

# Classification multi-modale de produits e-commerce

## Rakuten : rapport d'exploration et de prétraitement des données

RIZLÈNE BANAT – BRYAN KHAN MAHMOOD – ROMAIN MAZOYER – STEEVE TRINCAL

**Mots-clés :** data exploration, data analysis, data visualization, wordcloud, correlation, missing values, preprocessing, text cleaning, transformers, word embedding, text embedding, normalization, image preprocessing, feature engineering, over-sampling, under-sampling, pipeline.

**GitHub :** <https://github.com/fibonacci/rakuten-multimodal-classification>

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Contexte . . . . .	3
1.2	Objectifs . . . . .	3
1.3	Jeu de données . . . . .	3
<b>2</b>	<b>Exploration des données</b>	<b>4</b>
2.1	Catégories . . . . .	5
2.1.1	Distribution dans le jeu de données . . . . .	5
2.1.2	Champ lexical et corrélations . . . . .	5
2.2	Données textuelles . . . . .	7
2.2.1	Valeurs manquantes . . . . .	7
2.2.2	Analyse textuelle . . . . .	8
2.2.3	Distribution des longueurs . . . . .	10
2.3	Données imagées . . . . .	11
<b>3</b>	<b>Prétraitement des données</b>	<b>12</b>
3.1	Pipeline sur $X$ . . . . .	13
3.1.1	Nettoyage . . . . .	13
3.1.2	Traduction . . . . .	14
3.1.3	Vectorisation et normalisation . . . . .	14
3.1.4	Complétion des valeurs manquantes . . . . .	17
3.1.5	Création de variables . . . . .	17
3.2	Pipeline sur $X^{im}$ . . . . .	19
3.2.1	Redimensionnement . . . . .	19
3.2.2	Transformations numériques . . . . .	19
3.2.3	Transformations géométriques . . . . .	20
3.2.4	Création de variables . . . . .	20
3.3	Pipeline sur $Y$ . . . . .	20
3.3.1	Ré-échantillonnage . . . . .	20
3.3.2	Renommage . . . . .	21
3.4	Résultats du prétraitement . . . . .	21
3.4.1	Exemples issues de $X$ . . . . .	21
3.4.2	Exemples issues de $X^{im}$ . . . . .	22

# 1 Introduction

## 1.1 Contexte

Le projet porte sur la classification multimodale (données textuelles et imagées) à grande échelle de produits provenant du site e-commerce de Rakuten. Lors d'une recherche d'un produit sur le site, il est important pour l'acheteur comme pour le vendeur du produit que ce dernier soit facilement accessible là où on s'attend qu'il soit. Le vendeur doit voir son produit correctement catégorisé au sein du site pour s'assurer de sa visibilité, améliorant alors la probabilité que son produit soit acheté. L'acheteur quant à lui souhaite (*a minima*) que sa recherche l'amène à des produits correspondants à ce qu'il souhaite acheter. Ce défi est essentiel pour les marketplaces, car il permet d'améliorer la recherche, les recommandations et la compréhension des requêtes.

Les approches manuelles ou basées sur des règles ne sont pas adaptées à la diversité et au volume des produits. L'approche par le machine learning et l'intelligence artificielle constitue alors une solution adaptée à ce problème. L'objectif général est de prédire la catégorie d'un produit suivant le catalogue de Rakuten France, à partir des informations renseignées par le vendeur : le nom, la description et l'image du produit. L'enjeu réside notamment dans la gestion de données bruitées, déséquilibrées et issues de sources hétérogènes (vendeurs professionnels et particuliers).

## 1.2 Objectifs

Compte tenu de la diversité et du volume des données qui seront utilisées par les modèles, il est indispensable de réaliser un prétraitement complet et efficace de ces dernières. La qualité de ce prétraitement est naturellement conditionnée par la connaissance et la compréhension approfondie des données.

L'exploration des données constitue ainsi le premier objectif de ce projet. Dans cette phase, nous analysons l'ensemble des données disponibles et identifions les différents traitements à réaliser.

La phase exploratoire aboutit sur le second objectif : le prétraitement. Celle-ci intègre l'ensemble des conclusions faites à l'issue de l'exploration et permet l'application de différentes techniques de feature engineering aux variables explicatives.

A l'issue de ces étapes, les données doivent être utilisables telles quelles par les différents modèles de classification qui seront développés. Il convient donc de concevoir un prétraitement modulable permettant d'intégrer des transformations différentes en fonction du modèle utilisé.

L'objectif est de créer un modèle capable de classer automatiquement les fiches produits provenant de `X_test.csv` dans les bonnes catégories (variable `prdtypecode`). Cela est réalisable grâce au titre, à la description et à l'image des produits (si présente). Les avantages en créant ce type de modèle sont d'améliorer la recherche, les recommandations ou encore la gestion du catalogue.

Voici l'expertise de l'équipe :

- **Rizlène BANAT** : programmation python, mathématiques avancées, modèles d'IA, data science.
- **Bryan KHAN MAHMOOD** : programmation, clean code, librairies de data science, mathématiques avancées (statistiques, probabilités, algèbre linéaire, optimisation), rédaction de rapports, gestion de projet.
- **Romain MAZOYER** : programmation, mathématiques avancées et modèles d'IA, rédaction de rapports, gestion de projet.
- **Steeve TRINCAL** : développement web front-end, programmation, data science.

## 1.3 Jeu de données

Concernant le jeu de données utilisé, il s'agit de fichiers CSV récupérées [ici](#). Il est composé de trois fichiers principaux :

- `X_train.csv`
- `Y_train.csv`

- `X_test.csv`

ainsi que d'un dossier contenant les images des produits listés dans `X_train.csv` et `X_test.csv`. Ces données sont librement accessibles sur le site après la création d'un compte. Il convient de noter qu'il n'y a pas de fichier `Y_test.csv` associé au fichier `X_test.csv` car les données sont issues d'un data challenge dont `X_test.csv` est le fichier d'évaluation des participants. Ce fichier ne sera pas utilisé dans ce qui suit. Les fichiers `X_train.csv` et `Y_train.csv` contiennent donc les données d'entraînement **et** de test, qu'il faudra séparer.

## 2 Exploration des données

Dans cette partie,  $X$  désigne les données contenues dans `X_train.csv`,  $X^{im}$  les images et  $Y$  désigne celles contenues dans `Y_train.csv`. L'exploration a été réalisé en grande partie à l'aide des librairies usuelles : `pandas`, `matplotlib`, `seaborn`, `scikit-learn`, etc. Certaines explorations nécessitent des librairies spécifiques, notamment `nltk` ou encore `wordcloud`.

$X$  contient 84 916 lignes et 5 colonnes :

- `Unnamed`: 0 : colonne sans aucune information, contient les indices des lignes.
- `designation` : colonne représentant les titres / intitulés des produits.
- `description` : colonne représentant les descriptions des produits.
- `productid` : colonne représentant les codes permettant d'identifier les produits.
- `imageid` : colonne représentant les codes permettant d'identifier les images.

On a donc quatre colonnes contenant de l'information utile. Les colonnes `productid` et `imageid` ont des valeurs uniques et permettront par la suite de lier les données textuelles de  $X$  aux images du jeu de données. Cela permet aussi de s'assurer que chaque produit référencé est unique dans le jeu de données.

$X^{im}$  contient aussi 84 916 éléments au format `jpg`. Chaque image, dont on peut récupérer les champs `productid` et `imageid` associés, est unique et est reliée par l'intermédiaire de ses identifiants à un unique produit. Les images sont caractérisés par les éléments suivants :

- Dimension :  $500 \times 500$ .
- Canaux : trois canaux RGB.

Une image est donc représentable par un tenseur d'ordre 3 à 750 000 éléments.

Un résumé des différentes variables du jeu de données ainsi que leur propriétés sont présents dans le tableau suivant :

Nom de la colonne	Description	Disponibilité de la variable <i>a priori</i>	Type informatique	Taux de NA	Gestion des NA	Distribution des valeurs	Remarques sur la colonne
	Que représente cette variable en quelques mots ?	Pouvez vous connaître ce champ en amont d'une prédiction ? Aurez vous accès à cette variable en environnement de production ?	<code>int64</code> , <code>float</code> etc... Si "object", détaillez.	en %	Quelle mode de (non) - gestion des NA favorisez vous ?	Pour les variables catégorielles comportant moins de 10 catégories, énumérez toutes les catégories. Pour les variables quantitatives, détaillez la distribution (statistiques descriptives de base)	champs libre à renseigner
designation	Le titre du produit, accompagné d'un court texte resumant le produit.	oui	▼ string	0,00% -	Valeur unique	▼ ???	
description	Texte plus précis concernant le produit.	oui	▼ string	35,09%	Remplacement aléatoire, remplacement par les designation	Valeur unique	▼ Fort taux de NA hétérogène suivant les classes
productid	Identifiant unique du produit.	oui	▼ int64	0,00% -	Valeur unique	▼ -	
imageid	Identifiant unique de l'image associée au produit.	oui	▼ int64	0,00% -	Valeur unique	▼ -	
prdtypecode	Variable à prédire, représente les différentes classes existantes	non	▼ int64	0,00% -	Catégorielle - sup. à 10 catégories	▼	Catégories non explicites, demande un renommage

FIGURE 1 – Résumé des variables explicatives au sein du jeu de données textuelles.

Certains points sont plus amplement détaillés dans ce qui suit.

## 2.1 Catégories

$Y$  contient autant de lignes que  $X$  et 2 colonnes : Unnamed: 0 et prdtypecode. La première colonne contenant les indices des lignes est inutile, la seconde représente les différents codes des catégories de produits du site Rakuten. Il s'agit des valeurs que les modèles devront prédire *in fine* à l'aide des variables explicatives `designation`, `description` et des images associées.

On dénombre 27 catégories différentes au sein de  $Y$ , ce qui constitue un nombre modéré de catégories.

### 2.1.1 Distribution dans le jeu de données

Dans un problème de classification avec de nombreuses catégories comme c'est le cas ici, il est important de s'assurer de la bonne répartition des catégories au sein du jeu de données. Une étude de cette répartition montre que certaines catégories sont sous-représentées et qu'une catégorie est nettement sur-représentée par rapport aux autres.

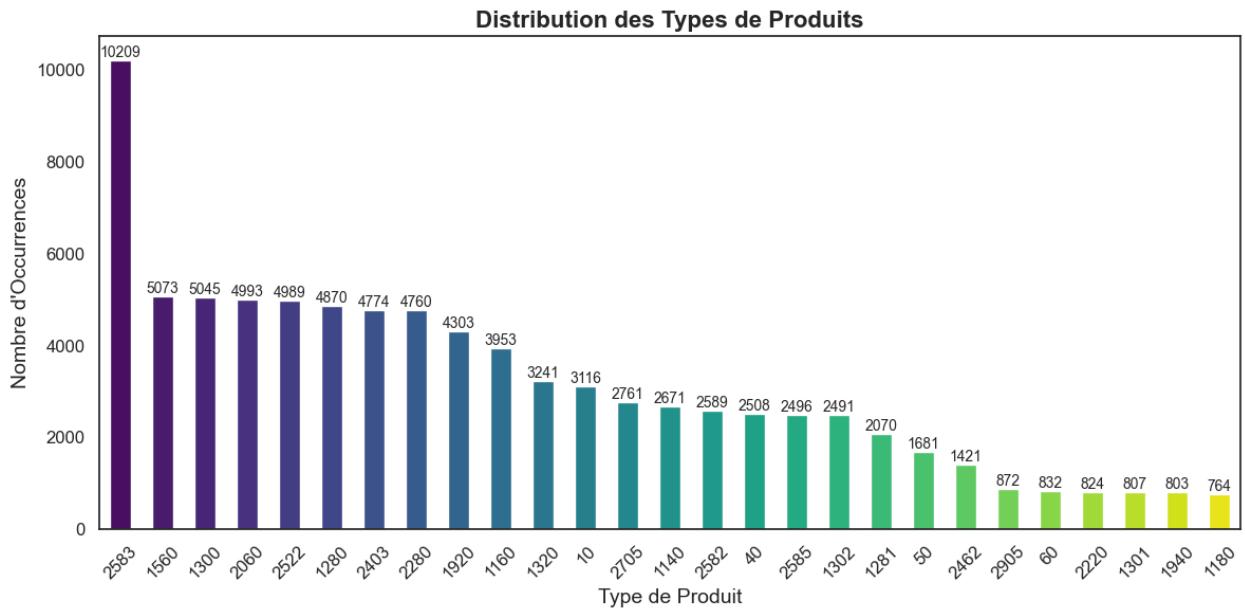


FIGURE 2 – Distribution des catégories au sein du jeu de données textuelles.

La catégorie de code 2583 est sur-représentée avec 10 209 produits, soit 12,02% du jeu de données entier. C'est plus que les deux catégories suivantes les plus représentées cumulées, et plus que les neufs catégories les moins représentées. Il est essentiel d'appliquer un ré-échantillonnage des données pour se prémunir des biais relatifs au déséquilibre inter-classe. Comme la catégorie 2583 semble être la seule sur-représentée, on pourra utiliser des techniques de sous-échantillonnage pour aligner le nombre d'échantillons avec les catégories correctement représentées plutôt que de sur-échantillonner toutes les autres catégories.

Un certain nombre de catégories sont sous-représentées, notamment les catégories 2905, 60, 2220, 1301, 1940, et 1180. Ces catégories devront être sur-échantillonnées pour éviter d'altérer le pouvoir prédictif des modèles sur ces dernières.

### 2.1.2 Champ lexical et corrélations

Les valeurs des classes sont des codes numériques non ordonnés qui n'apportent aucune information sur la nature du produit. Il sera donc pertinent de transformer les valeurs des classes pour faciliter l'analyse des prédictions faites par les modèles. Le graphique ci-dessous représente des nuages de mots construits pour quelques classes en concaténant les colonnes `designation` et `description`. Cette représentation permet d'associer à chaque classe une catégorie de produit par le biais du champ lexical mis en évidence par les nuages de mots (par exemple, le code 2583 fait référence aux produits étroitement liés au mot « piscine », et pourrait être substitué par une expression comme « produit aquatique »).

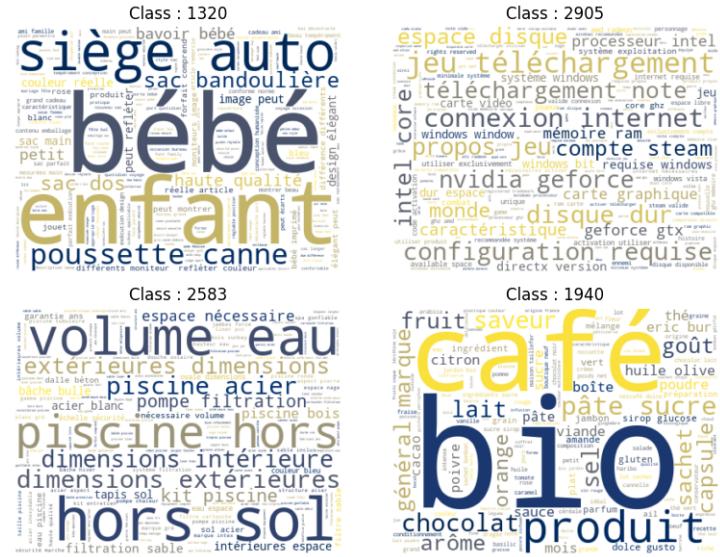


FIGURE 3 – Nuages de mots pour quatre catégories de produits. On identifie clairement un univers dans lequel les produits se rattachent.

Certaines catégories font ressortir des champs lexicaux rapprochés, pouvant indiquer que la séparation de certaines catégories pourrait être particulièrement subtile. Un moyen de s'assurer de la (dé-)corrélation des catégories entre elles est de construire une matrice de corrélation lexicale entre les catégories, où la corrélation est calculée à partir des données construites pour les nuages de mots :

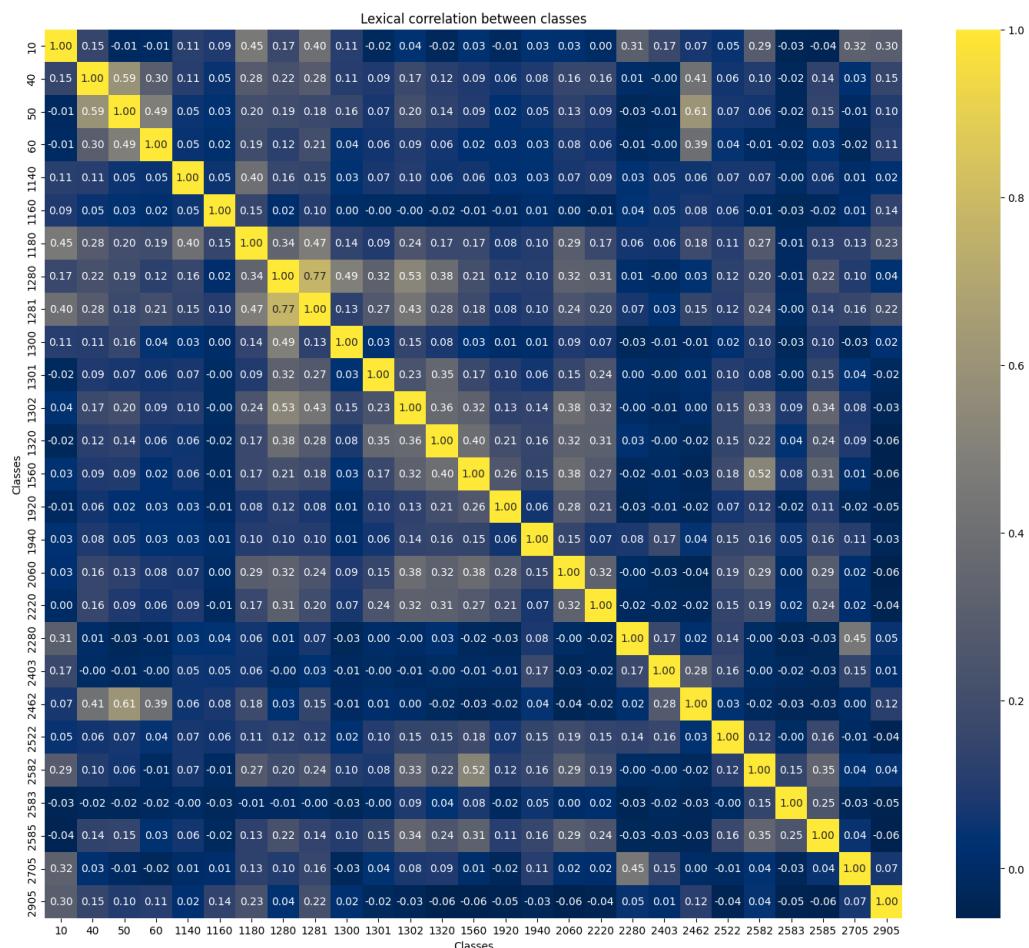


FIGURE 4 – Matrice de corrélation lexicale entre les classes. On identifie aisément les catégories aux champs lexicaux rapprochés et la catégorie 1280 qui semble avoir un champ lexical particulièrement étendu.

La matrice de corrélation permet de confirmer que la grande majorité des catégories sont bien décorréées. Cependant, certaines catégories ont une corrélation sensiblement élevée, notamment les paires de catégories (1280, 1281), (50, 2462), (1280, 1302), et (1560, 2582). Globalement, la catégorie 1280 est nettement plus corrélée aux autres catégories. D'une part, ces résultats sont en accord avec la variabilité des champs lexicaux mis en évidence dans les nuages de mots. D'autre part, le rapprochement lexical entre certaines catégories nécessite d'accorder une importance particulière à la manière dont les données textuelles seront nettoyées et encodées. En effet, si la vectorisation du texte ne permet pas de capturer les différences lexicales entre deux catégories rapprochées, la distinction entre ces dernières sera faible et les modèles auront plus de difficultés à les distinguer.

## 2.2 Données textuelles

L'essentiel de l'exploration des données textuelles au sein de  $X$  sera concentrée sur les colonnes `designation` et `description` qui sont les variables explicatives de  $X$ .

### 2.2.1 Valeurs manquantes

On relève 35,09% de valeurs manquantes au sein de  $X$ , toutes présentes dans la colonne `description`. La qualité de cette colonne est donc pour le moment limitée, et c'est un point particulièrement important qui sera intégré dans le prétraitement de cette variable. En effet, cette grande proportion de valeurs manquantes exclut l'idée de les supprimer ; la perte des données associées serait beaucoup trop importante.

Une analyse plus fine montre une forte disparité de la proportion de valeurs manquantes entre les catégories, comme l'illustre l'histogramme suivant :

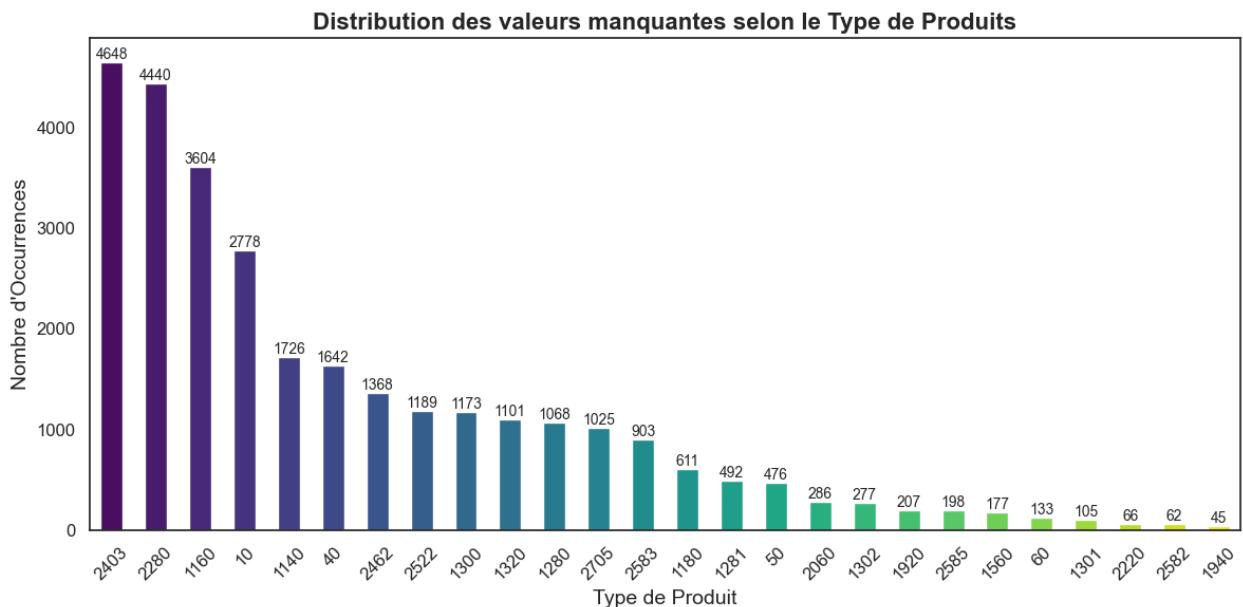


FIGURE 5 – Histogramme du nombre de valeurs manquantes pour chaque classe, dans l'ordre décroissant. On note que certaines classes ont un très grand nombre de valeurs manquantes, ce qui impose un traitement poussé.

On note tout d'abord qu'il y a huit catégories dont la proportion de descriptions manquantes est particulièrement élevée : les catégories 2403, 2462, 2280, 1160, 10, 1180, 40 et 1140. Le taux de valeurs manquantes pour ces catégories est indiqué dans le tableau ci-dessous :

Type de Produit	Analyse des Produits par Type		Nombre Total de Produits (non null)	Nombre de Produits
	Pourcentage Manquant (%)	Nombre		
2403	97.36%	126	4774	
2462	96.27%	53	1421	
2280	93.28%	320	4760	
1160	91.17%	349	3953	
10	89.15%	338	3116	
1180	79.97%	153	784	
40	65.47%	866	2508	
1140	64.62%	945	2671	
2705	37.12%	1736	2761	
1320	33.97%	2140	3241	
50	28.32%	1205	1681	
2522	23.83%	3800	4989	
1281	23.77%	1578	2070	
1300	23.25%	3872	5045	
1280	21.93%	3802	4870	
60	15.99%	699	832	
1301	13.01%	702	807	
1302	11.12%	2214	2491	
2583	8.85%	9306	10209	
2220	8.01%	758	824	
2585	7.93%	2298	2496	
2060	5.73%	4707	4993	
1940	5.60%	758	803	
1920	4.81%	4096	4303	
1560	3.49%	4896	5073	
2582	2.39%	2527	2589	
2905	0.00%	872	872	

FIGURE 6 – Taux de valeurs manquantes pour chaque classe. La volumétrie des descriptions non vides et le total pour chaque classe sont données dans les deux dernières colonnes.

Étant donné que ces catégories ont une proportion de valeurs manquantes très élevée, il est nécessaire de compléter la variable **description** avec des techniques robustes pour ne pas perdre d'information et voir certaines catégories mal prédites par les modèles car désavantagées par la qualité des données disponibles.

Sur l'ensemble des catégories, quatre catégories peuvent être considérées comme propres vis-à-vis des valeurs manquantes : les catégories 1920, 1560, 2582, et 2905. A noter que la catégorie 2905 n'a aucune valeur manquante ce qui est particulièrement rare sur des données réelles.

Plusieurs éléments permettent d'expliquer les valeurs manquantes de la variable **description** :

- La description sur le site de Rakuten n'est pas un champ obligatoire lors de l'ajout d'un produit. Les individus sont donc libres de renseigner ou non cette information.
- Les individus peuvent renseigner une description dans le champ dédié à l'intitulé du produit (représenté par la variable **designation**). On a alors une situation où l'intitulé *est* la description.

L'étude des longueurs des variables **designation** et **description** nous permet (entre autres) de mieux saisir l'origine de ces valeurs manquantes. Il est cependant nécessaire d'analyser en amont le contenu de ces variables, et notamment la composition des chaînes de caractères en leur sein.

### 2.2.2 Analyse textuelle

Les données textuelles réelles peuvent être fortement polluées par différents caractères ou symboles. Ces derniers détériorent la performance des modèles. Il est donc essentiel d'identifier les différents éléments indésirables présents dans nos variables textuelles. Les analyses des caractères présents dans les variables **designation** et **description** suggèrent qu'une part importante des descriptions contient du code HTML. Cette constatation peut être issue, entre autres, d'une technique défaillante de récupération des données. Il convient néanmoins de traiter ces cas et de nettoyer les descriptions de ce contenu, notamment à cause de l'impact de ce dernier sur les longueurs des chaînes de caractères. La variable **designation** est relativement propre vis-à-vis de ce type de contenu.

Des caractères spéciaux non standards sont présents dans ces variables et doivent être retirés pour assurer le maximum de compatibilité avec les modèles. Certains caractères ne contiennent ou ne contribuent à aucune information intéressante pour le travail de classification (symboles de monnaies étrangères, symboles mathématiques, ponctuations non standards). Il y 26 caractères distincts non

alpha-numériques dans la variable `designation` et 47 dans la variable `description`, qui devront être pris en compte dans le prétraitement des variables.

Outre les caractères indésirables, certaines portions de texte peuvent aussi causer des problèmes ou altérer la qualité des données. Ils seront retirés par listing explicite ou par des expressions régulières (par exemple des adresses mail, des liens URL, etc).

Il convient de noter que le nettoyage des variables textuelles conditionne les techniques de vectorisation qui seront utilisées pour encoder les variables. En effet, les techniques basées sur les mots et leur tokenisation (par exemple `TfidfVectorizer` de la librairie `sklearn`) demandent de réaliser un nettoyage particulièrement poussé (y compris la suppression de mots *vides*) pour assurer un encodage de qualité. A contrario, une vectorisation basée sur les transformers demande de conserver beaucoup plus d'éléments, y compris la ponctuation, les verbes et autres mots considérés comme *vides*, car ces derniers sont utilisés comme support pour extraire le sens et le contexte. Le nettoyage est alors allégé, mais l'encodage, plus lourd, nécessite plus de ressources et construit des vecteurs dont la dimension est particulièrement grande et est définie par le modèle de transformer choisi. On a donc moins de contrôle sur la manière dont la vectorisation est construite.

Une autre problématique à laquelle nous sommes confrontés est la langue dans laquelle la variable est renseignée. Le site e-commerce de Rakuten a une portée internationale. Les produits que l'on trouve peuvent tout à fait avoir été conçus pour une utilisation dans un pays autre que la France. En outre, la nature des produits peuvent induire plusieurs langues différentes au sein des variables textuelles. En effet, l'une des catégories *a priori* correspondante à l'univers du livre, est prédisposée à contenir dans la description des produits plusieurs langues différentes (auteur étranger, titre original, etc). On doit donc étudier les différentes langues présentes dans les variables textuelles.

La librairie `langdetect` permet d'extraire la proportion de chaque langue dans les descriptions :

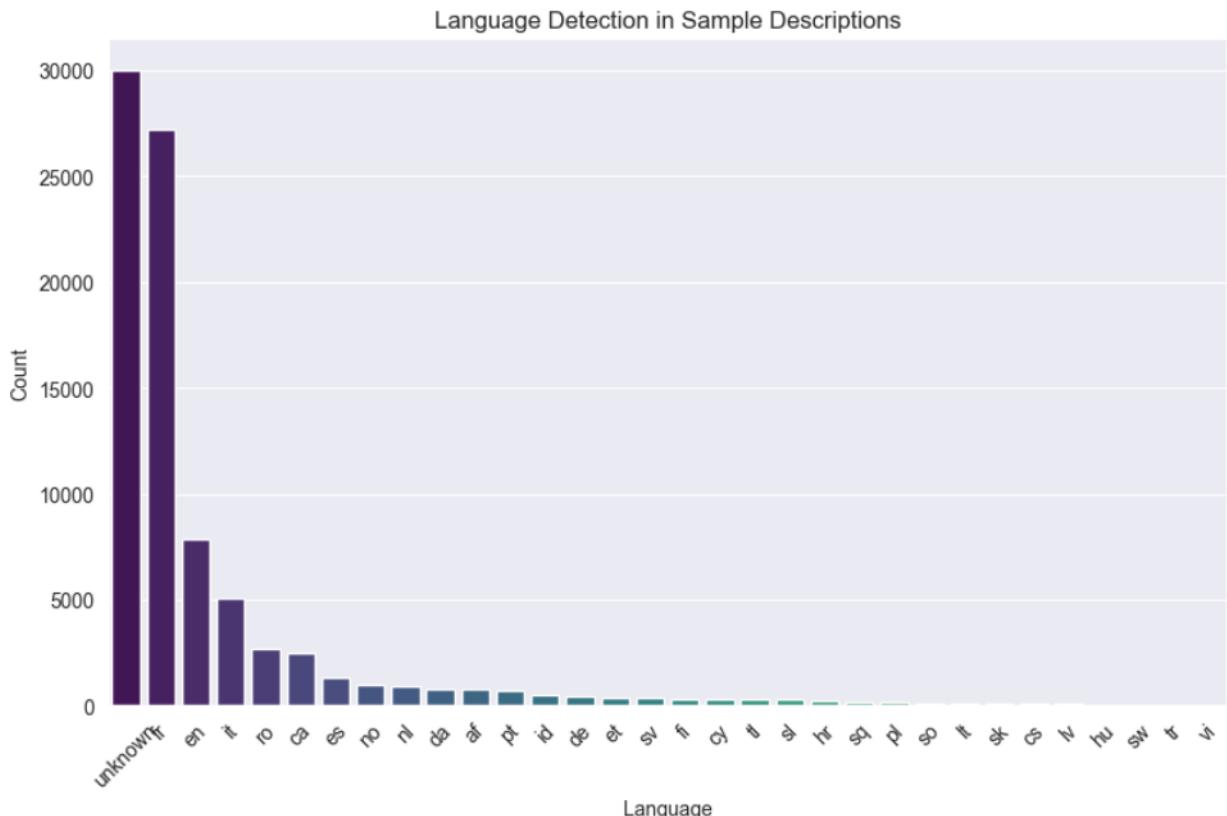


FIGURE 7 – Distributions des langues utilisées au sein des descriptions.

On note qu'une grande majorité des descriptions sont en français. Néanmoins, la proportion de langues étrangères n'est pas assez faible pour l'ignorer. C'est une information qui doit être prise en compte dans le prétraitement des variables textuelles. On peut noter l'avantage que possède un encodage par

des modèles de transformers multilingues, qui permet de conserver la qualité de l'encodage malgré la variété des langues présentes.

### 2.2.3 Distribution des longueurs

Une caractéristique intéressante à étudier sur les variables `designation` et `description` est la longueur des textes qu'elles contiennent. Tout d'abord, cela permet de vérifier que les textes ont des longueurs cohérentes avec ce qu'ils représentent. Pour un intitulé de produit, plusieurs dizaines à une centaine de caractères est tout à fait normal et pour une description, plusieurs centaines voire milliers de caractères est raisonnable. De plus, certaines techniques de vectorisation ont une taille de texte maximale supportée (par le biais d'une limite du nombre de tokens). On doit donc déterminer si les variables textuelles respectent ou pas ces limites et éventuellement choisir une stratégie pour résoudre ce problème.

Les graphiques ci-après illustrent la distribution des longueurs des textes au sein de la variable `designation`, en présence ou non de descriptions :

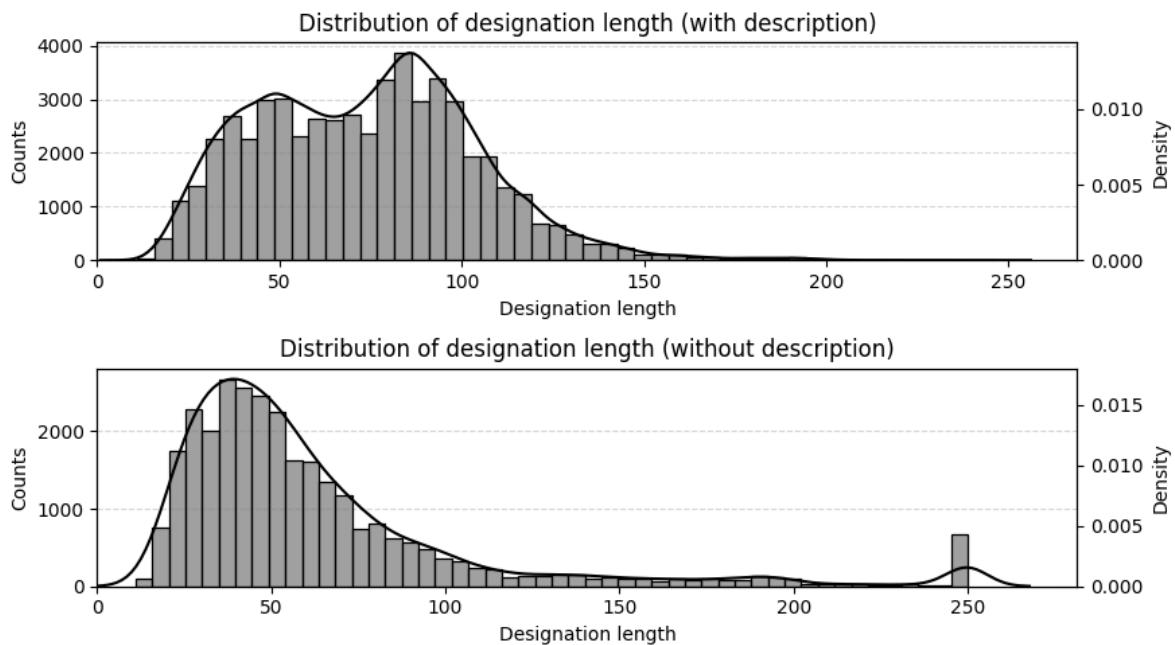


FIGURE 8 – Distribution des longueurs des intitulés en présence ou non de description. On identifie clairement un pic caractéristique en l'absence de descriptions.

A la vue de ces informations, on peut constater les choses suivantes :

- La longueur des intitulés des produits est en moyenne plus élevée et est plus homogène lorsqu'il y a une description ( $\mu = 73,67$ ;  $\sigma = 29,81$ ). Cela peut s'expliquer par le fait qu'un individu proposant une description à son produit accorde plus d'importance aux informations transmises aux acheteurs (effort de la part du vendeur, intégration des éléments non obligatoires, etc). En conséquence, les intitulés sont plus précis, donc plus détaillés, donc plus longs. Ils gardent cependant une taille raisonnable (conscience que l'intitulé doit être concis) étant donné qu'une description est présente pour détailler ces produits.
- A contrario, pour les produits sans description, la distribution est plus hétérogène et les longueurs des intitulés sont globalement plus faibles ( $\mu = 63,67$ ;  $\sigma = 46,36$ ). On constate un important pic en bout de queue de la distribution pour les longueurs entre 240 et 250 caractères. On peut d'abord expliquer les longueurs globalement plus faibles par le fait qu'un individu qui ne proposera pas de description accorde moins d'importance aux informations précises et détaillées contenues dans l'offre du produit ; la personne se limitera donc au strict minimum sur l'intitulé du produit (on ne peut en revanche pas conclure sur la qualité de l'intitulé : un intitulé court, simple, concis peut être tout à fait préférable à un intitulé plus détaillé et plus long). Concernant le pic sur la queue de la distribution, on peut raisonnablement faire l'hypothèse que certains vendeurs

intègrent la description du produit au sein de l'intitulé. On a alors un produit où l'intitulé *est* la description ; cette situation peut être (ou pas) volontaire de la part du vendeur. On note aussi que la queue de la distribution est bien plus importante que celle où une description est présente. Cette observation corrobore l'idée que des vendeurs intègrent la description au sein de l'intitulé du produit.

- Comme on n'observe aucune valeur supérieure à 250 caractères, il y a un nombre maximum de caractères possibles pour l'intitulé d'un produit. L'hypothèse sur le pic observé est confirmée ici dans le sens où les vendeurs se voient bloqués par la taille limite lorsqu'ils décrivent leurs produits (qui en principe peut facilement dépasser 250 caractères).

Outre les traitements textuels usuels (caractères spéciaux, etc), lorsqu'une donnée possède un intitulé long et ne possède pas de description, l'idée est de doper la description avec cet intitulé. Il faut alors déterminer dans quel cas un intitulé peut être considéré comme une description (critère sur la longueur, etc). En se basant sur les distributions ci-dessus, un seuil de 150 caractères peut être un candidat pertinent pour considérer qu'un intitulé est suffisamment long pour décrire le produit.

L'histogramme ci-dessous complète l'analyse précédente en relevant pour chaque catégorie le taux de valeurs manquantes et la longueur moyenne des textes de la variable `designation` :

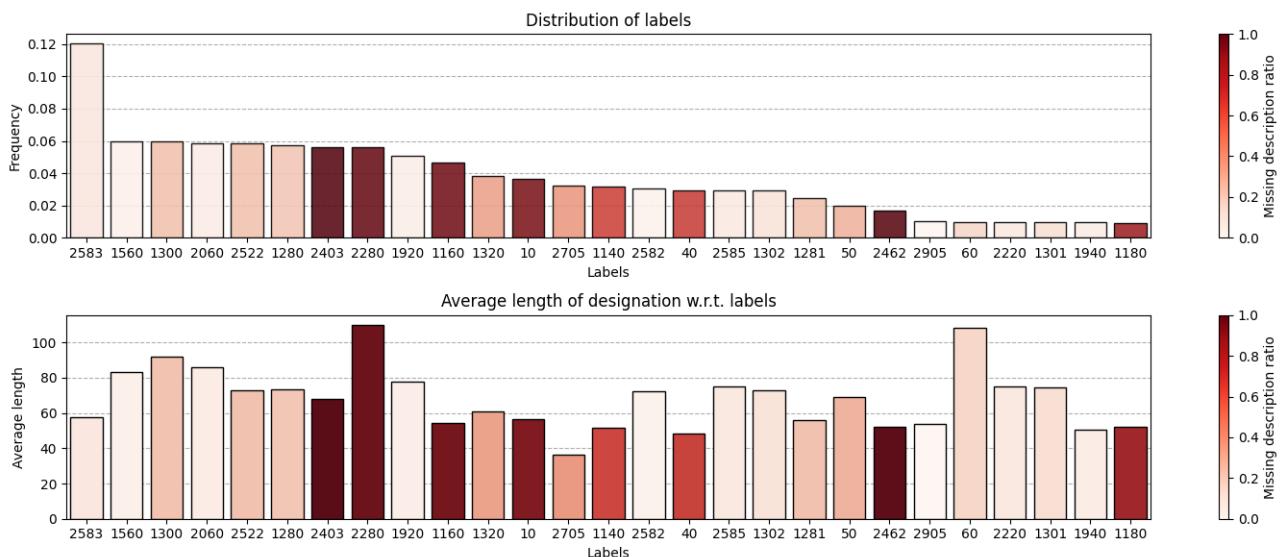


FIGURE 9 – En haut : distribution des catégories avec le taux de valeurs manquantes associé. En bas : longueur moyenne de l'intitulé pour chaque catégorie, avec le taux de valeurs manquantes associé.

On peut remarquer que les catégories dont le taux de valeurs manquantes est très élevé n'ont pas toujours les intitulés les plus longs. Cela ne rentre pas en désaccord avec les distributions analysées précédemment, mais met en avant le fait que l'intégration de la description au sein de l'intitulé par les vendeurs n'est pas la seule explication aux valeurs manquantes (voir 2.2.1), et qu'une description intégrée en tant qu'intitulé, même complète, peut tout à fait contenir une centaine de caractères et ce faisant ne pas augmenter significativement la longueur moyenne des intitulés associée à la catégorie.

## 2.3 Données imagées

Comme vu précédemment, la dimension de chaque image est (500, 500, 3). En considérant une représentation par des `float32` (en brut, les nombres sont des entiers entre 0 et 255, mais des normalisations pourront transformer ces entiers durant la phase de prétraitement), chaque image pèsera 3 Mo en mémoire, soit un total d'environ 255 Go pour l'ensemble des images. Sans réduction de l'image durant le prétraitement, il sera nécessaire de considérer une gestion par lots lors de l'injection des images dans les modèles pour éviter une saturation de la mémoire vive.

On note qu'aucune image n'est corrompue.

De part la nature du problème et du grand volume de données présent, il est nécessaire de s'assurer

que les modèles ne soient pas sujets à l'overfitting. Plusieurs transformations pourront être appliquées aux images :

- Redimensionnement pour alléger la taille des données fournies au modèle, éviter la surcharge mémoire, accélérer les traitements, conserver l'information utile de l'image et *de facto* réduire la complexité de la classification.
- Transformations géométriques pour enrichir les images. Il en résulte une plus forte variabilité au sein et entre les classes, limitant l'overfitting (on pense notamment à des rotations, symétries, zooms, etc).
- Transformations numériques pour enrichir les images. Cela passe par l'altération des valeurs numériques des pixels des images en utilisant des filtres (accentuation des contours, modification des contrastes, renversement des couleurs, etc) ou encore en perturbant la qualité en bruitant les images de manière contrôlée.

### 3 Prétraitement des données

Le prétraitement des données passe par la manipulation de données de natures sensiblement différentes. Il en résulte que le prétraitement doit être séparé en trois parties : un prétraitement dédié à  $X$ , un dédié à  $X^{im}$  et un dédié à  $Y$ . Notons que le prétraitement de  $Y$  n'est pas indépendant des autres étant donné que  $Y$  dépend de  $X$  et  $X^{im}$ . On doit donc s'assurer que les prétraitements soient réalisés dans le bon ordre.

Le nombre important de prétraitements identifiés dans l'exploration des données allié à la variabilité des modèles qui seront testés nous amène à concevoir un système structuré et surtout modulable (ajout, modification, suppression de prétraitement, etc) pour ne pas devoir réimplémenter l'ensemble de cette phase lors d'un changement de modèle ou de choix de prétraitement.

Les outils de prétraitement fournis par la librairie **scikit-learn** permettent de concevoir aisément des pipelines via la classe **Pipeline**. La seule restriction est de respecter le modèle de l'API **scikit-learn**. Les transformations que l'on souhaite appliquer doivent être implémentées en tant que classes héritant de celles de **scikit-learn**. Ces classes héritées devront implémenter deux méthodes : **fit** et **transform**. On trouvera ci-dessous un schéma de la structure du pipeline de prétraitement, dont le contenu des transformations pourra évoluer avec les modèles retenus :

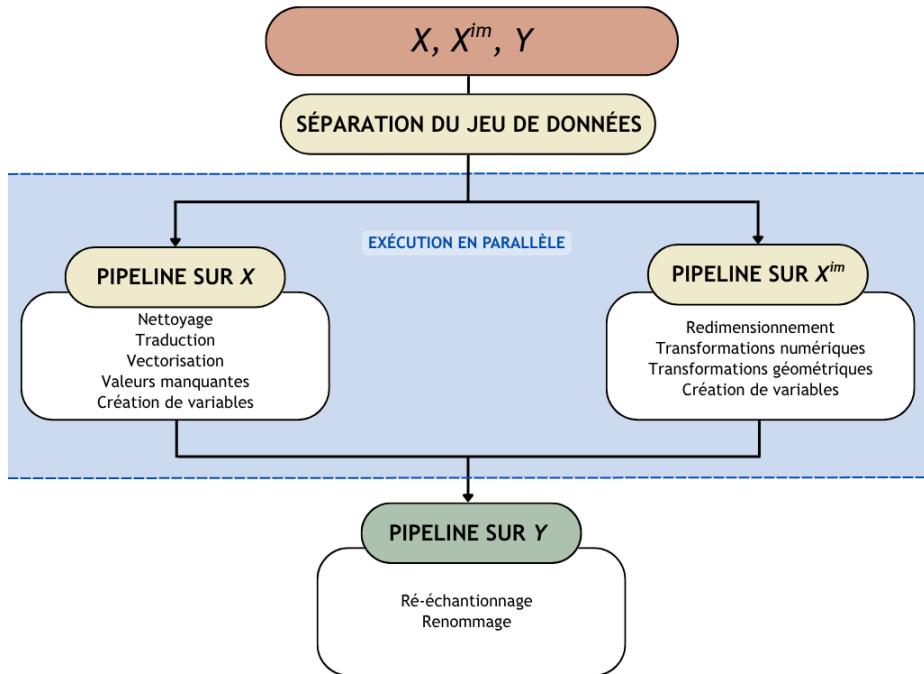


FIGURE 10 – Schéma de la structure du pipeline.

La première étape est comme le schéma l'indique de découper le jeu de données en jeu d'entraînement et de test.

Dans toute la suite, on désigne par  $X_{train}$ ,  $X_{train}^{im}$ ,  $Y_{train}$  les données d'entraînement et par  $X_{test}$ ,  $X_{test}^{im}$ ,  $Y_{test}$  les données de test.

Peu importe le modèle utilisé, les points essentiels relevés dans la phase d'exploration seront toujours intégrés dans les pipelines correspondants, car ce sont des points qui contribuent à la mauvaise qualité des données. Chaque transformation dont le fonctionnement repose sur des processus stochastiques sera contrôlée par une graine (typiquement via un paramètre `random_state`) pour assurer la reproductibilité des résultats.

### 3.1 Pipeline sur $X$

#### 3.1.1 Nettoyage

Le nettoyage des données désigne l'ensemble des opérations visant à détecter, corriger ou éliminer les données incorrectes, incohérentes, incomplètes ou dupliquées au sein d'un jeu de données. Cette étape constitue une phase critique du pipeline de traitement des données, en particulier dans les contextes de modélisation statistique, analyse exploratoire ou apprentissage automatique (Machine Learning). En effet, les modèles supervisés et non-supervisés sont extrêmement sensibles à la qualité des données d'entrée. Entraîner un modèle sur un corpus bruité ou corrompu introduit des biais ou des sur-apprentissages non pertinents (overfitting sur du bruit), ce qui compromet et réduit la performance sur données réelles. Par conséquent, le nettoyage des données devient un préalable indispensable à la phase de modélisation.

Dans le cas des données  $X_{train}$ , la description est notre variable qui sera sujette à ce nettoyage. En effet, le nettoyage est nécessaire pour diverses raisons :

- Correction des erreurs de saisie ou de format : dans notre jeu de données, beaucoup de descriptions étaient en format HTML, c'est pourquoi il était important de retirer les balises afin d'obtenir simplement le texte.
- Gestion des valeurs manquantes : en effet, comme vu précédemment, 35,09% des descriptions sont vides, c'est pourquoi nous détaillerons plus tard les différentes méthodes mises en place pour ces valeurs.
- Standardisation : uniformisation des formats, unités, encodages ou représentations textuelles. Dans notre cas, cela concerne la suppression des accents, des majuscules, des chiffres et symboles quelconques et des stopwords (« le », « la », « les », etc).

L'étape de standardisation est une étape sujette à adaptation selon le modèle de vectorisation utilisé (voir [2.2.2](#)).

Pour garantir la reproductibilité et l'intégration fluide dans un pipeline scikit-learn, un premier transformateur personnalisé a été défini : la classe `NettoyeurTexte`, héritée de `BaseEstimator` et `TransformerMixin`. Cela permet de chaîner le nettoyage textuel directement dans un pipeline de prétraitement ou de modélisation. Les avantages de l'utilisation d'un transformeur sont :

- Ecrire un pipeline clair et modulaire.
- D'un point de vue maintenance, chaque étape est séparée et indépendante. Le code est plus lisible, plus testable unitairement et plus facile à étendre.
- Une fois défini, le transformateur peut être réutilisé dans d'autres projets ou pipelines, tout en restant indépendant du dataset d'origine.

Tout cela est trouvable dans le fichier `Test.ipynb` sur le Github.

Concernant le fonctionnement de ce transformeur :

- Dans un premier temps, il traite les valeurs manquantes dans la colonne `description`. Dans le cas où elle est manquante, il va copier coller la `designation`.

- Ensuite, il applique un prétraitement linguistique avec un passage en minuscules, la suppression des accents afin de standardiser les caractères (ex. é → e), le nettoyage des balises HTML, l'élimination des caractères non alphabétiques (chiffres, ponctuation, symboles) et enfin la réduction des espaces multiples.
- Enfin, on utilise la lemmatisation avec la librairie `Spacy` avec le modèle `fr_core_news_sm`, léger mais optimisé pour la langue française. On finit par filtrer les tokens, en excluant les stopwords, les tokens de moins de 3 caractères et la ponctuation et les espaces.

La méthode `transform()` retourne une liste de chaînes nettoyées et lemmatisées, prêtes pour la vectorisation ou l'entraînement d'un modèle. Le nettoyage réalisé par `NettoyeurTexte` est adapté à l'utilisation de modèle de vectorisation tel que `TfidfVectorizer` de `scikit-learn`.

Un seconde classe dédiée au nettoyage a été implémentée, nommée `CharacterCleaner`, dans l'optique d'appliquer un encodage par des modèles de transformers (au sens des modèles de NLP) tels que ceux trouvés dans la librairie `sentence-transformers`.

La différence majeure entre `CharacterCleaner` et `NettoyeurTexte` réside dans la suppression ou la conservation de certains mots *vides*. On pense notamment aux mots de liaisons, déterminants, pronoms, et verbes qui servent de support aux transformers pour l'encodage d'une phrase.

La classe `CharacterCleaner` conserve uniquement les lettres, chiffres, la ponctuation standard et certains caractères spéciaux. Ces caractères peuvent être trouvés dans le fichier `textual_pipeline_components.py` disponible sur le dépôt GitHub dans les variables `ALPHANUM_CHARACTERS`, `PUNCTUATION_CHARACTERS` et `ACCENTS_CHARACTERS`.

La classe inclut le nettoyage du contenu HTML via les outils fournis par la librairie `BeautifulSoup4` ainsi que la suppression de certaines portions de textes indiquant des URLs ou des adresses mail via la librairie native `re`.

### 3.1.2 Traduction

Comme cela a été observé dans l'exploration des données, une multitude de langues ont été relevées. Il est donc important de traduire ces données dans une seule langue (ici le français, qui domine les langues présentes) pour les modèles mono-langage. Pour réaliser ces traductions, la librairie `transformers` a été utilisé, qui dispose de plusieurs modèles de traduction pré-entraînés permettant de traduire du texte en français.

Pour le modèle TF-IDF, la traduction automatique vers le français a été intégrée dans le pipeline de prétraitement. La langue de chaque description est automatiquement détectée à l'aide du module `langdetect`. Ce module repose sur un classifieur bayésien entraîné sur un large corpus multilingue. Il permet de distinguer les langues les plus courantes avec un niveau de précision acceptable, tout en restant léger et rapide pour un usage à grande échelle.

Les langues les plus représentés que nous avons décidé de traiter sont les suivants : anglais (en), italien (it), espagnol (es), néerlandais (nl) et le roumain (ro).

Pour les textes détectés dans ces langues, une traduction vers le français est effectuée à l'aide de modèles de la famille `Helsinki-NLP/OPUS-MT`, fournis via la bibliothèque `transformers`. Ces modèles sont des systèmes de traduction neuronale pré-entraînés sur des corpus multilingues à l'échelle du web. La traduction est intégrée dans la classe `NettoyeurTexte`, à l'étape de normalisation du texte (après mise en minuscule et suppression des accents, mais avant nettoyage et lemmatisation). Ce choix garantit que le contenu linguistique est correctement aligné avant les étapes de vectorisation.

### 3.1.3 Vectorisation et normalisation

Comme nous le savons, les modèles et algorithmes d'apprentissage automatique comprennent des données numériques. Or, dans notre cas, nous avons besoin d'une méthode permettant de passer de données textuelles (à savoir nos descriptions) à des données numériques. La vectorisation se présente

comme solution car c'est un processus qui consiste à convertir des données textuelles ou catégorielles en vecteurs numériques. En convertissant les données en données numériques, il est possible d'entraîner des modèle avec plus de précision. De plus, la vectorisation possède de nombreux avantages :

- Elle n'est pas seulement utile pour la conversion en format numérique, mais aussi pour la capture du sens sémantique. Hors dans nos données, le sens de la phrase contenue dans la description a une grande importance afin de pouvoir prédire son prdtypecode.
- La vectorisation peut réduire la dimensionnalité des données et les rendre plus efficaces. Cela peut s'avérer très utile lorsque vous travaillez sur un grand ensemble de données.
- De nombreux algorithmes d'apprentissage automatique nécessitent une entrée numérique, tels que les réseaux neuronaux, c'est pourquoi la vectorisation est favorable.

Deux approches ont été sélectionnés pour ce travail de vectorisation : TF-IDF de la librairie `scikit-learn` et le modèle `paraphrase-multilingual-MiniLM-L12-v2` fourni par la librairie `sentence-transformers`.

La méthode TF-IDF (Term Frequency – Inverse Document Frequency), qui permet de mesurer le poids d'un terme dans un document ou un corpus. Son but est de pondérer les termes (ou n-grammes) selon leur importance dans le document courant, tout en pénalisant ceux trop fréquents dans l'ensemble du corpus. Ce schéma est formellement défini comme :

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \log \left( \frac{N}{1 + \text{DF}(t)} \right),$$

où

- $\text{TF}(t, d)$  est la fréquence du terme  $t$  dans le document  $d$ .
- $\text{DF}(t)$  est le nombre de documents contenant le terme  $t$ .
- $N$  est le nombre total de documents dans le corpus.

Un pipeline scikit-learn est défini comme suit :

```
pipeline = Pipeline([
    ('nettoyage', NettoyeurTexte()),
    ('tfidf', TfidfVectorizer(
        max_features=1000,
        ngram_range=(1, 1),
        stop_words='english'
    ))
])
```

FIGURE 11 – Pipeline scikit-learn

On remarque ici :

- Implémentation de notre classe `NettoyeurTexte` qui applique un nettoyage linguistique avancé.
- Mise en place de la vectorisation avec `TfidfVectorizer` qui transforme la sortie textuelle nettoyée en une matrice sparse :

$$n_{\text{documents}} \times n_{\text{features}}, \quad \text{ici de dimension } N \times 1000.$$

`max_features = 1000` : conserve uniquement les 1000 tokens (colonnes) les plus fréquents et informatifs selon la pondération TF-IDF.

`ngram_range = (1, 1)` : utilise uniquement les **unigrammes**, c'est-à-dire des mots seuls sans combinaison de termes.

`stop_words = 'english'` : sans effet ici, car les stopwords français ont déjà été supprimés lors de la phase de nettoyage. Ce paramètre peut être désactivé ou personnalisé si besoin.

A l'issue de l'utilisation de notre pipeline, une variable contient la matrice TF-IDF sous forme sparse, économisant la mémoire. Chaque ligne représente un produit, chaque colonne un terme. Les poids TF-IDF reflètent l'importance relative de chaque mot dans chaque document.

Exemple fictif de matrice TF-IDF :							
	adapt	adhesif	ARRIERE	art	aussi	avoir	bleu
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.415983	0.000000
1	0.000000	0.000000	0.000000	0.820413	0.000000	0.095657	0.000000
2	0.169357	0.179139	0.146583	0.000000	0.129054	0.154866	0.118175
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.290655	0.000000
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.106308	0.000000
	caracteristique	confort	couleur	...	style	support	tactile
0	0.00000	0.000000	0.00000	...	0.000000	0.000000	0.000000
1	0.00000	0.000000	0.00000	...	0.000000	0.000000	0.000000
2	0.09351	0.142083	0.07601	...	0.238061	0.281544	0.160019
3	0.00000	0.000000	0.00000	...	0.000000	0.000000	0.000000
4	0.00000	0.000000	0.00000	...	0.000000	0.000000	0.000000
	tenir	tout	tre	vente	votre	vouloir	être
0	0.0000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000
1	0.0000	0.00000	0.000000	0.297858	0.000000	0.000000	0.000000
2	0.1522	0.00000	0.107367	0.000000	0.076152	0.000000	0.195175
3	0.0000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000
4	0.0000	0.15919	0.221107	0.000000	0.000000	0.294549	0.133978

FIGURE 12 – Exemple de matrice TF-IDF.

Le modèle `paraphrase-multilingual-MiniLM-L12-v2` est un modèle de transformers qui prend en entrée un texte (typiquement une ou plusieurs phrases) et renvoie un vecteur de taille 384. Il ne supporte pas plus de 512 tokens en entrée, ce qui nécessite de découper si besoin la description ou l'intitulé lors de l'encodage par ce modèle. Dans le cas où un texte dépasse les 512 tokens, le texte est découpé en  $p$  morceaux de 512 tokens maximum et chaque morceau est vectorisé. On prend alors la moyenne des  $p$  vecteurs obtenus pour conserver l'information de chaque partie du texte.

Il convient de noter que la valeur de la coordonnée  $i$  d'un vecteur ne représente pas un poids associé à un mot ou token précis du texte d'entrée, contrairement à ce que fournit la vectorisation TF-IDF. Les 384 dimensions de sortie permettent de décrire numériquement la richesse du langage dans son ensemble, de sorte que deux vecteurs *proches* représentent deux textes dont le sens et le contexte sont *proches*. La classe `Vectorizer` implémente l'utilisation du modèle de transformers en intégrant la segmentation en blocs de 512 tokens avec moyennage.

Les inconvénients de cette technique résident dans l'aspect *boîte noire* du modèle, dans le sens où l'encodage n'est pas ou peu déterminé par nos propres choix. De plus, la dimension de sortie n'est pas choisie mais imposée par le modèle utilisé. En contrepartie, le modèle supporte plusieurs langues, ce qui est particulièrement intéressant dans notre cas, où les données contiennent de nombreuses langues. Cela évite de devoir passer par une étape de traduction. De manière générale, la structure du modèle demande des prétraitements relativement légers du texte d'entrée (voir 3.1.1), ce qui permet un allègement du nettoyage et éventuellement une meilleure qualité de ce dernier, car il sera géré en partie par le modèle lui-même, qui a été entraîné avec des outils de pointe.

Les vecteurs générés par les méthodes précédentes ne sont pas normalisés par défaut. Il est donc important de le faire pour s'assurer que chaque composante est située sur la même plage de valeurs et éviter que certains modèles trop sensibles aux valeurs numériques ne divergent lors de l'apprentissage. On utilisera les transformers de la librairie `scikit-learn` pour normaliser les données. Plusieurs choix sont possibles, menant à une volonté différente :

- `StandardScaler` : standardisation des valeurs en rendant la distribution normale. Permet de

contrôler l'échelle de valeurs tout en accordant de l'importance à chaque composante.

- **RobustScaler** : analogue à **StandardScaler** mais moins sensible aux outliers. Par construction des vecteurs, nous n'avons pas d'outliers significatifs qui pourraient impacter la normalisation.
- **MinMaxScaler** : normalise en se basant sur le minimum et le maximum des valeurs. Permet d'accorder de l'importance aux composantes importantes (par exemple, accentuer le poids relatif des mots déterminé par **TfidfVectorizer**). Il peut en résulter de meilleures performances si l'importance initiale des mots vectorisés rend correctement compte de la réalité du texte encodé. Dans le cas contraire, cela peut détériorer la qualité finale de l'encodage.

### 3.1.4 Complétion des valeurs manquantes

Lorsque les données sont fortement impactées par des valeurs manquantes, il convient de les traiter avec une attention particulière. En particulier, la suppression ou l'ignorance de ces dernières ne sont pas des solutions convenables. On doit donc faire un choix pour imputer à ces valeurs manquantes des données qui permettront d'améliorer la qualité du jeu de données. On rappelle que seule la colonne **description** contient des valeurs manquantes.

Une première option, implémentée au sein de la pipeline, consiste à imputer les intitulés aux descriptions. Cela se justifie par le fait que les intitulés (colonne **designation**) sont de bonne qualité, mais aussi parce qu'une partie des descriptions manquantes s'expliquent par le fait qu'elles sont en réalité intégrées au sein des intitulés (on a alors déjà accès à toute l'information, elle est simplement mal placée). Parmi les inconvénients de cette stratégie, on peut noter la faible variabilité induite. Aucun nouveau contenu n'est généré, et les variables se corrèlent davantage entre elles pour les données où la description était manquante. L'avantage principal est la simplicité de cette méthode qui permet rapidement de gérer le problème des valeurs manquantes. La classe représentant cette méthode est **NaiveDescriptionImputer**.

Une méthode limitant la corrélation naturelle des variables est celle analogue à ce qui se fait dans le ré-échantillonnage des catégories. On complète les valeurs manquantes en tirant aléatoirement une valeur existante au sein de la catégorie. La classe **RandomDescriptionImputer** implémente cette technique. L'inconvénient de cette méthode repose sur la grande variabilité des données au sein de chaque catégories. Par exemple, dans la catégorie associée à l'univers de la piscine et de l'environnement aquatique, on y trouve des pompes et des piscines gonflables. Si une description associée à une piscine gonflable est manquante et que l'on tire aléatoirement la description d'une pompe, on constate que d'un point de vue humain, le remplacement est relativement absurde. Néanmoins, les pompes faisant partie de la catégorie, elles la caractérisent en partie et les modèles devraient ainsi être capables de tirer des prédictions de meilleure qualité avec la présence de cette description.

### 3.1.5 Crédation de variables

Dans l'optique d'enrichir le jeu de données, il peut être intéressant de réfléchir à la construction de nouvelles variables explicatives à partir de celles déjà présentes.

Concernant les données textuelles, plusieurs enrichissements peuvent être intégrés à partir des variables **designation** et **description**. Ces variables dérivées permettent de mieux capturer les spécificités des produits et de renforcer la capacité discriminante du modèle, en particulier dans des contextes où les données d'entrée sont textuelles, bruitées ou redondantes. Ces nouvelles variables, dont une liste non exhaustive est présentée ci-dessous, apportent une information structurelle ou linguistique complémentaire utile à la classification :

- Le nombre de mots et le nombre de caractères : ces métriques sont souvent corrélées à la complexité ou à la richesse de l'information fournie dans les informations du produit. Par exemple, une longue description peut correspondre à un produit technique (informatique, électronique), tandis qu'une courte peut indiquer un produit générique ou standardisé.
- Le ratio entre le nombre de mots uniques et le nombre total de mots : ce ratio permet d'estimer la diversité lexicale d'une description. Un ratio faible peut refléter une répétition excessive

(exemples : spams, templates automatiques, etc), tandis qu'un ratio élevé suggère une richesse sémantique plus forte.

- L'existence de chiffres dans la description : une variable booléenne est définie pour indiquer la présence de chiffres, souvent associés à des références techniques, dimensions, puissances ou quantités. Cela peut être un signal utile pour certains produits fortement liés à des spécifications numériques (ex. : ampoules, visseries, batteries...).
- La détection d'objet sur les images par des modèles pré-entraînés tels que ResNet ou YOLO permettent de faire ressortir dans certains cas les objets présents sur les images. Les corrélations entre les classes prédites par ces modèles et les catégories réelles des produits peuvent augmenter la richesse des variables et améliorer la classification. On peut retrouver les détails de ces corrélations sur le dépôt GitHub dans les fichiers `rakuten_resnet.ipynb` et `rakuten_yolo.ipynb`. On relève deux correspondances très intéressante entre deux catégories du modèle ResNet et des catégories de Rakuten, illustrées sur les histogrammes suivant :

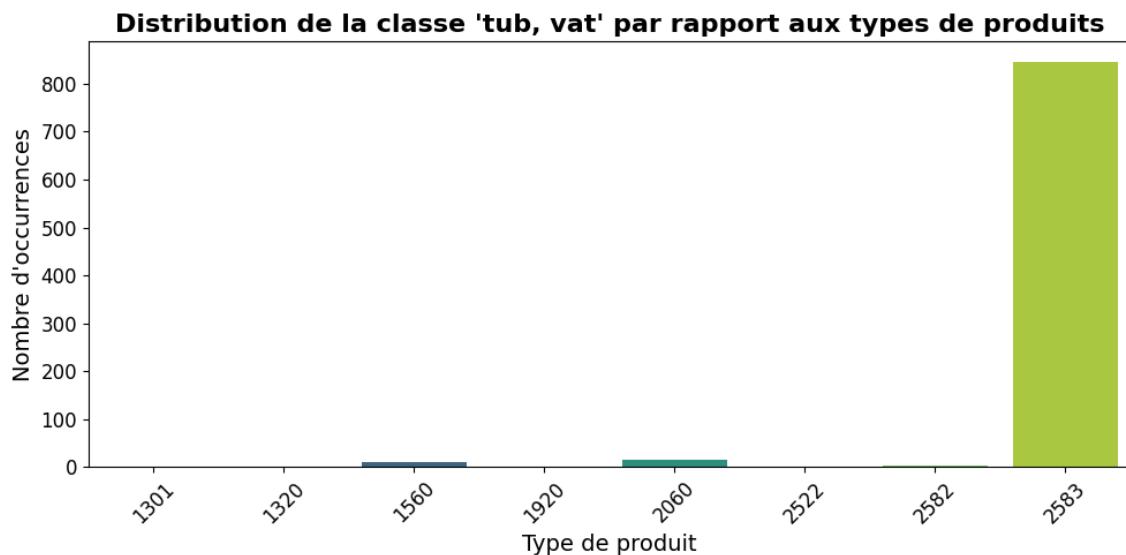


FIGURE 13 – Histogramme des objets classés dans la catégorie ResNet « *tub, vat* » en fonction des catégories de Rakuten.

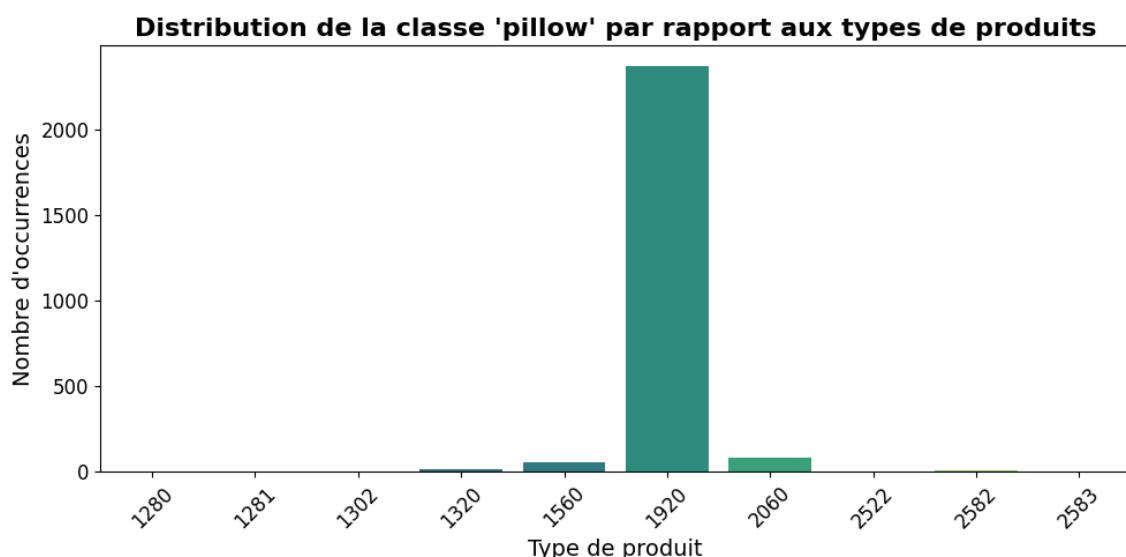


FIGURE 14 – Histogramme des objets classés dans la catégorie ResNet « *pillow* » en fonction des catégories de Rakuten.

Dans les deux situations précédentes, la classification de ResNet coïncide avec les catégories de

produits Rakuten. On pourra donc utiliser l'indexation des classes de ResNet (ou YOLO, de manière analogue) pour enrichir nos données. Outre les correspondances parfaites, l'important est qu'une catégorie de produits Rakuten soit associée à une ou quelques classes de ResNet (ou YOLO).

Selon la nature du modèle utilisé, ces nouvelles variables pourront être intégrées librement au sein de  $X$ ,  $X^{im}$ , ou les deux.

### 3.2 Pipeline sur $X^{im}$

L'ensemble des transformations présentées ci-dessous sont appliquées de manière aléatoire. Plus précisément, chaque image sera transformée (ou non) suivant les règles suivantes :

1. Tirage aléatoire des transformations à appliquer (par exemple bruit, contouring, rotation).
2. Sélection aléatoire des paramètres de chaque transformation retenue (par exemple intensité du bruit, angle de la rotation, etc).
3. Tirage aléatoire de l'ordre d'application des transformations. Toutes les transformations ne commutent pas avec les autres, augmentant la richesse des transformations possibles.
4. Application des transformations précédentes à l'image.

Cette succession d'étapes permet de construire des transformations presque sûrement uniques pour chaque image, optimisant la richesse des données. Par ailleurs, cette variété de changement au sein des images permettra aux modèles d'*apprendre* les bonnes caractéristiques des images et de limiter l'overfitting des informations inutiles présentes dans les données.

Toutes ces transformations sont parfaitement définies mathématiquement et *a fortiori* implémentables aisément, notamment en utilisant des librairies dédiées comme `opencv-python`.

#### 3.2.1 Redimensionnement

Le redimensionnement des images d'un jeu de données peut être imposé par plusieurs facteurs, notamment ceux relevés dans la phase d'exploration (voir 2.3). On applique donc un redimensionnement dont les dimensions de sortie peuvent être choisies librement. Certaines dimensions sont particulièrement intéressantes, notamment pour certains modèles de classification d'images pré-entraînés comme ResNet, où des images de taille  $224 \times 224$  sont appréciées.

#### 3.2.2 Transformations numériques

Une grande majorité des modèles sont sensibles aux plages de valeurs que prennent les données. Il est bien connu que la normalisation des données est une étape qui permet de sensiblement améliorer la performance de certains modèles. Dans les cas des images, bien que l'ensemble des valeurs (l'intensité des pixels sur les trois canaux) soit canalisé entre 0 et 255, il est important de les normaliser entre 0 et 1 pour améliorer la qualité des données. Entre autres, les modèles pré-entraînés demandent que les données fournies soient normalisées, ainsi cette étape élargit la compatibilité de nos données avec les modèles existants.

En complément de la normalisation des images, plusieurs transformations importantes et sources d'un enrichissement considérable de  $X^{im}$  ont été implémentées :

- Altération du contraste par l'application de filtres sur les images permettant d'intensifier ou d'atténuer le contraste des images.
- Intensification des contours par l'application de filtres sur les images.
- Renversement des couleurs par application de filtres négatifs.
- Perturbations indépendantes des canaux pour modifier les couleurs de l'image.
- Application de bruit permettant d'altérer la qualité globale de l'image.
- Application de flou permettant d'altérer la netteté de l'image.

### 3.2.3 Transformations géométriques

Pour enrichir davantage la qualité des images vis-à-vis des modèles qui seront utilisés, plusieurs transformations géométriques ont été intégrées dans le pipeline :

- Rotation des images autour d'un pixel donné.
- Symétrie des images par rapport à un axe donné.
- Translation des images suivant un vecteur donné.
- Zoom et dézoom des images focalisé sur un pixel donné.

De manière analogue aux transformations numériques des images (voir 3.2.2), les transformations ci-dessus seront choisies de manière aléatoire, que ce soit la sélection des transformations, l'ordre dans lesquelles elles s'appliquent et les valeurs des paramètres des transformations utilisées.

### 3.2.4 Crédit de variables

Dans le cadre de notre pipeline dédié aux données visuelles  $X^{im}$ , nous avons procédé à l'extraction de variables numériques dérivées directement des caractéristiques intrinsèques des images. Ces variables, aussi appelées *features statistiques*, permettent d'enrichir la représentation d'une image sans passer par des réseaux convolutifs. Elles servent notamment de base à certains modèles de machine learning classiques (comme XGBoost ou CatBoost) ou peuvent être utilisées comme features additionnelles dans des modèles hybrides. Parmi les variables construites, on relève :

- La moyenne des intensités par canal (rouge, vert, bleu).
- L'écart-type des intensités (permettant de capturer le contraste global d'une image).
- Le canal dominant de l'image, obtenu en comptant le nombre de pixels dominés par chaque canal.
- Les intensités extrémiales (indicateurs de bruit et/ou d'écrêtage).

Selon la nature du modèle utilisé, ces nouvelles variables pourront être intégrées librement au sein de  $X$ ,  $X^{im}$ , ou les deux.

## 3.3 Pipeline sur $Y$

### 3.3.1 Ré-échantillonnage

Comme cela a été observé durant la phase exploratoire, les catégories doivent être ré-échantillonnées. La catégorie sur-représentée 2583 sera sous-échantillonnée tandis que toutes les autres seront sur-échantillonnées pour rendre le jeu de données équilibré vis-à-vis des classes. Différentes options sont considérées, notamment les techniques basées sur les algorithmes SMOTE. La bibliothèque spécialisée `imbalanced-learn` implémente de nombreuses variétés d'algorithmes reposant sur cette idée.

La quantité cible de données pour chaque classe est choisie de sorte que la quantité totale de données soit inchangée : on aura donc 3145 données pour chaque classe. Ce choix pourra être adapté en fonction de la réponse des modèles à la volumétrie des données.

La classe `RandomOverSampler` permet un sur-échantillonnage efficace et rapide des catégories. Elle sera utilisée en premier lieu à la fois sur les données textuelles et imagées. Concernant les images, on pourra accentuer la variabilité de ces nouvelles données en appliquant une nouvelle transformation à l'image originale (avant pré-traitement), la rendant sensiblement différente de l'image transformée de la donnée sélectionnée aléatoirement.

Pour la classe 2583, on utilisera la classe `RandomUnderSampler` qui permet un sous-échantillonnage aléatoire.

Plusieurs éléments conditionnent le choix des techniques de ré-échantillonnage :

- Le volume de données associé au nombre de variables post-traitement (plusieurs centaines) entraîne des limitations matérielles. Un ré-échantillonnage basé sur des stratégies complexes nécessitant de nombreux calculs conduit à des temps de traitement pouvant être longs et une quantité de mémoire requise trop importante.
- La dimension de l'espace des données étant très élevée ( $> 100$ ), les techniques de ré-échantillonnage basées sur les distances (notamment KNN) sont exposées à la malédiction de la dimensionnalité, avec des distances potentiellement extrêmes entre toutes les données rendant la notion de proximité inadaptée.

Le ré-échantillonnage des catégories est appliqué après l'exécution du pipeline sur  $X$ .

### 3.3.2 Renommage

Le renommage des classes est une transformation non essentielle dans le sens où elle n'influence pas la qualité des données pour les modèles. En revanche, elle permet une meilleure interprétation des résultats de ces derniers, étant donné qu'une catégorie sera à l'issue du renommage décrite par des mots plutôt qu'un code utilisé par Rakuten en interne. On pourra donc avoir une meilleure vision de ce qui est correctement classifié et de ce qui ne l'est pas.

Pour effectuer ce renommage, on se base sur les données textuelles nettoyées. On génère des nuages de mots pour chaque catégorie et on sélectionne les mots les plus importants de chaque nuage pour représenter les catégories. Si deux catégories ont le même mot le plus important, on sélectionne alors pour chacune des deux leur deuxième mot important, et ainsi de suite jusqu'à obtenir des mots distincts. L'ensemble des mots choisis dans ce processus sont conservés pour identifier ces deux catégories. La classe `CatRenamer` implémente cette transformation.

Naturellement, une correspondance sera faite entre les catégories renommées et des valeurs numériques classiques pour une meilleure gestion par les modèles.

## 3.4 Résultats du prétraitement

### 3.4.1 Exemples issues de $X$

Voici un exemple de données textuelles avant traitement par le transformateur `CharacterCleaner` :

- `designation` : *La Guerre Des Tuques.*
- `description` : *Luc a des id&eacute;es de grandeur. Il veut organiser un jeu de guerre de boules de neige et s'arranger pour en &ecirc;tre le vainqueur incontest&eacute;. Mais Sophie s'en m&ecirc;le et chambarde tous ses plans....*

Suite au nettoyage, les variables deviennent :

- `designation` : *La Guerre Des Tuques.*
- `description` : *Luc a des idées de grandeur. Il veut organiser un jeu de guerre de boules de neige et s'arranger pour en être le vainqueur incontesté. Mais Sophie s'en mêle et chambarde tous ses plans....*

On note que les accents mal formatés ont été correctement traités.

L'exemple suivant est issue de la classe `NettoyeurTexte` :

- Avant : *<b>Lot de 30 Spots encastrable orientable BLANC avec GU10 LED de 5W &#61; 40W</b>*.
- Après : *lot spot encastrable orientable blanc led.*

On note les différences entre les deux nettoyages, imposées par les modèles d'encodage décrits précédemment (`TfidfVectorizer` et `Vectorizer`).

L'exemple ci-après illustre un cas où une traduction a été appliquée à la description (de l'anglais au français) :

- Avant : *pCan my 2-year-old play a game ? Yes Each game in our 2 year old collection is designed so that your child will experience what its like to take turns play within the rules (even loosely) and possibly learn a new skill like rolling a die. Unlike games for older children however you dont win or lose any of these game s- you play to explore you play to learn and you play to connect and have quality time with your little one. Since 1983 play has been the heart and soul of Peaceable Kingdom. Silly play special play wholehearted play We create experiences to help kids play well together so they can play well in the world. Our games and gifts connect kids and families encourage learning and self-expression and let imaginations soar. And if a piece of your Peaceable Kingdom game or gift gets lost in all that fun we will gladly send you a replacement piece - just contact us directly with your need When play comes from the heart and feeds the soul thats Peaceable Kingdom.p.*
- Après : *pcan mon year old jouer jeu oui chaque jeu notre collection être con sorte votre enfant aller vivre aimer jouer tour tour gle che pouvoir apprendre nouveau competence rouler mort contrairement jeu enfant gramme cependant gagner perdre jeu jouer explorer jouer apprendre jouer connecter avoir temps qualit votre petit depuis jeu avoir royaume paisible jeu stupide jeu cial jouer tout experience aider enfant jouer bien ensemble afin pouvoir bien jouer monde notre jeu cadeal relier enfant famille encourager apprendre exprimer laisser imagination envoler morceau votre jeu paisible royaume cadeau perdre tout plaisir aller volontier envoyer recharge juste contacter directement votre besoin lorsque jeu venir nourrir royaume paisible.*

Notons que le texte post-traitement inclut aussi le passage dans **NettoyeurTexte**, ce qui explique que de nombreux mots soient absents.

### 3.4.2 Exemples issues de $X^{im}$

On illustre ici quelques images originales du jeu de données qui ont été transformées :

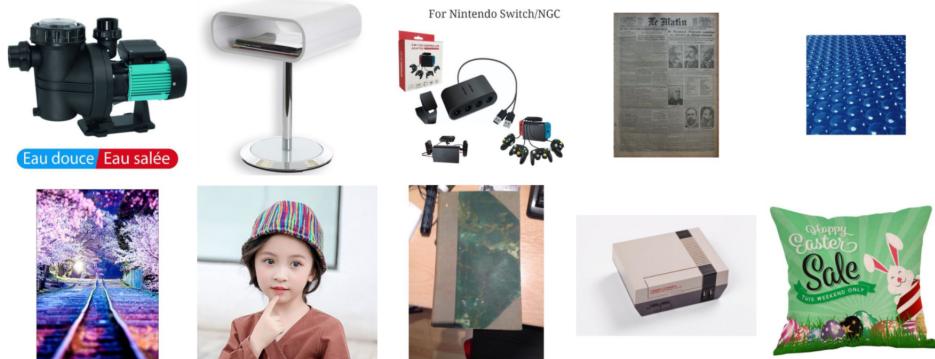


FIGURE 15 – Images originales du jeu de données.

Voici un exemple de rotations de ces images :



FIGURE 16 – Images après rotations aléatoires.

Les mêmes images subissant un bruit :



FIGURE 17 – Images après bruitage.

Comme expliqué dans la partie 3.2, ces transformations peuvent être cumulées. Les transformations ci-dessus ont été produites à titre d'exemples.