Extraction de données relatives aux produits alimentaires à partir de documents non structurés

Pierre MASSÉ

Juin 2020

Résumé

La gestion de l'information produit est devenu un enjeu de société majeur ces dernières années. Les scandales sanitaires récents ont déclenché une prise de conscience collective des consommateurs, en parallèle de la mise en place de réglementations de plus en plus contraignantes pour l'ensemble des acteurs de la filière [3][4]. À ce titre, le Groupe Pomona a lancé ces dernières années un projet majeur de refonte des processus et des outils de gestion de l'information produit.

La première filiale du Groupe a fait l'objet d'un déploiement réussi, mais cela a toutefois mis en évidence le fait que des gains à la fois en qualité et en productivité restent accessibles.

La mise en place d'outils mettant en oeuvre les principes du Machine Learning appliqués au traitement du langage permettrait d'aider les opérationnels de la gestion de l'information à interpréter plus vite et mieux les documents mis à disposition par les fournisseurs du Groupe.

Le présent rapport détaille la mise en place d'un outil permettant d'extraire les listes d'ingrédients des fiches techniques transmises par les fabricants des produits.

TABLE DES MATIÈRES

Première partie

CONTEXTE MÉTIER

Chapitre 1

DESCRIPTION DU GROUPE

L'objet de l'ensemble de cette première partie est de donner sur le Groupe Pomona des éclairages nécessaires à la compréhension du cas d'usage développé. Bien d'autres aspects sur la société, pourraient être mentionnés (ex : des indicateurs sur l'activité, l'histoire du Groupe...) mais ils seront omis car non indispensables à la compréhension du sujet. Plus de détails sur le Groupe sont accessibles sur le site web de la société[10].

1.1 Le métier du Groupe Pomona

Le Groupe Pomona est une société de distribution livrée de produits alimentaires à destination des professionnels des métiers de bouche. L'activité du Groupe consiste uniquement à acheter et revendre de la marchandise, à l'exclusion de toute activité de fabrication ou de transformation [1]. Le Groupe Pomona est une société de *distribution*. Elle ne possède d'ailleurs pas d'actif industriels (autre que des entrepôts logistiques) ni d'agréement pour transformer les marchandises.

Cette activité d'achat/vente se fait dans la majorité des cas sous le régime du *négoce*, à savoir que le Groupe acquiert la propriété des marchandises qu'il commercialise avant de la céder à ses clients. L'autre régime est celui dit de la *prestation (logistique)*. Dans ce cas, par le jeux d'écritures comptables, la valorisation du stock disparaît des comptes du Groupe. Néanmoins, indépendamment de cet aspect purement comptable, l'ensemble :

des flux de documents : commandes d'achat, factures fournisseur, commandes de vente, factures clients

des flux financiers : paiements fournisseur, paiements client

des flux physiques : réception et stockage, préparation et expédition

1. de très rares cas de transformation existent (ex : mûrissage de fruits, filetage de poisson) mais sont extrêmement exceptionnels

restent largement inchangés.

Pour résumer, l'activité de l'ensemble des entités du Groupe pourraient se résumer via le schéma présenté à la FIGURE 1 page 6

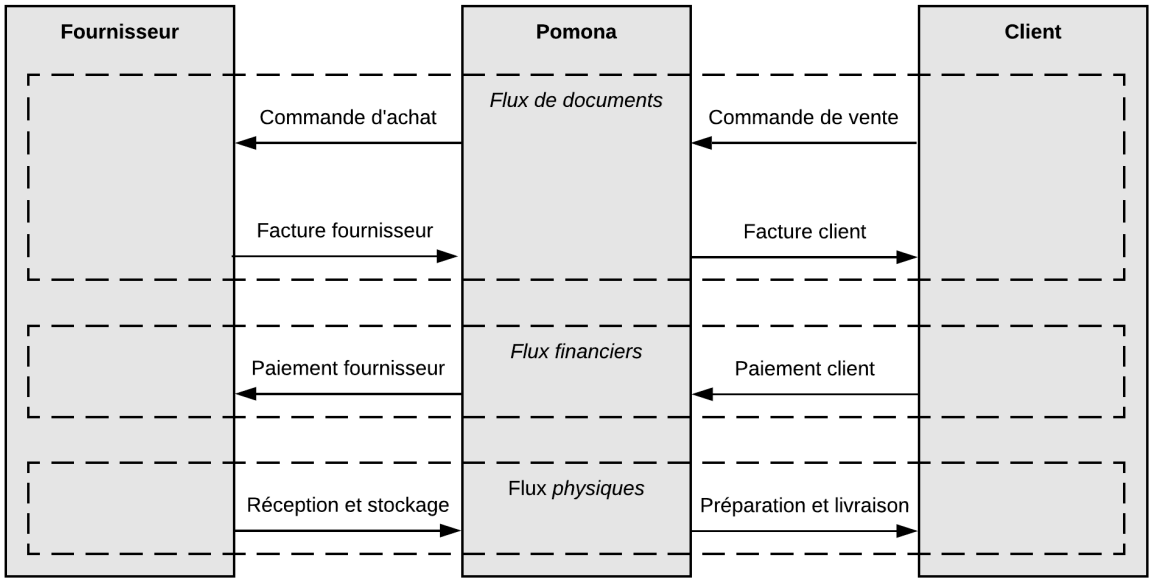


FIGURE 1 – Les flux métier avec les partenaires commerciaux

Le métier du Groupe est d'être un grossiste*, qui achète et revend des produits alimentaires*[2] sans produire ou transformer quoi que ce soit.

1.2 La décentralisation

Le Groupe Pomona est un Groupe fortement décentralisé, avec des organisations largement indépendantes les unes des autres.

1.2.1 Les Directions fonctionnelles

Pour des raisons évidentes de recherche de synergies ou de conformité réglementaires, certaines activités restent toutefois mutualisées à la maille du Groupe. Il s'agit des organisations suivantes :

La Direction Administrative et Financière (DAF) : regroupe les équipes comptables Groupe, l'audit interne et la consolidation financière

La Direction Qualité : est en charge de définir et contrôler l'application des standard de qualité

2. dans la grande majorité des cas, cf. *Les produits non-alimentaires* 1.2.3 page 9

La Direction des Systèmes d'Information (DSI) : développe et maintient en condition opérationnelles les systèmes d'information du Groupe

La Direction Technique et Logistique (DTL) : est en charge des projets immobiliers (entrepôts), des négociations avec les transporteurs et joue un rôle de conseil interne sur les sujets logistiques

La Direction des Ressources Humaines : se charge de l'ensemble des aspects en lien avec le recrutement, la paye et les sujets sociaux

La Direction Commerciale Groupe (DCG) : définit une stratégie et des bonnes pratiques commerciales et marketing

1.2.2 Les clients du Groupe

Afin de comprendre l'organisation du Groupe, il est nécessaire de connaître la typologie de ses clients. Comme mentionné précédemment, le Groupe s'adresse exclusivement aux professionnels des métiers de bouche. Aucune marchandise n'est vendue à des particuliers. Les principales typologies de clients sont les suivantes :

Les Sociétés de Restauration : elles exploitent les restaurants d'entreprise et certaines cantines d'établissement d'enseignement supérieur

Les Marchés Publics : regroupent les clients qui dépendent des collectivités (écoles, hôpitaux, prisons, ...)

La restauration commerciale : est l'ensemble des restaurants à vocation commerciale, qu'ils soient chaînés (hippopotamus, O'Tacos, ...) ou indépendants (« le restaurant du coin »)

Les spécialistes : il s'agit des détaillants spécialisés qui s'adressent aux particuliers. Boulangers, pâtissiers, bouchers, traiteurs, vente à emporter, ...

Les Grandes et Moyennes surfaces (la GMS) : sont les enseignes de la grande distribution. En général, l'accès à ces clients est compliqué par les règles mises en place par leurs centrales d'achat. Il représentent en général qu'un canal de vente d'opportunité.

Les trois premières de ces catégories représentent ce que l'on appelle la *Restauration Hors Domicile (RHD)* (ou parfois également la Restauration Hors Foyer, RHF).

1.2.3 Premier niveau de décentralisation : les branches

Le Groupe Pomona est divisé en branches, qui sont des organisations indépendantes et qui ont toute latitude pour gérer leurs stratégie et politique commerciales, la gestion de leurs achats, leur stratégie marketing, ...En particulier, les systèmes d'information ne sont pas identiques entre les différentes branches. Afin d'éviter de se concurrencer entre elles, leurs domaines d'activité respectifs ont été partitionnés par familles de produit commercialisés, segments client cibles et géographie.

Les branches RHD

Les branches RHD s'adressent comme leur nom l'indique aux client de la Restauration Hors Domicile (cf. section 1.2.2 page 7) en France. Elles se répartissent ce marché en travaillant des gammes de produits distinctes. Il s'agit des branches historiques du Groupe, qui représentent l'essentiel de son chiffre d'affaire. La répartition par produit est la suivante :

PassionFroid : spécialiste des *produits surgelés, de la viande fraîche et des produits laitiers*

ÉpiSaveurs : spécialiste des produits qui se conservent à température ambiante : *produits d'épicerie, conserves, boissons et consommables de cuisine non-alimentaires*

TerreAzur : spécialites des *Fruits et Légumes frais, et Produits De la Mer frais*

La non-concurrence entre les branches est assurée par le fait qu'elles ne commercialisent pas les mêmes produits. Bien que nommées RHD, elles peuvent également vendre leurs produits à la grande distribution, mais généralement ces marchés sont verrouillés par les centrales d'achat des grandes enseignes. De plus, les branches RHD utilisent le progiciel SAP comme système de gestion. La branche TerreAzur est en cours de déploiement, en 2020 environ les 2 tiers des succursales travaillent avec ce progiciel.

Les branches spécialistes

Les branches spécialistes s'adressent aux clients dits spécialistes (cf. section 1.2.2 page 7) en France. Elles sont en mesure de commercialiser tout type de produit pour répondre aux besoins de leurs clients. En particulier, elles peuvent tout à fait commercialiser certains produits qui sont également vendus par les branches RHD. Elles se répartissent la clientèle spécialiste de la manière suivante :

Délice et Création : s'addresse aux *Boulangers et Pâtissiers*

Saveurs d'Antoine : s'adresse aux *Bouchers, Charcutiers et Traiteurs*

Relais d'Or : s'addresse à la *restauration indépendante nomade*

Comme pour les branches RHD, ces branches peuvent lorsqu'elles en ont l'opportunité vendre leurs produits à la GMS.

L'étranger

Bien que le Groupe Pomona soit une société dont l'essentiel de l'activité est faite sur le marché français, deux réseaux sont en cours de constitution sur des pays limitrophe. Ces branches sont susceptibles de travailler tout type de produit, à destination de tout type de client. Elles sont positionnées sur les marchés suivants :

Pomona Suisse : présente sur le marché Suisse

Pomona Iberia : présente sur le marché Espagnol

On peut synthétiser la répartition de l'activité par branche de la manière présentée à la FIGURE 2 page 10.

Les produits non-alimentaires

Si l'essentiel des produits commercialisés par les branches du Groupe sont des produits alimentaires, comme évoqué précédemment une partie de l'activité commerciale se fait tout de même autour de produits non-alimentaires. Ces produits restent malgré tout destinés exclusivement aux professionnels des métier de bouche, et il s'agit de consommables (par opposition à des articles d'électroménager, de la vaisselle non-jetable, ...).

On distingue en général deux catégories de produits non-alimentaires :

— les produits dits « d'hygiène »

— les produits dits « de chimie »

Les produits de chimie regroupent les produits qui doivent faire l'objet d'une fiche de données de sécurté au sens du réglement Européen No 1907/2006 dit « REACH »(Registration, Evaluation, Authorisation and Restriction of Chemicals) [2].

Les produits d'hygiène regroupent tous les autres produits non-alimentaires. L'appelation « d'hygiène »est donc réductricte, dans la mesure où cette large famille regroupe les consommables de nettoyage (éponges, papiers absorbants, ...) mais également tout type d'autres consommables (serviettes de tables, gobelets en plastiques, pics à brochettes, boîtes de produits à emporter, ...).

La commercialisation de produits non-alimentaires existe au sein du Groupe, mais on se focalisera pour la suite sur les produits alimentaires qui reste le coeur de métier.

1.2.4 Le second niveau de décentralisation : les succursales

Chacune des branches est elle-même à son tour décentralisée en un réseau d'entrepôts régionaux : les succursales (parfois également appelées simplement « régions »). Ces succursales sont gérées comme des PME indépendantes, avec un directeur et un compte de résultat qui leur est propre. Si certaines négociation avec des fournisseurs ou des clients nationaux sont parfois menée par les branches, les succursales sont autonomes dans :

— la définition de leur assortiment, même si des contraintes s'appliquent

— la stratégie de développement commercial

— la négociation des prix d'achat

— la négociation des prix de vente

— la politique de rémunération de leurs employés

À ce titre, elles ont leurs propres équipes d'achat, leurs équipes commerciales (télévente et vente route), leurs équipes administratives et évidemment leurs équipes logistiques (essentiellement en entrepôt et les chauffeurs livreurs en charge des livraisons client).

Certaines activités restent de la responsabilité des équipes centrales des branches, comme : la négociation

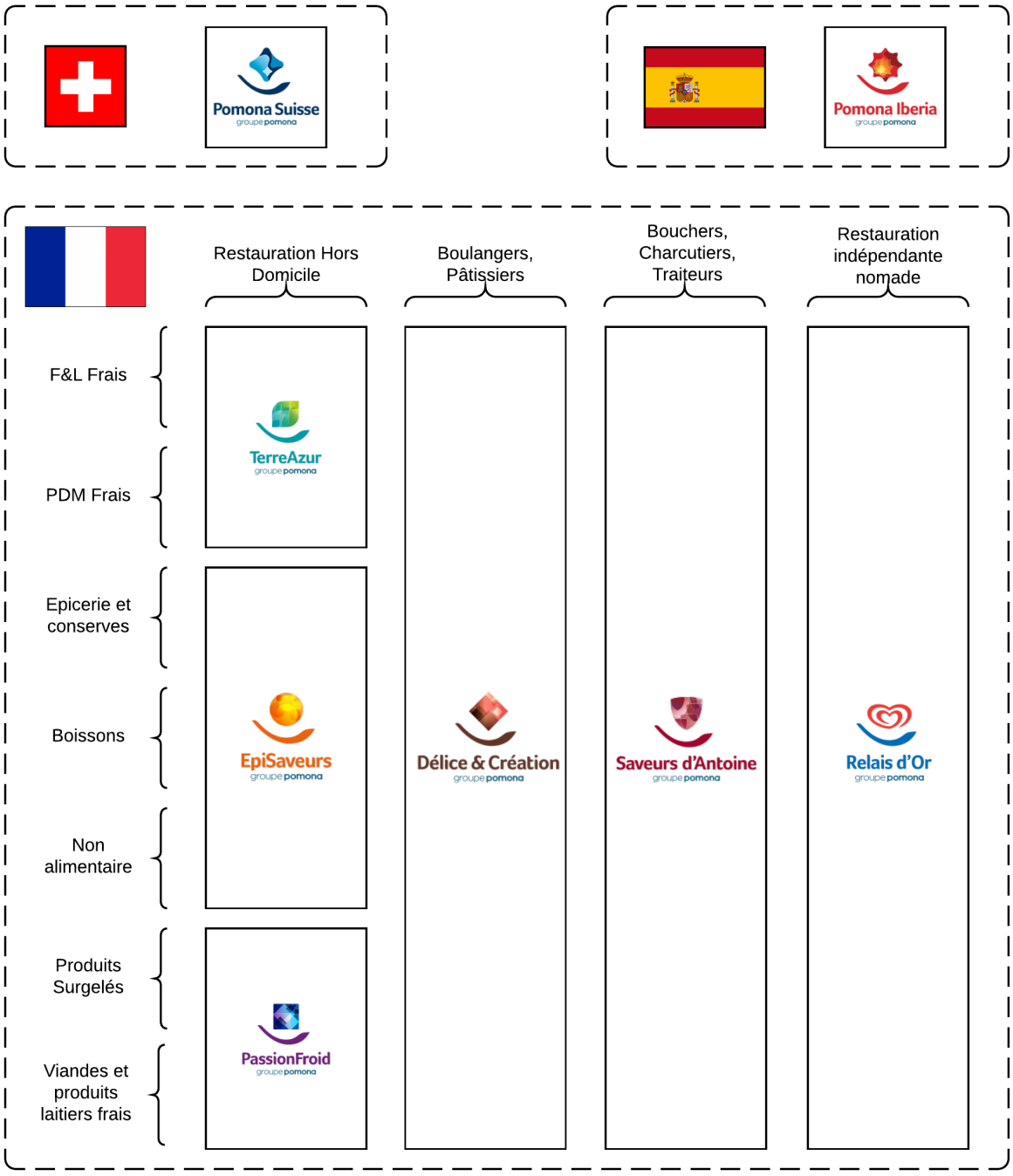


FIGURE 2 – La répartition de l'activité des branches

avec les clients ou les fournisseurs nationaux, la constitution de l'assortiment commun (les produits que toutes les succursales doivent détenir), la gestion des référentiels de données de base métier, ...

Un exemple de maillage régional est présenté en FIGURE 3 page 11, sachant que ce maillage régional est différent pour chacune des branches.

FIGURE 3 – Le maillage régional de la branche ÉpiSaveurs

Chapitre 2
LA GESTION DE L'INFORMATION PRODUIT

Ce chapitre a pour vocation à éclairer les aspects métier en lien avec la gestion de l'information produit. C'est le seul processus métier qui sera détaillé dans la mesure où c'est uniquement celui qu'il est nécessaire de connaître pour comprendre les cas d'usage développés ultérieurement.

2.1 L'information produit

2.1.1 Utilisations de l'information produit

Conformité réglementaire

La gestion de l'information produit est essentiellement une contrainte réglementaire à statisfaire. Comme mentionné au préambule, la réglementation autour de l'information des consommateur s'est sans cesse complétée au cours des dernières années. Un des textes centraux est le règlement n°1169/2011 dit INCO (INformation COnsommateur)[3][4]. C'est ce règlement qui définit l'ensemble des informations qui doivent être étiquetées sur le produit (liste d'ingrédients, tableau de données nutritionnelles, ...), mais également affichée au client lors de commande en ligne sur les sites de e-commerce. Il s'agit principalement d'informations relatives à la sécurité alimentaire (ex : les allergènes) ou la santé (ex : informations nutritionnelles).

Attentes client

Les consommateurs finaux (les « convives ») étant de plus en plus sensibles au contenu de leur assiète, les clients du Groupe sont de plus en plus demandeurs d'information relatives aux produits qu'ils commandent. Ils demandent donc régulièrement des informations qui vont au-delà de ce qui est normalement prévu par la réglementaition.

De plus, sur certains marchés pour lesquels des contrats courant sur de longues périodes - jusqu'à un an - sont établis (les marchés publics sont très concernés), il n'y a pas d'échantillonnage des produits. La seule manière pour ces clients d'évaluer la qualité des produits est de se référer aux documents contenant les informations produit, fournis par les distributeurs.

Gestion

Certaines informations relatives au produits sont nécessaires pour des raison de gestion adminsitrative. Par exemple, la gestion des taxes sur les produits alimentaires est complexe :

— les taux de TVA sont variables en fonction du type de produit

— des taxes spécifiques s'appliquaient aux produits contenant de l'huile ou de la farine

— des règlements particuliers s'appliquent aux alcools

— ...

D'autres informations, comme la nomenclature douanière, sont nécessaires pour effectuer les déclarations auprès des douanes européennes.

Un autre type d'information capital pour la gestion des flux d'achat et de vente sont les informations logistiques, qui définissent par exemple le nombre d'unités consommateur dans le colis, le nombre de colis sur une palette, ...Une gestion rigoureuse de ces information est indispensable pour que les flux d'achat ou de vente soient correctement exécutés (que les quantités commandées soient les bonnes, que les montants facturés soient corrects, ...).

2.1.2 Des produits bruts aux produits transformés

Le niveau d'exigence en termes d'information produit est variable en fonction du niveau de transformation de ce produit. Par exemple, sur des fruits et légumes frais, à peu de choses près seul le pays d'origine doit être affiché au client. Sur une barre chocolatée, ou un plat cuisiné, il sera nécessaire d'afficher :

— une liste d'ingrédients (mettant en évidence les allergènes)

— un tableau de données nutritionnelles (protéines, gludcides, ...)

— une dénomination réglementaire

2.1.3 Les grands types d'information

On se focalisera dans ce paragraphe sur les informations relatives aux *produits alimentaires*.

La composition

La première grande famille de données réglementaires sont les données de composition. Elles détaillent quels sont les ingrédients qui sont mis en oeuvre dans la fabrication des produits. Évidemment, la composition a en général plus de sens que pour les produits transformés que pour les produits bruts. Elle peut prendre la forme d'un texte listant la liste des ingrédients (l'étiquetage de ce texte est en général obligatoire sur les emballages des produits), ou d'un tableau.

Les ingrédients incluent également les additifs. Il s'agit de substances ajoutées à la recette pour répondre à des fonctions particulières (colorant, exhausteur de goût, émulsifiant). Elles ne représentent en général un pourcentage en masse négligeable dans la composition totale du produit.

Le pourcentage en masse est parfois inclus sur certains ingrédients. La règlementation l'oblige dans certains cas, par exemple quand l'ingrédient en question est mentionné dans la dénomination du produit (pour une *tarte aux framboises*, la proportion de framboise doit être mentionnée dans la composition).

Enfin, un aspect à la fois règlementaire et particulièrement important est la présence d'allergènes dans la composition. Le règlement INCO[3][4] impose de mettre en évidence les allergènes relevant d'une des 14 catégories suivantes :

1. Céréales contenant du gluten, à savoir blé, seigle, orge, avoine, épeautre, kamut ou leurs souches hybridées, et produits à base de ces céréales
2. Crustacés et produits à base de crustacés
3. Œufs et produits à base d' œufs
4. Poissons et produits à base de poissons
5. Soja et produits à base de soja
6. Lait et produits à base de lait (y compris le lactose)
7. Fruits à coque, à savoir : amandes, noisettes, noix, noix de cajou, noix de pécan, noix du Brésil, pistaches, noix de Macadamia ou du Queensland, et produits à base de ces fruits
8. Céleri et produits à base de céleri
9. Moutarde et produits à base de moutarde
10. Graines de sésame et produits à base de graines de sésame
11. Anhydride sulfureux et sulfites
12. Lupin et produits à base de lupin
13. Mollusques et produits à base de mollusques

Il peut y avoir deux niveaux de présence d'un allergène dans un produit (au-delà de la simple absence) :

intentionnellement mis en oeuvre : dans le cas où un ingrédient allergène fait volontairement partie de la recette. Ex : présence de moutarde dans un plat cuisiné.

contamination croisée : par exemple lorsque le produit fini est issu d'une chaîne de transformation qui traite un ingrédient allergène, mais que cet ingrédient ne fait pas partie de la recette. Ce cas de figure est en général mis en évidence par des mentions telles que *"Peut contenir des traces de soja"* ou bien *"Transformé dans un atelier processant également des fruits à coques et du sésame"*.

Les informations nutritionnelles

Une autre grande famille d'information produit sont les informations nutritionnelles. Elles détaillent la quantité des principaux nutriments contenus dans les produits. Certains d'entre eux sont rendus obligatoires par le règlement INCO [3][4] cf. l'exemple de tableau à la Table 1 page 15, et d'autres sont optionnels, comme par exemple la quantité de fer, de calcium, ...

Informations nutritionnelles	Pour 100g	Pour un biscuit	% des AJR pour un biscuit
Énergie	1674 kJ 398 kcal	209 kJ 50 kcal	3 %
Protéines	3.0 g	1.0 g	3 %
Matières grasses	13.0 g	1.6 g	2 %
dont acides gras saturés	5.8 g	0.7 g	4 %
Glucides	66 g	8.2 g	3 %
dont sucres	48 g	6.1 g	7 %
Fibres alimentaires	2.5 g	0.3 g	
Protéines	3.3 g	0.4 g	1 %
Sel	0.41 g	0.05 g	1 %

TABLE 1 – Exemple de tableau de données nutritionnelles

La réglementation rend obligatoire de mentionner les informations nutitionnelles de cette table pour 100g, ou 100mL de produit (pour les boissons).

Les informations nutritionnelles peuvent également se présenter sous forme d'allégations, qui ont des définitions précises dans la réglementation. Ces allégations peuvent être : *sans sel, faible en sucres, riche en fibres, ...*

Les origines

Du fait de la complexification des opérations de transformation et de la complexification des flux d'échanges de marchandises, l'origine des produits alimentaires est une notion qui n'est pas définie avec précision dans l'absolu. Il n'y a donc pas non plus de réglementation précise sur le sujet, si ce n'est que l'information produit doit toujours être présentée de manière loyale au consommateur. On peut se donner une règle simple pour définir l'origine d'un produit alimentaire : plus il est brut, plus va compter l'origine de ses ingrédients ; plus il est transformé, plus va compter le lieu de dernière transformation.

Par exemple, sur des morceaux piécés de viande fraîche, on aura des origines multiples en fonction du pays de naissance, d'élevage ou d'abattage de la bête. Et à l'inverse, sur un steak haché cette information n'aura aucun sens dans la mesure où il est produit d'un assemblage de « minerais »pouvant provenir de multiples pays. L'industriel pourra choisir de communiquer sur le fait que la viande a été transformée en steak dans telle usine par exemple.

Les données logistiques

On appelle données logistiques essentiellement le plan de palettisation et de conditionnement du produit. Il s'agit de la définition de la « hiérarchie logistique »du produit. Cette hiérarchie se base d'abord sur la définition d'une « unité de base »qui est la plus petite unité légalement détaillable (i.e. qui porte l'ensemble des informations réglementaire pour sa commercialisation). Ces notions ont été standardisées par l'organisme international de standardisation GS1[6]. Deux exemples pour illustrer :

— pour un boîte de sachets de thé, l'unité de base est la boite car les sachets de thé ne portent pas les informations nécessaires à leur commercialisation

— pour un paquet de barres chocolatées (comme celles qu'on peut trouver au détail en boulangerie), l'unité de base est la barre car elle porte l'ensemble des mentions réglementaires sur son emballage

La hiérarchie logistique est à la fois :

— la définition des niveaux successifs d'emballage des produits : combien d'unités de base dans un paquet, combien de paquets dans un carton, combien de cartons sur une palette, ...

— la définition du contenu de l'unité de base (ex : le nombre de sachets de thé, le nombre de doses dans une boîte d'aides culinaires, ...)

Les données logistiques concernent également les durées de vie du produit (type de durée de vie : Date Limite de Consommation ou Date de Durabilité Minimale ; ainsi que la durée en jours entre la fin de production du produit et son expiration).

Parfois, certaines contraintes d'approvisionnement peuvent être mentionnées :

— unités commandables (ex : on ne peut commander que des cartons complets)

— multiples de commande (ex : on ne peut commander les cartons que 10 par 10 pour des raisons de montage des palettes)

— minimum de commande (ex : il faut commander au minimum 30 cartons)

mais elles sont dépendantes d'un accord entre l'industriel et son client distributeur et ne sont donc pas à proprement parler des informations produit.

Les données administratives et financières

Les données dites administratives et financières regroupent le reste des informations de gestion pour lesquelles il existe des contraintes réglementaires. Il s'agit :

— du taux de TVA du produit

— de sa nomenclature douanière et du pays d'origine au sens de la déclaration d'échange de biens[5]

— de toute autre taxe applicable au produit

Les labels

Afin de garantir des qualités spécifiques à certains produit, des organismes de certification ont mis en place des labels pouvant s'appliquer aux produits. En général, ils se basent sur des cahiers des charges et peuvent être assortis d'audits de certification ou de contrôle. Ils peuvent garantir des méthodes de production ou transformation, des lieux de production, des caractéristiques de leurs ingrédients, des pratiques commerciales équitables, ...Les types de labels les plus connus sont :

— les produits Biologiques

— les origines protégées (Appellation d'Origine Protégée, Indication Géographique Protégée, *viandes* de France, Bleu Blanc Cor, Régions UltraPériphériques d'Europe...)

— les pratiques commerciales équitables (Max Havelaar, ...)
— les modes de production respectueux de l'environnement (Aquaculture Stewardship Council, Marine Stewardship Council, Roundtable on Sustainable Palm Oil, Nordic Swan, ...)
— la qualité « générale »des produits (Label Rouge, ...)

Les données marketing

Certaines données marketing font également partie de l'information produit. La plus évidente est la marque commerciale du produit, qui parfois définit totalement le produit. Par exemple, on sait ce qu'est un Snickers, de la Mousline, du Nutella, ...Les produits peuvent également porter d'autres allégations marketing, non réglementaires ou labelisantes : Élu produit de l'année, Vu à la télé, Issu de notre savoir-faire centenaire, ...

2.2 Le processus

2.2.1 Le fournisseur est propriétaire des informations produit

Comme présenté à la section 1.1 page 5, le Groupe Pomona n'a pas d'activité de fabrication ou de transformation de marchandises. À ce titre, l'ensemble des données produits ne peuvent être déterminées que par les fournisseurs de ces produits. L'ensemble des entités du Groupe s'appuient donc sur les données transmises par les industriels ou producteurs de marchandises.

Il peut arriver que certains produits soient achetés par Pomona à d'autres négociants non-producteurs. Dans ce cas, de la même manière que le Groupe Pomona a la responsabilité de collecter puis transmettre les informations produit à ses clients, ces fournisseurs négociants doivent eux-même aller chercher l'information produit et la transmettre à Pomona.

Dans tous les cas, c'est le fournisseur *qui est responsable de produire et de transmettre l'information produit aux entités du Groupe Pomona.*

2.2.2 La notion de produit et d'article

Un mot sur la modélisation des données adoptée est nécessaire pour comprendre les grandes lignes du processus. Comme il a été vu à la section 1.2.3 page 7, certains produits sont susceptibles d'être commercialisés par plusieurs branches du Groupe.

De plus, du fait que les systèmes d'information ne sont pas identiques entre les branches, certaines contraintes imposent parfois des différences de modélisation, des duplications volontaires de codes pour répondre à ces contraintes. Une illustration de ce point pour clarifier : la facturation client pour une canette de soda peut se faire au litre (permet de comparer les prix entre les différents conditionnement et les différentes marques) ou à la cannette (permet de se faire une idée du coût portion d'un produit). Or, la possibilité de

pouvoir facturer un même article dans plusieurs unités différentes n'est pas une fonctionnalité offerte par tous les systèmes d'information. En particulier, ÉpiSaveurs peut gérer dans ce cas un unique article et le facturer dans l'unité de son choix en fonction des demandes des clients. Mais Délice et Création (qui possède un système d'information différent) doit dupliquer cet article car une seule unité de facturation est possible pour un article donné.

Enfin, au-delà des contraintes liées au SI, certaines pratiques imposent de laisser aux branche une indépendance forte dans la gestion de leurs référentiels articles. Il faut savoir que commercialiser sous un même code article des produits qui sont similaires permet d'obtenir des gains de productivité à plusieurs niveaux :

— on économise des emplacements en entrepôt (un emplacement ne peut contenir qu'un article)
— on gagne du temps administratif dans la gestion des prix : le foisonnement d'articles impose de gérer plus de prix client
— ...

La contrepartie à adopter cette pratique est qu'il n'est alors plus possible de différencier ces produits similaires, par exemple pour leur appliquer des prix de vente distincts, ou bien offrir la possibilité à un vendeur de garantir au client la livraison d'un produit plutôt que l'autre. Néanmoins, en fonction de la clientèle adressée, certains produits pourront être considérés comme similaires, alors que pour d'autres ils ne seront pas interchangeables. Cet exemple est détaillé dans la FIGURE 4 page 19.

On différencie donc les deux notions suivantes :

les produits : ils représentent une marchandise physique produite par un fournisseur. Ce sont les produits qui portent les *informations produit* décrites à la section 2.1.3 page 13. Un produit ne peut appartenir qu'à un seul fournisseur. Le référentiel produit est unique pour l'ensemble du Groupe.

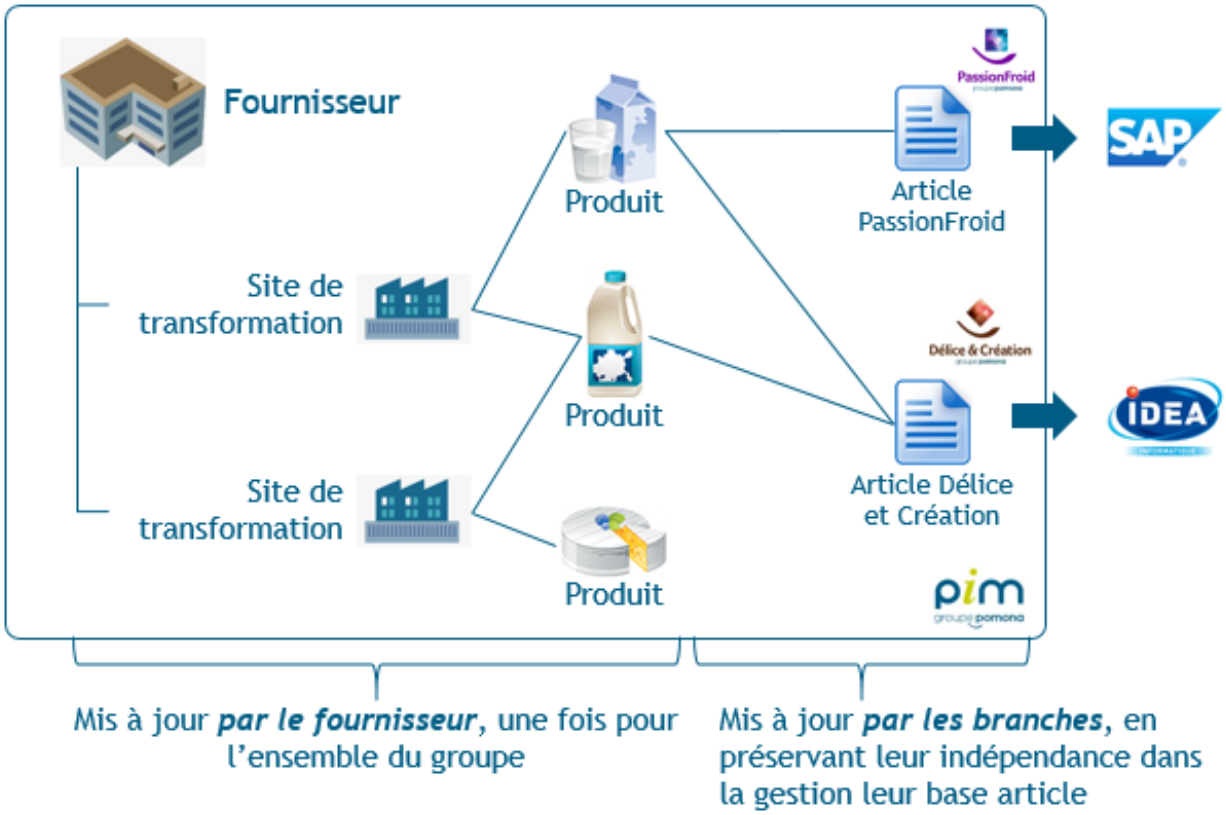
les articles : ce sont les objets qui sont gérés par les branches dans leurs systèmes de gestion respectifs. Leurs attributs sont très liés au système d'information qui les porte. Chaque branche gère de manière autonome son référentiel article, incluant les liens qui sont faits entre produits et articles.

Cette modélisation permet de répondre à l'ensemble des contraintes présentées dans ce paragraphe.

2.2.3 Les contrôles

Il a été vu que le Groupe Pomona - dans sa qualité de distributeur - n'est pas en mesure de déterminer seul les informations produit sur les marchandises qu'il commercialise. Néanmoins, comme cela a été vu dans la section 2.1.1 page 12, il est nécessaire que les différentes entités du Groupe soient en possession d'une information produit fiable. Or, avoir des données de qualité nécessite des efforts de la part des métiers, en particulier lorsque le processus n'est pas entièrement porté en interne dans la société. À ce titre, plusieurs étapes de contrôle ont été définies dans le processus de gestion du référentiel de données produit et article :

lorsque le fournisseur a saisi les données produit : la personne à l'origine de la demande de référencement (en général, un acheteur) doit contrôler la cohérence des données produit



Dans cet exemple fictif, le type de conditionnement du lait n'a aucun impact sur les clients de la branche Délice et Création. Elle commercialise donc sous un même code article deux produits distincts.
Pour des raisons de contrainte de conservation, la branche PassionFroid a quant à elle choisit de ne commercialiser qu'un seul produit - en brique. Elle pourrait choisir d'ouvrir un nouveau code article pour le lait en bouteille (non représenté sur le schéma ci-dessus).

FIGURE 4 – La distinction entre produit et article

lorsque le demandeur a demandé la création d'un article : le gestionnaire de référentiel valide à nouveau la cohérence des informations produit

après la création article, de manière asynchrone : le service qualité contrôle par échantillonnage les données d'une partie des produits et articles qui ont été modifiés pendant une période.

Le retour d'expérience montre que ces contrôles, loin d'être redondants, sont nécessaires pour avoir une qualité de données acceptable. Ces processus de contrôle sont décrits à la FIGURE 5 page 20.

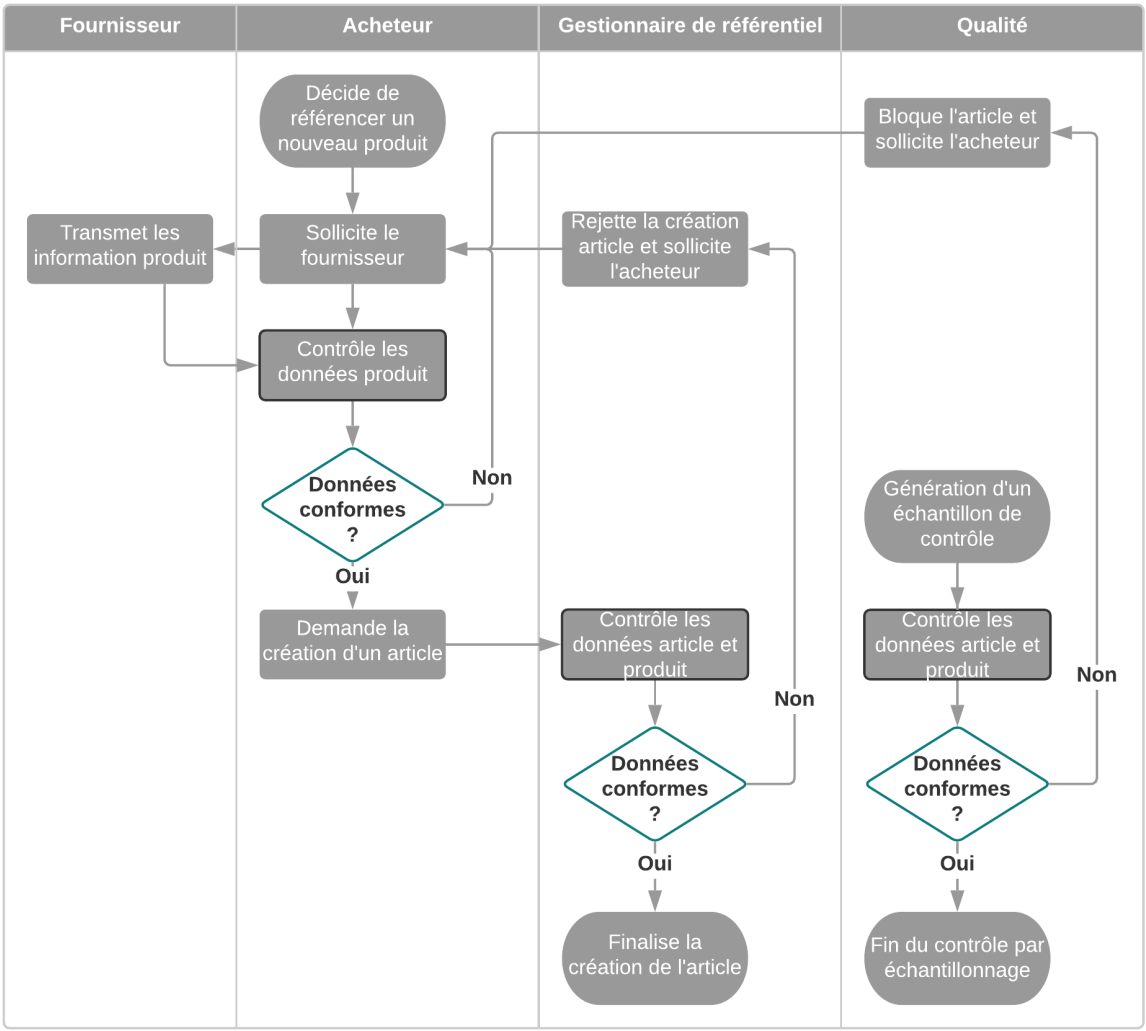


FIGURE 5 – Le processus de création article

Les contrôles effectués à chacune des étapes sont les suivants :

contrôle de la complétude des données : vérification que les données transmises comportent l'ensemble des données attendues

contrôle de cohérence entre les données : vérification que les informations transmises sont cohérentes entre elles (ex : un allergène présent dans la liste d'ingrédients du produit a bien été signalé comme allergène par ailleurs)

contrôle de la cohérence avec les pièces jointes : en plus de données structurées, les fournisseurs transmettent également des fichiers portant des informations produit (ex : l'étiquette produit ou le visuel de l'emballage). Ces pièces jointes sont décrites à la section 4.3 page 32. La personne en charge du contrôle vérifie que les données transmises sont cohérentes avec ces documents.

2.3 Les outils informatiques associés

Comme vu dans la description des branches du Groupe (voir section 1.2.3 page 7), les outils informatiques ne sont pas tous les mêmes sur l'ensemble des branches. Ainsi, les outils utilisés pour la gestion de l'information produit ne sont pas les mêmes.

2.3.1 Les branches faiblement outillées

Les branches étrangères (Pomona Suisse, Pomona Iberia), spécialistes (Délice et Création, Saveurs d'Antoine) et la branche TerreAzur sont aujourd'hui faiblement outillées. Cela signifie que l'information produit est en général stockée uniquement sous la forme de fichiers (essentiellement les pièces jointes, décrites à la section 4.3 page 32). L'ensemble des échanges avec les fournisseurs se font par mail, et les articles sont créés directement dans les systèmes de gestion par les gestionnaires de référentiel. Les liens entre les articles et les informations produit ne sont pas matérialisés dans les systèmes informatiques.

2.3.2 Le GIP

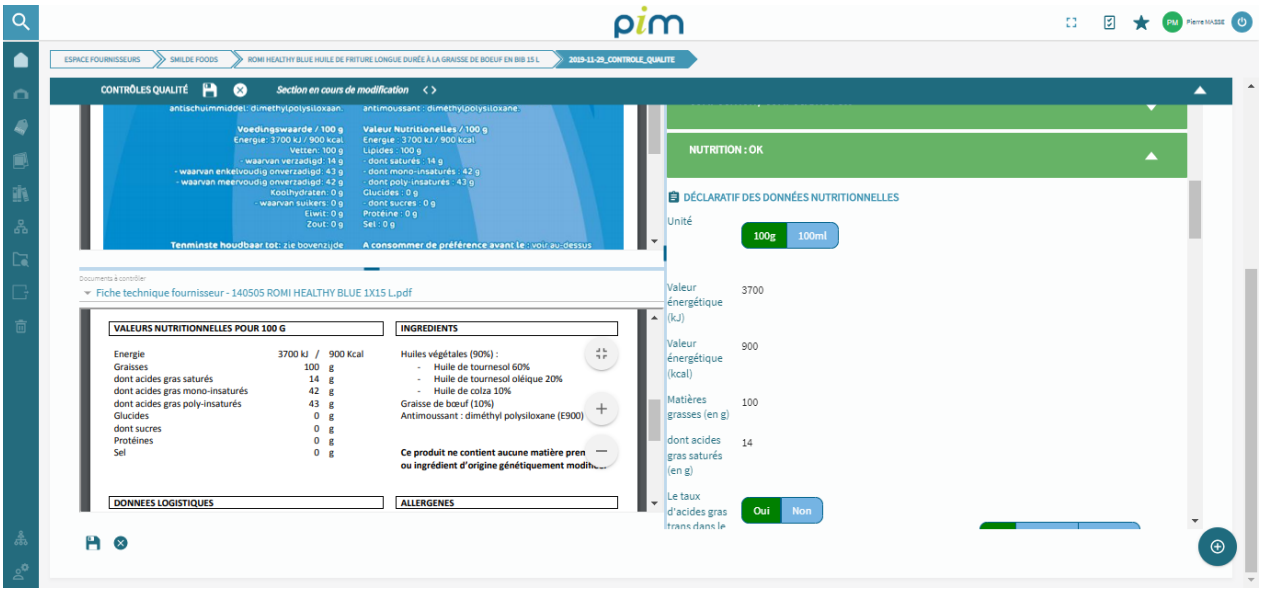
Le GIP (Gestion de l'Information Produit) est utilisé sur la branche PassionFroid. C'est un système de gestion de l'information produit qui est maintenant obsolescent et en cours de remplacement. Il a toutefois le mérite de permettre le stockage dans une application des données et des pièces jointes relatives aux produits, avec la possibilité d'accéder aux informations produit à partir des identifiants des articles. C'est ce système qui a permis de pouvoir alimenter les sites de e-commerce PassionFroid et ÉpiSaveurs avec les informations produit. Il est toutefois ancien, et ne propose pas de fonctionnalité d'export en masse fiable. Il s'agit d'une application qui n'est pas ouverte aux utilisateurs externes au Groupe, et les échanges avec les fournisseurs passent donc par des échanges de mails.

2.3.3 Le PIM

Le PIM (Product Information Management) est un système de gestion de l'information produit qui a été mis en production en mai 2019, pour la branche ÉpiSaveurs.

Description générale de l'outil

C'est un système qui porte l'ensemble du processus de gestion de l'information produit, tel que décrit à la FIGURE 5 page 20. Il est accessible aux fournisseurs du Groupe, qui viennent directement mettre à disposition les données et les pièces jointes. Cet outil porte entre autres les fonctionnalités de contrôle des informations, comme illustré à la FIGURE 6 page 22.



Cette capture d'écran montre l'outil de contrôle des données produit. Sur la partie gauche, le contenu des pièces jointes est affiché (visuel de l'emballage en haut, fiche technique en bas), sur la droite les données qui ont été transmises par le fournisseur.

FIGURE 6 – Une capture d'écran du PIM

La GDSN

La GDSN (Global Data Synchronization Network) est un réseau d'échange de données produit entre industriels, distributeurs, restaurateurs, ...Ce réseau est exploité par des opérateurs privés, mais le format et la chorégraphie des échanges a été standardisé par l'organisme de standardisation GS1. Son schéma de principe est décrit à la FIGURE 7 page 23. Sans rentrer dans le détail, au sein du Groupe Pomona l'utilisation qui en est faite est de récupérer les informations depuis ce réseau d'échange, afin de préalimenter les données produit pour les fournisseurs. Cette foncitonnalité permet de faire gagner du temps aux fournisseurs pour leur éviter une partie de ressaisie, mais également de limiter les erreurs. Toutefois, cette fonctionnalité ne permet pas à elle seule de garantir une parfaite qualité de données. Il s'agit uniquement d'un « tuyau », si les données en entrée ne sont pas correctes, elles ne seront pas correctes en sortie. Pour aller plus loin dans la compréhension de ce réseau, il est possible de consulter les ressources mises en ligne par GS1 [7][8].

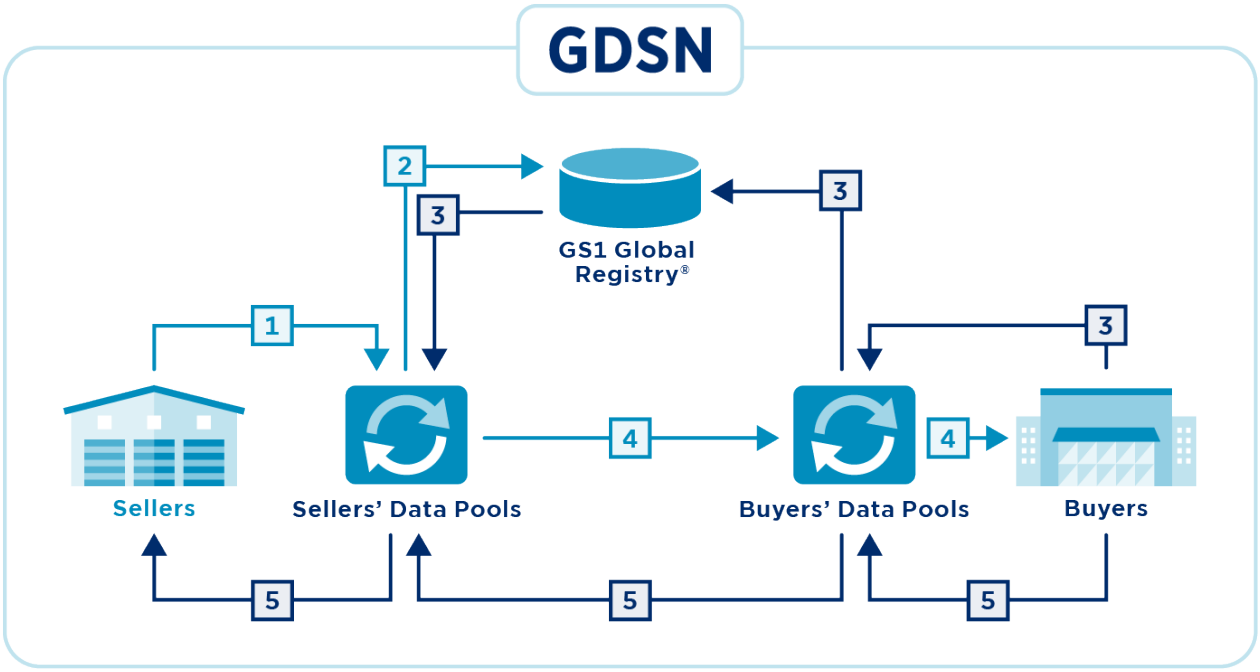


FIGURE 7 – Schéma de principe de la GDSN

L'identification des objets dans le PIM

Un dernier point à connaître à propos du PIM, est la manière d'identifier l'ensemble des objets en son sein. Chaque objet géré (ainsi que toute version archivée) porte un identifiant unique, nommé *uid* qui est totalement univoque. Pour la suite, on se basera sur ces uid pour faire référence à des produits stockés dans le PIM. Une illustration est présentée à la FIGURE 8 page 24.

L'uid du produit affiché est mis en évidence dans l'url

FIGURE 8 – L'uid d'un produit

Les API

Un des aspects intéressants du PIM pour l'exploitation en masse des données produit, est qu'il expose des API permettant d'aller requéter l'ensemble de son contenu. Cela concerne à la fois les données dites structurées, mais également les pièces jointes. Cela rend les données produit de la branche ÉpiSaveurs bien plus simplement accessibles que celles des autres branches.

Deuxième partie

LES DONNÉES

Chapitre 3
LE PÉRIMÈTRE PRODUIT

3.1 Accessibilité de la donnée en fonction des branches

Comme vu à la section 2.3 page 21, les systèmes d'information associés à la gestion de l'information produit offrent des niveaux d'accès hétérogènes à la donnée produit. Le récapitulatif par branche est le suivant :

ÉpiSaveurs : on peut simplement accéder à l'ensemble des données produit, structurées, non structurées (i.e. textes longs) et pièces jointes

PassionFroid : on a uniquement la possibilité d'exporter manuellement les données structurées articles depuis le système de gestion SAP. Elles permettent de produire quelques analyses quantitatives. Il est difficile de faire des exports en masse de l'outil de gestion de l'information produit GIP (cf. section 2.3.2 page 21).

TerreAzur : idem PassionFroid, si ce n'est qu'en plus le système GIP n'est pas utilisé au sein de cette branche.

Délice et Création : le système d'information ne permet pas d'exporter les données et donc de produire des indicateurs détaillés. On peut toutefois avoir des informations quantitatives de la part des opérationnels.

Saveurs d'Antoine : idem Délice et Création

Pomona Suisse : la branche est en cours de structuration, et les référentiels articles ne sont pas partagés entre les succursales. Il n'est pas possible d'obtenir d'information quantitative sur ces données.

Pomona Iberia : idem Pomona Suisse

Pour les analyses quantitatives, on pourra se baser sur des extractions uniquement pour les branches RHD (ÉpiSaveurs, PassionFroid, TerreAzur). L'ensemble des analyses portant sur les branches RHD sont produites sur la base d'extractions de leur système de gestion SAP.

3.2 Analyses quantitatives

Les graphes de cette section ont été produits via le code présenté en annexe, au chapitre C page 46. Les données pour les branches spécialistes (Délice et Création et Saveurs d'Antoine) sont issues d'informations fournies par le métier, hors système. Dans l'ensemble de cette section, on raisonnera à la maille *article* (cf. la définition article vs. produits, présentée à la section 2.2.2 page 17).

3.2.1 Comparatifs entre les branches

En termes de volumétrie article (cf. FIGURE 9 page 26, c'est TerreAzur qui possède le référentiel le plus étendu (environ 62 000 articles de marchandises actifs). Cela s'explique par le fait que cette branche commercialise essentiellement des produits bruts, non-préemballés (ex : des cagettes de fruits ou de légumes). Or, ces produits ne sont pas clairement identifiés, par exemple par un GTIN. Au démarrage de cette branche, afin de limiter la charge sur les gestionnaires de référentiels, le parti a été pris de créer en avance de phase l'ensemble des articles susceptibles d'être commercialisés. Cela s'est traduit par la création d'un grand nombre d'articles, du fait de l'application « brutale »de la combinatoire des différents critères pouvant définir un produit. Un exemple (fictif) serait, sur les pommes :

— 8 variétés possibles (Gala, Golden, ...)

— 4 calibres possibles

— 6 conditionnements possibles (plateau 6kg, plateau 4,5kg, ...)

— 2 catégories (I, II)

— 8 origines (France, Espagne, ...)

ce qui donne un total de 3072 articles uniquement sur cette gamme de produits.

Viennent ensuite PassionFroid, et ÉpiSaveurs, qui sont les autres « grosses »branches historiques du Groupe.

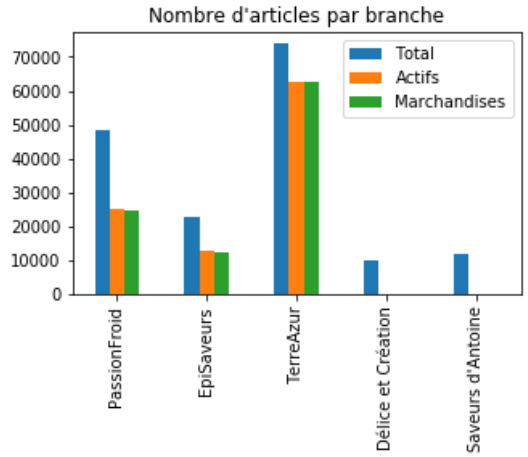


FIGURE 9 – Volumétrie article par branche

Branche	Total	Actifs	Marchandises
PassionFroid	48478	24898	24554
EpiSaveurs	22498	12798	12241
TerreAzur	73804	62789	62710
Délice et Création	10000	-	-
Saveurs d'Antoine	12000	-	-

TABLE 2 – Volumétrie article par branche

Une analyse du recouvrement des référentiels montre que dans l'ensemble, les branches ne travaillent pas les mêmes articles (cf. FIGURE 10 page 27). PassionFroid commercialise certains produits des branches ÉpiSaveurs et TerreAzur, mais cela s'explique par une petite entité luxembourgeoise qui travaille des produits de tout type de stockage. Une réserve toutefois par rapport à cette analyse de recouvrement produit : elle sous-estime vraisemblablement lesdits recouvrements, dans la mesure où la présence de doublons n'est pas prise en compte.

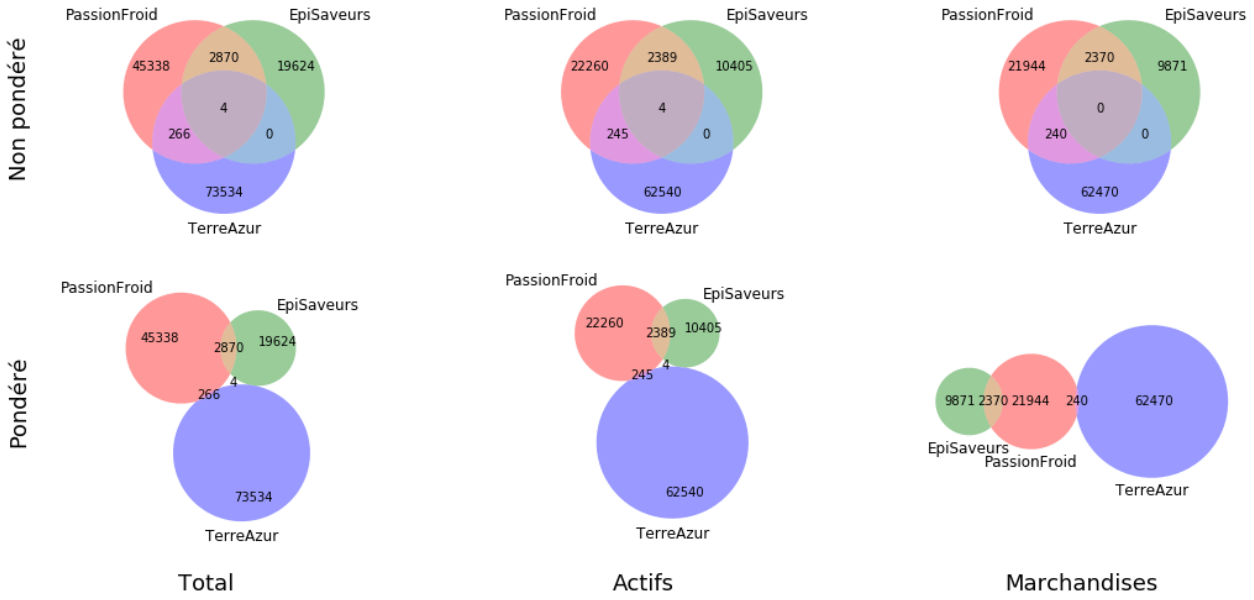


FIGURE 10 – Recouvrements entre branches RHD

3.2.2 Les grands types de produits

Comme montré à la FIGURE 11 page 28, on voit bien (en plus du fait que les articles étaient peu partagés entre les branches) que :

— les deux types d'articles (négoce et presté) sont utilisés par les 3 branches

— c'est tout de même PassionFroid qui fait l'utilisation majoritaire d'articles prestés

— les branches *ne* partagent *pas* l'utilisation des autres « catégories » (groupe de marchandises, conditions de stockage, hiérarchie produit, ...)

Les indicateurs sont également récapitulés dans la TABLE 3 page 29.

Possibilité d'extension : - ajouter la vision produit (via les FIA) - faire une sorte d'heatmap qui montre la forte correlation entre les différentes variables catégorielles (et les définitions associées. Ex : ZELAB + SA = Saurisserie, etc...)

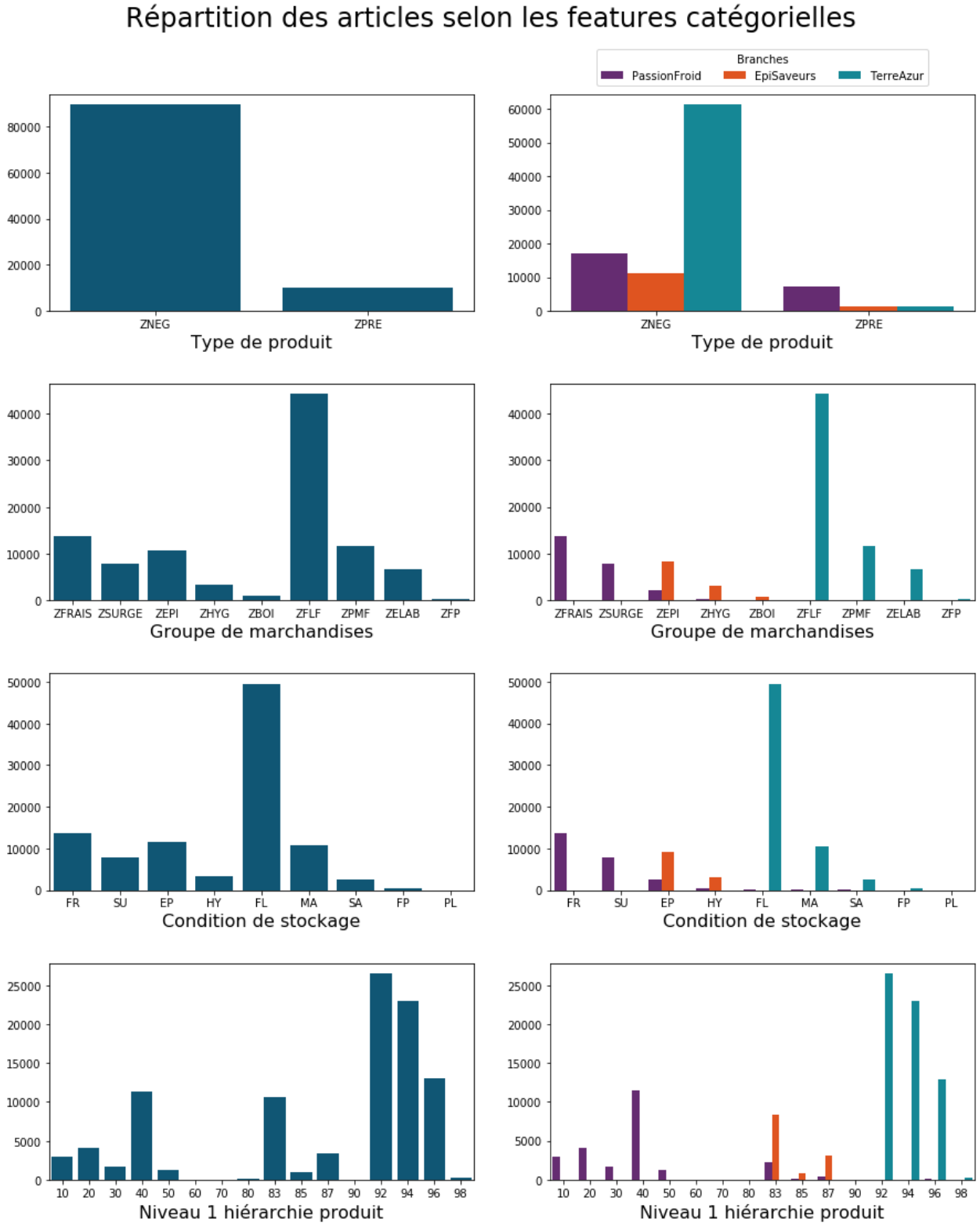


FIGURE 11 – Répartition des articles en fonction des variable catégorielles

Branche Type de produit	PassionFroid	EpiSaveurs	TerreAzur	Total
ZNEG - Article de négoce	17166	11048	61273	89487
ZPRE - Article de prestation	7388	1193	1437	10018

Branche Groupe de marchandises	PassionFroid	EpiSaveurs	TerreAzur	Total
ZSURGE - Surgelés	7756	-	-	7756
ZFRAIS - Frais	13785	6	4	13795
ZEPI - Epicerie	2298	8305	-	10603
ZBOI - Boissons	126	826	-	952
ZHYG - Hygiène	350	3078	-	3428
ZFLF - Fruits et Légumes	4	-	44133	44137
ZPMF - Produits de la mer	142	-	11594	11736
ZELAB - Produits élaborés	91	-	6644	6735
ZFP - Fleurs et plantes	-	-	297	297
ZAUTRE - Autres	2	26	38	66

Branche Condition de stockage	PassionFroid	EpiSaveurs	TerreAzur	Total
SU - Surgelés	7758	-	-	7758
FR - Frais	13781	6	3	13790
EP - Epicerie	2430	9155	-	11585
HY - Hygiène	344	3080	-	3424
FL - Fruits et légumes	78	-	49508	49586
MA - Marée	126	-	10501	10627
FP - Fleurs et plantes	-	-	286	286
SA - Saurisserie	34	-	2408	2442
PL - Publicié	2	-	1	3

Branche Niveau 1 hiérarchie produit	PassionFroid	EpiSaveurs	TerreAzur	Total
10 - Beurre, oeufs, fromage	3010	6	1	3017
20 - Elaborés	4150	2	6	4158
30 - Garnitures et fruits	1701	-	-	1701
40 - Produits carnés	11413	-	-	11413
50 - Produits de la mer	1214	-	2	1216
60 - Consommables	1	-	-	1
70 - Emballage	-	1	-	1
80 - Publicité sur le lieu de vente	34	25	37	96
83 - Epicerie	2306	8296	-	10602
85 - Liquides	135	836	-	971
87 - Hygiène et entretien	348	3075	-	3423
90 - Services	10	-	-	10
92 - Fruits	35	-	26543	26578
94 - Légumes	37	-	22929	22966
96 - Produits de la mer Frais	160	-	12891	13051
98 - Fleurs - plantes	-	-	301	301

TABLE 3 – Utilisation des variables catégorielles article au sein des branches RHD

Chapitre 4
LES DONNÉES UTILISABLES, ISSUES DU PIM

Comme vu au chapitre 3 page 25, les données produit ne sont simplement accessibles que pour la branche ÉpiSaveurs. On se focalisera donc sur cette branche pour la suite de cette étude, ainsi que sur les produits alimentaires (en excluant donc les produits d'hygiène et de chimie, cf. section 1.2.3 page 9). Contrairement au chapitre précédent, ici on travaillera à la maille *produit* (cf. la distinction produit vs. article section 2.2.2 page 17).

4.1 Données structurées

Les données dites structurées sont l'ensemble des données qui peuvent prendre leurs valeurs dans un domaine restreint. Par exemple, ce sont les données booléennes, les choix issus de listes déroulantes, les valeurs numériques...Les principales données structurées pour les produits alimentaires dans le PIM sont :

le code du produit : calculé par le système

le fournisseur : référence croisée vers le code du fournisseur

le type de produit : épicerie, boisson alcoolisée, hygiène, chimie, boisson non-alcoolisée

le type d'unité de base : paquet, boîte, sachet, rouleau, bouteille, pot, ...

le GTIN du produit : identifiant numérique unique, utilisé entre autres pour l'étiquetage sous forme de code à barres [9]

les poids : brut, net, net égoutté (pour les conserves)

le volume : pour les produits liquides

les durées de vie : le type (Date Limite de Consommation ou Date de Durabilité Minimale) et la durée (totale à fin de production, garantie à livraison)

les modes de conservation avant/après ouverture : à température ambiante, au réfrigérateur puis à consommer sous 2 jours, ...

les labels : le(s) label(s) s'appliquant au produit (cf. section 2.1.3 page 16)

les régimes particuliers : Halal, Casher, Sans porc, Végétarien, Végétalien, ...

les caractéristiques spéciales : sans OGM, non traité par ionisation

la présence d'allergènes : le niveau de présence de chacun des 14 allergènes réglementés (cf. section 2.1.3 page 13) : absence, présence ou traces

les matières grasses utilisées : palme, beurre, coco, tournesol, palmiste, ...

les additifs présents : les codes Exxx et les fonctions des additifs mis en oeuvre [1][11]

les données nutritionnelles obligatoires : pour 100g ou 100mL, valeur énergétique (en kJ et kcal), matières grasses, dont acides gras saturés, Glucides, dont sucres simples, Fibres, Protéines, simplement

les données nutritionnelles facultatives : vitamines, minéraux, omégas, ...

les allégations nutritionnelles : riche en, faible en, sans,...associé à un nutriment défini dans les 2 points précédents

le nutriscore : note allant de A à E, définie dans la loi Santé de janvier 2016

le taux de TVA : un des quatre taux définis dans la réglementation française

le code nomenclature douanière : code identifiant les marchandises défini par les douanes pour la Déclaration d'Échange de Biens [5]

le pays d'origine pour la DEB : le pays d'origine à déclarer dans la Déclaration d'Échange de Biens [5]

les informations logistiques : il s'agit du plan de conditionnement et de palettisation du produit. Elles regroupent les différents niveaux et les quantités pour passer de l'un à l'autre (ex : 3 boites dans un cartons, 64 cartons dans une palette), les poids et dimensions de ces niveaux logistiques, leurs GTIN, ...

Ajouter un tableau d'exemple avec ces données, en distingant sous forme de multiples tableaux (la logistique à part, etc...) Faire un notebook pour cela.

4.2 Données non structurées

4.2.1 Les libellés

Plusieurs libellés permettent de faire référence à chaque produit, avec des usages distincts :

Libellé temporaire unité de besoin : il s'agit simplement d'un libellé qui est choisi par la personne à l'origine de la demande de référencement produit (souvent l'acheteur, cf. FIGURE 5 page 20), afin que le fournisseur comprenne sur quel produit porte le référencement.

Désignation du produit fournisseur : c'est le libellé qui est donné par le fournisseur afin d'identifier simplement son produit.

Code article interne fournisseur : c'est l'identifiant du produit, dans le système de gestion du fournisseur. Il s'agit généralement de codes, avec autant de formats distincts que de fournisseur venant contribuer.

Marque commerciale du produit : il s'agit du nom de la marque commerciale du produit.

Dénomination réglementaire : c'est le libellé qui doit décrire de manière neutre le produit, sans notion liée au marketing (telle que la marque, entre autres), et qui doit obligatoirement figurer sur l'emballage du produit.

Ajouter ici quelques exemples issus du PIM sur ces libellés.

4.2.2 Les listes d'ingrédients

L'autre grand type de donnée non structurées sont les listes d'ingrédients. La construction des listes d'ingrédients doit suivre les règles suivantes, même si l'application n'est pas toujours parfaitement respectée :

— elle doit détailler l'ensemble des ingrédients, y compris les additifs et les arômes

— elle doit être triée par ordre d'importance pondérale décroissante (i.e. les ingrédients les plus représentatifs en poids doivent être cités en premier)

— la quantité de certains ingrédients (en pourcentage de la masse) par exemple ceux mis en valeur sur l'étiquetage ou dans la dénomination de vente (ex. gâteau aux fraises, pizza au jambon)

Même s'il ne s'agit pas d'une exigence réglementaire, le Groupe Pomona demande à ses fournisseurs de ne pas distinguer les ingrédients par phase comme cela se fait parfois. Cela signifie, par exemple, séparer une partie de la composition du produit (la pâte de la garniture pour une tarte, la sauce et les raviolis, ...). De telles pratiques peuvent parfois induire le consommateur en erreur, comme par exemple dans la liste d'ingrédients suivante (s'applique à des chips de légumes) :

Légumes 64% (betterave, panais, carottes, patates douces), huile de tournesol, sel marin.

Sans l'artifice d'avoir regroupé les légumes en une seule phase, le premier ingrédient de la liste aurait pu être l'huile de tournesol, qui est un ingrédient moins attractif pour un consommateur de chips de légumes.

Les contraintes ci-dessus s'appliquant aux listes d'ingrédients font qu'en général, il s'agit d'une énumération d'ingrédients, sans doublon.

Fous-y des exemples ici aussi.

4.3 Pièces jointes

Dans chacune des sections, mentionner la volumétrie de données accessibles (avec les facettes migration, statuts, & compagnie) et tout

4.3.1 Fiches techniques fournisseur

4.3.2 Étiquettes produit

4.3.3 Fiches logistiques fournisseur

4.3.4 Fiches techniques et argumentaires Pomona

4.4 Récapitulatif de la complétude des données

Mettre ici un ou plusieurs tableaux récapitulatifs illustrant les données possédées quantitativement.

4.5 Analyse qualitative des données

Montrer qu'un sondage basique fait que la qualité actuelle est perfectible

Mettre également la distribution numérique des produits par fournisseur et insister sur la difficulté posée par de multiples formats

Dire ici qu'il y a finalement beaucoup de pdf qui possèdent des textes extractibles vs. uniquement des images.

4.6 Les données « manuellement étiquetées »

Montrer comment elles ont été produites

Expliciter les règles de gestion qui ont été listées pendant l'étiquetage manuel

Evaluer la cohérence entre étiquettes manuelles et contenu du PIM

Troisième partie

LES OBJECTIFS DE CE PROJET

Chapitre 5

LES CAS D'USAGE

5.1 Objectifs : Qualité et productivité

5.2 La préalimentation d'information

5.3 Le contrôle à la saisie fournisseur

5.4 L'aide aux vérifications Pomona

5.5 Les contrôles en masse asynchrones

Chapitre 6

LES TYPES DE DONNÉES À RÉCUPÉRER

6.1 La composition produit

6.2 Les données nutritionnelles

6.3 Les données logistiques

Chapitre 7

LE CHOIX DU CAS D'USAGE

7.1 Les multiples formats

7.2 Les informations « spatialisées »

Au vu des différentes contraintes listées plus haut, on s'attachera à extraire les listes d'ingrédients *des produits* alimentaires *de la branche* EpiSaveurs *depuis* les fiches techniques fournisseur*, en se basant sur* le contenu textuel *de ces documents.*

Quatrième partie

CONSTRUCTION DU MODÈLE

Chapitre 8

LES PRINCIPES GÉNÉRAUX

8.1 Contenu du texte d'une liste d'ingrédients

En général, chaque ingrédient sera présent une seule fois dans la liste.

Le calcul d'embeddings via des modèles tels que SVD ou Word2Vec fait peu de sens.

l'extraction des textes se fait au format Bag Of Words*, sans utiliser de notion d'IDF. L'utilsation de TF semble églament sujette à caution.*

8.2 Limitation à l'identification des listes d'ingrédients

On est sur une taxonomie d'informations limitée dans les fiches techniques.

On pourrait envisager de classifier l'ensemble des textes présents dans les fiches techniques.

Mais l'absence de données étiquetées rend cette tâche impossible. La charge d'étiquetage d'un nombre représentatif de blocs de texte de fiches techniques est trop importante pour être mise en oeuvre dans le cadre de ce projet.

8.3 Conversion de documents en texte

dire ici qu'on utilise principalement pdfminer vs. d'autres outils d'OCR.

De plus, on partira dans un premier temps sur une transformation basique d'un document en texte, sans passer par une analyse de la localisation des textes sur le document.

Chapitre 9
CONSTRUCTION D'UN MODÈLE SIMPLE « OUVERT »

Expliciter le principe de ce modèle avec un schéma simple.

Pas de mesure possible de la performance

9.1 Extraction des données

Ne garder que produits d'épicerie et boissons non alcoolisées

9.2 Conversion en blocs de texte

9.3 Train/Test split

9.4 Entrainement du modèle

9.5 Calcul de la similarité

9.6 Illustration des résultats obtenus

Chapitre 10
UTILISATION DES DONNÉES MANUELLEMENT ÉTIQUETÉES

Expliciter pourquoi on ne peut pas faire tourner (référence parties précédentes) sur l'ensemble des données

10.1 Chargement des données manuellement étiquetées

10.2 Train/Test split

10.3 Entraînement du modèle

10.4 Illustration des prédictions obtenues

Chapitre 11

MESURE DE LA PERFORMANCE

11.1 Précision

11.1.1 Approche naïve

11.1.2 Avec du « text-postprocessing »

11.2 Similarité cosinus

11.3 Fonction de *loss* spécifique

Expliciter les diverses distances, et pourquoi certaines sont plus pertinentes que d'autres.

Ex : on ne garde pas la distance de Hamming

11.3.1 Distance de Levenshtein

11.3.2 Distance de Dameray-Levenshtein

11.3.3 Distance de Jaro

11.3.4 Distance de Jaro-Wrinkler

11.4 Cross-validation des modèles précédents

11.4.1 Modèle « ouvert »

11.4.2 Modèle entraîné sur les données étiquetées manuellement

Chapitre 12

TRANSFER LEARNING

12.1 Principe du pré-entraînement

Expliquer qu'il s'agit d'une approche hybride des 2 modèles précédents

12.2 Illustration de l'impact sur la performance

Chapitre 13

HYPERPARAMETER TUNING

13.1 Les paramètres ajustables

13.2 Application d'une grid search

Cinquième partie

TRAVAUX SUBSÉQUENTS

Chapitre 14

OPÉRATIONNALISATION DE CETTE MAQUETTE

14.1 Client et sponsor métier

14.2 Définition des règles de gestion

14.3 Mise en place d'une organisation projet

14.4 Industrialisation du code

Prochaines étapes : opérationnalisation via API

Documentation

14.5 Monitoring de la performance du modèle

Chapitre 15
EXTENSION DES FONCTIONNALITÉS OFFERTES

15.1 Prise en compte de nouveaux types de pièces jointes

15.2 Utilisation d'outil d'OCR pour les pdf non structurés

15.3 Mise en place d'outil de spatialisation des textes

15.4 Construction d'outils d'extraction de données connexes à la composition

15.5 Élargissement aux données nutritionnelles

15.6 Extraction d'informations complémentaires

15.7 Évaluation de la performances sur d'autres familles de produits

Sixième partie

ANNEXES

Annexe A

FIGURES, TABLEAUX ET BIBLIOGRAPHIE

LISTE DES TABLEAUX

TABLE DES FIGURES

BIBLIOGRAPHIE

[1] Conseil de l'Union Européenne. Règlement n°1333/2008 sur les additifs alimentaires, dec 2008. https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2008:354:0016:0033:FR:PDF.

[2] Conseil de l'Union Européenne. Règlement n°1907/2006 dit REACH, dec 2006. https://bit.ly/2Jm05v9.

[3] Conseil de l'Union Européenne. Règlement n°1169/2011 dit INCO, nov 2011. https://www.senat.fr/europe/textes_europeens/ue0120.pdf.

[4] Direction Générale de la Concurrence, de la Consommation et de la Répression des Fraudes. Étiquetage des denrées alimentaires : nouvelles règles européennes, jan 2015. https://www.economie.gouv.fr/dgccrf/etiquetage-des-denrees-alimentaires-nouvelles-regles-europeennes.

[5] Direction Générales des Douanes et Droits Indirects. Notions essentielles sur la Déclaration d'Échanges de Biens. https://www.douane.gouv.fr/notions-essentielles-sur-la-declaration-dechanges-de-biens.

[6] GS1. GDSN Trade Item Implementation Guide, nov 2019. https://www.gs1.org/docs/gdsn/tiig/3_1/GDSN_Trade_Item_Implementation_Guide.pdf.

[7] GS1 France. Le réseau GDSN, le canal pour l'échange d'informations produits. https://www.gs1.fr/Notre-offre/Le-reseau-GDSN-le-canal-pour-l-echange-d-informations-produits.

[8] GS1 Global. Global Data Synchronisation Network. https://www.gs1.org/services/gdsn.

[9] GS1 Global. GS1 General Specifications. https://www.gs1.org/docs/barcodes/GS1_General_Specifications.pdf.

[10] Groupe Pomona. Site institutionnel du groupe pomona. https://www.groupe-pomona.fr/.

[11] Wikipedia. Liste des additifs alimentaires. https://fr.wikipedia.org/wiki/Liste_des_additifs_alimentaires.

Annexe B

EXEMPLE DE DOCUMENTS FOURNISSEUR

B.1 Fiches techniques

B.2 Étiquettes produit

B.3 Fiches logistiques

Annexe C

LE CODE UTILISÉ

Analyse quantitative

Pierre Massé

April 17, 2020

1 Analyse quantitative multibranche

[1]:
```
# data analysis
import pandas as pd
pd.options.display.width=108
import numpy as np

# visualization
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib_venn import venn3_unweighted, venn3
import matplotlib as mpl
# mpl.rcParams['text.usetex'] = True
# plt.rcParams['text.latex.preamble'] = [r'\usepackage{lmodern}']

# utils
from pathlib import Path
```

Définition des couleurs :

[2]:
```
c_pomona = tuple(val / 255 for val in [0, 92, 132])
c_terreazur = tuple(val / 255 for val in [0, 152, 170])
c_episaveurs = tuple(val / 255 for val in [255, 69, 0])
c_passionfroid = tuple(val / 255 for val in [109, 32, 124])
c_deliceetcreation = tuple(val / 255 for val in [97, 45, 28])
c_saveursdantoine = tuple(val / 255 for val in [156, 34, 63])
```

On charge les données d'un fichier exporté du système de gestion des branches RHD (SAP).

[3]:
```
path = Path('..') / 'data' / 'export2020.csv'

types = {
    'material': 'object',
    'branch': 'int',
    'plant': 'object',
    'type': 'object',
    'designation': 'object',
    'del_mand': 'bool',
    'del_plant': 'bool',
    'march_group': 'object',
    'storage_cond': 'object',
    'hier': 'object',
}
df = pd.read_csv(path,
                 sep=';',
                 encoding='latin-1',
                 engine='python',
                 header=0,
                 skipfooter=1, # footer line with totals in export
                 dtype=types,
                 true_values=['X'], # for del_mand and del_plant
                 false_values=['', np.nan], # for del_mand and del_plant
                )
```

```
df = df[types.keys()] #filter and reorder columns
```

Parmi les colonnes conservées, on a : - le code article (material)

- le code de branche de création (branch).
  - 1: PassionFroid
  - 2: EpiSaveurs
  - 3: TerreAzur
- le code d'activation sur une branche (plant).
  - 1PPF: PassionFroid
  - 2PES: EpiSaveurs
  - 3PTA: TerreAzur
- le type d'article (type). Seuls ZNEG et ZPRE représententent des aricles de marchandises.
  - ZNEG: Négoce
  - ZPRE: Prestation
  - ZENG: Article d'engagement (fictif pour facturation)
  - ZEMB: Article d'emballage (ex: palette)
  - ZSER: Article de service
- le libellé de l'article (designation)
- si l'article est marqué pour suppression pour toutes les branches (del_mand)
- si l'article est marqué pour suppression sur la branche mentionnée dans la colonne plant (del_plant).
- le groupe de marchandises (march_group) :
  - ZSURGE: Surgelés
  - ZFRAIS: Frais (PassionFroid)
  - ZEPI: Epicerie
  - ZBOI: Boissons
  - ZHYG: Hygiène et chimie
  - ZFLF: Fruits et légumes (TerreAzur)
  - ZPMF: Produits de la mer (TerreAzur)
  - ZFP: Fleurs et plantes
  - ZELAB: Produits élaborés (TerreAzur)
- la condition de stockage (storage_cond) :
  - FR: Frais (PassionFroid)
  - SU: Surgelé,
  - EP: Epicerie,
  - AL: Alcool
  - HY: Hygiène et chimie
  - FL: Fruits et légumes (TerreAzur)
  - FP: Fleurs et plantes
  - MA: Marée
  - SA: Saurisserie (produits élaborés de la mer)
  - SE: Articles de Service
  - PL: Articles de publicité
- la hiérarchie produit (hier). Un plan de classement sur 6 niveaux, représentés par 2 caractères numériques chacun.

On crée une nouvelle feature qui correspond au niveau 1 de la hiérarchie produit.

[4]:
```
# Creation of first level of product hierarchy
df.loc[:, 'hier1'] = df.hier.str[:2]
```

On définit un dictionnaire permettant de rappeler les libellés long des divers codes présents dans le dataset.

[5]:
```
# Label names
lab = {'type': 'Type de produit',
       'march_group': 'Groupe de marchandises',
       'storage_cond': 'Condition de stockage',
       'hier1': 'Niveau 1 hiérarchie produit',
       '1PPF': 'PassionFroid',
       '2PES': 'EpiSaveurs',
       '3PTA': 'TerreAzur',
       'ZNEG': 'Article de négoce',
       'ZPRE': 'Article de prestation',
       'ZSURGE': 'Surgelés',
       'ZFRAIS': 'Frais',
       'ZEPI': 'Epicerie',
       'ZBOI': 'Boissons',
       'ZHYG': 'Hygiène',
```

```
    'ZFLF': 'Fruits et Légumes',
    'ZPMF': 'Produits de la mer',
    'ZELAB': 'Produits élaborés',
    'ZFP': 'Fleurs et plantes',
    'ZAUTRE': 'Autres',
    'SU': 'Surgelés',
    'FR': 'Frais',
    'EP': 'Epicerie',
    'AL': 'Alcool',
    'HY': 'Hygiène',
    'FL': 'Fruits et légumes',
    'MA': 'Marée',
    'FP': 'Fleurs et plantes',
    'SA': 'Saurisserie',
    'PL': 'Publicié',
    '10': 'Beurre, oeufs, fromage',
    '20': 'Elaborés',
    '30': 'Garnitures et fruits',
    '40': 'Produits carnés',
    '50': 'Produits de la mer',
    '60': 'Consommables',
    '70': 'Emballage',
    '80': 'Publicité sur le lieu de vente',
    '83': 'Epicerie',
    '85': 'Liquides',
    '87': 'Hygiène et entretien',
    '90': 'Services',
    '92': 'Fruits',
    '94': 'Légumes',
    '96': 'Produits de la mer Frais',
    '98': 'Fleurs - plantes',
   }
```

[6]:
```
df.loc[[5000, 90000, 100000, 130000, 110000] , :]
```

[6]:

	material	branch	plant	type	designation	del_mand	del_plant	march_group	storage_cond	hier	hier1
5000	15712	2	2PES	ZNEG	PSVNX CERN BRISURE S/AZ SAC 1KGX12 CERNO	True	True	ZEPI	EP	832020500505	83
90000	153086	3	3PTA	ZNEG	MANGUE KENT 351/550G PAD 12F DELIC BR°	False	False	ZFLF	FL	920518010405	92
100000	165387	1	1PPF	ZNEG	SALADE PLT 1KGX12 HAMAL	False	False	ZFRAIS	FR	202520150505	20
130000	203582	1	1PPF	ZPRE	EFFILOCHE BOEUF BARBACOA (2KGX6)/12KG CS	False	False	ZSURGE	SU	401015051505	40
110000	177238	2	2PES	ZNEG	COMP POIRE ALL BIO BTE 5/1X3 STM	False	False	ZEPI	EP	832005451505	83

On va définir deux masques, permettant de filtrer : - les articles actifs (i.e. non supprimé niveau mandant ni branche) - les articles actifs de marchandises (i.e. qui ne sont pas des articles “spéciaux”)

[7]:
```
active_mask = ~df.del_mand & ~df.del_plant
active_march_mask = active_mask & df.type.isin(['ZNEG', 'ZPRE'])
```

On peut calculer la volumétrie d’articles et la représenter comme un histogramme. Les données de Délice et Création et Saveurs d’Antoine sont issue d’estimations fournies par le métier.

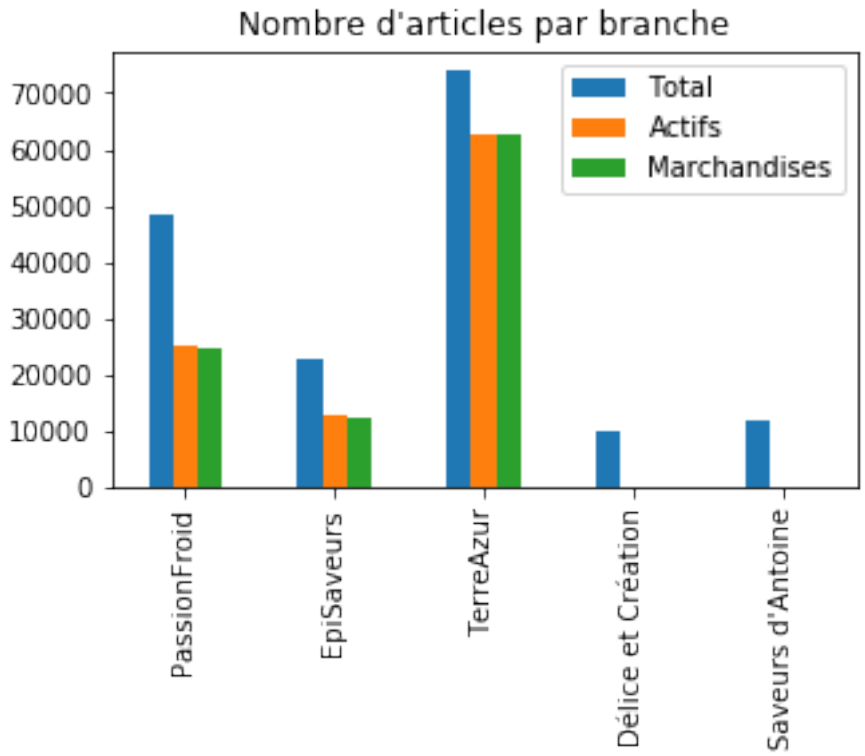
[8]:
```
counts = df.groupby('plant')['material'].count().rename('Total')
filtered_counts = df[active_mask].groupby('plant')['material'].count().rename('Actifs')
filtered_counts2 = df[active_march_mask].groupby('plant')['material'].count().rename('Marchandises')

report = pd.concat([counts, filtered_counts, filtered_counts2], axis=1)
report.loc['Délice et Création', :] = [10000, np.nan, np.nan]
report.loc['Saveurs d\'Antoine', :] = [12000, np.nan, np.nan]
report.rename({'1PPF': 'PassionFroid',
```

```
                '2PES': 'EpiSaveurs',
                '3PTA': 'TerreAzur'},
               inplace=True)
report.index.rename('Branche', inplace=True)
report = report.astype('Int64')
report.to_latex(Path('..') / 'tbls' / 'Articles par branche.tex',
                bold_rows=True,
                column_format='lccc',
                na_rep='-'
               )
report
```

[8]:

Branche	Total	Actifs	Marchandises
PassionFroid	48478	24898	24554
EpiSaveurs	22498	12798	12241
TerreAzur	73804	62789	62710
Délice et Création	10000	NaN	NaN
Saveurs d'Antoine	12000	NaN	NaN

[9]:
```
fig, ax = plt.subplots(figsize=(5,3))
report.plot(kind='bar', ax=ax)
ax.set_title('Nombre d\'articles par branche')
ax.set_xlabel('')
fig.savefig(Path('..') / 'img' / 'Articles par branche.png', bbox_inches='tight')
```



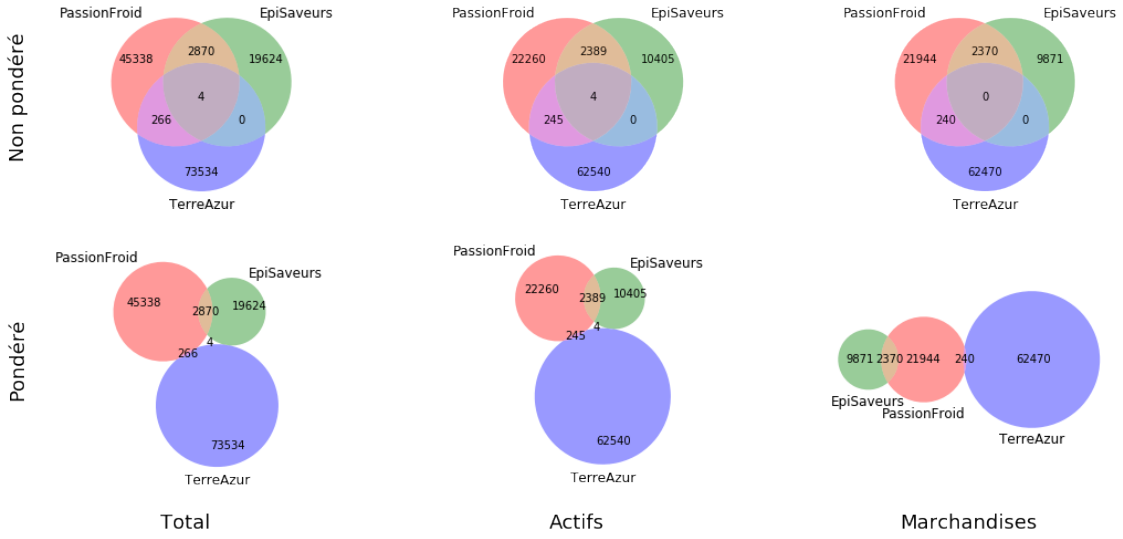
On peut également contruire le diagramme de Venn des articles pour les branches RHD :

[10]:
```
# Filtering the dataset with active materials, and active merchandize materials
branch_sets = [set(df.loc[df.plant == branch_, 'material']) for branch_ in ['1PPF', '2PES', '3PTA']]

filtered_df = df.loc[active_mask]
```

```
filtered_sets = [set(filtered_df.loc[filtered_df.plant == branch_, 'material']) for branch_ in ['1PPF',␣
↪'2PES', '3PTA']]

filtered_march_df = df.loc[active_march_mask]
filtered_march_sets = [set(filtered_march_df.loc[filtered_march_df.plant == branch_, 'material'])
                       for branch_ in ['1PPF', '2PES', '3PTA']]
```

[11]:
```
# This function is used to add label on Venn diagrams axes without showing spines
# (matplotlib-venn disables totally axis's, and spines need to get erased after
# axis's reactivation)
def labelize(ax, label, where='bottom', **kwargs):
    ax.set_axis_on()
    for spine in ['top', 'bottom', 'left', 'right']:
        ax.spines[spine].set_visible(False)
        if where == 'bottom':
            ax.set_xlabel(label, **kwargs)
        elif where == 'left':
            ax.set_ylabel(label, **kwargs)
        else:
            raise ValueError(f"Unexpected 'where' argument: {where}")
```

[12]:
```
# Construction of the diagrams
scope = ['Total', 'Actifs', 'Marchandises']
types = ['Non pondéré', 'Pondéré']
nrows, ncols = len(types), len(scope)

fig, axs = plt.subplots(nrows, ncols, sharex='col', sharey='row', figsize=(18, 8))

for col, source_df in enumerate([branch_sets, filtered_sets, filtered_march_sets]):
    for row, venn_kind in enumerate([venn3_unweighted, venn3]):
        venn_kind(source_df, set_labels=['PassionFroid', 'EpiSaveurs', 'TerreAzur'], ax=axs[row, col])
        if col == 0:
            labelize(axs[row, col], types[row], where='left', fontsize=18, labelpad=10)
        if row == 1:
            labelize(axs[row, col], scope[col], where='bottom', fontsize=18, labelpad=40)

# Ajusting the min and max of axes lims, as they are not the same by default
xmin = min([axs[row][col].get_xlim()[0] for row in range(nrows) for col in range(ncols)])
xmax = max([axs[row][col].get_xlim()[1] for row in range(nrows) for col in range(ncols)])
ymin = min([axs[row][col].get_ylim()[0] for row in range(nrows) for col in range(ncols)])
ymax = max([axs[row][col].get_ylim()[1] for row in range(nrows) for col in range(ncols)]) + 0.1

for row in range(nrows):
    for col in range(ncols):
        axs[row, col].set_xlim(xmin, xmax)
        axs[row, col].set_ylim(ymin, ymax)

# Saving the file to disk so that it is included in the report
fig.savefig(Path('..') / 'img' / 'Diagrammes de Venn articles.png', bbox_inches='tight')
```



On peut constater que les articles utilisés par les 3 branches RHD sont des articles “spéciaux”.

[13]:
```
df[df.material.isin(df.material.value_counts()[df.material.value_counts() >= 3].index)]
```

[13]:

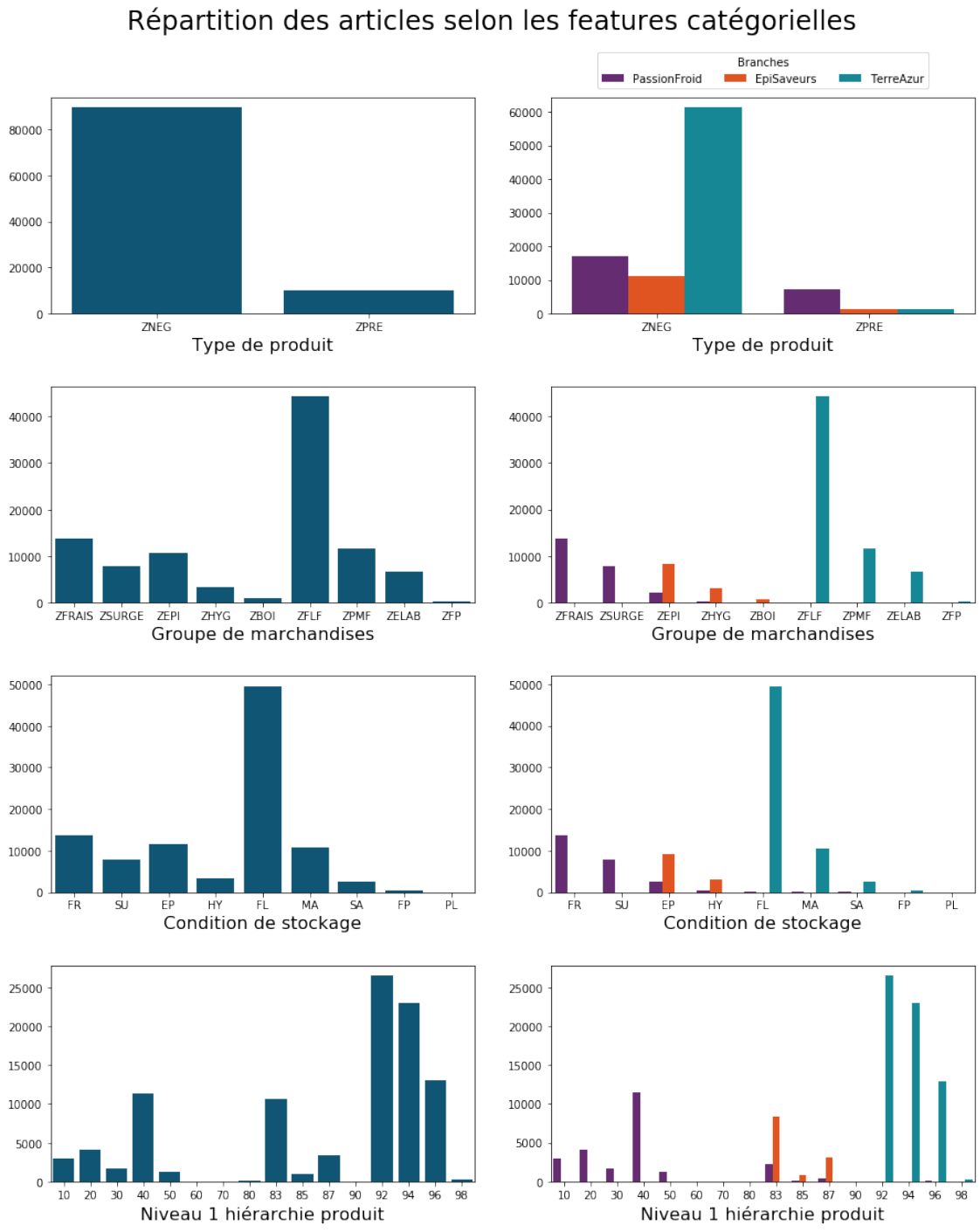
	material	branch	plant	type	designation	del_mand	del_plant	march_group	storage_cond	hier	hier1
144564	DECOMPTE	1	2PES	ZSER	ARTICLE DE DECOMPTE CONDITIONS ARRIERES	False	False	ZAUTRE	NaN	900505050505	90
144565	DECOMPTE	1	3PTA	ZSER	ARTICLE DE DECOMPTE CONDITIONS ARRIERES	False	False	ZAUTRE	NaN	900505050505	90
144566	DECOMPTE	1	1PPF	ZSER	ARTICLE DE DECOMPTE CONDITIONS ARRIERES	False	False	ZAUTRE	NaN	900505050505	90
144612	FC41849	1	1PPF	ZSER	RÉGUL SURFACTURATION DÉCONDITIONNEMENT	False	False	ZAUTRE	NaN	900505050505	90
144613	FC41849	1	2PES	ZSER	RÉGUL SURFACTURATION DÉCONDITIONNEMENT	False	False	ZAUTRE	NaN	900505050505	90
144614	FC41849	1	3PTA	ZSER	RÉGUL SURFACTURATION DÉCONDITIONNEMENT	False	False	ZAUTRE	NaN	900505050505	90
144642	LOT_ENGT	1	1PPF	ZENG	LOT ENGAGEMENT	False	False	NaN	NaN	NaN	NaN
144643	LOT_ENGT	1	3PTA	ZENG	LOT ENGAGEMENT	False	False	NaN	NaN	NaN	NaN
144644	LOT_ENGT	1	2PES	ZENG	LOT ENGAGEMENT	False	False	NaN	NaN	NaN	NaN
144719	S_PALETTE_PERDUE	3	3PTA	ZEMB	PALETTE 80X120 PERDUE	False	False	ZAUTRE	NaN	700510050505	70
144720	S_PALETTE_PERDUE	3	2PES	ZEMB	PALETTE 80X120 PERDUE	False	False	ZAUTRE	NaN	700510050505	70
144721	S_PALETTE_PERDUE	3	1PPF	ZEMB	PALETTE 80X120 PERDUE	False	False	ZAUTRE	NaN	700510050505	70

On peut ensuite essayer de représenter les comptages d’articles sur les diverses variables catégorielles.

[14]:
```
# Definition of feature and order to show
features = {'type': None,
           'march_group': ['ZFRAIS', 'ZSURGE', 'ZEPI', 'ZHYG', 'ZBOI', 'ZFLF', 'ZPMF', 'ZELAB', 'ZFP'],
           'storage_cond': ['FR', 'SU', 'EP', 'HY', 'FL', 'MA', 'SA', 'FP', 'PL'],
           'hier1': None,
          }
```

```
# Definition of color palette
palette = {'1PPF': c_passionfroid,
           '2PES': c_episaveurs,
           '3PTA': c_terreazur,
          }
```

[15]:
```
fig, axs = plt.subplots(nrows=len(features), ncols=2, figsize=(13, 15))
# for each feature, draw counts without and with hue
for idx, (feature, order) in enumerate(features.items()):
    # drawing without hue
    sns.countplot(data=df.loc[active_march_mask],
                  x=feature,
                  order=order,
                  ax=axs[idx][0],
                  color=c_pomona)
    # remove y label, and set x label to full length text
    axs[idx][0].set_ylabel('')
    axs[idx][0].set_xlabel(lab[feature], fontsize=16)
    # drawing with hue
    sns.countplot(data=df.loc[active_march_mask],
                  x=feature,
                  hue='plant',
                  order=order,
                  palette=palette,
                  ax=axs[idx][1],
                 )
    # remove y label, and set x label to full length text
    axs[idx][1].set_ylabel('')
    axs[idx][1].set_xlabel(lab[feature], fontsize=16)
    # hide legend for each axis
    axs[idx][1].legend().set_visible(False)

# redraw legend for the whole figure, above, centered and
# expanded
handles, labels = axs[3][1].get_legend_handles_labels()
fig.legend(handles,
           [lab[label] for label in labels],
           ncol=len(handles),
           title='Branches',
           loc='center',
           bbox_to_anchor=(0, 1, 1, 0.25),
           bbox_transform=axs[0][1].transAxes,
           # mode='expand',
          )

# adding a title
fig.suptitle('Répartition des articles selon les features catégorielles',
             fontsize=24,
             y=1.025,
             va='bottom',
            )

# adding padding between plots
fig.tight_layout(pad=3.0)

# saving to disk
fig.savefig(Path('..') / 'img' / 'Repartition articles categories.png', bbox_inches='tight')
```



[16]:
```
def long_lab(label):
    if label in lab:
        return(label + ' - ' + lab[label])
    else:
        return(label)

for feature in features.keys():
    # Construct the pivot table for the feature
```

```
    piv = pd.pivot_table(df.loc[active_march_mask],
                         columns='plant',
                         index=feature,
                         values='material',
                         aggfunc='count',
                         fill_value='-',
                        )
    # Reorder indices so that they follow the order defined in
    # lab dictionary
    if np.all(piv.index.isin(lab.keys())): # check to avoid filtering piv!
        piv = piv.reindex([key for key in lab.keys() if key in piv.index])

    # Rename indices, columns and axes for pretty printing
    piv = (piv.rename(long_lab, axis=0)
              .rename(lab, axis=1)
              .rename_axis(lab[feature])
              .rename_axis('Branche', axis=1))
    print(piv)
    print('------------------------------------------------------------------')
    piv.to_latex(Path('..') / 'tbls' / ('Repartition par ' + feature + '.tex'),
                 bold_rows=True,
                 column_format='lccc',
                 na_rep='-',
                )
```

```
Branche                      PassionFroid  EpiSaveurs  TerreAzur
Type de produit
ZNEG - Article de négoce            17166       11048      61273
ZPRE - Article de prestation         7388        1193       1437
------------------------------------------------------------------
Branche                   PassionFroid EpiSaveurs TerreAzur
Groupe de marchandises
ZSURGE - Surgelés                 7756          -         -
ZFRAIS - Frais                   13785          6         4
ZEPI - Epicerie                   2298       8305         -
ZBOI - Boissons                    126        826         -
ZHYG - Hygiène                     350       3078         -
ZFLF - Fruits et Légumes             4          -     44133
ZPMF - Produits de la mer          142          -     11594
ZELAB - Produits élaborés           91          -      6644
ZFP - Fleurs et plantes              -          -       297
ZAUTRE - Autres                      2         26        38
------------------------------------------------------------------
Branche                PassionFroid EpiSaveurs TerreAzur
Condition de stockage
SU - Surgelés                  7758          -         -
FR - Frais                    13781          6         3
EP - Epicerie                  2430       9155         -
HY - Hygiène                    344       3080         -
FL - Fruits et légumes           78          -     49508
MA - Marée                      126          -     10501
FP - Fleurs et plantes            -          -       286
SA - Saurisserie                 34          -      2408
PL - Publicié                     2          -         1
------------------------------------------------------------------
Branche                              PassionFroid EpiSaveurs TerreAzur
Niveau 1 hiérarchie produit
10 - Beurre, oeufs, fromage                  3010          6         1
20 - Elaborés                                4150          2         6
30 - Garnitures et fruits                    1701          -         -
40 - Produits carnés                        11413          -         -
50 - Produits de la mer                      1214          -         2
60 - Consommables                               1          -         -
70 - Emballage                                  -          1         -
80 - Publicité sur le lieu de vente            34         25        37
83 - Epicerie                                2306       8296         -
85 - Liquides                                 135        836         -
```

87 - Hygiène et entretien	348	3075	-
90 - Services	10	-	-
92 - Fruits	35	-	26543
94 - Légumes	37	-	22929
96 - Produits de la mer Frais	160	-	12891
98 - Fleurs - plantes	-	-	301

C.1 Gestion du fichier de configuration

Ce petit module a pour but de permettre de gérer les paramètres du progamme dans un fichier de configuration (afin de simplifier la maintenance). Il est utilisé dans l'ensemble des autres modules de ce projet.

```
"""Configuration Module

This module provides a class that enables to fetch configuration stored in the
cfg folder and make it available to other python modules.
"""

import yaml
import os


class Config:

    def __init__(self, env):
        if env not in {'dev', 'int', 'rec', 'qat', 'prd'}:
            raise ValueError(f'Specified env : {env} not expected')
        self.env = env
        self.path = os.path.join(os.path.dirname(__file__),
                                 '..',
                                 'cfg',
                                 'config.yaml')
        stream = open(self.path, 'r')
        data = yaml.safe_load(stream)
        config_keys = data['cross-env'].keys()
        for k in config_keys:
            setattr(self, k, data['cross-env'][k])
        config_keys = data[env].keys()
        for k in config_keys:
            setattr(self, k, data[env][k])
```

Un exemple de fichier de configuration (dont certains champs ont été anonymisés pour des raisons de confidentialité) est présenté ci-dessous. TODO ! ! Mettre le fichier ici.

C.2 Extraction des données du PIM

```
"""PIM API Module

This module aims to enable to fetch data from PIM system, into local folders.
"""

import requests
import os
import json
import warnings
import pandas as pd
import numpy as np
```

```
import copy
import threading
from requests.exceptions import ConnectionError

from . import conf

warnings.simplefilter('always', UserWarning)


class Requester(object):
    """Requester class to retrieve information from PIM
    """
    def __init__(self, env, proxies='default', auth=None):
        self.cfg = conf.Config(env)
        self.session = requests.Session()
        if not auth:
            self.session.auth = (self.cfg.user, self.cfg.password)
        else:
            self.session.auth = auth
        if proxies == 'default':
            try:
                target_proxies = self.cfg.proxies
            except (AttributeError):
                print('No proxy conf found for env : \'{self.cfg.env}\'.'
                      'No proxy will be used.')
                target_proxies = None
        else:
            target_proxies = proxies
        self.session.proxies = target_proxies
        if proxies == 'default':
            # We try the connection with the default proxy conf
            # If it fails, we retry with no proxy configuration
            # (e.g. in the case of working from outside the network)
            try:
                self.check_connection()
            except (ConnectionError):
                self.session.proxies = None
                self.check_connection()
        else:
            # If a proxy configuration has been passed as an argument, we
            # only validate the connexion with that configuration.
            self.check_connection()
        self.rlock = threading.RLock()
        self.result = []
        try:
            self._load_directory()
        except FileNotFoundError:
            warnings.warn('No directory found for current env : '
                          f' \'{self.cfg.env}\'. A new directory should be '
                          'set.')

    def check_connection(self):
        """Checks wether the connection with the environments works
```

```
    The methods tries to get content of PIM system homepage. It returns
    True if the request is handled properly, or raises an exception.
    Note: this method does NOT check whether credentials are correct.
    """
    resp = self.session.get(self.cfg.baseurl)
    if resp.status_code != 200:
        raise ConnectionError('Connection could not be validated during '
                              'check_connection method call.for '
                              f'environment : \'{self.cfg.env}\'')
    return(True)

def check_credentials(self):
    """Checks wether the credentials provided allow to connect to PIM

    The methods tries to get content of PIM root document. It returns
    True if the request is handled properly, or raises an exception.
    Note: if the connection is invalid for another reason (e.g. incorrect
    proyx configuration), an exception will be raised. the
    `check_connection` method should be used prior to checking the
    credentials.
    """
    resp = self.session.get(self.cfg.baseurl +
                            self.cfg.suffixid +
                            self.cfg.rootuid)
    if resp.status_code != 200:
        raise ConnectionError('Connection could not be validated during '
                              'check_credentials method call for '
                              f'environment : \'{self.cfg.env}\'')
    return(True)

def fetch_all_from_PIM(self, nx_properties='*', max_page=None,
                       page_size=None,):
    """Fetches all product data from PIM into result

    This method fetches all product data from PIM. It implements
    multithreading to speed up retrieval.
    """
    query = (f"SELECT * "
             f"FROM Document "
             f"WHERE ecm:primaryType='pomProduct' "
             f"AND ecm:isVersion=0")
    headers = {'Content-Type': 'application/json',
               'X-NXproperties': nx_properties}
    params = {'query': query}
    url = (self.cfg.baseurl +
           self.cfg.suffixid +
           self.cfg.rootuid + '/' +
           '@search')
    result_count = self.query_size(headers, params, url)
    self.result = []
    max_page = max_page if max_page else self.cfg.maxpage
    page_size = page_size if page_size else self.cfg.pagesize
    params['pageSize'] = page_size
    thread_count = result_count // page_size + 1
```

```
    if max_page != - 1 and thread_count > max_page:
        thread_count = max_page
        warnings.warn(f'\nMax size reached ! \n'
                      f'Only {max_page * page_size} results will be '
                      f'fetched out of {result_count} results\n')
    threads = []
    for page_index in range(thread_count):
        t = threading.Thread(target=self.get_page_from_query,
                             args=(url, headers, params, page_index))
        t.start()
        threads.append(t)
    for t in threads:
        t.join()
    print('Done')

def get_page_from_query(self, url, headers, params, currentPageIndex=0):
    """Fetches data for a single page of results

    This methods fetches data for a prepared query for a single page. It
    is used to implement multithreading for `fetch_all_from_PIM`
    """
    try:
        local_params = copy.deepcopy(params)
        local_params['currentPageIndex'] = currentPageIndex
        resp = self.session.get(url,
                                headers=headers,
                                params=local_params)
        with self.rlock:
            self.result.append(resp)
    except Exception as e:
        print('An error occured in this thread!')
        print(e)

def fetch_list_from_PIM(self, iter_uid, batch_size=50, nx_properties='*'):
    """Fetches data from an uid iterable, from PIM

    This method fetches the data from PIM based on a iterable of uids.
    It creates threads to fetch the complete list, and bases on the
    Nuxeo @search API with a WHERE clause. It may therefore be less
    fast than a full fetch.
    Due to http limitations, it requires that no more than 50 uid be
    retrieved at a time in a single thread. Failing to enforce will result
    in incomplete responses an missed results.
    """
    if batch_size > 50:
        raise ValueError(f'batch_size needs to be < 50. '
                         f'Call with:{batch_size}')
    query = (f"SELECT * "
             f"FROM Document "
             f"WHERE ecm:primaryType='pomProduct' "
             f"AND ecm:isVersion=0")
    headers = {'Content-Type': 'application/json',
               'X-NXproperties': nx_properties}
    url = (self.cfg.baseurl +
```

```
            self.cfg.suffixid +
            self.cfg.rootuid + '/' +
            '@search')
        uid_lists = [iter_uid[i:i+batch_size]
                     for i in range(0, len(iter_uid), batch_size)]
        thread_list = []
        self.result = []
        for uid_list in uid_lists:
            uid_string = ', '.join(["'" + str(uid) + "'" for uid in uid_list])
            local_query = query + f" AND ecm:uuid in ({uid_string})"
            params = {'query': local_query}
            t = threading.Thread(target=self.get_page_from_query,
                                 args=(url, headers, params))
            t.start()
            thread_list.append(t)
        for thread in thread_list:
            thread.join()
        print('Done')

    def query_size(self, headers, params, url):
        """Runs a single result query to get the number of results

        This methods takes a query to be sent to Nuxeo in order to count the
        number of results.
        It then execute the query with page_size=1 and max_page=1 to fetch a
        single result, and extracts the total number of results from the
        response.
        """
        loc_headers = copy.deepcopy(headers)
        loc_headers['X-NXproperties'] = ''
        loc_params = copy.deepcopy(params)
        loc_params['pageSize'] = 1
        loc_params['currentPageIndex'] = 0
        resp = self.session.get(url,
                                headers=loc_headers,
                                params=loc_params)
        return(resp.json()['resultsCount'])

    def _root_path(self):
        """Returns the root path for the dumps
        """
        return(os.path.join(os.path.dirname(__file__),
                            '..',
                            'dumps',
                            self.cfg.env))

    def dump_data_from_result(self, filename='data.json',
                              update_directory=True, root_path=None):
        """Dumps data from result attribute as JSON files

        This method dumps data from result as JSON files.
        Note that result MUST be an iterable of responses (be it from PIM or
        from disk), each response should have an 'entries' list of documents.
        """
```

```
    if update_directory and not hasattr(self, '_directory'):
        self._load_directory()
    now = pd.Timestamp.now(tz='UTC')
    threads = []
    for single_result in self.result:
        t = threading.Thread(target=self.dump_data_from_single_result,
                             args=(single_result, filename, now),
                             kwargs={'update_directory': update_directory,
                                     'root_path': root_path})
        t.start()
        threads.append(t)
    for t in threads:
        t.join()
    print('Done')

def dump_data_from_single_result(self, single_result, filename, now,
                                 update_directory=True, root_path=None):
    """Dumps data from a single result

    This method dumps data from a single result passed as an argument.
    It is used for multithreading the result list.
    """
    try:
        doc_list = single_result.json()['entries']
        s_list = []
        if root_path is None:
            root_path = self._root_path()
        for document in doc_list:
            path = os.path.join(root_path, document['uid'])
            if not os.path.exists(path):
                os.makedirs(path)
            full_path = os.path.join(path, filename)
            with open(full_path, 'w+') as outfile:
                json.dump(document, outfile)
            s_list.append(pd.Series(now,
                                    index=[document['uid']],
                                    name='lastFetchedData'))
        df = pd.concat(s_list, axis=0)
        if update_directory:
            with self.rlock:
                self._directory.update(df)
                self._save_directory()
    except Exception as e:
        print('An error occured in this thread!')
        print(e)

def dump_files_from_result(self, update_directory=True, root_path=None):
    """Dumps attached files from result items on disk

    This method dumps files from PIM on disk.
    It also updates the local directory with current datetime.
    Note that result MUST be an iterable of responses (be it from PIM or
    from disk), each response should have an 'entries' list of documents,
    and each document mention the files attached to it.
```

```
    Attached files definition MUST be set in the config.yaml file.
    """
    if update_directory and not hasattr(self, '_directory'):
        self._load_directory()
    now = pd.Timestamp.now(tz='UTC')
    threads = []
    print(f'Launching {len(self.result)} threads.')
    for single_result in self.result:
        t = threading.Thread(target=self.dump_files_from_single_result,
                             args=(single_result, now),
                             kwargs={'update_directory': update_directory,
                                     'root_path': root_path})
        t.start()
        threads.append(t)
    for t in threads:
        t.join()
    print('Done')

def dump_files_from_single_result(self, single_result, now,
                                  update_directory=True, root_path=None):
    """Dumps attached files from items on disk - for a single result

    This method dumps files from a single result into the disk. It is used
    for multithreading in `dump_files_from_result` method.
    """
    try:
        doc_list = single_result.json()['entries']
        s_list = []
        if root_path is None:
            root_path = self._root_path()
        for document in doc_list:
            path = os.path.join(root_path, document['uid'])
            if not os.path.exists(path):
                os.makedirs(path)
            self.dump_attached_files(document, path)
            s_list.append(pd.Series(now,
                                    index=[document['uid']],
                                    name='lastFetchedFiles'))
        df = pd.concat(s_list, axis=0)
        if update_directory:
            with self.rlock:
                self._directory.update(df)
                self._save_directory()
        print('Thread complete!')
    except Exception as e:
        print('An error occured in this thread!')
        print(e)

def _dump_file(self, file_url, path, filename='file'):
    """Dumps a file on fisk from its url"""
    resp = self.session.get(file_url,
                            auth=(self.cfg.user, self.cfg.password),
                            stream=True)
    full_path = os.path.join(path, filename)
```

```
        with open(full_path, 'wb') as outfile:
            outfile.write(resp.content)

    def dump_attached_files(self, document, path):
        """Dumps attached files from a nuxeo document

        This function fetches attached files from a nuxeo document (stored as
        a JSON).
        Files definitions are stored in the configuration file.
        """
        for filekind, filedef in self.cfg.filedefs.items():
            try:
                pointer = document
                for node in filedef['nuxeopath']:
                    pointer = pointer[node]
                nxfilename = pointer['name']
                url = (self.cfg.baseurl +
                       self.cfg.suffixfile +
                       self.cfg.nxrepo +
                       document['uid'] + '/' +
                       filedef['nuxeopath'][-1])
                ext = Requester.compute_extension(nxfilename)
                filename = filedef['dumpfilename'] + '.' + ext
                self._dump_file(url, path, filename=filename)
            except TypeError:
                pass

    def get_directory(self, **kwargs):
        """Get the uid directory for the environment as a pandas DataFrame

        This function fetches the uid directory data for the current
        environment as a pandas DataFrame.
        To fetch all the results, set max_page attribute to -1.
        This function returns data as is from PIM, and requires it to be
        formatted as a directory later on with `_init_as_directory` or
        `_format_as_directory` methods. (else, columns are not consistent with
        directory definition)
        """
        self.fetch_all_from_PIM(nx_properties='', **kwargs)
        df_list = []
        for single_result in self.result:
            df_list.append(pd.DataFrame(single_result.json()['entries'])
                           .set_index('uid'))
        df = pd.concat(df_list)
        df['lastModified'] = pd.to_datetime(df.loc[:, 'lastModified'])
        return(df)

    @staticmethod
    def _directory_headers():
        """Returns the directory headers
        """
        headers = ['type',
                   'title',
                   'lastModified',
```

```
                'lastRefreshed',
                'lastFetchedData',
                'lastFetchedFiles']
        return(headers)

    def _load_directory(self, filename=None):
        """Loads directory from disk into the tool attribute
        """
        directory_filename = filename if filename else self.cfg.uiddirectory
        full_path = os.path.join(self._root_path(), directory_filename)
        self._directory = (pd.read_csv(full_path,
                                       encoding='utf-8-sig',
                                       parse_dates=['lastModified',
                                                    'lastRefreshed',
                                                    'lastFetchedData',
                                                    'lastFetchedFiles'])
                           .set_index('uid'))
        self._directory = Requester._format_as_directory(self._directory)

    @staticmethod
    def _init_as_directory(df):
        """Sets initial dates values for dataframes to be used as directories
        """
        df['lastRefreshed'] = pd.Timestamp.now(tz='UTC')
        df['lastFetchedData'] = np.nan
        df['lastFetchedFiles'] = np.nan
        return(Requester._format_as_directory(df))

    @staticmethod
    def _format_as_directory(df):
        """Formats a dataframe as a directory from dtypes point of view

        This method formats a dataframe columns as to be consistent with
        directory definition. It DOES NOT set initial values (see
        `_init_as_directory`).
        """
        types = {'lastFetchedData': 'datetime64[ns, UTC]',
                 'lastFetchedFiles': 'datetime64[ns, UTC]'}
        df = df.astype(types)
        df = df.loc[:, Requester._directory_headers()]
        return(df)

    def reset_directory(self, page_size=None, max_page=None, filename=None):
        """COMPLETELY RESETS the directory for the environment

        Warning: this function will completely reset the directory for the
        current environment. It should only be used when first setting the
        directory or if it requires being remade from scratch.
        Warning: this function should never be used with max_page parameter
        different from -1 as it could result in incomplete data to erase
        current directory. Only relevant usage of max_page != -1 is for
        debugging purpose.
        """
        self._directory = Requester._init_as_directory(
```

```
            self.get_directory(max_page=max_page, page_size=page_size))
        self._save_directory()

    def refresh_directory(self, max_page=None, filename=None, page_size=None):
        """Refreshes the directory for the environment

        Warning: this function should never be used with max_page parameter
        different from -1 except for debugging purpose. Yet, doing so does
        not corrupt or lose data whatsoever.
        """
        new_dir = self.get_directory(max_page=max_page, page_size=page_size)
        new_dir = Requester._init_as_directory(new_dir)
        self._load_directory(filename=filename)
        cur_dir = Requester._format_as_directory(self._directory)
        cur_dir.update(new_dir)
        cur_dir = pd.concat([cur_dir,
                             new_dir[~new_dir.index.isin(cur_dir.index)]])
        self._directory = cur_dir
        self._save_directory()

    def _save_directory(self, filename=None):
        """Saves current directory to disk

        This methods saves the current directory to disk.
        """
        directory_filename = filename if filename else self.cfg.uiddirectory
        full_path = os.path.join(self._root_path(), directory_filename)
        self._directory.to_csv(full_path, encoding='utf-8-sig')

    def modified_items(self, what='any', max_results=None):
        """Returns modified items according to directory

        This methods compares the date of last modification stored in the
        directory to the date of last fetching of data or attached files.
        If what = 'any', all outdated items will be returned
        If what = 'data', it will return only items with outdated data
        If what = 'files', it will return only items with outdated files
        max_results enables to fetch only a limited count of results.
        This returns an uid list
        """
        if not hasattr(self, '_directory'):
            self._load_directory()
        df = self._directory
        if what == 'any':
            mask = (df.lastFetchedFiles.isna() |
                    (df.lastModified > df.lastFetchedFiles) |
                    df.lastFetchedData.isna() |
                    (df.lastModified > df.lastFetchedData))
        elif what == 'data':
            mask = (df.lastFetchedData.isna() |
                    (df.lastModified > df.lastFetchedData))
        elif what == 'files':
            mask = (df.lastFetchedFiles.isna() |
                    (df.lastModified > df.lastFetchedFiles))
```

```
    else:
        raise ValueError(f"Unexpected 'what' argument: {what}")
    uid_list = mask.index[mask].tolist()
    if max_results:
        uid_list = uid_list[:max_results]
    return(uid_list)

def modification_report(self):
    """Prints some elements about current state

    This methods bases on current state of directory.
    """
    if not hasattr(self, '_directory'):
        self._load_directory()
    print(f'Number of items: {len(self._directory)}')
    print(f'Number of items with outdated data: '
          f'{len(self.modified_items(what="data"))}')
    print(f'Number of items with outdated files: '
          f'{len(self.modified_items(what="files"))}')

@staticmethod
def compute_extension(filename):
    """Computes the extension from a filename

    Returns the extension. If the filename has no "." (dot) in it, returns
    an empty string. If the computed extension has strictly more than 4
    characters, returns the empty string."""
    splitted = filename.split('.')
    if len(splitted) == 1:
        return('')
    elif len(splitted[-1]) > 4:
        return('')
    else:
        return(filename.split('.')[-1].lower())

def result_to_dataframe(self, record_path=None, meta=None, mapping=None,
                        index=None):
    """Formats result content as a dataframe with defined format

    record_path and meta are pandas json_normalize method arguments
    mapping is a key : adress mapping that maps return dataframe keys to
    data adress in the JSON. Reminder: json_normalize default separator
    is '.'
    index is the field identifier(s) for the field(s) to be used as index
    """
    result_json = [result.json() for result in self.result]
    if mapping:
        with CleanJSONDataFrame(result_json,
                                record_path=record_path,
                                meta=meta) as df:
            for key, path in mapping.items():
                try:
                    df.df[key] = df.df[df.prefix + path]
                except KeyError:
```

```
                    df.df[key] = np.nan
            df = df.df
        else:
            df = pd.json_normalize(result_json,
                                   record_path=record_path,
                                   meta=meta)
        if index:
            df.set_index(index, inplace=True)
        return(df)

    def file_report_from_result(self, mapping, index=None, record_path=None):
        """Returns a dataframe about the files contained in the result

        This methods analyzes the content of the result, and generates a
        dataframe which enables the analysis of the files on the products
        Files to report on are based on the content of the config.yaml
        configuration file.
        """
        if not record_path:
            record_path = 'entries'
        for filekind, filedef in self.cfg.filedefs.items():
            mapping[filekind] = '.'.join(filedef['nuxeopath']) + '.name'
        df = self.result_to_dataframe(record_path=record_path,
                                      mapping=mapping,
                                      meta=None,
                                      index=index)
        for filekind in self.cfg.filedefs:
            df['has_' + filekind] = df[filekind].notna()
            del(df[filekind])
        return(df)


class CleanJSONDataFrame(object):
    """Context manager class for cleanly importing data from JSON

    This class enables to create a context manager that enables to read JSON
    data into a dataframe, by specifying the data to keep (this feature is not
    provided yet by the pandas json_normalize). It does so by following these
    steps:
    - read data from a JSON into a new dataframe
    - enable one to duplicate loaded data into new fields
    - delete the loaded data when exiting, thus keeping only duplicated data
    Complicated prefix is set to avoid duplicates in column names.
    """
    def __init__(self, data, record_path=None, meta=None,
                 prefix='_prev_duplc1'):
        self.df = pd.json_normalize(data, record_path=record_path,
                                    meta=meta, record_prefix=prefix,
                                    meta_prefix=prefix)
        self.prefix = prefix
        self.columns = list(self.df.columns.values)

    def __enter__(self):
        return(self)
```

```
    def __exit__(self, exc_type, exc_val, exc_tb):
        self.df.drop([c for c in self.columns], axis=1, inplace=True)
```

C.3 Conversion des pièces jointes en textes

```
"""PIM PDF Module

This module aims to parse the content of PDF files into text.
"""

import pandas as pd
from multiprocessing import cpu_count
from pathos.multiprocessing import ProcessPool as Pool
from io import StringIO, BytesIO
from functools import partial

from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from pdfminer.pdfdocument import PDFDocument
from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.pdfpage import PDFPage
from pdfminer.pdfparser import PDFParser


class PDFDecoder(object):
    """Tool that provides basic pdf decoding functionnalities
    """

    @staticmethod
    def content_to_text(content,
                        none_content='raise'):
        """Decodes the binary passed as argument

        content arg must be a Bytesio object.
        none_content arg must be raise (if it is not expected to have an empty
        input, the default) or to_empty (which will cause to return an empty
        string)
        """
        if none_content not in {'raise', 'to_empty'}:
            raise ValueError(f'Unexpected value for none_content parameter. '
                             f'Got {none_content} but only \'raise\' or '
                             f'\'to_empty\' are expected.')
        if not content.read():
            if none_content == 'raise':
                raise RuntimeError(f'PDFminer got an empty bytesIO object to '
                                   f'parse')
            if none_content == 'to_empty':
                return('')
        output_string = StringIO()
        parser = PDFParser(content)
        doc = PDFDocument(parser)
        rsrcmgr = PDFResourceManager()
```

```
        device = TextConverter(rsrcmgr, output_string,
                               laparams=LAParams())
        interpreter = PDFPageInterpreter(rsrcmgr, device)
        for page in PDFPage.create_pages(doc):
            interpreter.process_page(page)
        return(output_string.getvalue())

    @staticmethod
    def path_to_text(path, missing_file='raise'):
        """Decodes file at local path in the form of a long string
        """
        if missing_file not in {'raise', 'ignore'}:
            raise ValueError(f'Unexpected value for missing_file parameter. '
                             f'Got {missing_file} but only \'raise\' or '
                             f'\'ignore\' are expected.')
        try:
            with open(path, mode='rb') as content:
                return(PDFDecoder.content_to_text(content))
        except FileNotFoundError:
            if missing_file == 'raise':
                raise
            if missing_file == 'ignore':
                print(f'File not found at path: {path}')
                return('')
        except Exception as e:
            print(e)
            print(f'An error happened at path: {path}')
            return('')

    @staticmethod
    def text_to_blocks(text, split_func=lambda x: x.split('\n\n')):
        """Splits text passed as an argument with splitter function

        split_func should be defined as to return a list like object.
        """
        return(split_func(text))

    @staticmethod
    def text_to_blocks_series(text, index=None,
                              split_func=lambda x: x.split('\n\n'),
                              return_type='along_index'
                              ):
        """Splits text passed as an argument (using splitter func) to a Series

        The return_type can be:
            - 'along_index': the return is a pandas Series of length the number
                             of blocks (with the index provided)
            - 'as_list'    : the return is a pandas Series of length 1, with
                             the provided scalar as index, and the value of the
                             Series being the blocks as a list of strings
        If return_type is 'along_index' (the default), the index arg is passed
        as provided to pd.Series constructor, or if it is a scalar it is
        broadcasted as a constant on all values.
        """
```

```
    blocks_list = PDFDecoder.text_to_blocks(text,
                                            split_func=split_func,
                                            )
    if return_type not in {'along_index', 'as_list'}:
        raise ValueError(f'Unexpected value for return_type parameter. '
                         f'Got {return_type} but only \'along_index\' or '
                         f'\'as_list\' are expected.')
    if return_type == 'as_list':
        return(pd.Series([blocks_list], index=[index]))
    elif return_type == 'along_index':
        try:
            return(pd.Series(blocks_list, index=index))
        except TypeError:
            index = [index] * len(blocks_list)
            return(pd.Series(blocks_list, index=index))

@staticmethod
def path_to_blocks(path, split_func=lambda x: x.split('\n\n'),
                   missing_file='raise'):
    """Decodes file at local path in the form of a list of blocks

    Blocks are part of the original string separated by at least 2
    carriage returns (i.e. with at least a single blank line between them)
    """
    text = PDFDecoder.path_to_text(path, missing_file=missing_file)
    return(PDFDecoder.text_to_blocks(text, split_func=split_func))

@staticmethod
def path_to_blocks_series(path,
                          index=None,
                          split_func=lambda x: x.split('\n\n'),
                          missing_file='raise'):
    """Decodes file at local path in the form a pd Series of blocks
    """
    text = PDFDecoder.path_to_text(path, missing_file=missing_file)
    return(PDFDecoder.text_to_blocks_series(text,
                                            split_func=split_func,
                                            index=index))

@staticmethod
def paths_to_blocks(path_series, split_func=lambda x: x.split('\n\n'),
                    missing_file='raise'):
    """Decodes files for each path in path list as a blocks Dataseries

    Blocks are part of the original string after the splitting function
    has been applied.
    The input must be a pandas dataseries of paths.
    The output is another pd Series, with the same indexes as the initial
    series (broadcasted to match the block count for each path)
    """
    ds_list = []
    for uid, path in path_series.items():
        ds = (PDFDecoder
              .path_to_blocks_series(path,
```

```
                                    split_func=split_func,
                                    index=uid,
                                    missing_file=missing_file))
            ds_list.append(ds)
        return(pd.concat(ds_list, axis=0))

    @staticmethod
    def threaded_paths_to_blocks(path_series, processes=None,
                                 split_func=lambda x: x.split('\n\n'),
                                 missing_file='raise',
                                 ):
        """Threaded version of paths_to_blocks method

        It takes as input a series which index is the uid of the products,
        and the values are the path to the document.
        processes argument is the number of processes to launch. If omitted,
        it defaults to the number of cpu cores on the machine.
        """
        processer = partial(PDFDecoder.path_to_blocks_series,
                            split_func=split_func, missing_file=missing_file)
        processes = processes if processes else cpu_count()
        print(f'Launching {processes} processes.')

        # Pool with context manager do not seem to work due to issue 38501 of
        # standard python library. It hangs when running tests through pytest
        # see: https://bugs.python.org/issue38501
        # Below content should be tested again whenever this issue is closed
        #
        # with Pool(nodes=processes) as pool:
        #     ds_list = pool.map(processer,
        #                        path_series, path_series.index)
        #
        # End of block

        # This temporary solution should be removed when tests mentioned above
        # are successful.
        # This just closes each pool after execution or exception.
        try:
            pool = Pool(nodes=processes)
            pool.restart(force=True)
            ds_list = pool.map(processer,
                               path_series,
                               path_series.index)
        except Exception:
            pool.close()
            raise
        pool.close()
        # End of block

        return(pd.concat(ds_list, axis=0))

    @staticmethod
    def threaded_contents_to_text(content_series,
                                  processes=None,
```

```
                            none_content='raise',
                            ):
    """Threaded version of content_to_text method

    It takes as input a series which index is the uid of the products,
    and the values are the content (in the form of bytes) of the
    documents.
    processes argument is the number of processes to launch. If omitted,
    it defaults to the number of cpu cores on the machine.
    none_content arg can be 'raise' (default) or to_empty
    """
    processer = partial(PDFDecoder.content_to_text,
                        none_content=none_content,
                        )
    processes = processes if processes else cpu_count()
    print(f'Launching {processes} processes.')
    in_ds = content_series.apply(BytesIO)

    # Pool with context manager do not seem to work due to issue 38501 of
    # standard python library. It hangs when running tests through pytest
    # see: https://bugs.python.org/issue38501
    # Below content should be tested again whenever this issue is closed
    #
    # with Pool(nodes=processes) as pool:
    #     tuples = (list(in_ds.index),
    #               pool.map(processer, in_ds))
    #
    # End of block

    # This temporary solution should be removed when tests mentioned above
    # are successful.
    # This just closes each pool after execution or exception.
    try:
        pool = Pool(nodes=processes)
        pool.restart(force=True)
        tuples = (list(in_ds.index), pool.map(processer, in_ds))
    except Exception:
        pool.close()
        raise
    pool.close()
    # End of block

    ds = pd.Series(tuples[1], index=tuples[0])
    return(ds)

@staticmethod
def threaded_texts_to_blocks(text_series, processes=None,
                             split_func=lambda x: x.split('\n\n'),
                             return_type='along_index'
                             ):
    """Threaded version of text_to_blocks_series method

    It takes as input a series which index is the uid of the products,
    and the values are the content (in the form of bytes) of the
```

```
        documents..
        processes argument is the number of processes to launch. If omitted,
        it defaults to the number of cpu cores on the machine.
        As for text_to_blocks_series function, return_type can be 'along_axis'
        or 'list_like'.
        """
        processer = partial(PDFDecoder.text_to_blocks_series,
                            split_func=split_func,
                            return_type=return_type)
        processes = processes if processes else cpu_count()
        print(f'Launching {processes} processes.')

        # Pool with context manager do not seem to work due to issue 38501 of
        # standard python library. It hangs when running tests through pytest
        # see: https://bugs.python.org/issue38501
        # Below content should be tested again whenever this issue is closed
        #
        # with Pool(nodes=processes) as pool:
        #     ds_list = pool.map(processer, text_series, text_series.index)
        #
        # End of block

        # This temporary solution should be removed when tests mentioned above
        # are successful.
        # This just closes each pool after execution or exception.
        try:
            pool = Pool(nodes=processes)
            pool.restart(force=True)
            ds_list = pool.map(processer, text_series, text_series.index)
        except Exception:
            pool.close()
            raise
        pool.close()
        # End of block

        ds = pd.concat(ds_list, axis=0)
        return(ds)
```

C.4 Identification des listes d'ingrédients

```
"""PIM Estimator module

This modules enables to create an estimator to identify which block is the
ingredient list from an iterable of text blocks.
"""

import os
from io import BytesIO

import numpy as np
import pandas as pd
from pathlib import Path
```

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.exceptions import NotFittedError
from sklearn.utils.validation import check_is_fitted
from scipy.sparse.linalg import norm as sparse_norm

from .pimapi import Requester
from .pimpdf import PDFDecoder
from .conf import Config


class CustomTransformer(object):
    """Abstract class for custom transformers
    """
    def __init__(self, source_col, target_col, target_exists):
        self.source_col = source_col
        self.target_col = target_col
        self.target_exists = target_exists

    def raise_if_not_a_df(self, X):
        if not isinstance(X, pd.DataFrame):
            raise TypeError(f'This transformer expects a pandas Dataframe '
                            f'object. Got an object of type \'{type(X)}\' '
                            f'instead')

    def check_target_exists(self):
        if self.target_exists not in {'raise', 'ignore', 'overwrite'}:
            raise ValueError(f'target_exists parameter should be set to '
                             f'\'raise\' or \'ignore\' or \'to_nan\'. Got '
                             f'\'{self.target_exists}\' instead.')

    def raise_if_target(self, X):
        if self.target_col in X.columns and self.target_exists == 'raise':
            raise RuntimeError(f'Column \'{self.target_col}\' already exists '
                               f'in input DataFrame.')

    def raise_if_no_source(self, X):
        if self.source_col and self.source_col not in X.columns:
            raise KeyError(f'Input DataFrame has no \'{self.source_col}\' '
                           f'column.')

    def fit(self, X, y=None):
        self.check_target_exists()
        self.raise_if_not_a_df(X)
        self.raise_if_target(X)
        self.raise_if_no_source(X)

    def transform(self, X):
        check_is_fitted(self)
        self.check_target_exists()
        self.raise_if_not_a_df(X)
        self.raise_if_target(X)
        self.raise_if_no_source(X)
        if self.target_col in X.columns and self.target_exists == 'ignore':
            return(X)
```

```
    def fit_transform(self, X, y=None):
        return(self.fit(X).transform(X))

    def get_params(self, deep=True):
        parms = dict()
        parms['source_col'] = self.source_col
        parms['target_col'] = self.target_col
        parms['target_exists'] = self.target_exists
        return(parms)

    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return(self)


class IngredientExtractor(object):
    """Estimator that identifies the most 'ingredient like' block from a list
    """
    def __init__(self):
        """Constructor method of ingredient extractor"""
        pass

    def fit(self, X, y=None):
        """Fitter method of ingredient extractor

        X is an iterable of ingredient lists in the form of strings
        y is just here for compatibility in sklearn pipeline usage
        """
        self._count_vect = CountVectorizer()
        self.vectorized_texts_ = self._count_vect.fit_transform(X)
        self.vocabulary_ = self._count_vect.vocabulary_
        self.mean_corpus_ = self.vectorized_texts_.mean(axis=0)
        return(self)

    def predict(self, X):
        """Predicter method of ingredient extractor

        X is a list of text blocks.
        This methods returns the index of the text block that is most likely
        to hold the ingredient list"""
        X_against_ingred_voc = self._count_vect.transform(X)
        X_norms = sparse_norm(CountVectorizer().fit_transform(X), axis=1)
        X_dot_ingred = np.array(X_against_ingred_voc.sum(axis=1)).squeeze()
        pseudo_cosine_sim = np.divide(X_dot_ingred,
                                      X_norms,
                                      out=np.zeros(X_norms.shape),
                                      where=X_norms != 0)
        self.similarity_ = pseudo_cosine_sim
        return(np.argmax(pseudo_cosine_sim))

    def show_emphasize(self, X):
        """Method that prints strings with words from vocabulary emphasized
```

```
    """
    for text in self.emphasize_texts(X):
        print(text)

def emphasize_texts(self, X):
    """Method that returns strings with words from vocabulary emphasized

    This method shows how some candidates texts are projected on the
    vocabulary that has been provided or gotten from fitting.
    It is useful to see how different blocks compare.
    X argument is an iterable of block candidates.
    """
    check_is_fitted(self)
    preprocessor = self._count_vect.build_preprocessor()
    tokenizer = self._count_vect.build_tokenizer()
    vocabulary = self._count_vect.vocabulary_
    emphasized_texts = []
    for block in X:
        text = self.emphasize_words(block,
                                    preprocessor=preprocessor,
                                    tokenizer=tokenizer,
                                    vocabulary=vocabulary,
                                    )
        emphasized_texts.append(text)
    return(emphasized_texts)

def emphasize_words(self,
                    text,
                    preprocessor=None,
                    tokenizer=None,
                    vocabulary=None,
                    ansi_color='\033[92m',  # green by default
                    ):
    """Method that returns a string with words emhasized

    This methods takes a string and returns a similar string with the words
    emphasized (with color markers)
    """
    check_is_fitted(self)
    ansi_end_block = '\033[0m'
    if not preprocessor:
        preprocessor = self._count_vect.build_preprocessor()
    if not tokenizer:
        tokenizer = self._count_vect.build_tokenizer()
    if not vocabulary:
        vocabulary = self._count_vect.vocabulary_
    preprocessed_text = preprocessor(text)
    tokenized_text = tokenizer(preprocessed_text)
    idx = 0
    emphasized_text = ''
    for token in tokenized_text:
        if token in vocabulary:
            while preprocessed_text[idx: idx + len(token)] != token:
                emphasized_text += text[idx]
```

```
                idx += 1
            emphasized_text += (ansi_color + text[idx: idx + len(token)] +
                                ansi_end_block)
            idx += len(token)
        emphasized_text += text[idx:]
        return(emphasized_text)

    def score(self, X, y):
        """Scorer method of ingredient extractor estimator

        X is an iterable of ingredient lists in the form of string
        y is the target as the index of the correct block.
        """
        pass


class PIMIngredientExtractor(IngredientExtractor):
    """Wrapped estimator that directly extracts the ingredient list from uid
    """
    def __init__(self, env='prd', **kwargs):
        self.requester = Requester(env, **kwargs)
        super().__init__()

    def compare_uid_data(self, uid):
        check_is_fitted(self)
        print(f'Fetching data from PIM for uid {uid}...')
        self.requester.fetch_list_from_PIM([uid])
        try:
            ingredient_list = (self.requester.result[0]
                               .json()['entries'][0]['properties']
                               ['pprodc:ingredientsList'])
        except IndexError:
            raise IndexError(f'Fetching data with uid {uid} ')
        print('-------------------------------------------------------------')
        print(f'Ingredient list from PIM is :\n\n{ingredient_list}')
        print('\n-------------------------------------------------------------')
        print(f'Supplier technical datasheet from PIM for uid {uid} is:')
        nuxeo_path = (self.requester.cfg.filedefs['supplierdatasheet']
                      ['nuxeopath'])
        pointer = self.requester.result[0].json()['entries'][0]
        for node in nuxeo_path:
            pointer = pointer[node]
        file_url = pointer['data']
        print(file_url)
        print('-------------------------------------------------------------')
        print(f'Downloading content of technical datasheet file...')
        self.resp = self.requester.session.get(file_url,
                                               stream=True)
        resp = self.resp
        print('Done!')
        print('-------------------------------------------------------------')
        print(f'Parsing content of technical datasheet file...')
        blocks = (PDFDecoder.content_to_text(BytesIO(resp.content))
                  .split('\n\n'))
```

```
        idx = self.predict(blocks)
        print('Done!')
        print('------------------------------------------------------------')
        print(f'Ingredient list extracted from technical datasheet:\n')
        print(blocks[idx])
        print('\n------------------------------------------------------------')

    def print_blocks(self):
        blocks = (PDFDecoder.content_to_text(BytesIO(self.resp.content))
                  .split('\n\n'))
        for i, block in enumerate(blocks):
            print(i, ' | ', block, '\n')


class PathGetter(CustomTransformer):
    """Class that gets path for documents on disk

    This class aims to compute the path to documents, in order to
    fetch documents from the correct folder (depending on whether
    they are from train set or from ground truth)
    All these can be set at initialization, if such is not the case
    then their values is gotten from the configuration file.
    """
    def __init__(self,
                 env='prd',
                 ground_truth_uids=None,
                 train_set_path=None,
                 ground_truth_path=None,
                 path_factory=lambda x: x,
                 filename_factory=lambda x: 'FTF.pdf',
                 source_col=None,
                 target_col='path',
                 target_exists='raise'
                 ):
        self.env = env
        self.ground_truth_uids = ground_truth_uids
        self.train_set_path = train_set_path
        self.ground_truth_path = ground_truth_path
        self.path_factory = path_factory
        self.filename_factory = filename_factory
        super().__init__(source_col=source_col, target_col=target_col,
                         target_exists=target_exists)

    def fit(self, X, y=None):
        """No fit is required for this class.
        """
        super().fit(X)
        self.cfg = Config(self.env)
        if not self.train_set_path:
            self.train_set_path = os.path.join(*self.cfg.trainsetpath)
        if not self.ground_truth_path:
            self.ground_truth_path = os.path.join(*self.cfg.groundtruthpath)
        self.fitted_ = True
        return(self)
```

```
    def transform(self, X):
        """Returns the paths for the uids"""
        super().transform(X)
        df = X.copy()
        df[self.target_col] = None
        for uid in X.index:
            if uid in self.ground_truth_uids:
                path = os.path.join(self.ground_truth_path,
                                    self.path_factory(uid),
                                    self.filename_factory(uid),
                                    )
            else:
                path = os.path.join(self.train_set_path,
                                    self.path_factory(uid),
                                    self.filename_factory(uid),
                                    )
            df.loc[uid, self.target_col] = path
        return(df)

    def get_params(self, deep=True):
        parms = super().get_params()
        parms['env'] = self.env
        parms['ground_truth_uids'] = self.ground_truth_uids
        parms['train_set_path'] = self.train_set_path
        parms['ground_truth_path'] = self.ground_truth_path
        parms['path_factory'] = self.path_factory
        parms['filename_factory'] = self.filename_factory
        return(parms)


class ContentGetter(CustomTransformer):
    """Class that fetches the content of documents on disk

    This class fetches the data from documents on disk as bytes.
    It requires a dataframe with a path column"""
    def __init__(self,
                 missing_file='raise',
                 target_exists='raise',
                 source_col='path',
                 target_col='content',
                 ):
        self.missing_file = missing_file
        super().__init__(source_col=source_col,
                         target_col=target_col,
                         target_exists=target_exists,
                         )

    def fit(self, X, y=None):
        super().fit(X)
        if self.missing_file not in {'raise', 'ignore', 'to_nan'}:
            raise ValueError(f'missing_file parameter should be set to '
                             f'\'raise\' or \'ignore\' or \'to_nan\'. Got '
                             f'\'{self.missing_file}\' instead.')
```

```
        self._raise_if_no_file(X)
        self.fitted_ = True
        return(self)

    def _raise_if_no_file(self, X):
        if self.missing_file == 'raise':
            mask = pd.DataFrame(index=X.index)
            mask['file_exists'] = X['path'].apply(ContentGetter.file_exists)
            if not mask['file_exists'].all():
                example_uid = mask.loc[~mask['file_exists']].index[0]
                example_path = X.loc[example_uid, 'path']
                raise RuntimeError(f'No file found for uid \'{example_uid}\' '
                                   f'at path \'{example_path}\'')

    def transform(self, X):
        super().transform(X)
        self._raise_if_no_file(X)
        X = X.copy()
        mask = pd.DataFrame(index=X.index)
        mask['file_exists'] = X['path'].apply(ContentGetter.file_exists)
        mask['target'] = X['path'].apply(ContentGetter.read_to_bytes)
        if self.missing_file == 'to_nan':
            idx_to_update = mask.index
        else:
            idx_to_update = mask['file_exists']
        X.loc[idx_to_update, 'content'] = mask.loc[idx_to_update, 'target']
        return(X)

    @staticmethod
    def read_to_bytes(path):
        try:
            return(Path(path).read_bytes())
        except FileNotFoundError:
            return(None)

    @staticmethod
    def file_exists(path):
        path = Path(path)
        return(path.is_file())

    def get_params(self, deep=True):
        parms = super().get_params()
        parms['missing_file'] = self.missing_file
        return(parms)


class PDFContentParser(CustomTransformer):
    """Class that parses pdf content to text

    This class converts a file content (in the form of bytes) into text, using
    pimpdf functionalities (based on pdfminer.six)
    """
    def __init__(self,
                 target_exists='raise',
```

```
                source_col='content',
                target_col='text',
                none_content='raise'
                ):
        self.none_content = none_content
        super().__init__(source_col=source_col,
                         target_col=target_col,
                         target_exists=target_exists,
                         )

    def fit(self, X, y=None):
        super().fit(X)
        self.fitted_ = True
        return(self)

    def transform(self, X):
        super().transform(X)
        X = X.copy()
        tran = (PDFDecoder
                .threaded_contents_to_text(X[self.source_col],
                                           none_content=self.none_content))
        X[self.target_col] = tran
        return(X)

    def get_params(self, deep=True):
        parms = super().get_params()
        parms['none_content'] = self.none_content
        return(parms)


class BlockSplitter(CustomTransformer):
    """Class that splits texts into blocks

    This class converts a text string into blocks (a list of string), using
    the splitter function provided
    """
    def __init__(self,
                target_exists='raise',
                source_col='text',
                target_col='blocks',
                splitter_func=(lambda x: x.split('\n\n'))
                ):
        self.splitter_func = splitter_func
        super().__init__(target_exists=target_exists,
                         source_col=source_col,
                         target_col=target_col,
                         )

    def fit(self, X, y=None):
        super().fit(X)
        self._check_splitter_callable()
        self.fitted_ = True
        return(self)
```

```
    def _check_splitter_callable(self):
        self.splitter_func('')

    def transform(self, X):
        super().transform(X)
        X = X.copy()
        blocks = (PDFDecoder
                  .threaded_texts_to_blocks(X[self.source_col],
                                            split_func=self.splitter_func,
                                            return_type='as_list',
                                            ))
        X[self.target_col] = blocks
        return(X)

    def get_params(self, deep=True):
        parms = super().get_params()
        parms['splitter_func'] = self.splitter_func
        return(parms)


class SimilaritySelector(CustomTransformer):
    """Class that select the most similar block from a block list

    This class provides functionnalities to fit an estimator on a topic
    specific vocabulary, and to retrieve the best candidate amongst these
    blocks.
    It can append a new column to an input pandas DataFrame with the best
    candidate.
    """
    def __init__(self,
                 source_col='blocks',
                 target_col='predicted',
                 target_exists='raise',
                 fit_col='Ingrédients',
                 count_vect_kwargs=dict(),
                 ):
        super().__init__(source_col=source_col,
                         target_col=target_col,
                         target_exists=target_exists,
                         )
        self.fit_col = fit_col
        self.count_vect_kwargs = count_vect_kwargs

    def fit(self, X, y=None):
        super().fit(X)
        self.count_vect = CountVectorizer(self.count_vect_kwargs)
        self.count_vect.fit(X[self.fit_col].fillna(''))
        self.source_count_vect = CountVectorizer()
        self.fitted_ = True
        return(self)

    def predict(self, block_list):
        check_is_fitted(self)
        try:
```

```
            check_is_fitted(self.source_count_vect)
        except NotFittedError:
            self.source_count_vect.fit(block_list)
        X_norms = sparse_norm(self.source_count_vect.transform(block_list),
                              axis=1)
        X_against_ingred_voc = self.count_vect.transform(block_list)
        X_dot_ingred = np.array(X_against_ingred_voc.sum(axis=1)).squeeze()
        pseudo_cosine_sim = np.divide(X_dot_ingred,
                                      X_norms,
                                      out=np.zeros(X_norms.shape),
                                      where=X_norms != 0)
        self.similarity_ = pseudo_cosine_sim
        return(block_list[np.argmax(pseudo_cosine_sim)])

    def transform(self, X):
        super().transform(X)
        X = X.copy()
        block_texts = (text for block in X[self.source_col] for text in block)
        self.source_count_vect.fit(block_texts)
        X[self.target_col] = X[self.source_col].apply(self.predict)
        return(X)
```