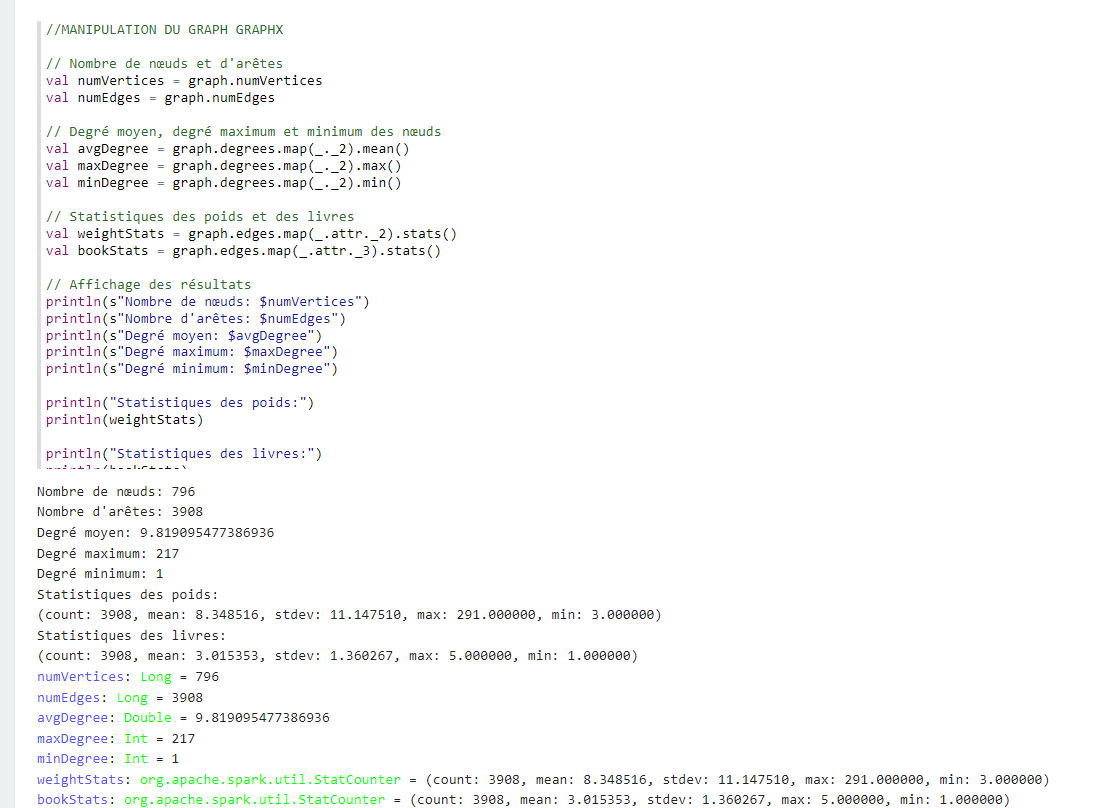


Figure 36 des statistiques sur le graph

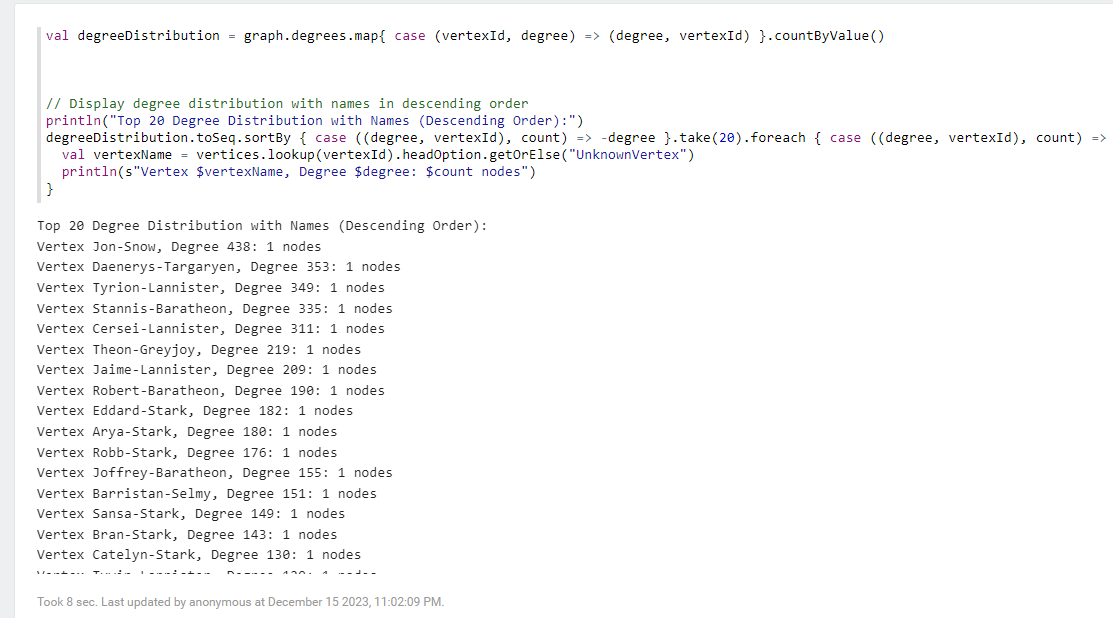


Figure 37 afficher les degrées des noeuds

Maintenant on applique les algorithmes sur GraphX :

## PageRank Algorithm

Un algorithme d'analyse de liens utilisé pour mesurer l'importance relative des nœuds dans un graphe orienté ou non orienté. Il attribue des scores aux nœuds en fonction de la structure du graphe, en tenant compte du nombre et de la qualité des liens.



Figure 38 PageRank Algorithm

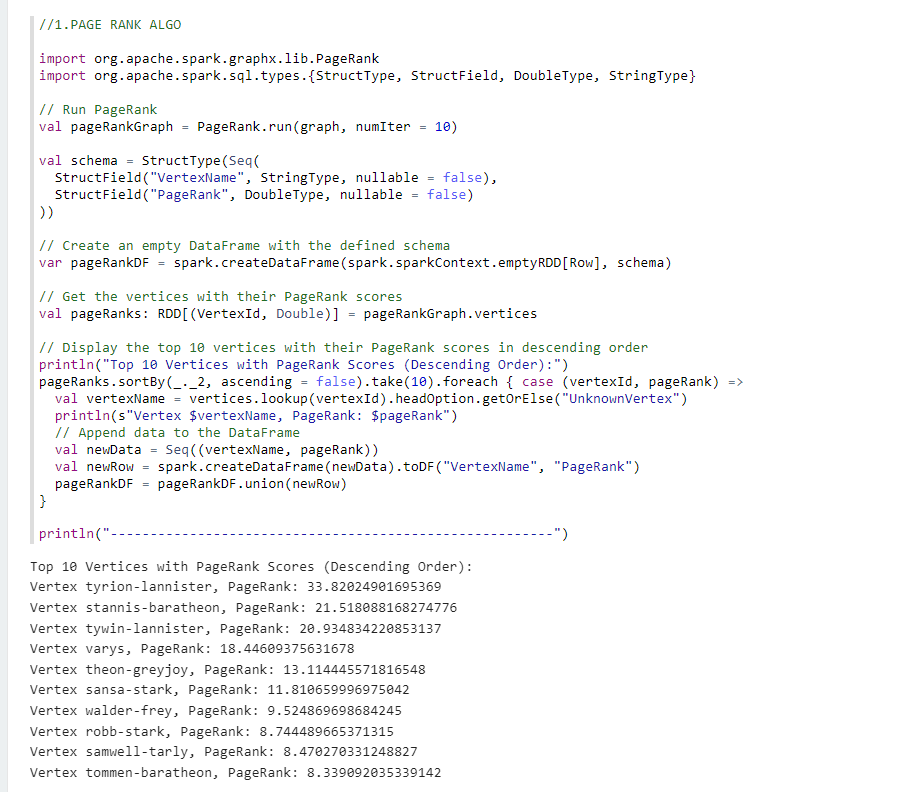


Figure 39 resultat de l'algorithme

**Top 10 Sommets :** Les 10 premiers sommets affichés représentent les nœuds les plus influents du graphe, selon l'algorithme PageRank. Ces sommets ont une probabilité plus élevée d'être visités lors d'un parcours aléatoire du graphe.

**Distribution de l'Influence** : Les scores PageRank fournissent une mesure de la distribution de l'influence à travers le réseau. Les sommets en haut de la liste ont une influence significative sur d'autres sommets du graphe.

**Identification de Points Centraux :** Les sommets en haut de la liste, tels que "Tyrion-Lannister" avec un PageRank de 33.82, peuvent être considérés comme des points centraux dans le réseau, indiquant peut-être leur rôle central ou leur importance dans le contexte du graphe.

## Connected Components Algorithm

Un algorithme qui identifie les composantes connexes dans un graphe non orienté. Les composantes connexes sont des sous-ensembles de nœuds dans lesquels chaque nœud est relié à tous les autres par un chemin.

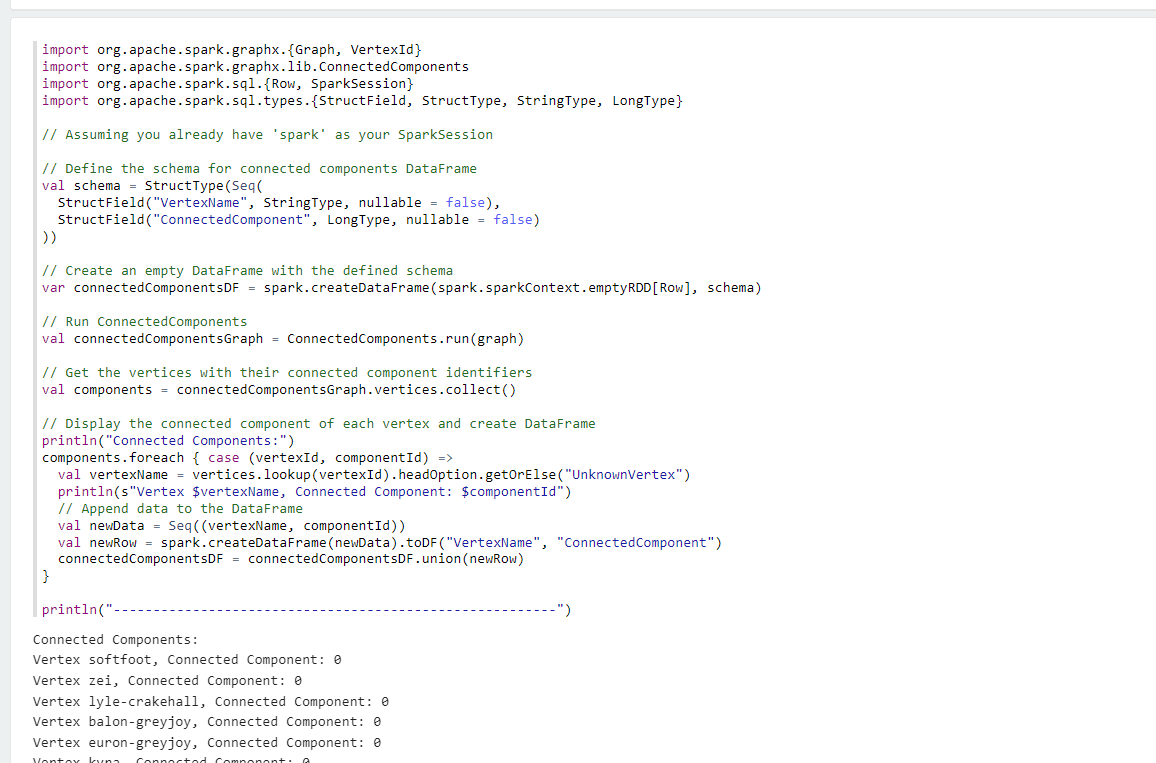


Figure 40 Connected Components Algorithm

Le fait que tous les sommets appartiennent au composant connecté "0" suggère qu'ils sont tous connectés les uns aux autres dans le graphe, directement ou indirectement, formant ainsi un seul ensemble ou une seule composante connectée.

## Label Propagation Algorithm

Un algorithme itératif où les nœuds d'un graphe se voient attribuer des étiquettes en fonction de l'étiquette majoritaire de leurs voisins. Cela se répète jusqu'à ce que chaque nœud ait une étiquette qui est la plus fréquente parmi ses voisins.

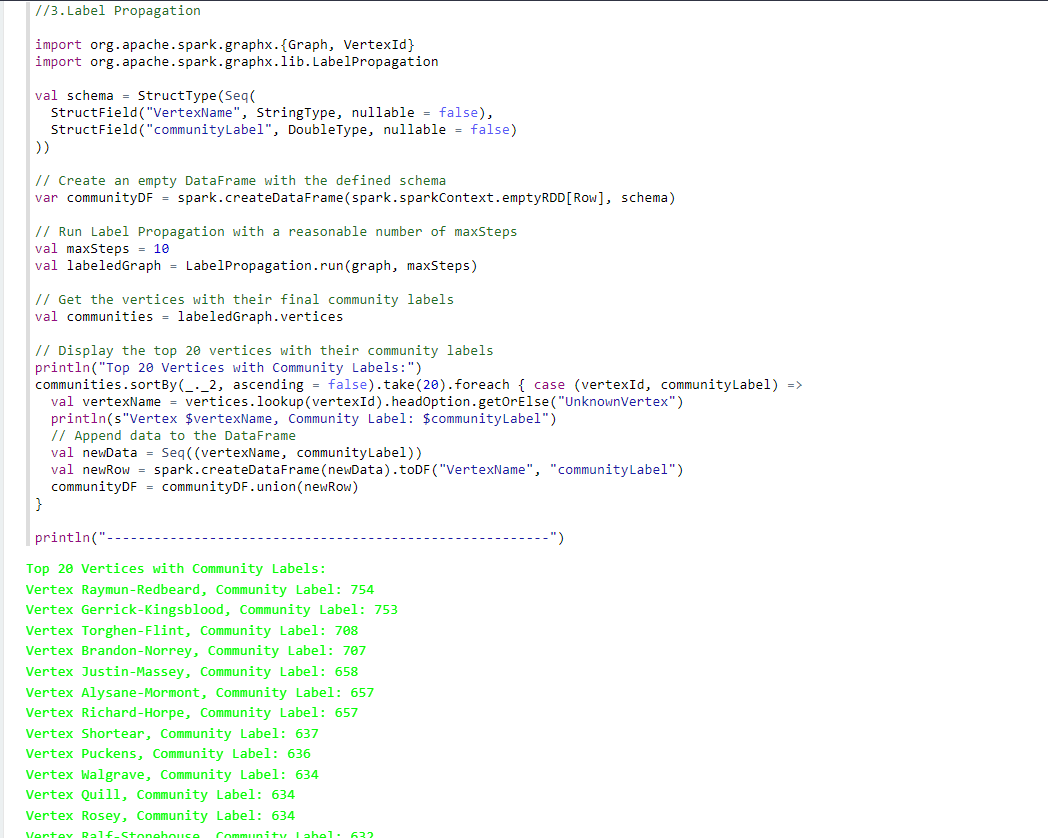


Figure 41 label propagation algorithme

**Diversité des Communautés :** Les sommets affichés avec leurs étiquettes de communauté représentent une variété de personnages issus de différentes régions ou factions dans le contexte du graphe. Par exemple, "Raymun-Redbeard" et "Gerrick-Kingsblood" appartiennent à des communautés distinctes (étiquettes 754 et 753).

**Top 20 Vertices :** Les 20 premiers sommets affichés avec leurs étiquettes de communauté offrent un aperçu des différentes communautés identifiées par l'algorithme.

## Triangle Count Algorithm

Un algorithme qui compte le nombre de triangles dans un graphe. Un triangle est une structure composée de trois nœuds interconnectés, et le comptage de triangles est utile pour comprendre la connectivité et la structure locale d'un graphe.

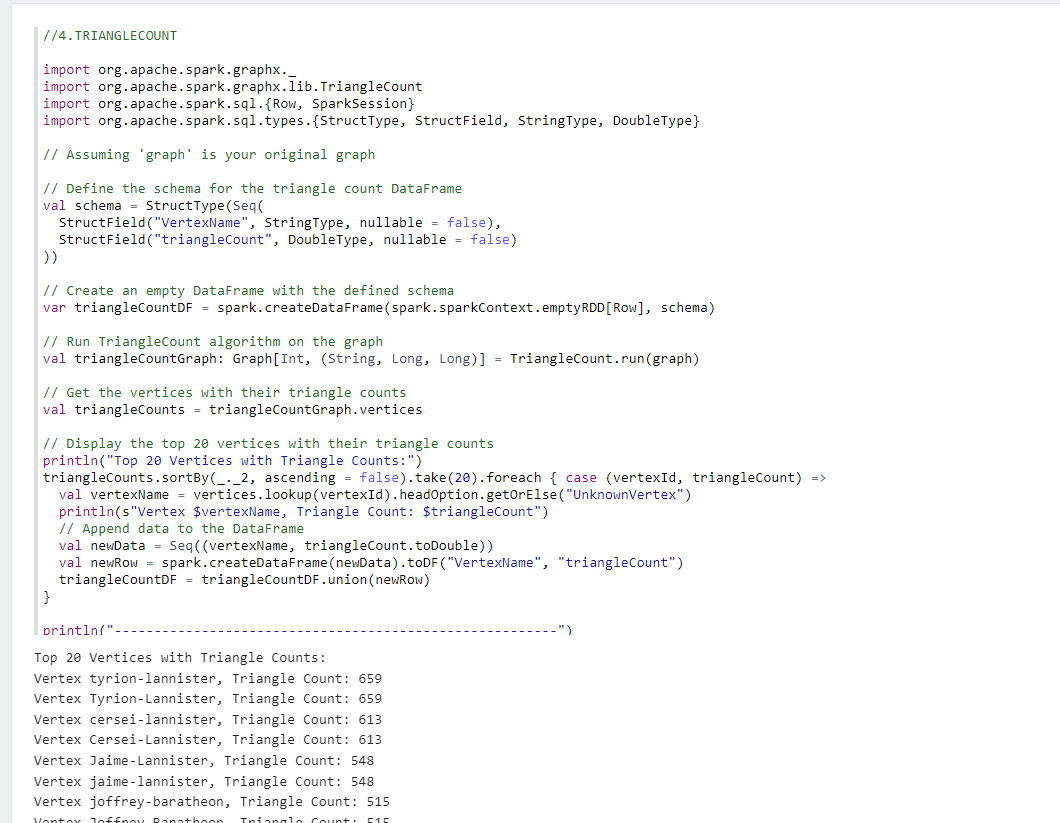


Figure 42 triangle count algorithme

**Sommet le plus Central :** Le sommet "tyrion-lannister" est celui qui participe au plus grand nombre de triangles, avec un triangleCount de 659. Cela suggère que Tyrion Lannister est fortement central dans des structures triangulaires du réseau.

**Interconnexion entre Personnages :** Les sommets les plus élevés dans la liste représentent des personnages qui sont fortement interconnectés avec d'autres personnages du réseau. Par exemple, les membres de la famille Lannister (tyrion-lannister, cersei-lannister, jaime-lannister) et d'autres personnages clés comme sansa-stark, jon-snow, et robert-baratheon sont bien représentés.

**Analyse des Communautés :** Des groupes de sommets interconnectés peuvent indiquer des communautés ou des groupes fortement liés entre eux. Les membres d'une famille ou les alliés politiques peuvent former des sous-groupes avec une participation triangulaire significative.

## Strongly Connected Algorithm

Un algorithme similaire à celui des composantes connexes, mais appliqué aux graphes orientés. Il identifie des sous-graphes dans lesquels chaque nœud est accessible à partir de tous les autres, formant ainsi des composantes fortement connexes.



Figure 43 strongly connected algorithme

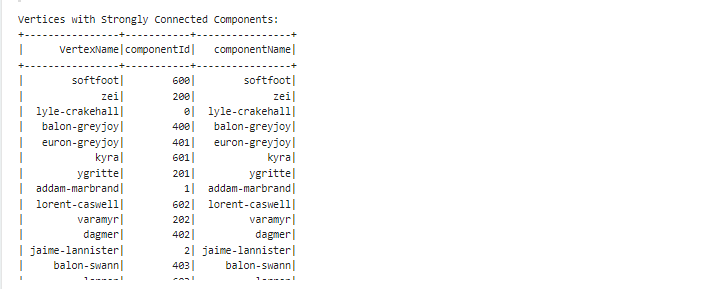


Figure 44 resultat de l'algorithme

# Visualisation des résultats GraphX via tableau :

Maintenant on visualiser les résultats des algorithmes sur GraphX :

* PageRank Algorithm (Algorithme PageRank) :

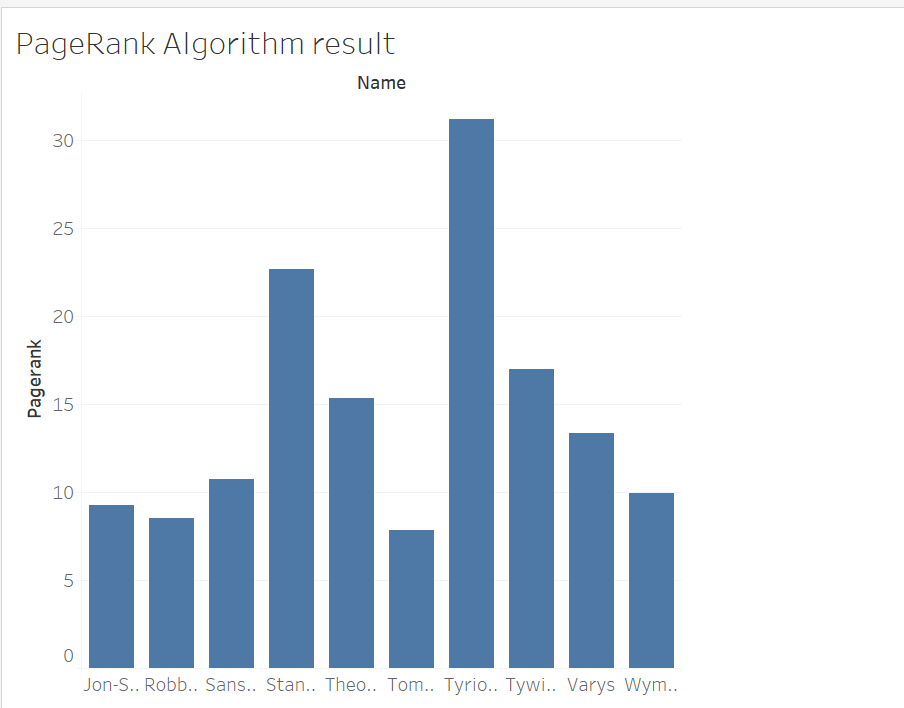


Figure 45 Page Rank algorithme

* Connected Components (Composantes Connexes) :

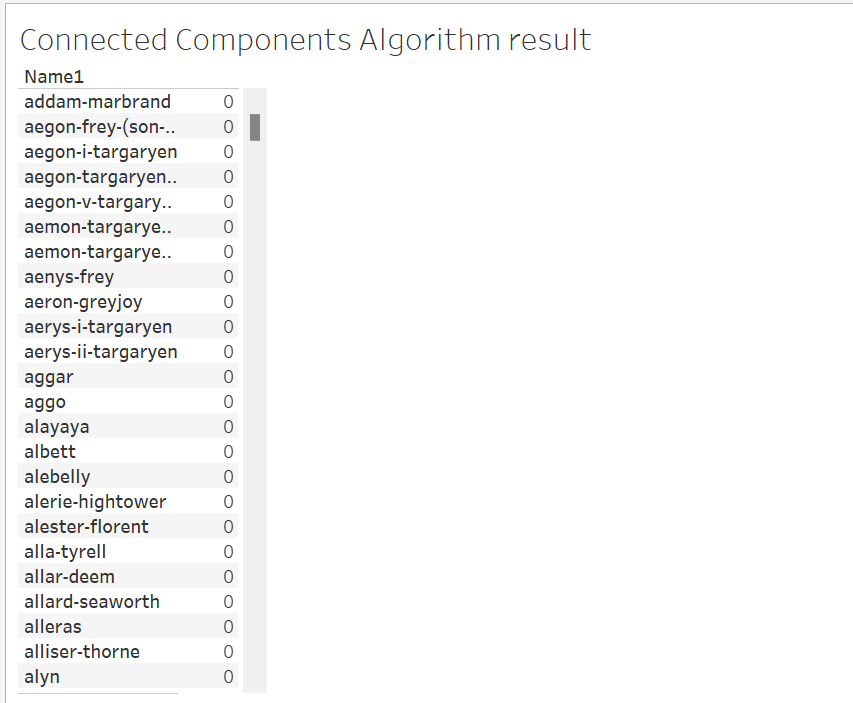


Figure 46 connected algorithme

* Label Propagation Algorithm (Algorithme de Propagation d'Étiquettes) :

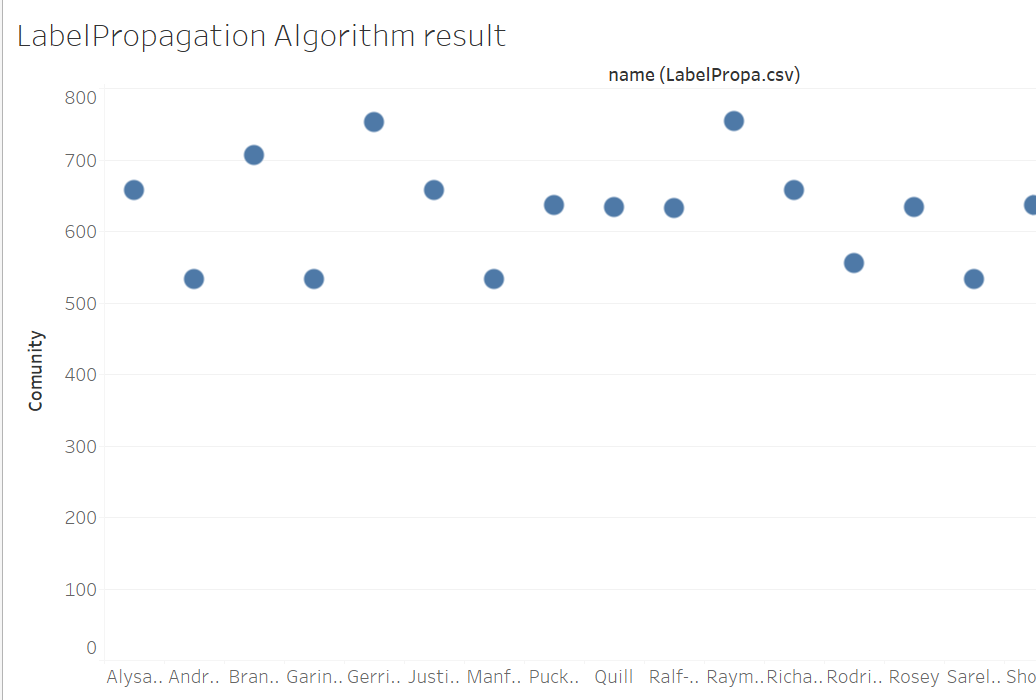


Figure 47 Label Propagation Algorithm

* Triangle Count (Comptage de Triangles) :

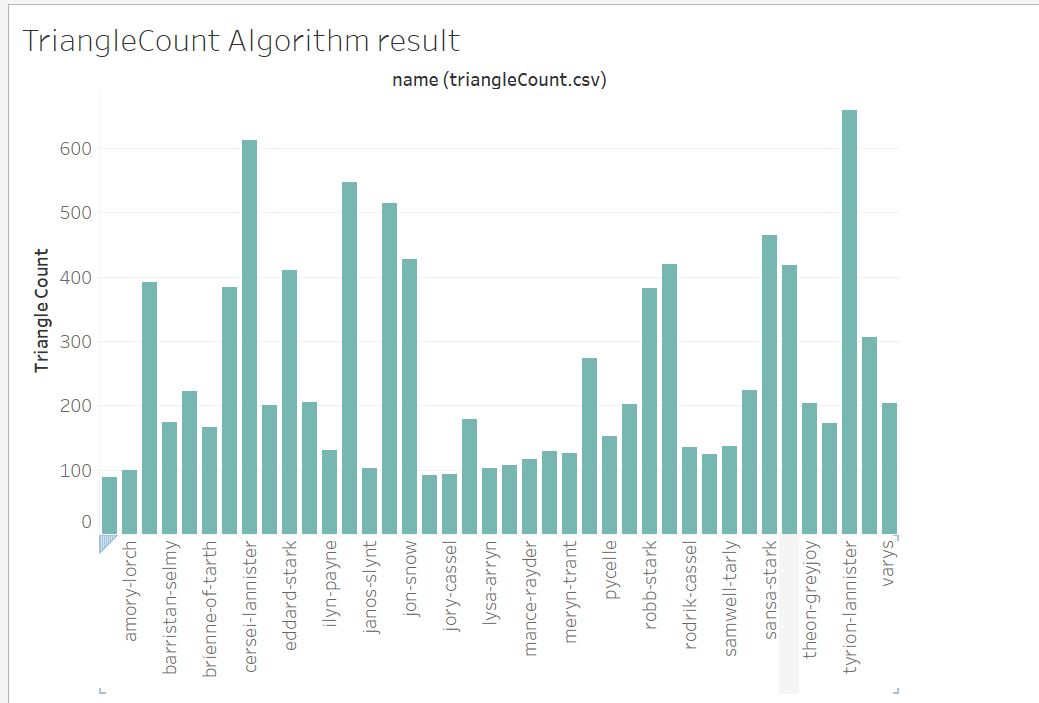


Figure 48 Triangle Count

* Strongly Connected Components (Composantes Fortement Connexes) :

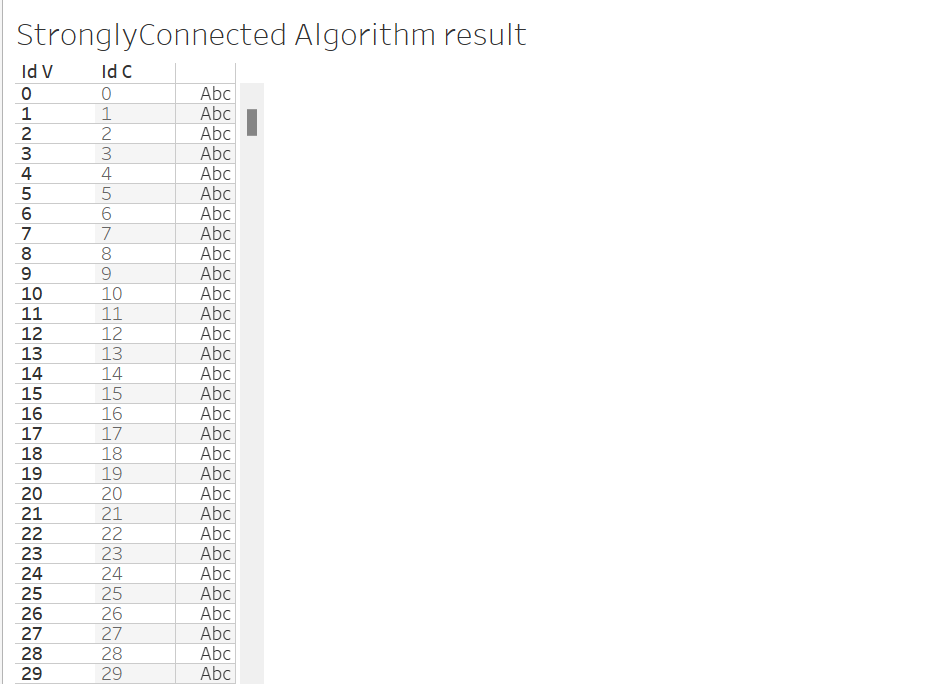


Figure 49 strongly Connected algorithme

* Dashboard :

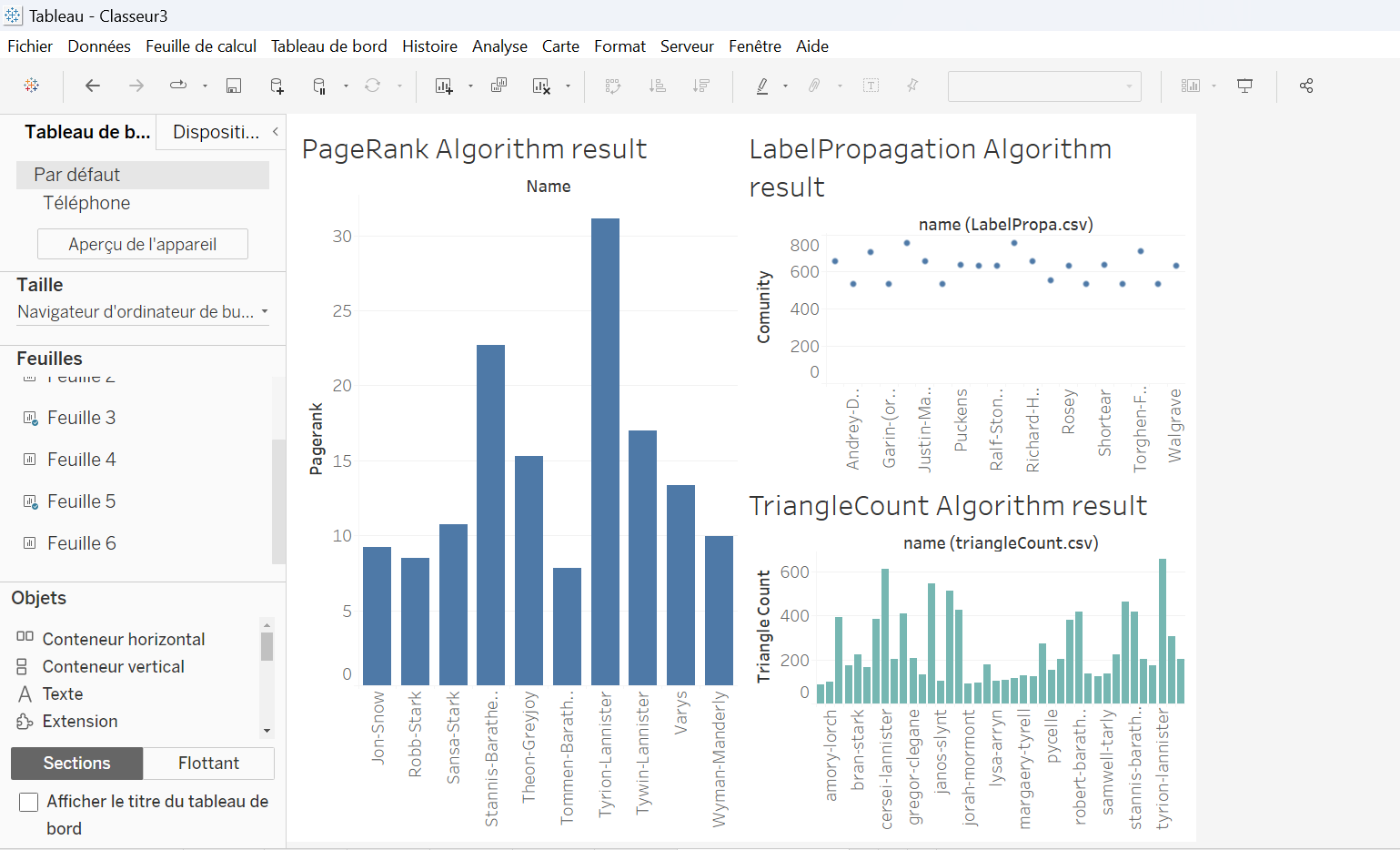


Figure 50 Dashboard

# Spark ML :

Il faut créer un nouveau notebook pour la partie SparkML

## Récupération des données

* Récupération des données d’après neo4j en utilisant pyspark :

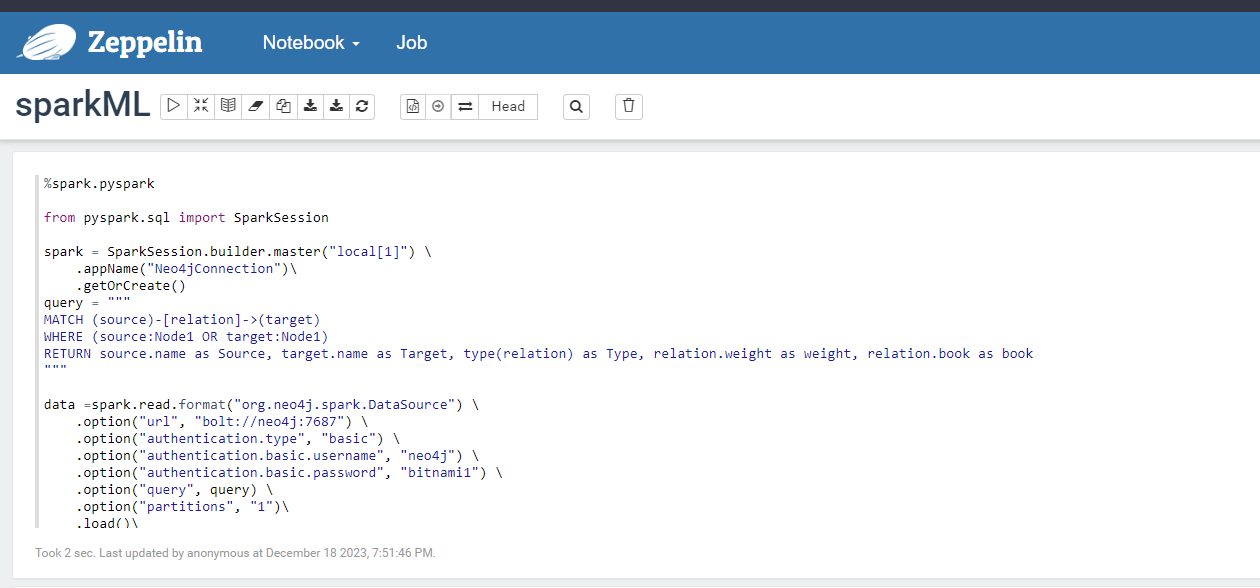


Figure 51 récupération des données

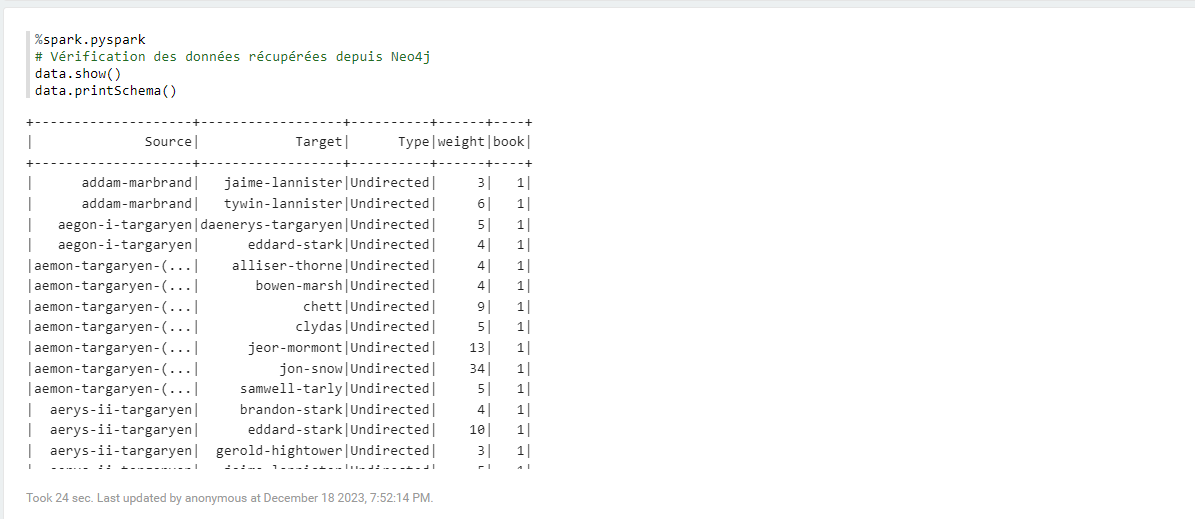
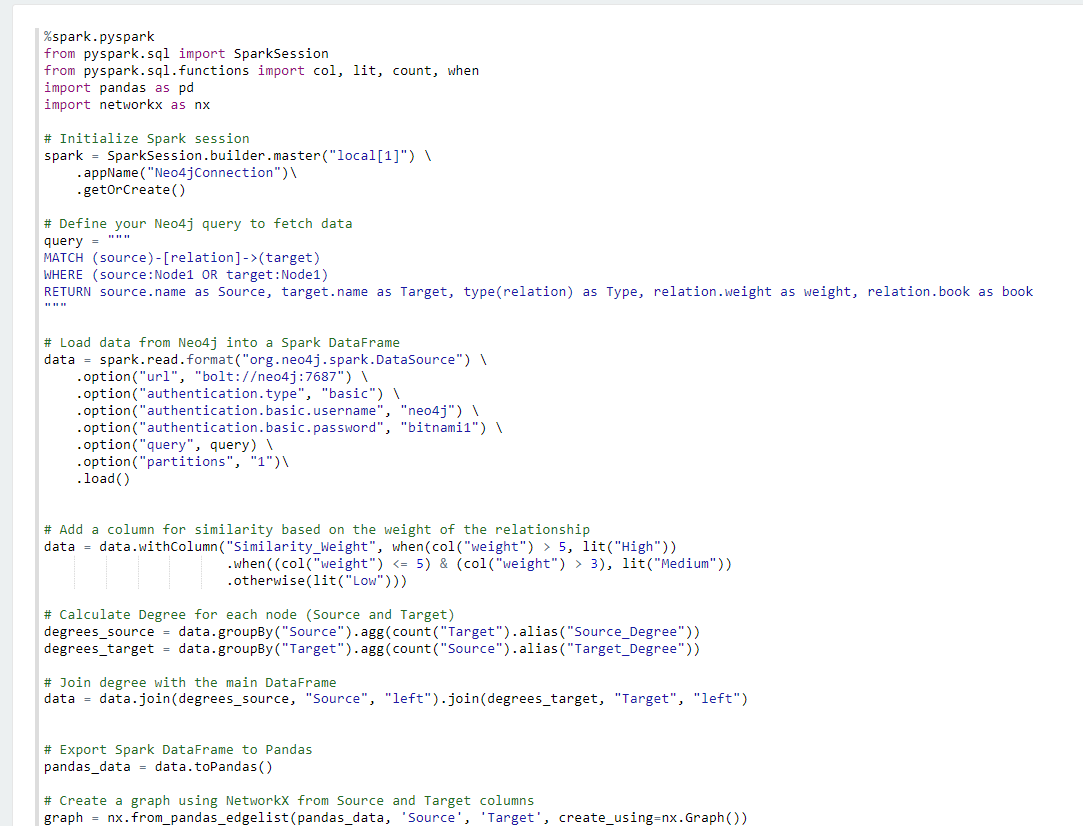


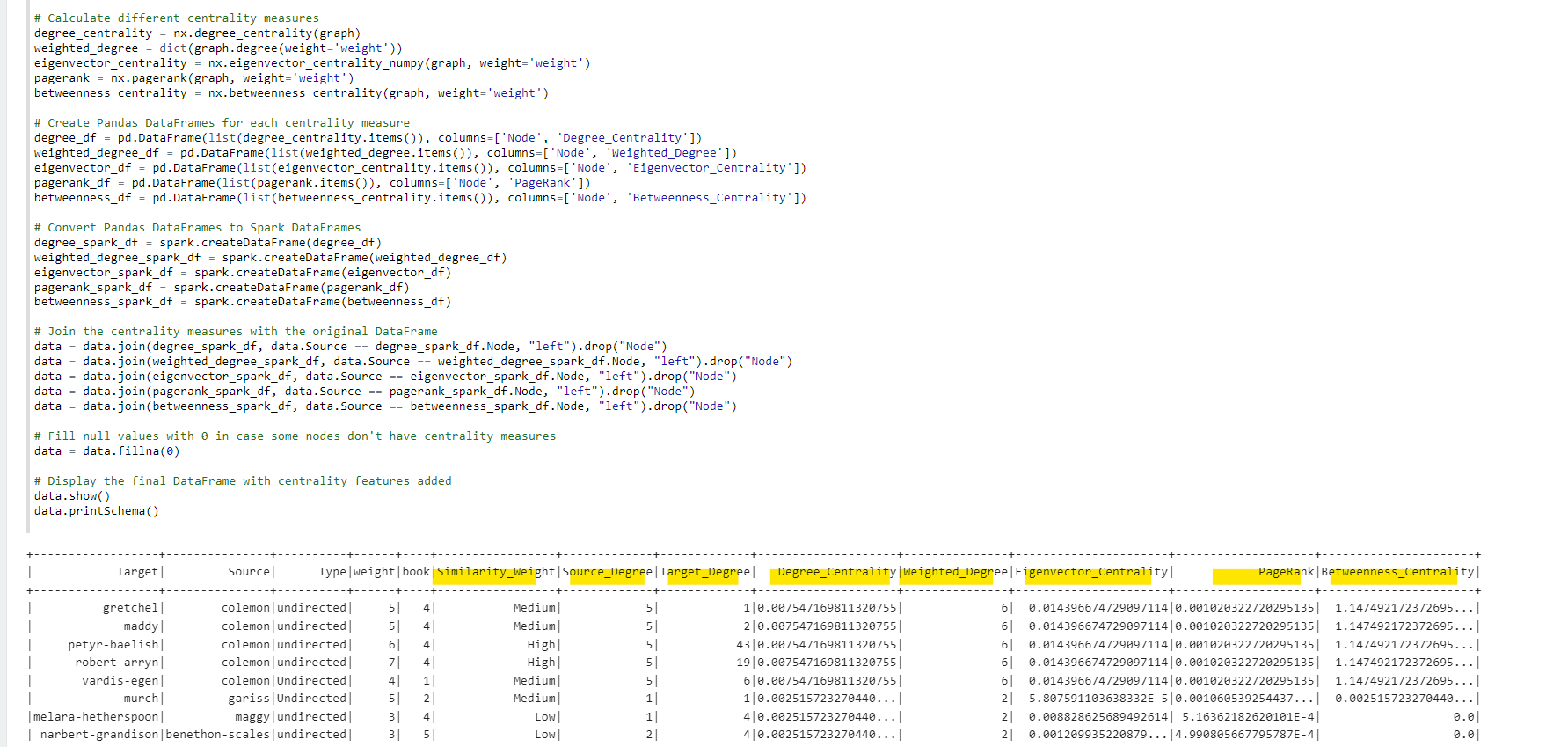
Figure 52 affichage des donées

## Enrichir le dataset

Pour enrichir le jeu de données, vous pouvez ajouter de nouvelles fonctionnalités telles que la similarité des poids ("similarity weight") ainsi que différentes mesures de centralité telles que la centralité degré ("Degree Centrality"), le degré pondéré ("Weighted Degree"), la centralité vectorielle ("Eigenvector Centrality"), le PageRank, et la centralité d'intermédiarité ("Betweenness Centrality").

* **Centralité de Degré (Degree Centrality) :**La centralité de degré mesure le nombre de liens qu'un nœud a dans un réseau. Un nœud avec un degré élevé est considéré comme important car il est directement connecté à un grand nombre d'autres nœuds.
* **Degré Pondéré (Weighted Degree) :**Le degré pondéré est similaire à la centralité de degré, mais il prend en compte les poids des liens. Cela signifie qu'au lieu de simplement compter le nombre de liens, on considère également les valeurs ou les poids associés à ces liens.
* **Centralité de Vecteur Propre (Eigenvector Centrality) :**La centralité de vecteur propre évalue l'importance d'un nœud en tenant compte non seulement de ses liens directs mais aussi de l'importance des nœuds auxquels il est connecté. Les nœuds connectés à d'autres nœuds importants auront une centralité plus élevée.
* **PageRank :**PageRank est une mesure de la importance d'une page Web développée par Google. Dans le contexte des réseaux, PageRank évalue la probabilité qu'un nœud soit visité de manière aléatoire. Les nœuds qui sont liés à beaucoup d'autres nœuds importants ont un score PageRank plus élevé.
* **Centralité d'Intermédiarité (Betweenness Centrality) :**La centralité d'intermédiarité mesure la fréquence à laquelle un nœud agit en tant qu'intermédiaire sur le chemin le plus court entre deux autres nœuds du réseau. Un nœud avec une centralité d'intermédiarité élevée est crucial pour maintenir la communication entre différentes parties du réseau.





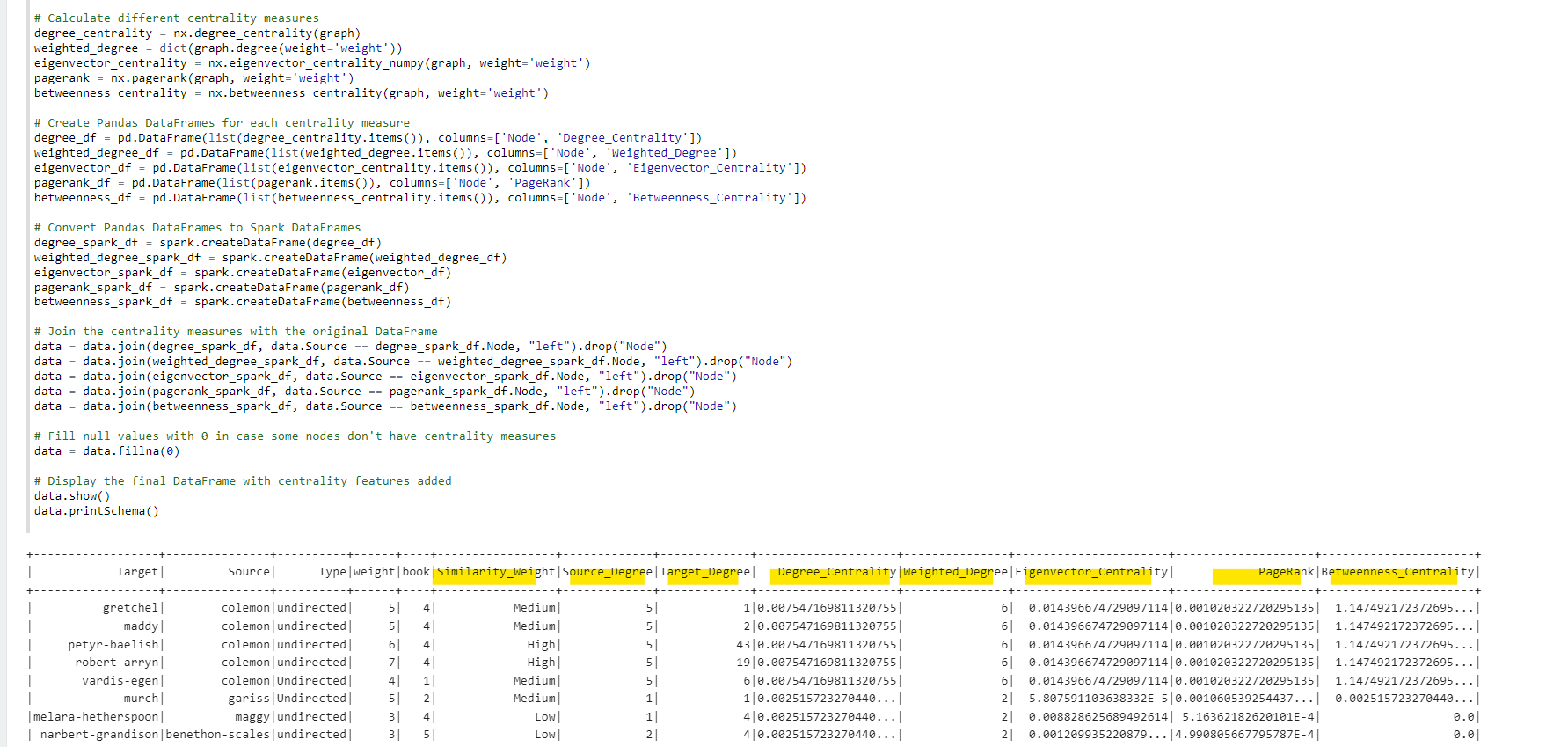


Figure 53 resultat

## Application de quelques algorithms de prédiction :

### Prediction du Degree\_Centrality :

Ce code utilise la régression linéaire pour prédire la centralité degré ('Degree\_Centrality') en utilisant différentes caractéristiques ('weight', 'book', 'Source\_Degree', 'Target\_Degree', 'Weighted\_Degree', 'Eigenvector\_Centrality', 'PageRank', 'Betweenness\_Centrality') du DataFrame. Ensuite, il évalue la performance du modèle en utilisant des métriques telles que MSE, RMSE et R² sur les données de test.

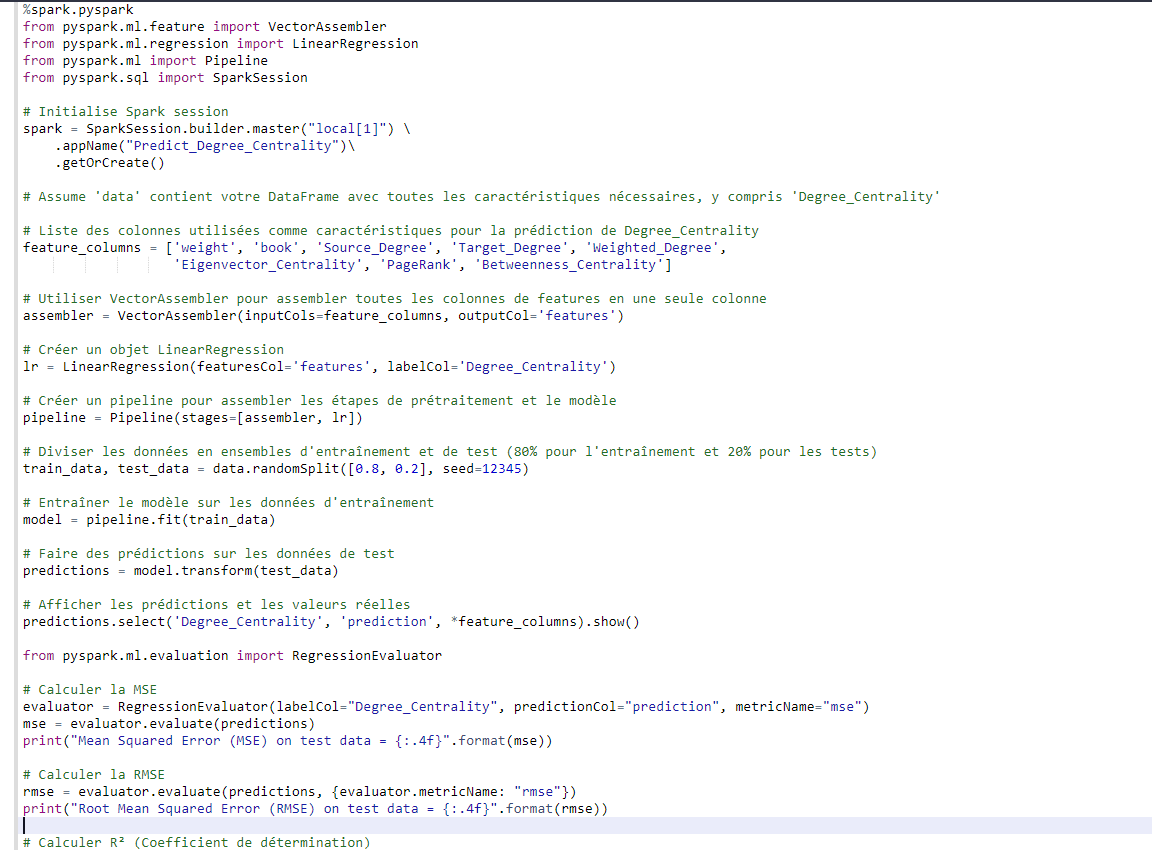


Figure 54 Prediction du Degree\_Centrality

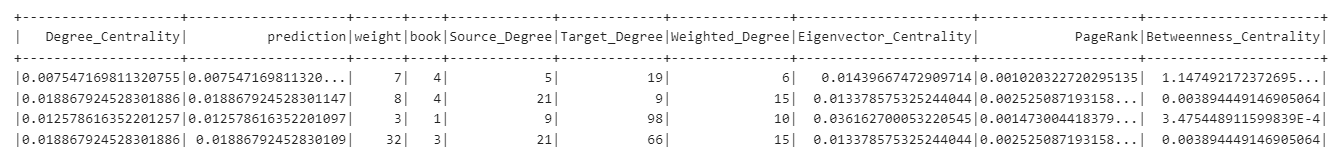


Figure 55 resultat



Figure 56 les métriques du model

### Prediction du PageRank :

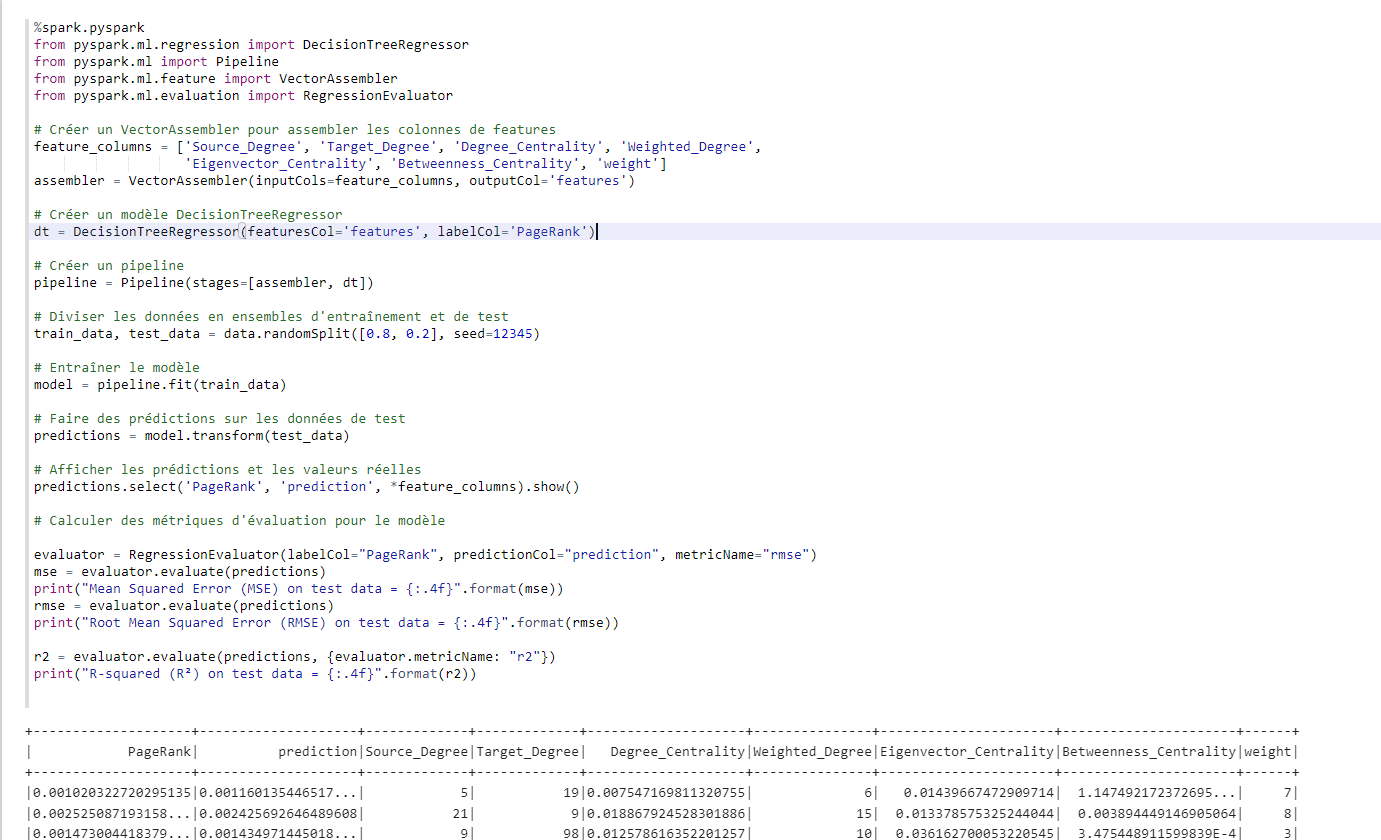


Figure 57 Prediction du PageRank



Figure 58 les metriques du model

### Prediction du Betweenness\_Centrality

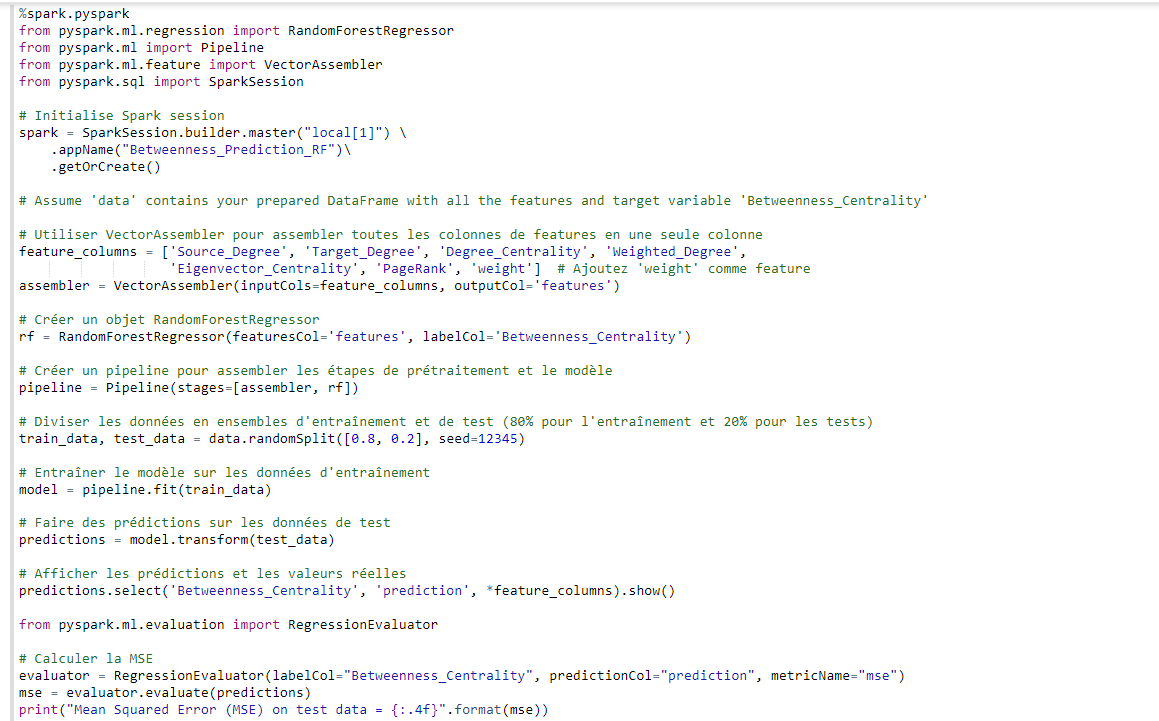


Figure 59 Prediction du Betweenness\_Centrality

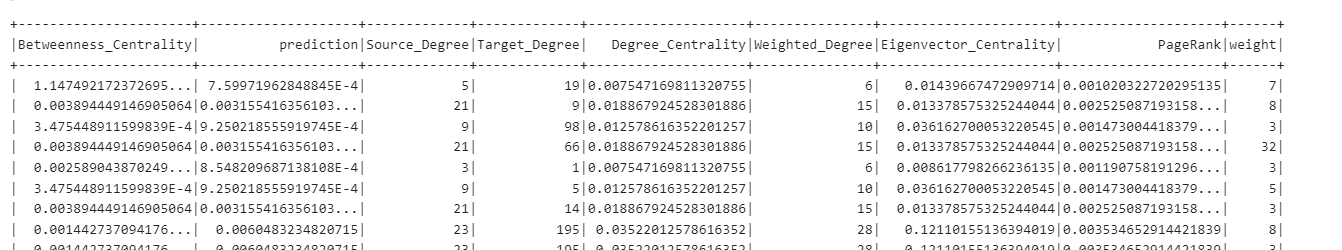


Figure 60 resultat



Figure 61 les metiriques du model

### Prediction du Weighted\_Degree

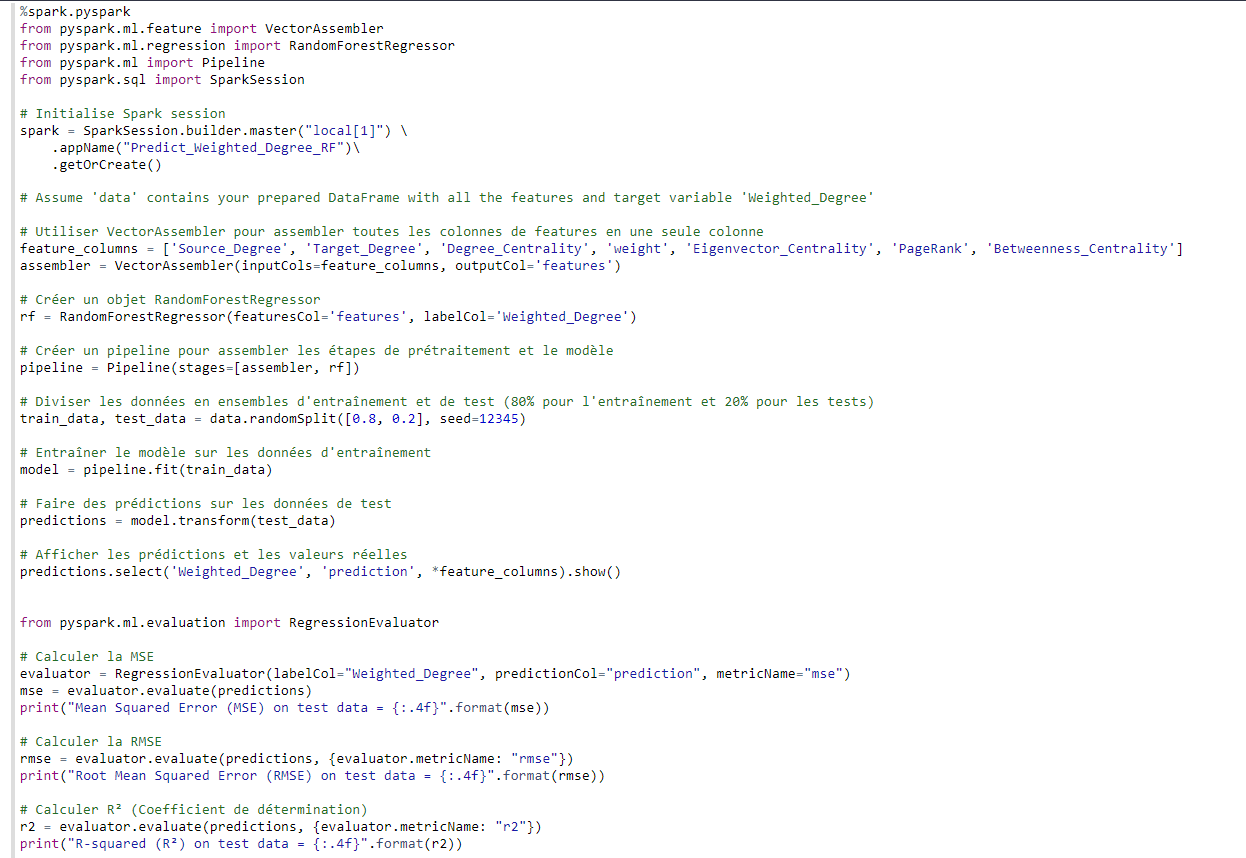


Figure 62 Prediction du Weighted\_Degree

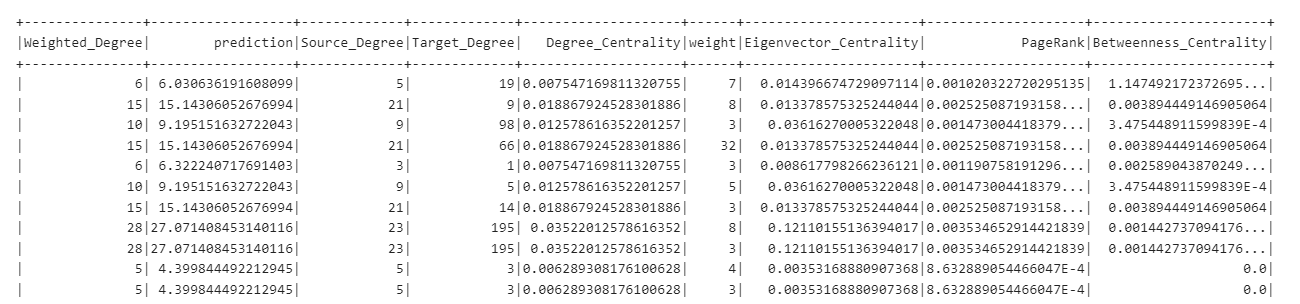


Figure 63 resultat



Figure 64 les métriques

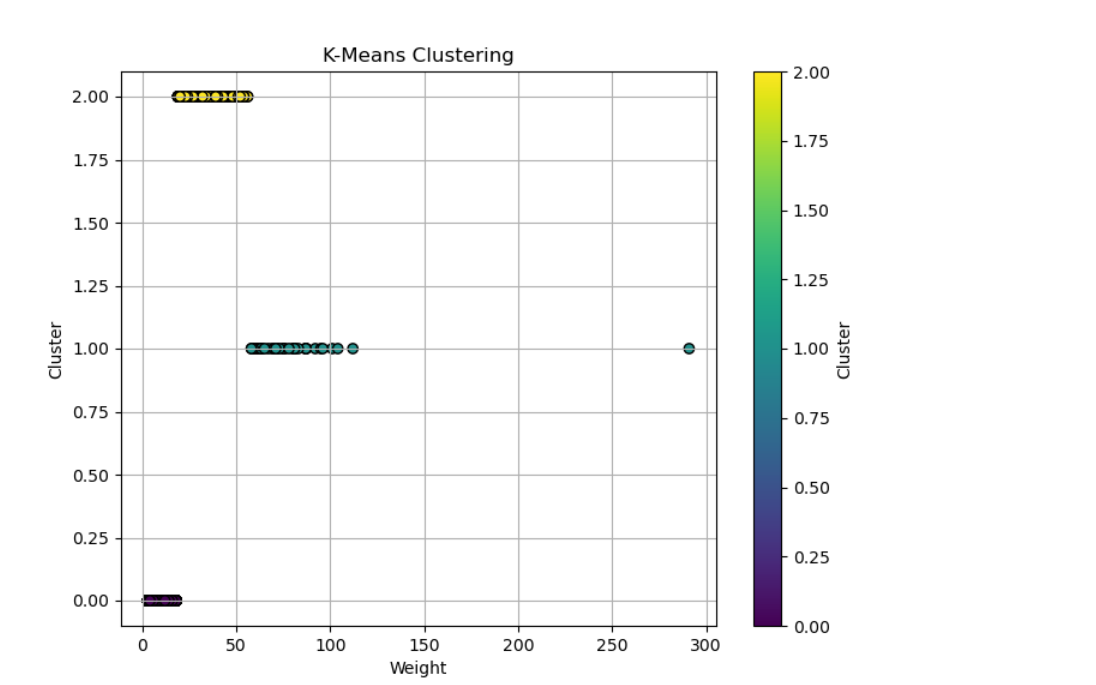
## : Clustring du Weight (KMeans)



Figure 65 le clustring avec Kmeans



Figure 66 affichage des clusters



On a trouvé la presence de 3 clusters des personnes et une valeur aberrant.