

**Forage de données**

Génération des règles d’association

Ayman Chafni & Nick Molgesson Claveus

**Prof :** [Mehdi Adda](https://portail.uqar.ca/user/view.php?id=7245&course=18349)

Le 27/02/2021

Table de matières :

[Introduction 3](#_Toc65345637)

[Partie 1 : 3](#_Toc65345638)

[Préparation du jeu de données 3](#_Toc65345639)

[Le problème à résoudre 3](#_Toc65345640)

[Traitement du Jeu de donnée 3](#_Toc65345641)

[Apriori: 4](#_Toc65345642)

[FP-Growth: 5](#_Toc65345643)

[Les meilleures règles trouvées 6](#_Toc65345644)

[Apriori: 6](#_Toc65345645)

[FP-Growth: 6](#_Toc65345646)

[Performance : 6](#_Toc65345647)

[Partie 2 : 7](#_Toc65345648)

[Pseudocode : 7](#_Toc65345649)

[Code Source : 8](#_Toc65345650)

[Exécution : 14](#_Toc65345651)

[Conclusion : 16](#_Toc65345652)

[Références : 16](#_Toc65345653)

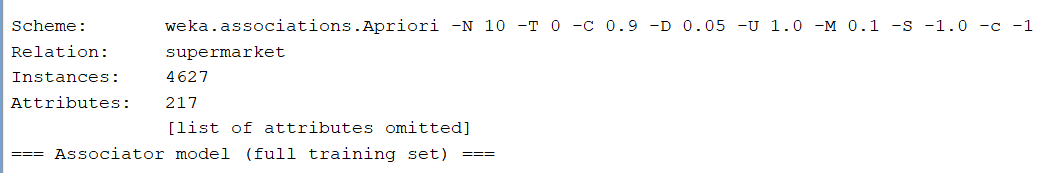
# Introduction

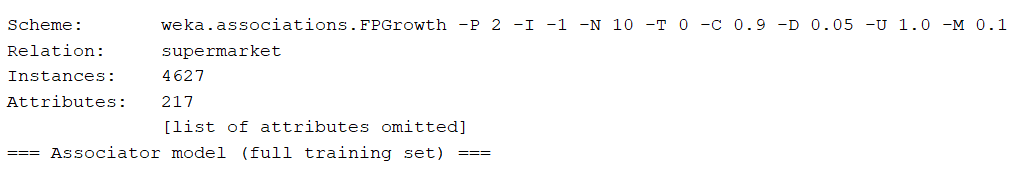
Notre travail consiste à analyser un jeu de données, premièrement avec l’application Weka. Pour cela on a utilisé le jeu de données supermarket pour notre travail. Avec ce jeu de donnée on veut analyser les données pour voir quels sont les produits les plus vendus au supermarché et quelles sont les probabilités qu’une personne va acheter ces produits simultanément. Pour ce faire, on va utiliser les deux algorithmes Apriori et FP Growth, avec une comparaison entre l’implémentation et les performances des deux algorithmes. Dans un deuxième temps, on va implémenter l’algorithme Apriori à zéro avec le langage Python.

# Partie 1 : WEKA

## Préparation du jeu de données

Pource jeu de données on a 4627 données en entrées ou en instance et on a 217 attributs. On a utilisé le concept d’association pour faire le traitement des données.





## Le problème à résoudre

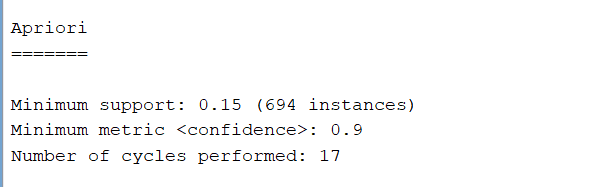
Encore connue sous le nom d’analyse d’affinité ou analyse du panier de consommation, la tâche d’association en fouille de données vise à voir quelles sont les variables qui vont ensemble. Parmi les algorithmes d’association

## 

## Traitement du Jeu de donnée

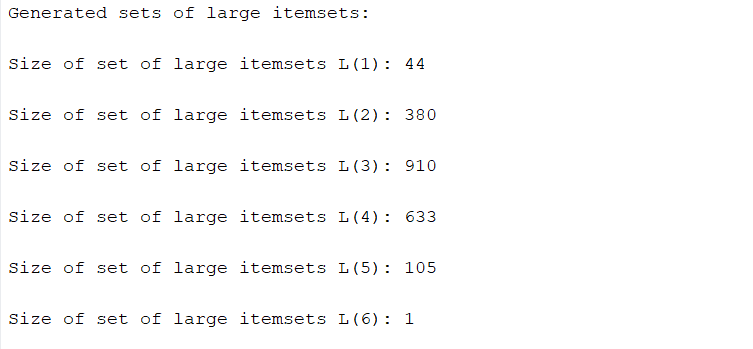
Nous avons utilisé l’algorithme Apriori et l’algorithme FP Growth pour faire l'analyse du jeu de données. Il s’agit de trouver des règles du type si X alors Y avec un certain niveau de probabilité. Deux métriques sont utilisées pour caractériser généralement la qualité d’une règle d’association : le support et la confiance. Le support décrit la probabilité d’existence de X et Y au sein du jeu de données, dans l’analyse du jeu de donnée le minimum support est de 0.19 qui respecte les 694 instances traitées. La confiance décrit quant à elle la probabilité d’existence de Y dans l’ensemble de données contenant X. Le minimum confiance est 0.9.

### Apriori:



Dans la figure ci-dessous vous voyez les ensembles d’itemsets générés par l’algorithme d’Apriori. Ces ensembles L(K) contiennent tous les itemsets fréquents de taille K, c’est-à-dire qu’un itemset de cet ensemble est acheté plus que 15% de fois (min support= 0.15)

* 44 des 1-itemsets sont fréquents
* 380 des 2-itemsets sont fréquents
* 910 des 3-itemset sont fréquents
* 633 des 4-itemset sont fréquents
* 105 des 5-itemsets sont fréquents
* 1 des 6-itemsets sont fréquents



Dans la deuxième partie, on donne plus de détails sur le fonctionnement de l’algorithme Apriori.

### FP-Growth:

FP-Growth est un algorithme qui utilise la stratégie "diviser pour régner" (divide-and-conquer). Il compresse les itemsets fréquents représentés dans la base de données sous la forme d'un arbre appelé arbre de modèle fréquent ou arbre FP dont les branches représentent toutes les associations fréquentes probables.

En fait, FP-Growth est basé sur la propriété suivante : Un sous-Itemset d’un itemset fréquent est un itemset fréquent, et son dual : un sur-itemset d’un itemset non fréquent est non fréquent.

Au lieu de générer des itemsets et parcourir la base de données ou la liste de transactions à chaque fois pour vérifier leurs fréquences comme dans le cas d’Apriori, FP-Growth ne parcourt la liste des transactions que deux fois. La première est pour extraire tous les items singuliers fréquents. La deuxième est pour extraire de chaque transaction l’itemset le plus grand contenant les items extraits dans le premier parcours.

Ensuite, on trie les items de chaque itemset selon un ordre décroissant du support. L'étape suivante serait de construire l’arbre, la racine de cet arbre contient la valeur nulle. Chaque autre nœud contient les items singuliers avec leurs 'frequency count’ dans la séquence où il appartient. On commence par la suite par prendre chaque itemset qui est déjà ordonné et ajouter ses items dans l’ordre dans l’arbre d'une façon à créer une branche dont chaque sous-branche est un itemset fréquent. A chaque fois qu’une séquence d’items se répète ( existe dans un autre itemset ) on incrémente le ‘frequency count’ pour ses items dans l’arbre. Si l’itemset ne contient aucune séquence qui se répète, on crée la séquence avec ‘frequency count’ égal à 1.

i)Après la création de notre arbre FP, on construit la ‘conditional pattern base’, cad, pour chaque item singulier fréquent, on liste les séquences qui lui ont donné lieu, avec le 'frequency count’ dans la séquence où il appartient.

Ensuite, on construit le ‘conditional FP tree’, on prend chaque ligne dans notre ‘conditional pattern base’, et on en extrait les séquences communes entre ses éléments avec l’addition de ses 'frequency count’.

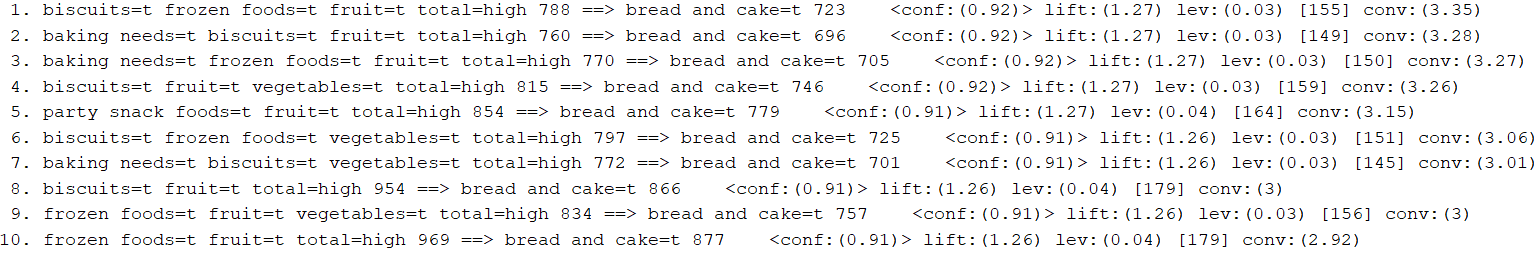
Comme dernière étape, on ajoute à ces séquences l’item qui les a générées dans l'étape i). Chaque combinaison possible (nombre items > 1) du résultat obtenu est un itemset fréquent avec comme ‘frequency count’ de tous ces itemsets le nombre stocké avec cette séquence.

Il ne nous reste maintenant que de construire les règles d’association, dont le processus de génération est élaboré en détail dans la deuxième partie.

## Les meilleures règles trouvées

On a trouvé les mêmes résultats pour les deux algorithmes.

### Apriori:



### FP-Growth:

On peut mettre à profit les techniques d’association pour déceler les liens éventuels entre les différents produits vendus dans un supermarché. On peut ainsi noter que chaque fois que les biscuits, frozen foods, fruits sont achetés par 788 personnes, 722 personnes sont également achetées bread and cake. On note donc une certaine association entre ses produits avec 92% de taux de confiance. On pourra entreprendre comme action de disposer le rayon « biscuits, frozen foods, fruits » à proximité de celui concernant la « bread and cake » afin d’amener le client à ne pas fournir d’effort pour aller dans le rayon « bread and cake » ou que celui-ci se rappelle qu’il doit acheter des bread and cakes au cas où il aurait oublié.

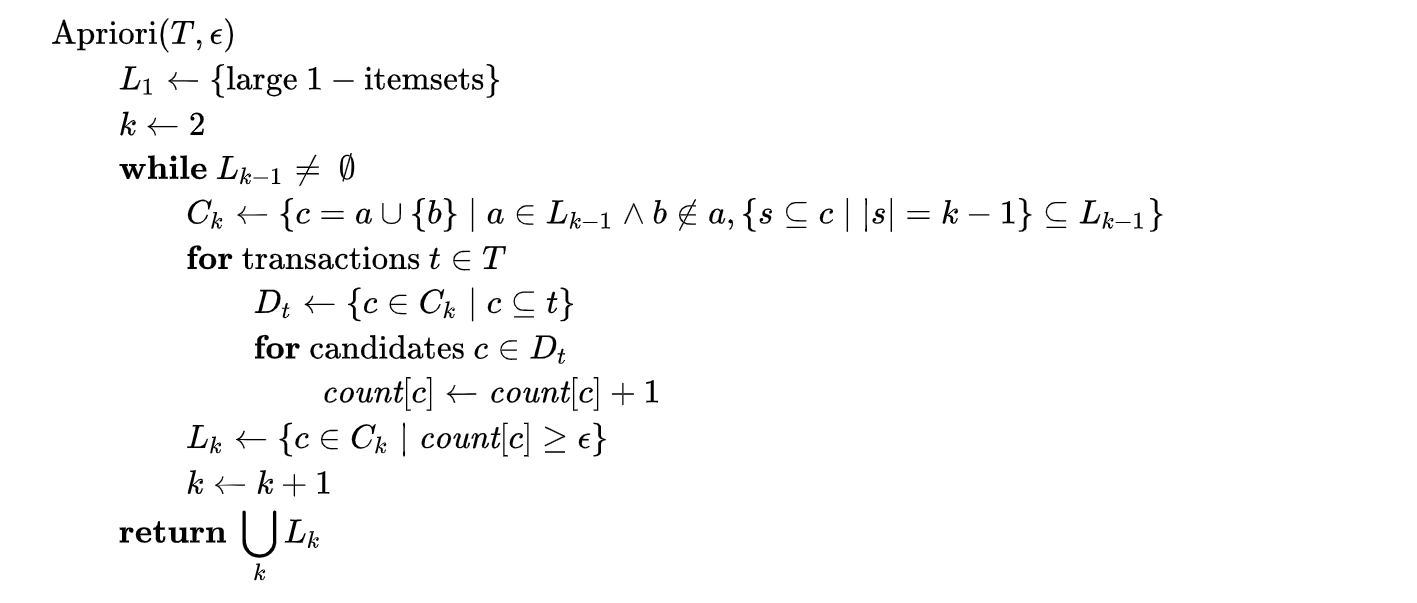
## 

## Performance :

Plusieurs études ont démontré que l’algorithme FP-Growth est plus rapide par rapport à Apriori. Le temps d'exécution du processus augmente linéairement avec l'augmentation du nombre d'itemsets tandis que le processus d’Apriori est comparativement plus lent avec un temps d'exécution qui augmente de façon exponentielle avec l'augmentation du nombre d'itemsets. Cependant, ce n’est pas toujours le cas. Lorsque la base de données est très volumineuse, l'algorithme FP-Growth peut ne pas tenir dans la mémoire partagée.

# Partie 2 : Implémentation d’Apriori

## Pseudocode :



Interprétation :

L’algorithme Apriori permet de trouver des règles d’association à partir des données sous formes d’ensembles de transaction, son fonctionnement est assez simple :

Dans un premier lieu, on construit un ensemble L1 contenant tous les subsets de longueur égale à 1 des transactions sans qu’on répète une subset, en second lieu, on commence à itérer cet ensemble en prenant à la fois un itemset de L(k-1), dont les itemsets sont de tailles k-1, et en ajoutant à cet itemset a chaque fois un item de L1 qui n’existe pas dans L(k- 1), on obtient par la suite un nouveau itemset de longueur k qu’on met dans un ensemble Ck à condition que tous ses sous-ensembles de longueur k- 1 sont fréquents (c.à.d., ils appartiennent à L(k-1)), parce qu’on sait qu’un itemset non fréquent, ses super-itemset sont aussi non fréquents. Du coup, cela nous permet d’alléger le temps de calcul. Par la suite, on prend les itemsets de Ck qui ont un support supérieur au minSupp et on les met dans l’ensemble Lk et on incrémente k. On arrête quand L(k-1) est vide, c.à.d. que les itemsets générés de taille k-1 sont non fréquents. On retourne à la fin l’ensemble de tous les Lk.

Ceci n’est que la première partie de l’algorithme (représenté par le pseudo code ci-dessus), qui permet d’obtenir tous les itemsets fréquents de l’ensemble des transactions. Maintenant, il faut obtenir la liste des règles d’association à partir de ces itemsets. Pour cela, on aura besoin d’un autre paramètre qui est le minimum de confiance des règles d’association (cet algorithme prend alors trois paramètres : le minimum support, le minimum de confiance et l’ensemble des transactions). Dans cette deuxième partie, on prend à la fois un Lk (k>1) et pour chaque itemset de Lk, on itère sur les subsets de cet itemset : pour chaque itération, on calcule le support de l’itemset divisé par le support du subset, si le rapport trouvé est supérieur ou égal à notre minimum de confiance, nous créons alors une règle qui lie ce subset au reste de l’itemset (->) avec une la confiance calculée.

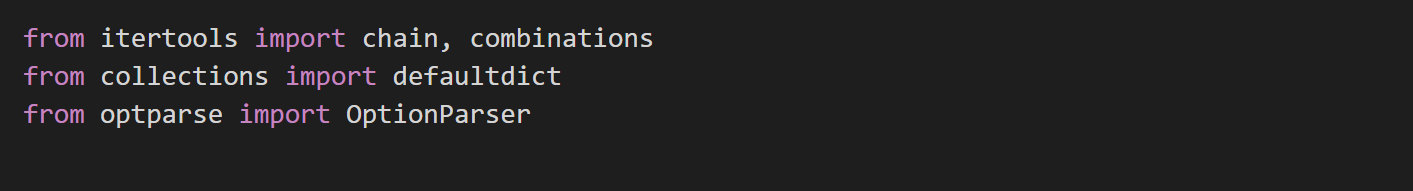
## Code Source :

Ce code est importé depuis un repo sur GitHub, c’est celui qu’on a utilisé avec des petites modifications. Notre travail consistait à vraiment comprendre son fonctionnement avant de l’utiliser. On a envisagé l’utilisation de ce code car il est bien factorisé et même en reproduisant notre propre version, elle ne serait pas très différente car le principe est le même.

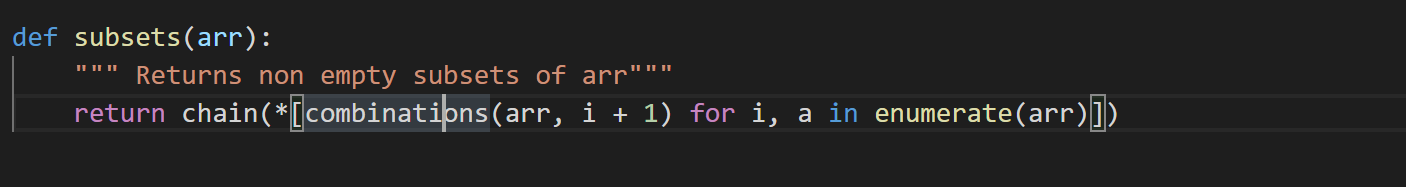
Dans ce qui suit, une explication du rôle de chaque fonction ou portion de code.

Dans un premier temps, on importe les modules nécessaires :

* chain et combinations : pour la manipulation et le découpage des listes
* defaultdict : pour la création et la manipulation des dictionnaires
* OptionParser : pour pouvoir exécuter le fichier avec des paramètres à injecter dans la ligne de commande



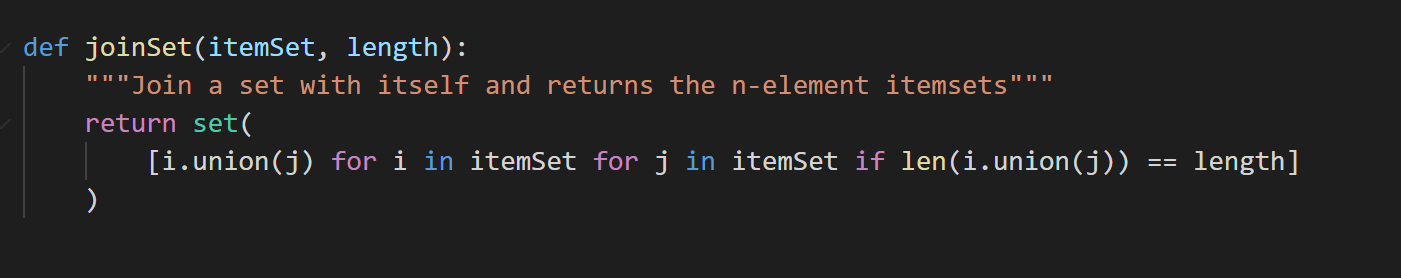
‘subsets’ est une fonction qui prend en paramètre une liste et retourne toutes les combinaisons de toute taille des éléments de cette liste.



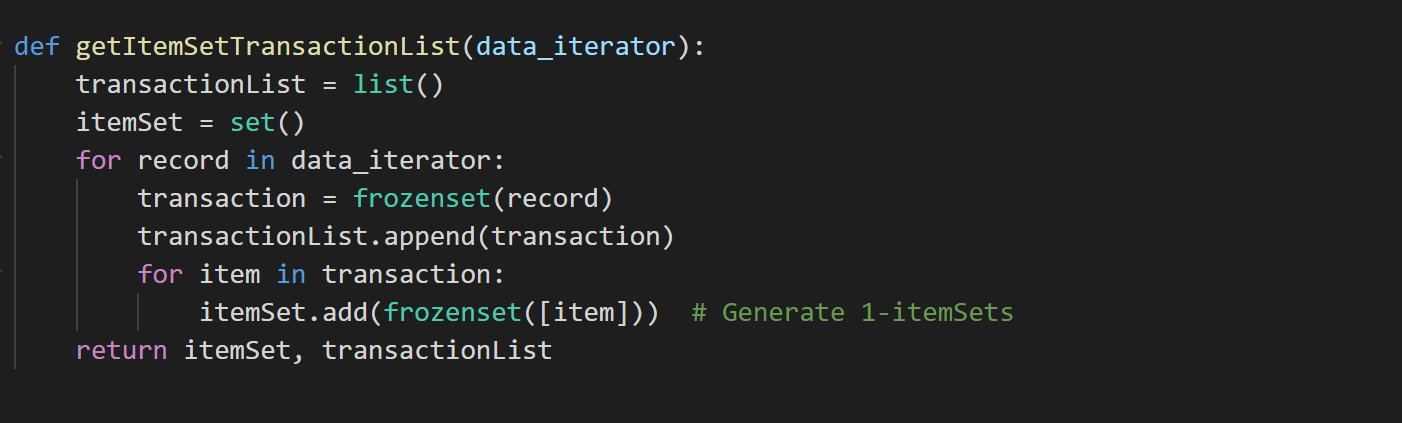
‘returnItemWithMinSupport’ est une fonction qui prend un itemset Lk qui regroupe les itemsets de taille k générés à partir d’itemsets de tailles k-1 en ajoutant un unique item et à chaque fois qu’on trouve que itemset de Lk est un sous-itemset d’une transaction de transactionList, on a un dictionnaire déclaré globale freqSet où on incrémente la valeur liée a l’itemset, on a aussi un dictionnaire déclaré local qui est un dictionnaire qui enregistre les fréquences aussi et que l’on utilise pour alléger les vérifications lorsqu’on veut calculer le support des itemsets de Lk. Ensuite, on vérifie si chaque itemset respecte le la condition du minSupport, si c’est le cas, on l’ajoute à un ensemble temporaire \_itemSet, et on retourne ce dernier.



‘joinSet’ est une fonction qui prend en paramètre un itemset et un entier length, et retourne l’union de cet itemset avec lui-même a condition que l’union soit de taille length. L’union est sans répétition des items. On va l’utiliser lors de la génération de nouveaux itemsets en ajoutant un item. Cette méthode nous permet de créer des itemsets tout en s’assurant que tous les sous-itemsets sont fréquents.

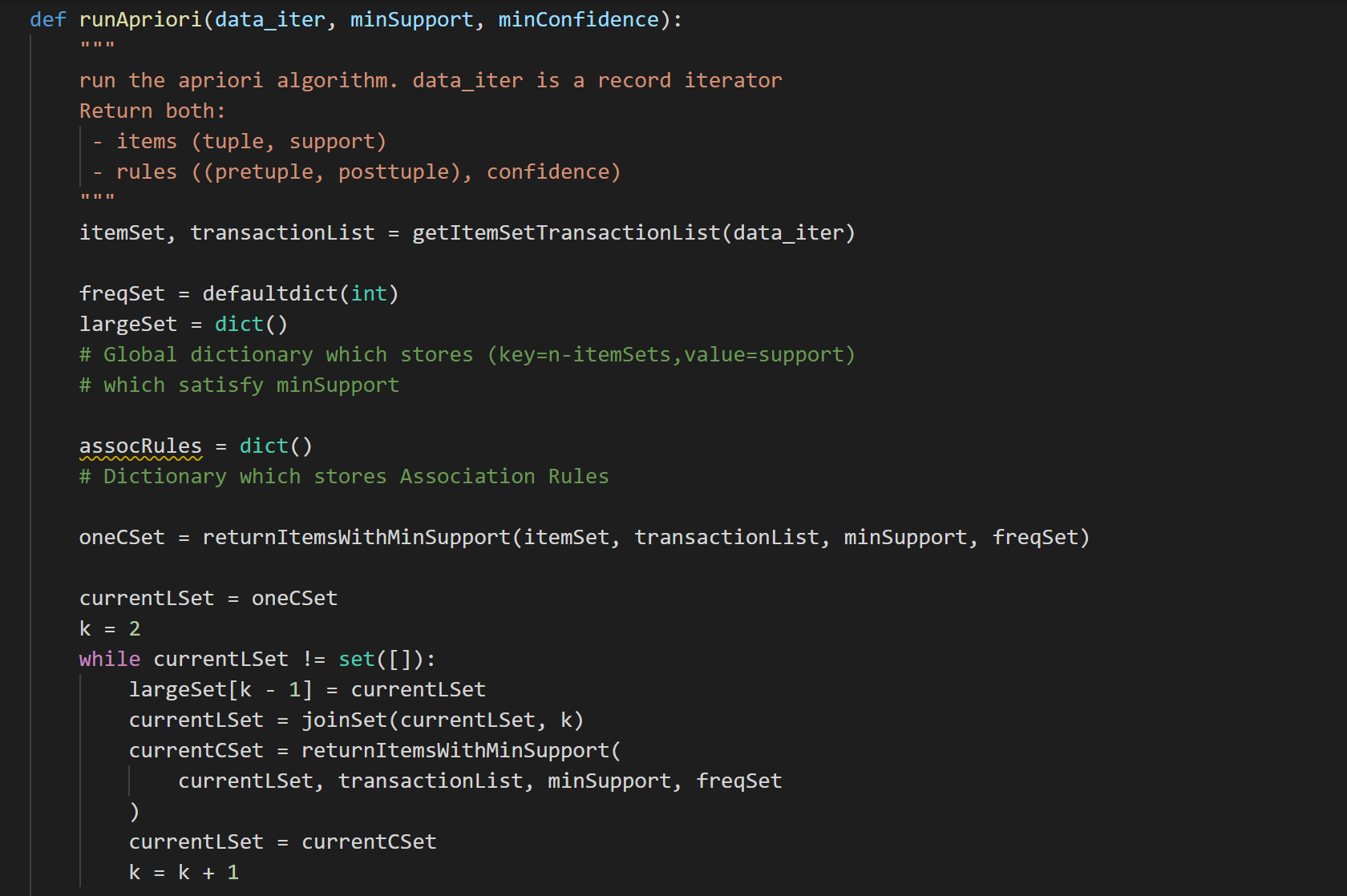


Cette fonction prend comme argument les lignes de transactions lus à partir du fichier et retourne une liste qui comporte les transactions et un ensemble qui comporte les 1-itemsets (unitaires).



La fonction ‘runApriori’ contient comme son nom l’indique l’implémentation de l’algorithme Apriori, qui se fait comme mentionné plus haut en deux étapes :

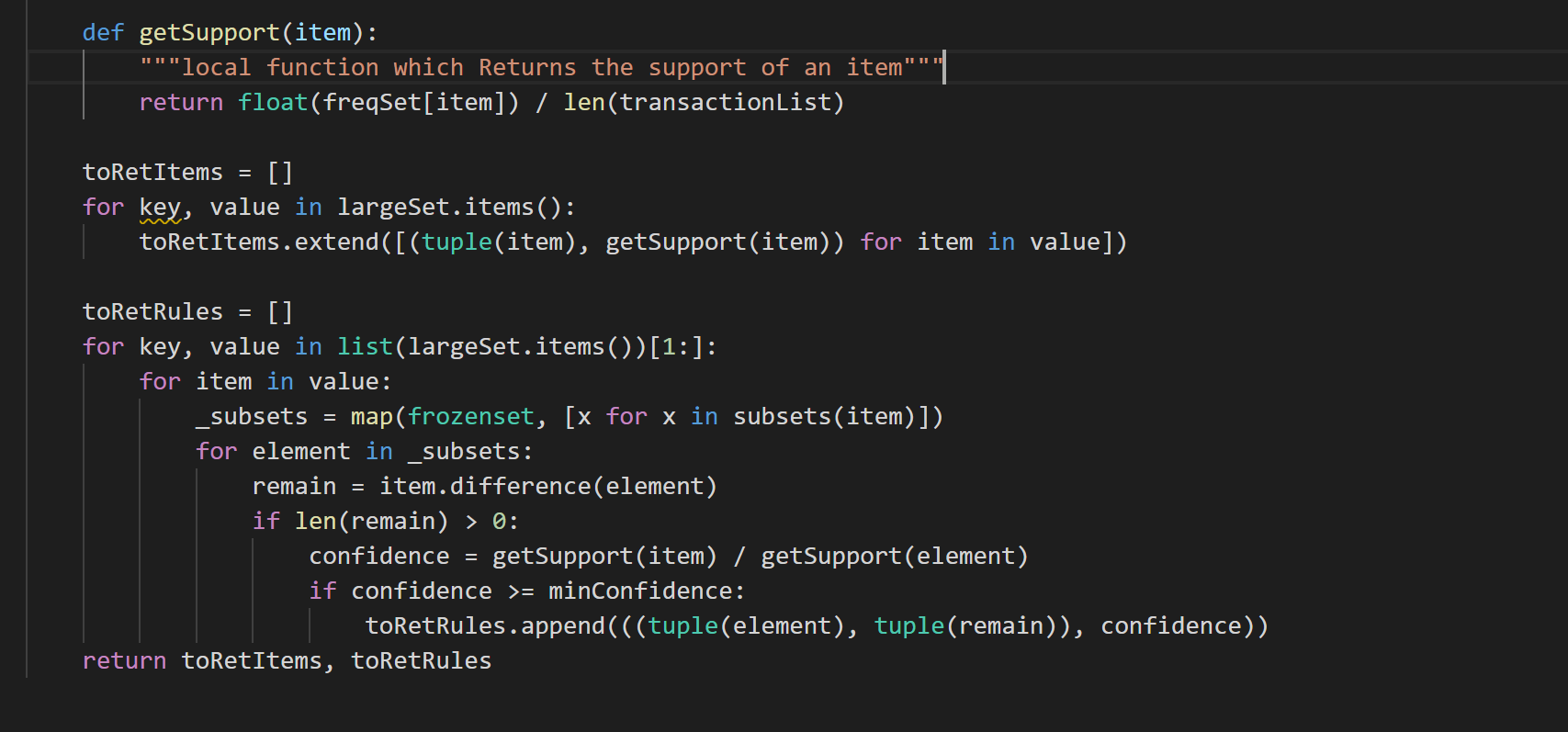
La première étape consiste à construire un dictionnaire ‘largeSet’ contenant tous les itemsets fréquents avec comme clés les tailles des itemsets (k) et comme valeurs les ensembles d’itemsets fréquents de même taille Lk.



A l’intérieur de cette fonction, on définit une fonction local ’getSupport’ qui permet de calculer le support d’un itemset et ce utilisant le dictionnaire global ‘freqSet’.

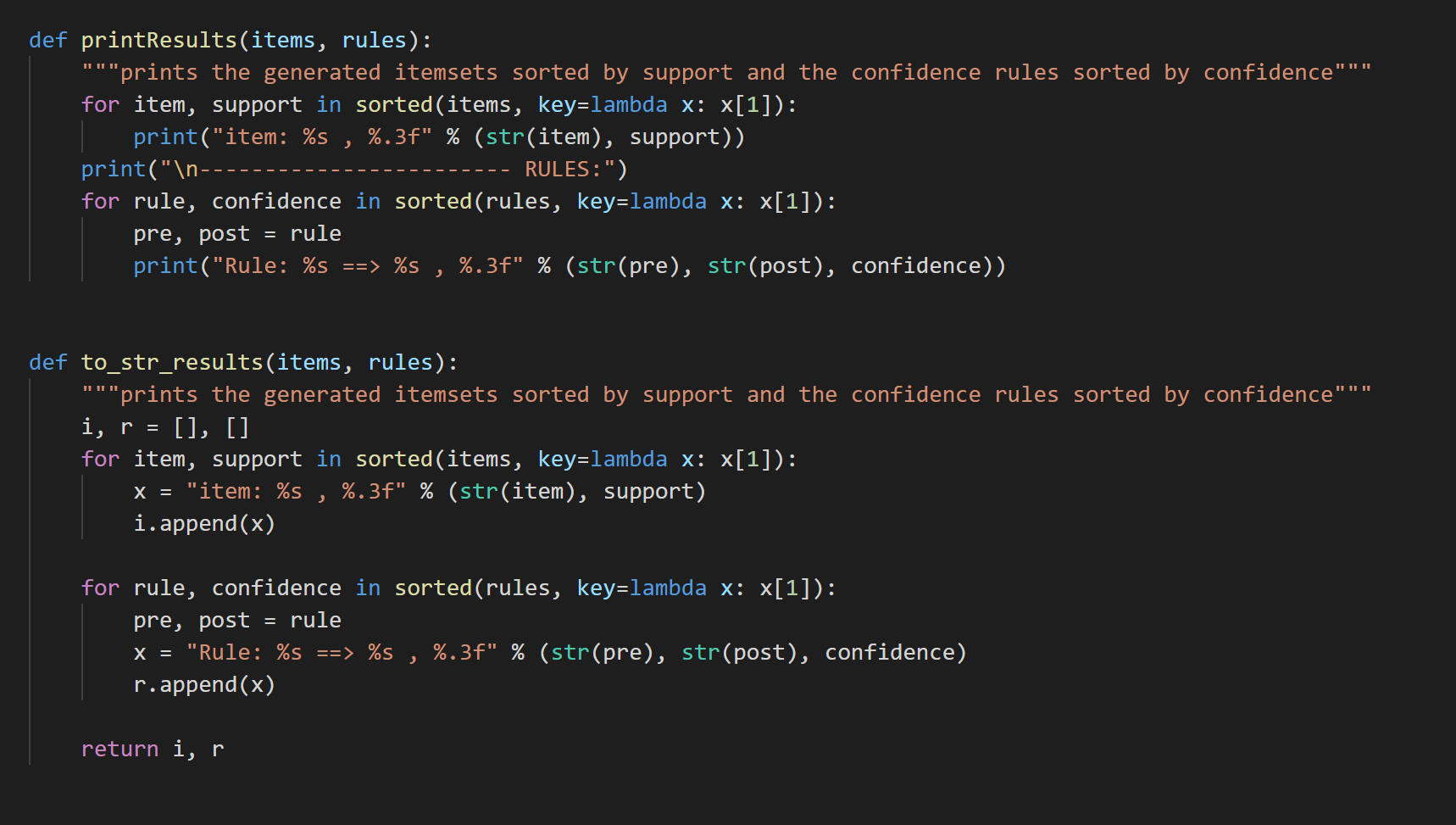
On utilise une liste toRetItems pour sauvegarder les itemsets de ‘largeSet’ avec leurs supports, on le fait pour des buts d’affichage, pour que les résultats soient tries selon le support lors de l’affichage.

La deuxième étape d’Apriori est l’étape de recherche de règles d’associations. Pour ce faire, on prend à chaque fois un Lk (k > 1, car aucune information ne peut être extraite à partir des 1-itemsets) et pour chaque itemset dans Lk, on prend a chaque fois un de ses subsets qu’on retrouve par la méthode ‘subsets’ et on calcul la confiance de la règle « subset => reste (reste du subset pour compléter l’itemset) » et pour cela, il suffit de diviser le support de l’itemset sur le support du subset. Si la règle satisfait la condition de la confiance, on l’ajoute à la liste toRetRules.

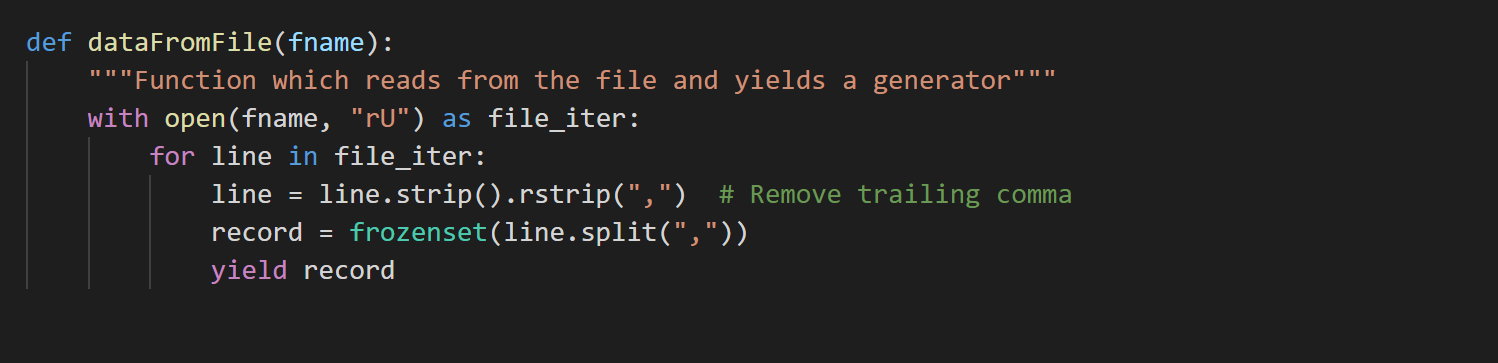


Pour l’affichage des résultats, on utilise la fonction ‘printResults’ qui permet d’afficher les itemsets fréquents par ordre croissant en fonction du support grâce à la fonction sorted () où on précise qu’on va trier suivant le deuxième élément de la liste par le paramètre ‘key’ (lambda est pour déclarer une fonction anonyme).

La fonction ‘to\_str\_results’ a le même fonctionnement mais elle retourne les itemsets fréquents et les règles dans des listes.

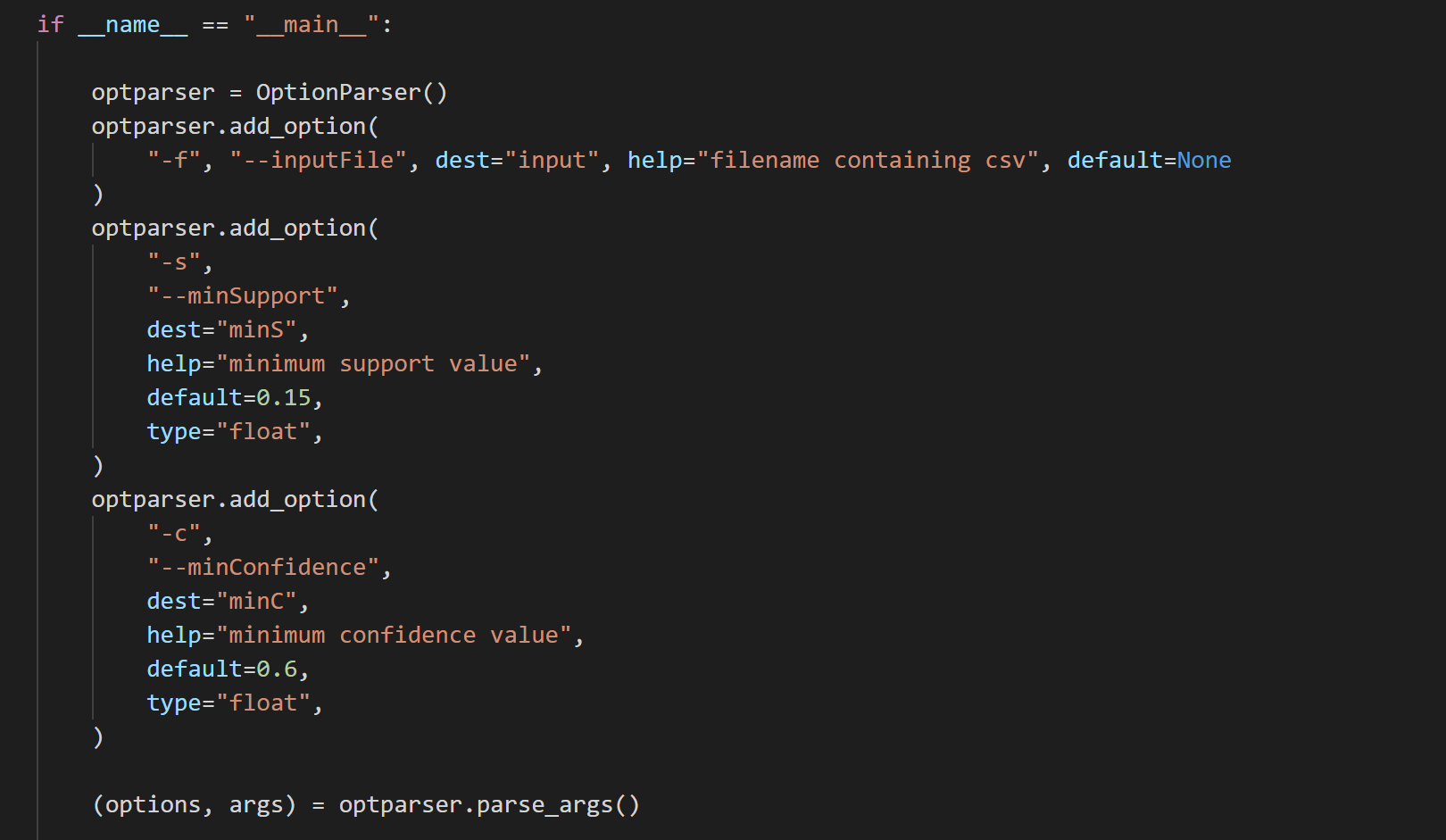


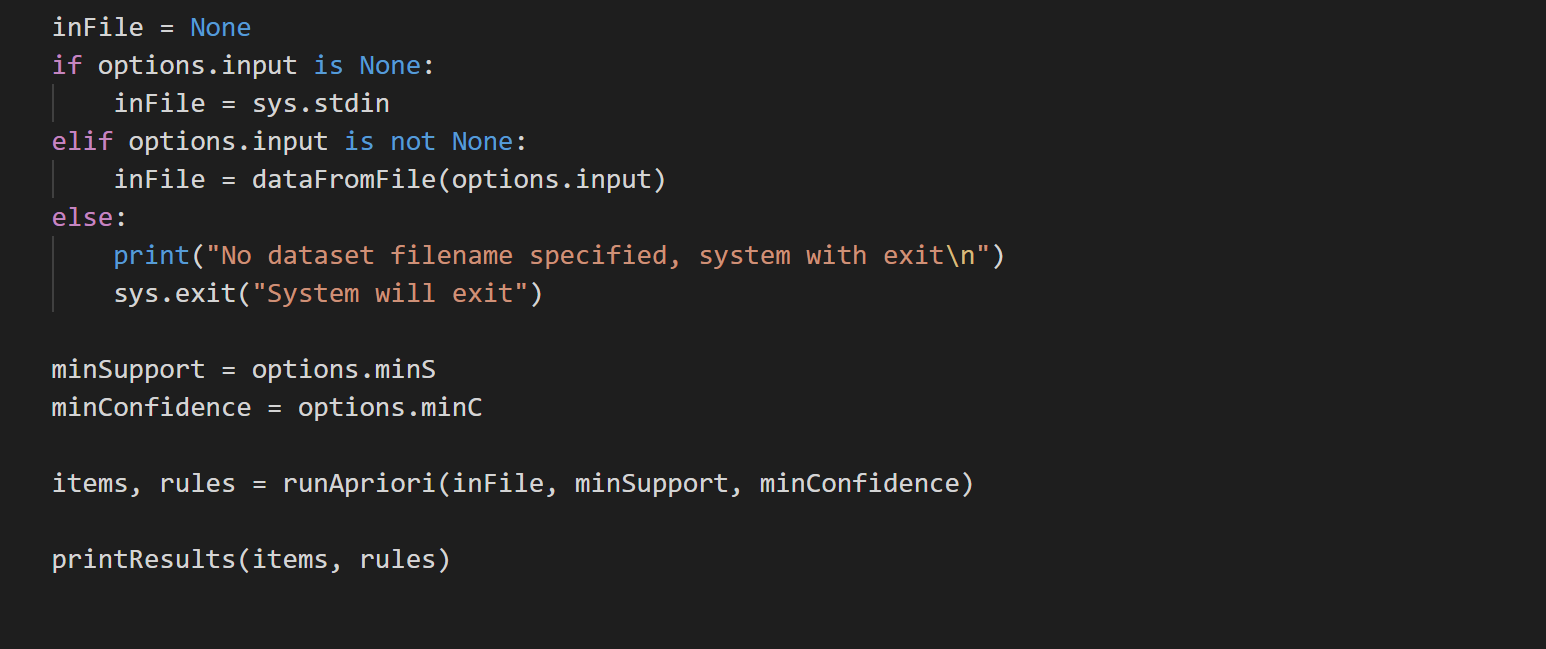
La fonction dataFromFile permet de lire le fichier texte et lire ligne par ligne, en reconnaissant le passage d’un item à un autre d’une transaction par une virgule.



Pour l’exécution du code, on commence par la création de paramètres d’exécution par la classe

OptionParser où on définit le paramètre -f : le nom du fichier des transactions, -s : support minimal et -c : confiance minimale.





Pour exécuter le code ci-dessus, il suffit d’exécuter la ligne de commande :

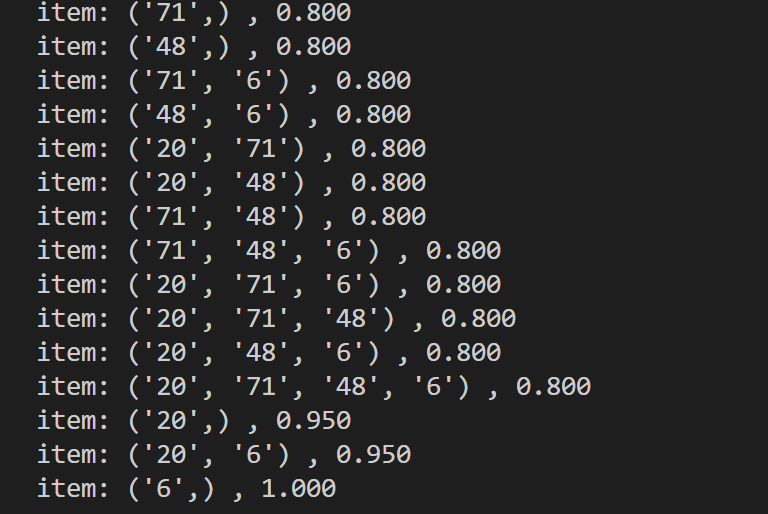
« cheminFichier/apriori.py -f nFile -s minSup -c minConf ». Si on ne précise pas minSup ou minConf, ils sont par défaut 0.15 et 0.6 respectivement.

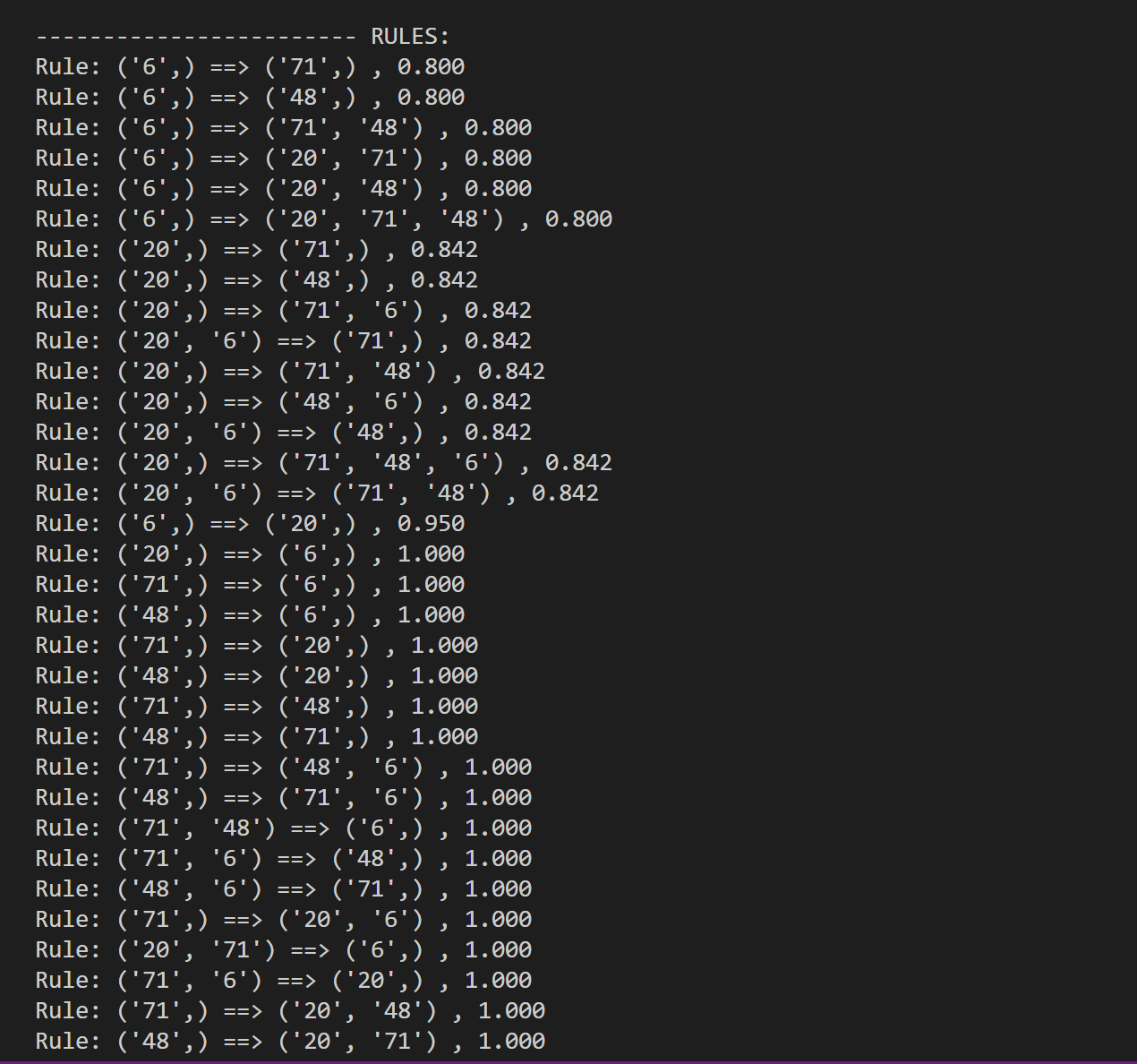
## Exécution :

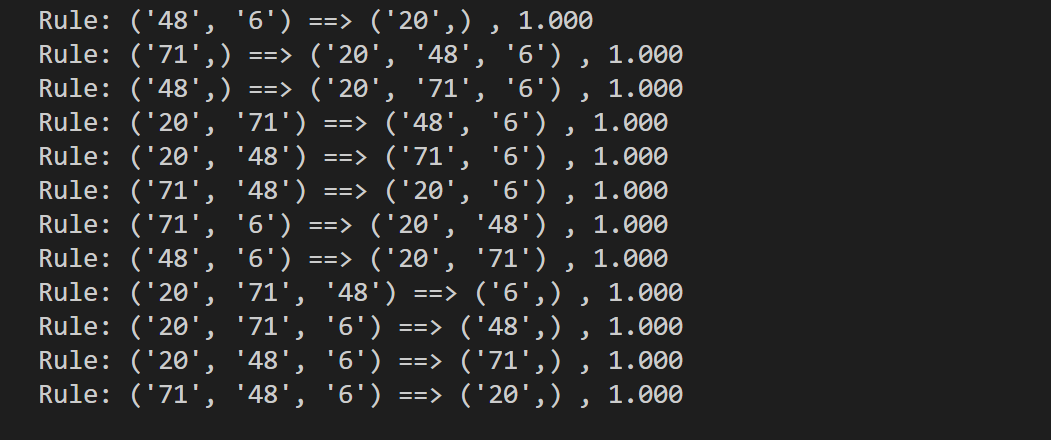
On a exécuté le code avec minSup = 0.8 et minConf = 0.8 et on a obtenu le résultat ci-dessous.

Nous avons choisi de grandes valeurs pour minSup et minConf pour ne pas avoir beaucoup d’items et de règles et alléger le rapport.









# Conclusion :

La recherche des règles d’association est une technique très utilisée de la fouille de données qui nous permet de mettre en évidence des relations qui ne sont pas évidentes à trouver. Elle est utilisée dans plusieurs domaines surtout le domaine de la vente où on a besoin d’améliorer l’expérience du client et en même temps augmenter les ventes. Il y a plusieurs algorithmes de recherche d’associations, les plus populaires sont Apriori, FP-Growth et Eclat. Généralement FP-Growth est plus performant comparé à Apriori, mais notre choix d’algorithme doit tenir en compte plusieurs paramètres notamment la largeur de notre base de données, la largeur des itemsets et les détails de notre implémentation.

# Références :

* <https://en.wikipedia.org/wiki/Apriori_algorithm>
* <https://github.com/asaini/Apriori/blob/python3/apriori.py>
* <https://www.softwaretestinghelp.com/fp-growth-algorithm-data-mining/>
* <https://fr.slideshare.net/CHOUAIBELHACHIMI/fp-growth-algorithm-108820971>
* <https://www.youtube.com/watch?v=yCbankIouUU&ab_channel=StudyKorner>
* <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.403.1361&rep=rep1&type=pdf>