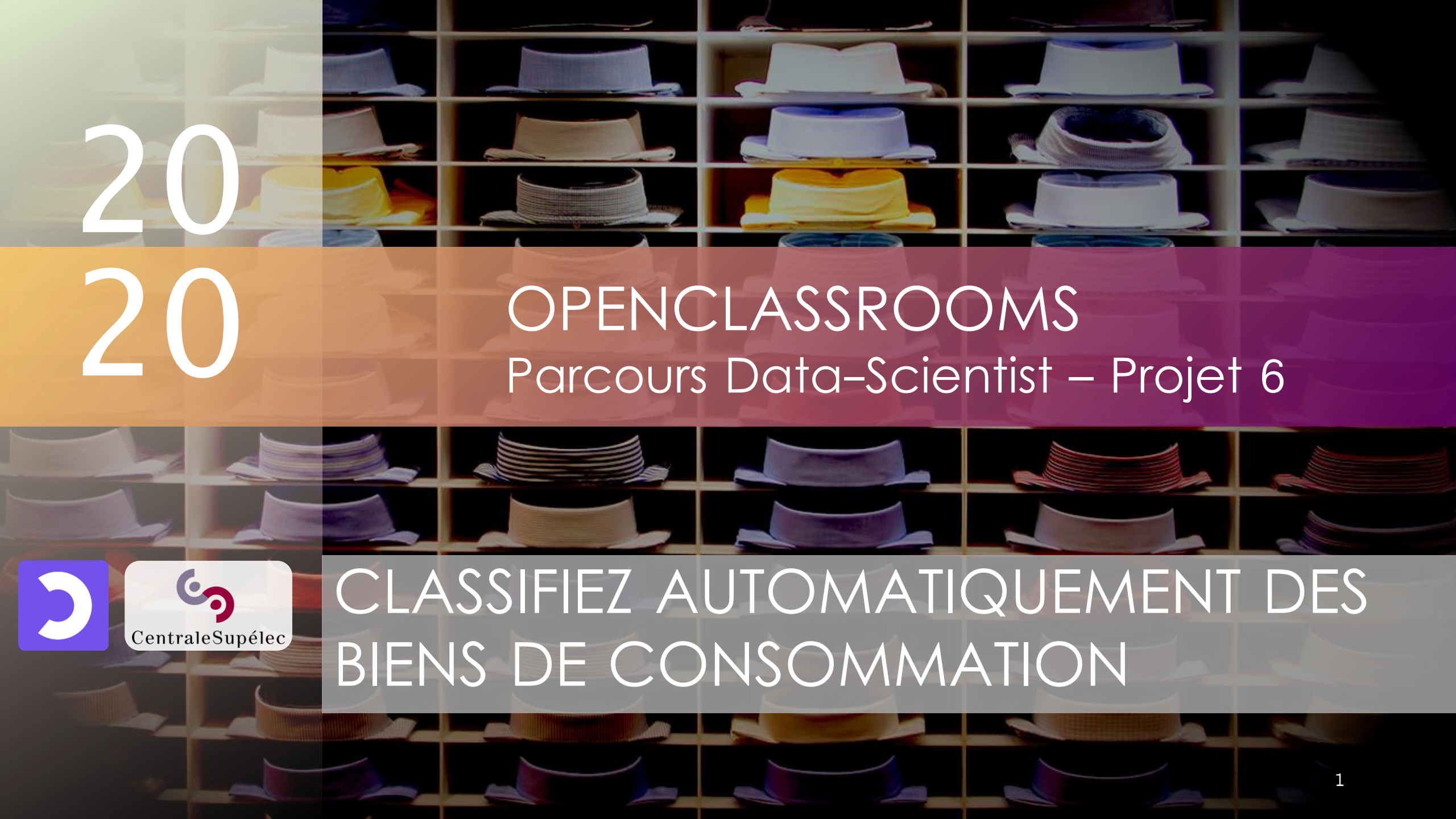


2020

OPENCLASSROOMS

Parcours Data-Scientist – Projet 6

CentraleSupélec

CLASSIFIEZ AUTOMATIQUEMENT DES BIENS DE CONSOMMATION

SOMMAIRE

- Problématique
- Présentation des données (structure / contenu du dataset)
- Nettoyage des données
- Analyse univariée
- Exploration du texte
- Analyse visuelle
- Transfert Learning : utiliser un CNN pré–entrainé pour classifier les images
- Conclusion et perspectives

PROBLEMATIQUE

- **Historique :** Sur la place de marché, des vendeurs proposent des articles à des acheteurs en postant une photo et une description.
- **Problème :** l'attribution de la catégorie d'un article est effectuée manuellement => peu fiable.
- **Objectif :** étude de faisabilité d'un moteur de classification d'articles sur la base d'image et de descriptions.

PRESENTATION DES DONNEES

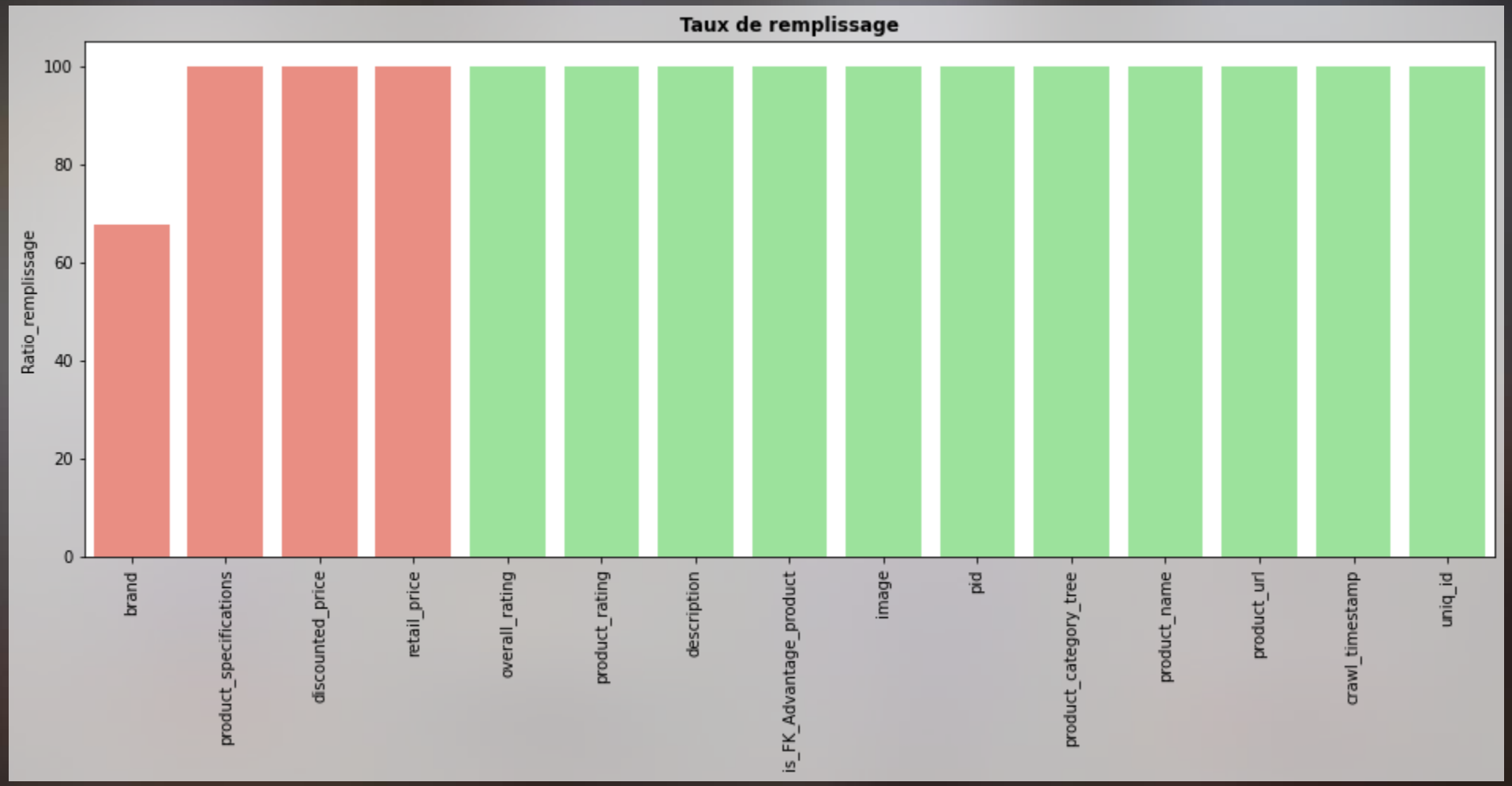
Structure du dataset, d'où proviennent les données ?

Contenu du dataset, taux de remplissage

STRUCTURE DU DATASET

- **Jeu de données :** disponible sur Amazon AWS
- **Format de fichier :** .csv et .jpg
- **Taille :** 329 Mo
- **Nombre de colonnes / lignes :** 1050 lignes & 15 colonnes

CONTENU DU DATASET



- Majorité de colonnes remplies
- 1 colonne (brand) remplie à presque 70%
- 3 colonnes avec peu de valeurs manquantes

=> imputation possible sans trop de biais

NETTOYAGE DES DONNEES & FEATURE ENGINEERING

NETTOYAGE DES DONNEES & FEATURE ENGINEERING

- Pas de datasets à fusionner
- Pas de doublons
- Pas de variables manquantes

2 étapes de Nettoyage – F.E.

NETTOYAGE DES DONNEES

Formatage des colonnes

Imputation des valeurs manquantes

```
# Conversion de crawl_timestamp au format date
import datetime

df['crawl_timestamp'] = pd.to_datetime(df['crawl_timestamp_2'], infer_datetime_format=True)
df = df.drop(['crawl_timestamp_2'], axis=1)
df.info()
```

FEATURE ENGINEERING

Formatage des colonnes

Imputation des valeurs manquantes

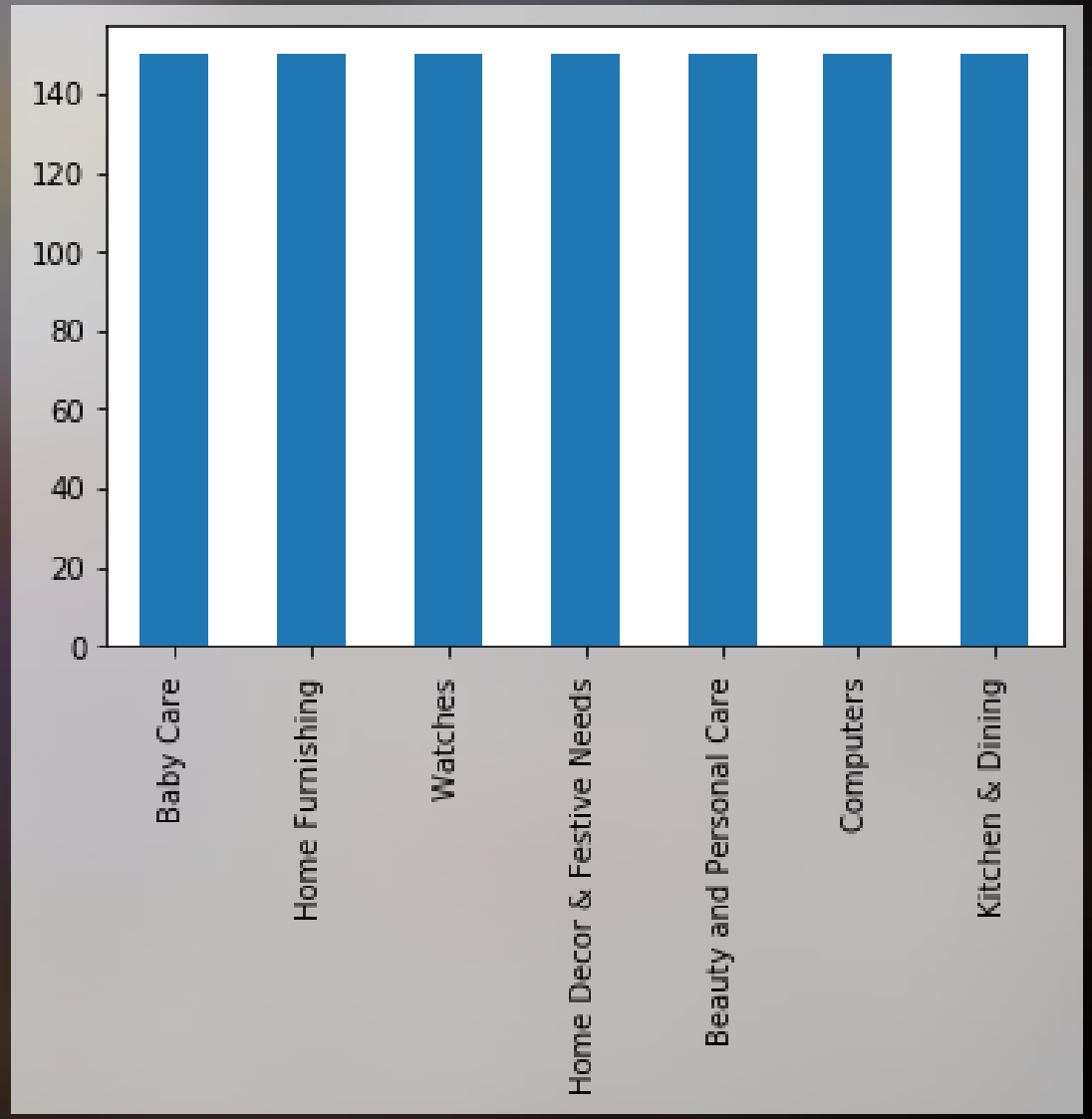
```python
# Importation des bibliothèques
import numpy as np
from sklearn.impute import KNNImputer
from sklearn.preprocessing import OrdinalEncoder

# Instancier les deux paquets à utiliser
oe = OrdinalEncoder()
imputer = KNNImputer()

# On définit une liste des variables catégorielles à encoder (tout ce qui n'est pas numérique)
cat_cols = df.select_dtypes(include=['object', 'bool', 'datetime64[ns]']).columns.tolist()

# Fonction d'imputation EID (Encodage, Imputation, Décodage)
def EID(data):
    global df_imputed

    # Fonction d'encodage
    def encode(data):
        '''fonction d'encodage des données en lieu et place des données originales'''
```

ANALYSE UNIVARIÉE

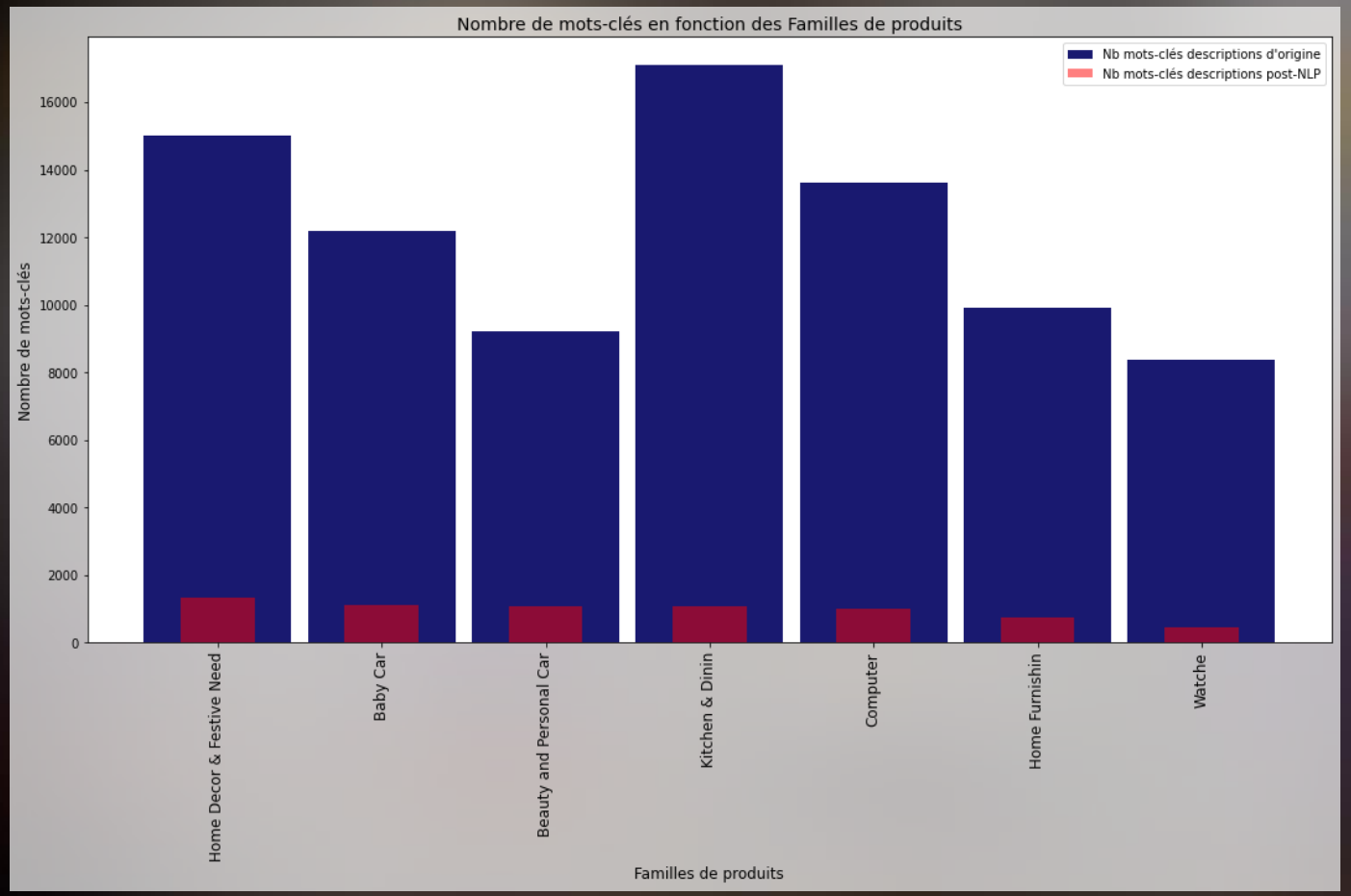
Distribution des variables

ANALYSE UNIVARIÉE

- 1050 produits, chacune ayant une illustration et une description (=> 1050 images au format *.jpg*)
- Pour catégoriser ces produits, il est nécessaire de n'en récupérer que la famille => racine de famille de produit
- 7 familles de produits
- Répartition équivalente du nombre de produits par famille (~150 produits/famille)



ANALYSE UNIVARIÉE

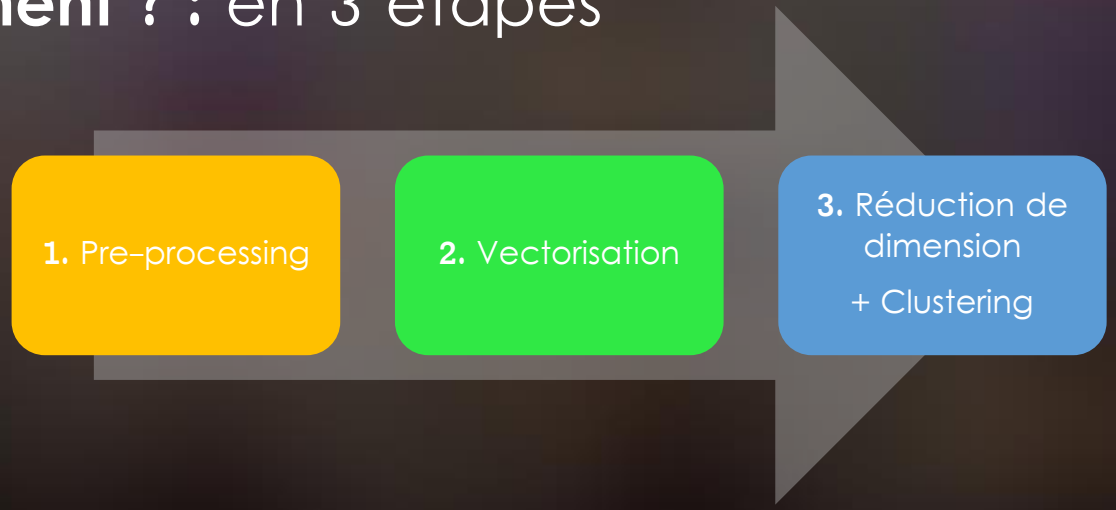


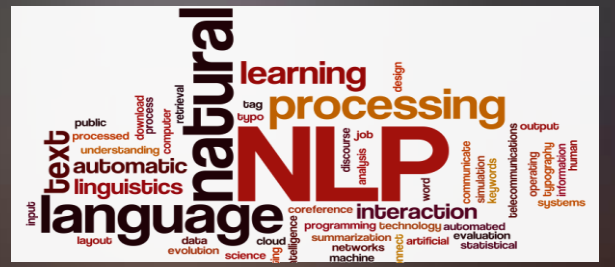
- Mots-clés nombreux par famille de produits (notamment les articles de cuisine, + 16000 mots)
- Le traitement de texte *NLP* (*Natural Language Processing*) via la librairie *nltk* permet de réduire drastiquement ce nombre
- Après ce traitement, les articles de décoration prennent la tête du classement

EXPLORATION DU TEXTE

Peut-on classifier les produits en fonction de leur description ?

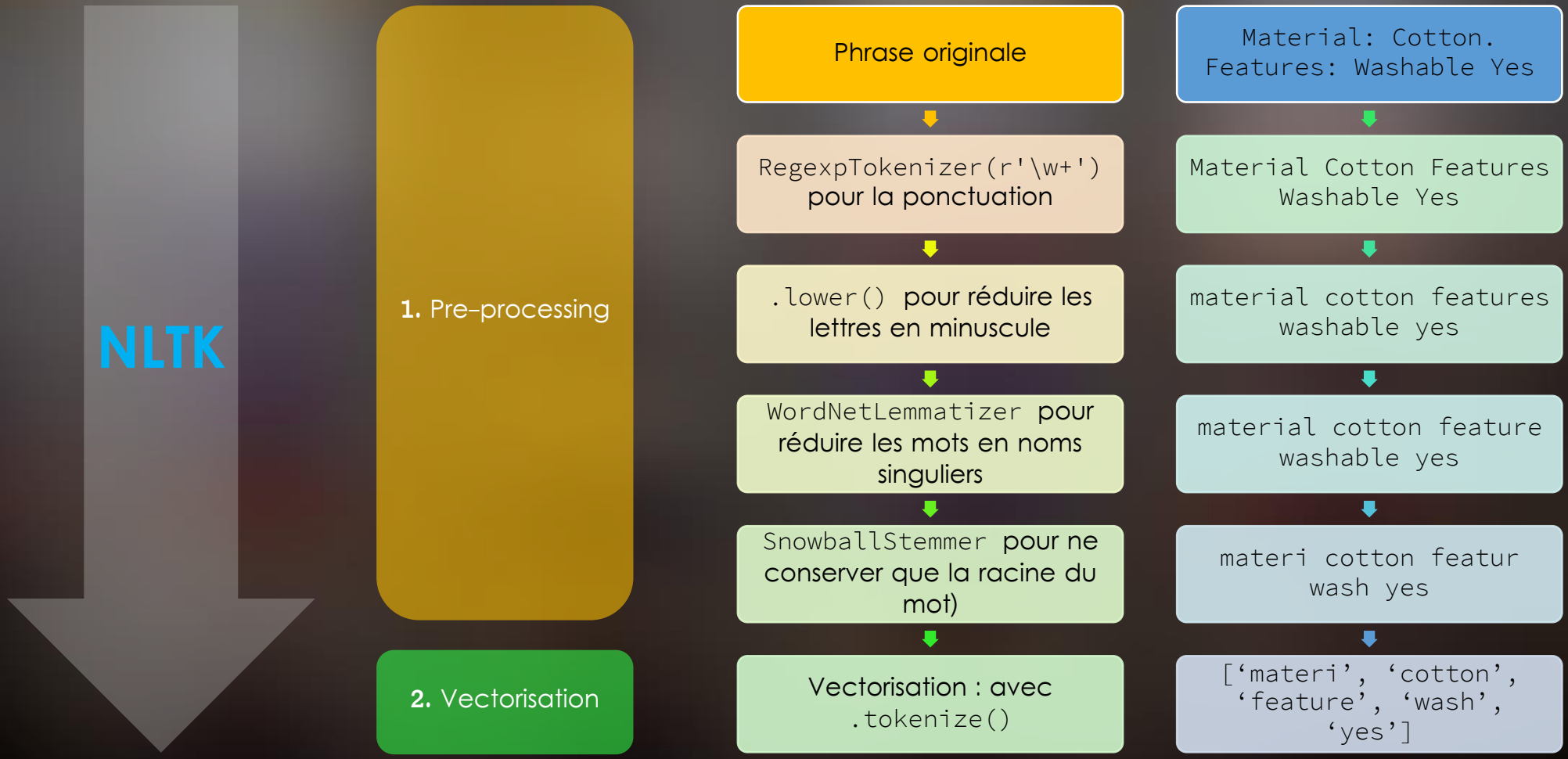
- **Proposition :** utiliser l'analyse naturelle du langage NLP *(« Natural Language Processing »)*
- **Comment ? :** en 3 étapes





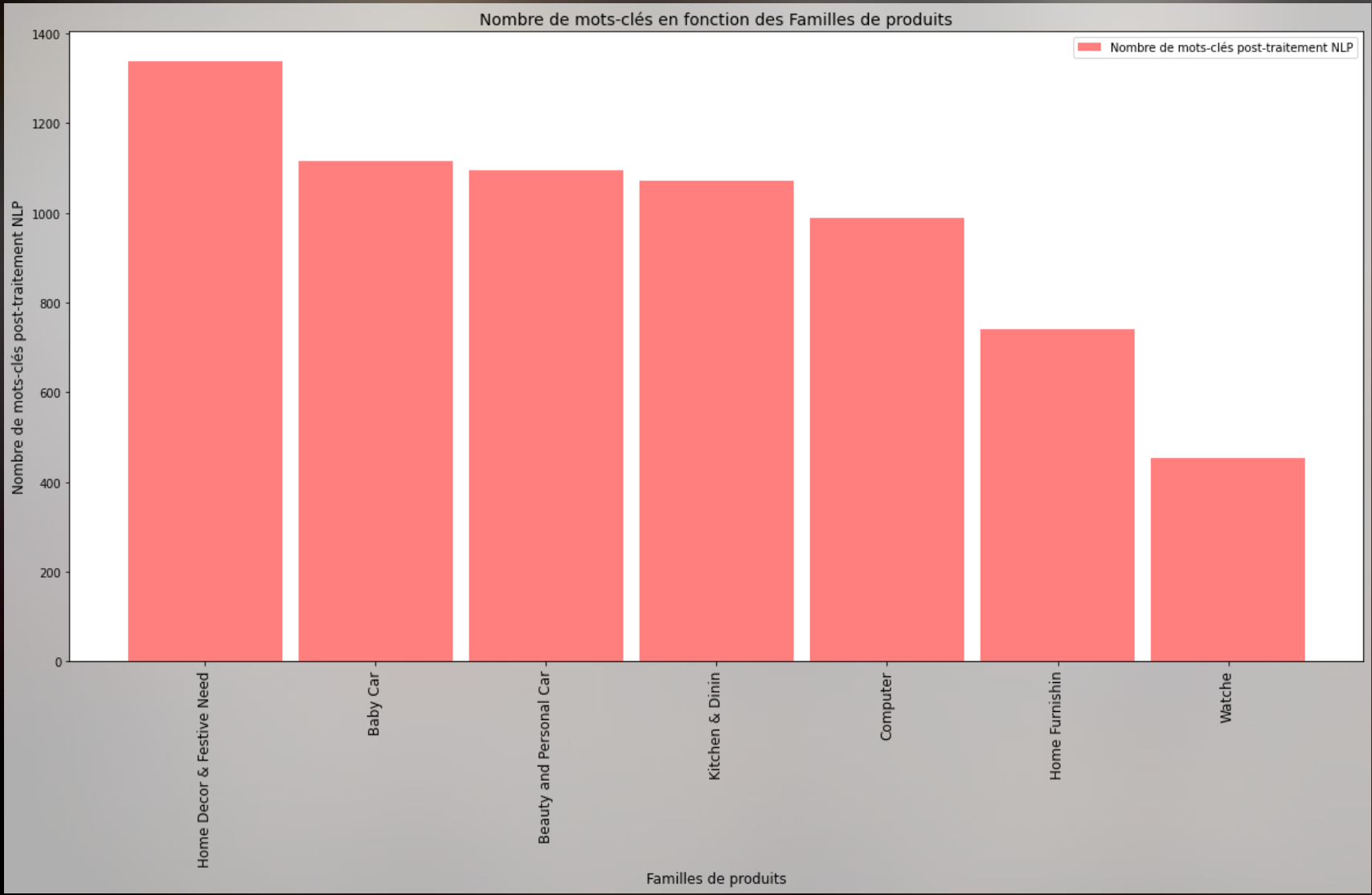
- **NLP :**
  - Rôle : extraire des informations et une signification d'un contenu textuel
  - Méthode : utilisation de NLTK *(« Natural Language Toolkit »)*, l'un des librairies les plus utilisées de NLP

Etapes du traitement de texte



EXPLORATION DU TEXTE

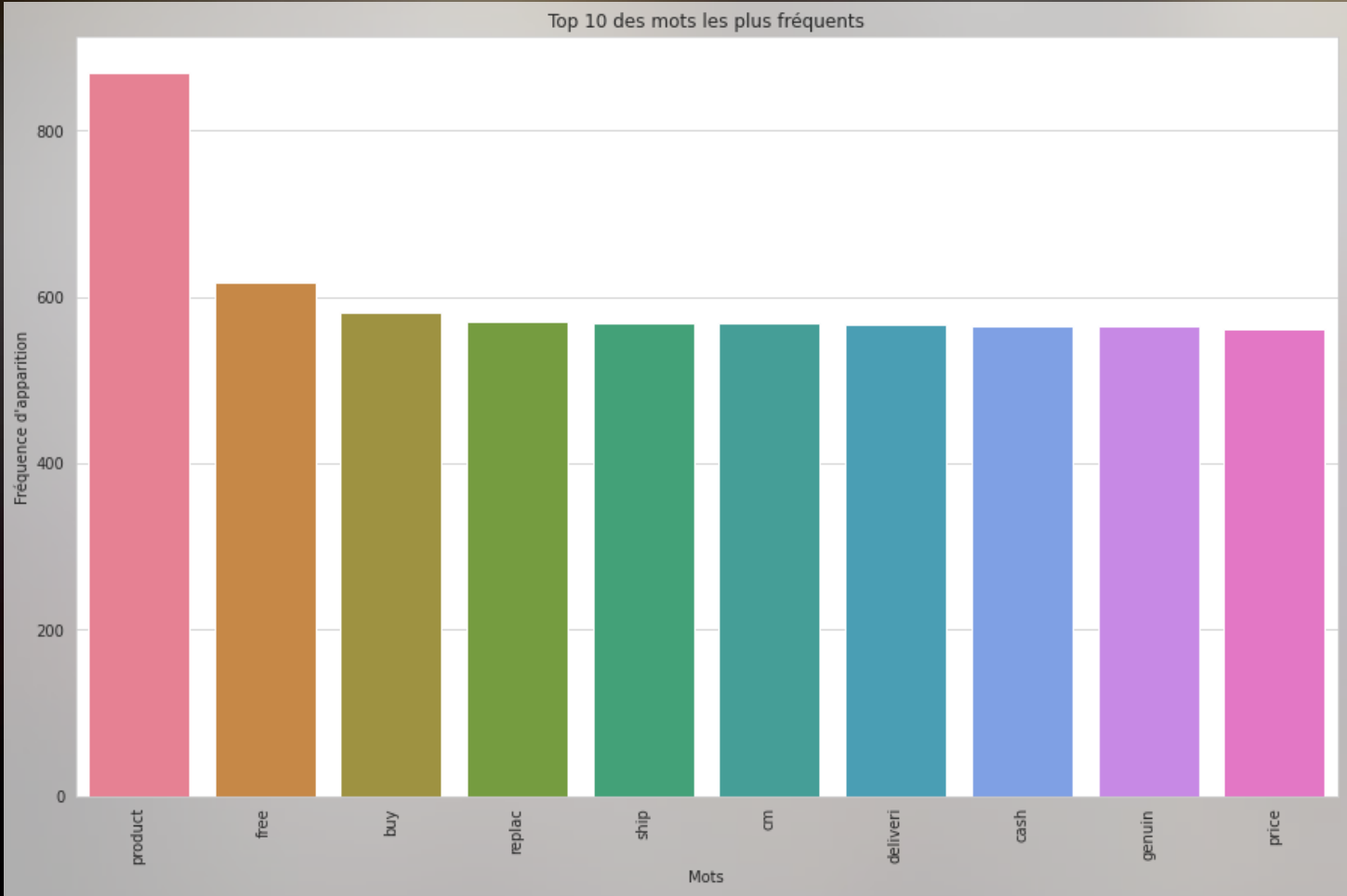
Quelques observations suite à ces 2 étapes

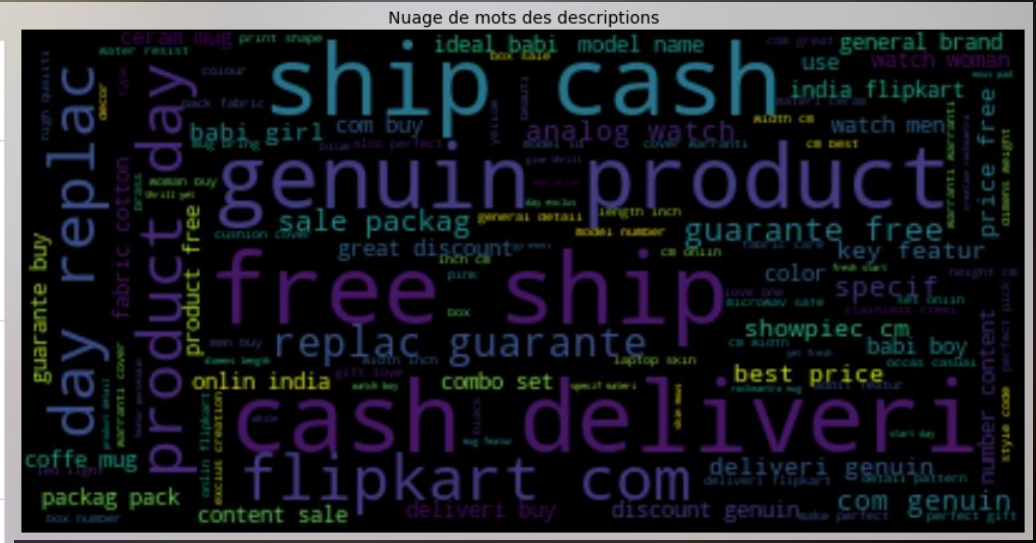


- La famille de produits *Home Decor & Festive Need* comprend le plus de mots variés (~1350 mots)
- A l'inverse, la famille de produits Watche ne possède que ~500 mots différents.

EXPLORATION DU TEXTE

Quelques observations suite à ces 2 étapes





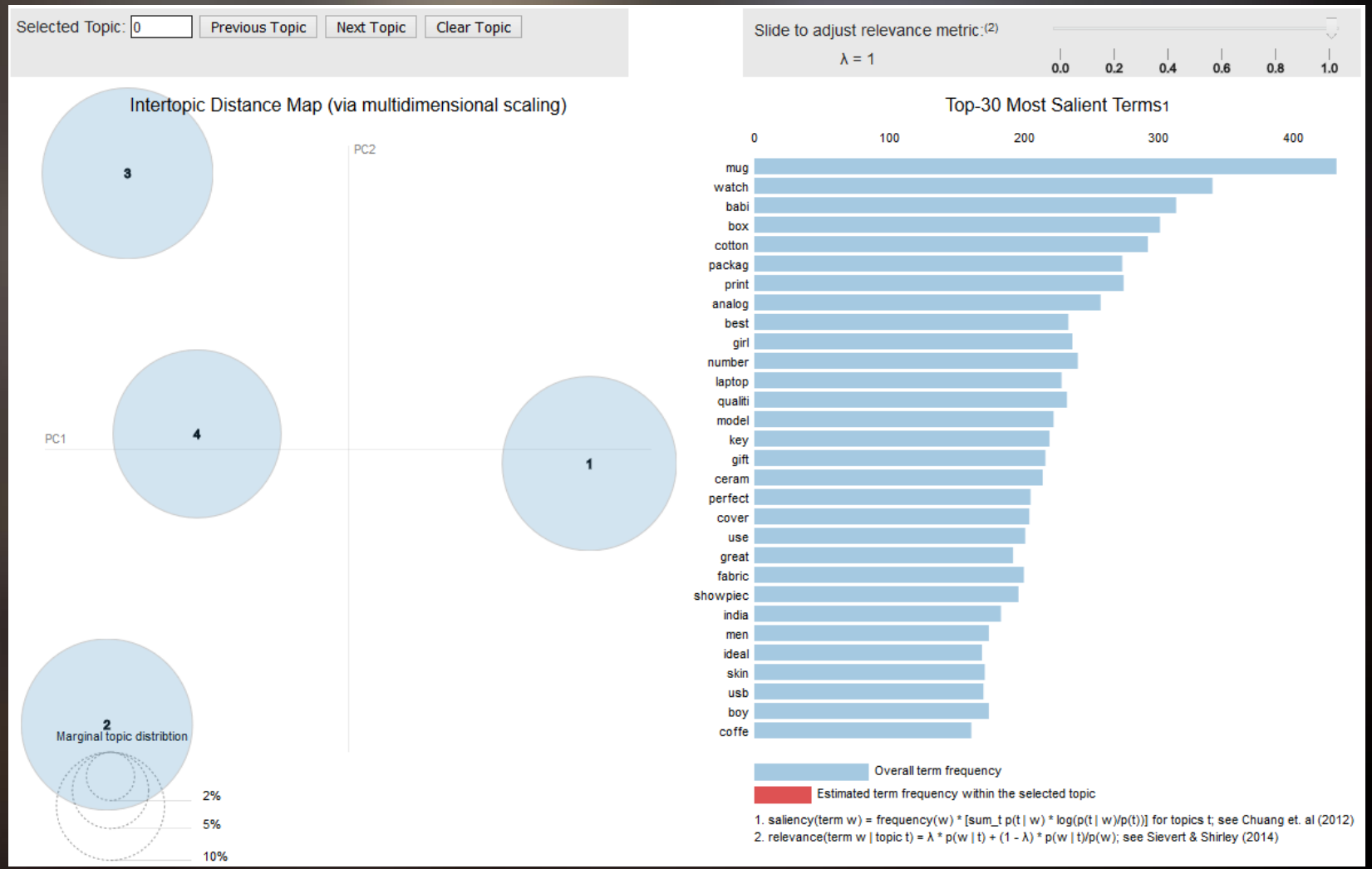
- Le nuage de mots met en évidence des mots liés à l'acte d'achat en tant que tel
- Le mot le plus utilisé est « *product* » (~830 fois), loin devant « *free* » et le reste dont la fréquence est similaire.

EXPLORATION DU TEXTE

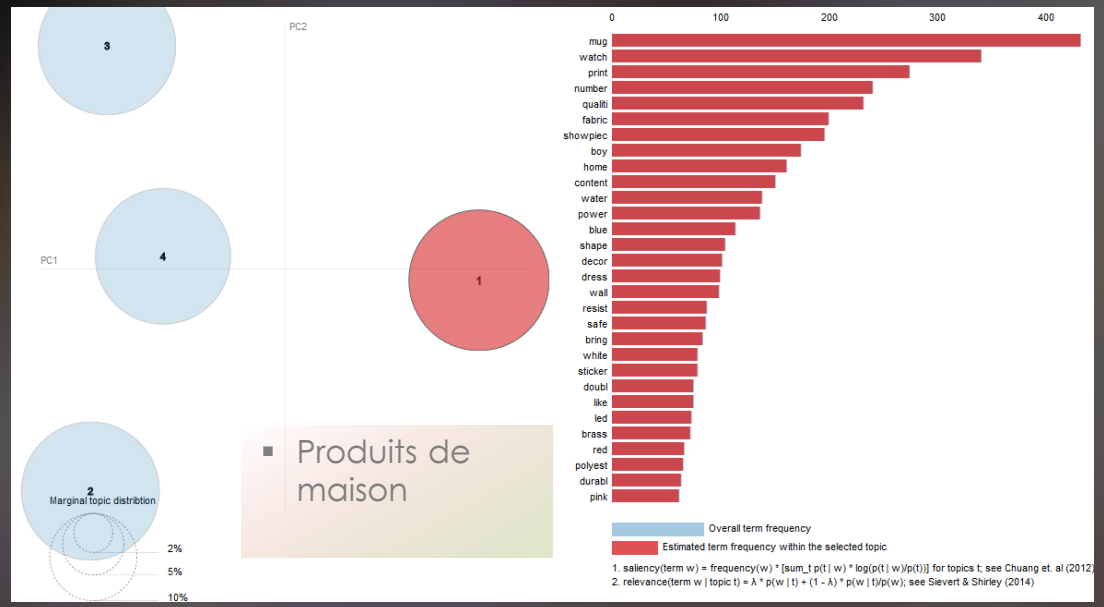
3. Réduction de dimension
+ Clustering

Visualisation par LDA Latent Dirichlet Association (pyLDAvis)

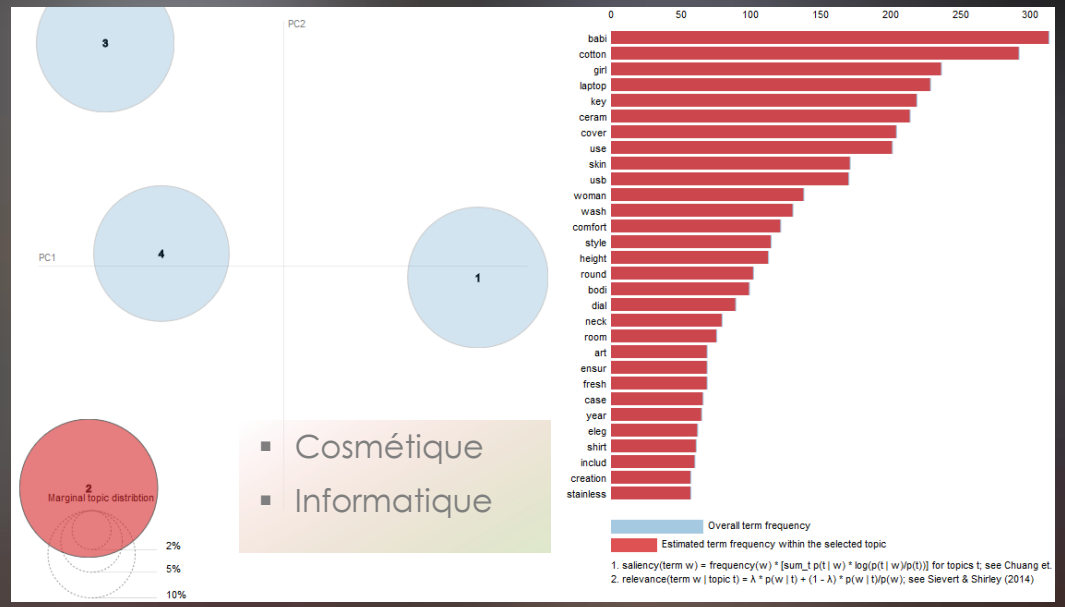
- Le LDA est une méthode de classification de texte non supervisée qui suit la logique : *mots* > *topics* (ensemble de mots dans un cluster) > *documents* (possèdent un topic dominant)
- LDA peu utiliser 2 méthodes :
  - *TF (Term Frequency)* : on ne se soucie pas si le mot est commun ou non (les articles, pronoms, etc. ont le même poids que n'importe quel mot)
  - *TF–IDF (Term Frequency–Inverse Document Frequency)* : plus un mot est fréquent (c'est le cas des articles, pronoms), plus son poids est faible.
- **TF–IDF est donc la meilleur méthode dans notre cas de figure**



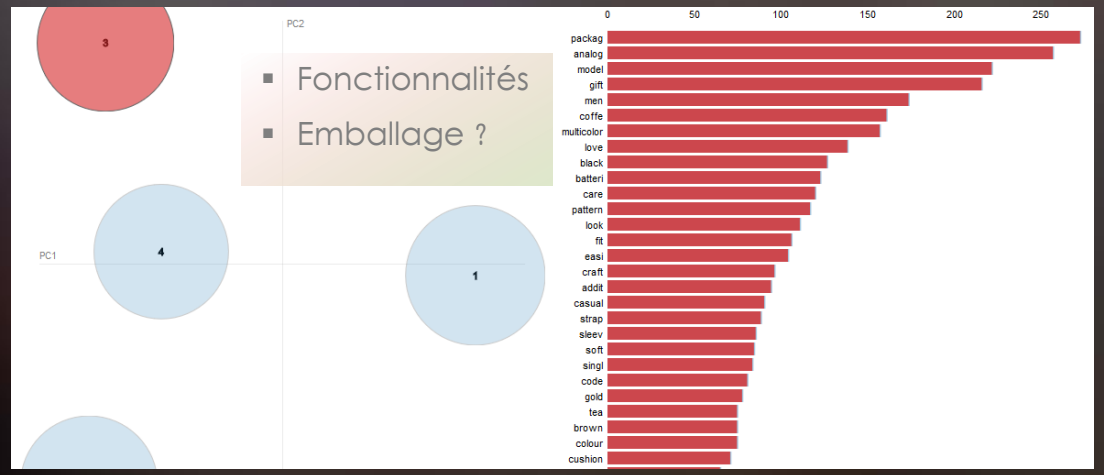
EXPLORATION DU TEXTE



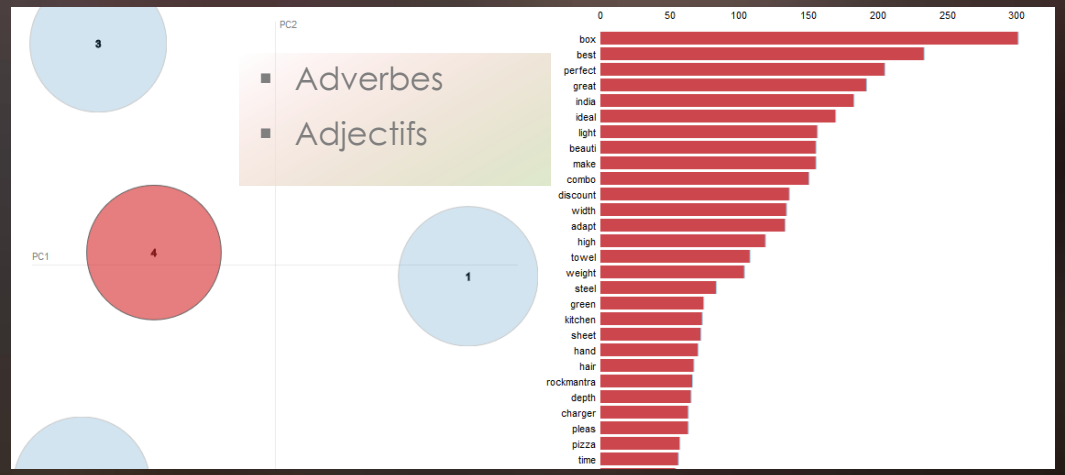
- Produits de maison



- Cosmétique
- Informatique



- Fonctionnalités
- Emballage ?



- Adverbes
- Adjectifs

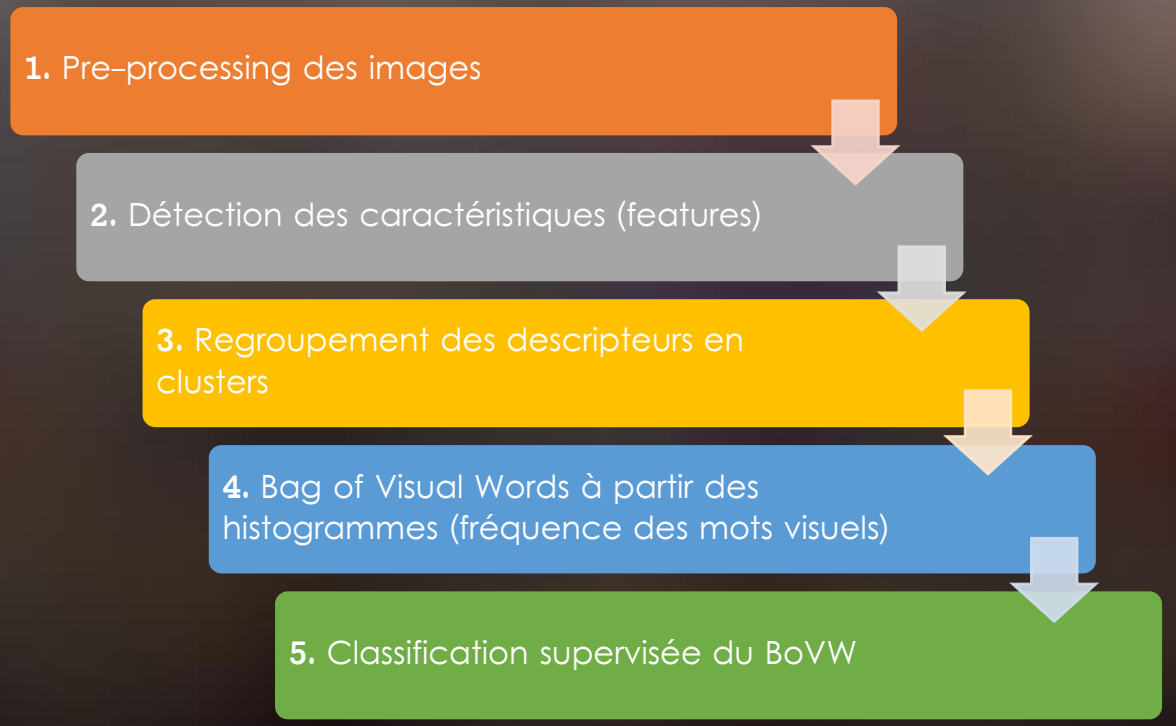
- Au-delà de 4 clusters, certains se mélangent fortement.
- Ici, les clusters sont biens séparés les uns des autres. Il est donc plus facile pour l'équipe marketing d'étudier ces groupes.
- Cependant, d'un point de vue thématique, les clusters ne sont pas assez explicites (volume de données trop faible ?)

ANALYSE VISUELLE

ANALYSE VISUELLE

Peut-on alors classifier les produits en fonction de leur image ?

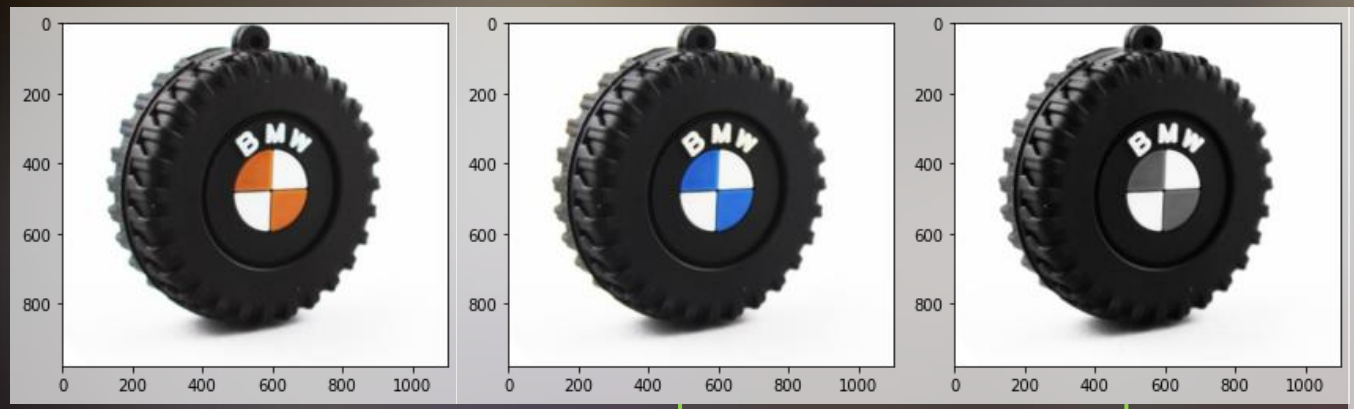
- **Proposition :** utiliser OpenCV et un algorithme de classification
- **Comment ? :** en 5 étapes

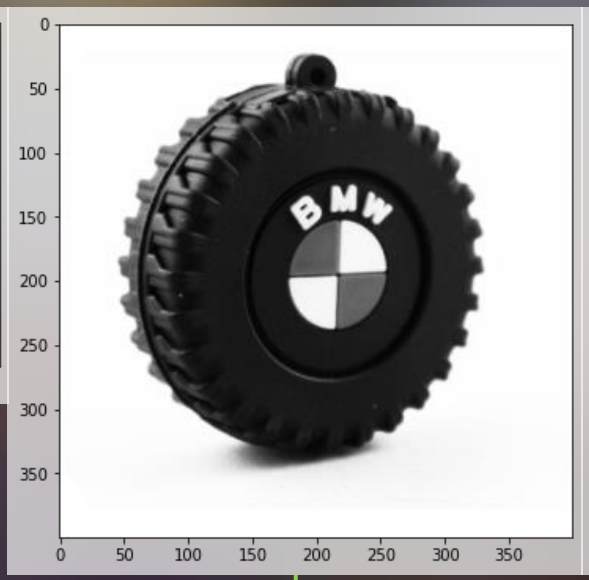


ANALYSE VISUELLE

1. Pre-processing des images

Image originale





```
# Les images étant au format BGR, Affichage des couleurs réelles
plt.imshow(cv2.cvtColor(imtest, cv2.COLOR_BGR2RGB))
```

```
# conversion en noir & blanc
img_gray = cv2.cvtColor(imtest, cv2.COLOR_BGR2GRAY)
plt.imshow(img_gray, cmap = 'gray')
```

```
# Redimmensionnement en carré sans distortion de l'image

def redim_carre(img, size):
    # get image dimensions
    h, w = img.shape[:2]

    # dif = max (height, width)
    dif = h if h > w else w

    # define interpolation for zooming and shrinkage
    interpolation = cv2.INTER_AREA if dif > size else cv2.INTER_CUBIC

    # for square images
    if h == w:
        return cv2.resize(img, (size, size), interpolation)

    # for non square images
    x_pos = (dif - w)//2
    y_pos = (dif - h)//2

    # define mask for both color and back and white images
    if len(img.shape) == 2:
        mask = np.full((dif, dif), 255, dtype=img.dtype)
        mask[y_pos:y_pos+h, x_pos:x_pos+w] = img[:h, :w]
    else:
        mask = np.full((dif, dif, img.shape[2]), 255, dtype=img.dtype)
        mask[y_pos:y_pos+h, x_pos:x_pos+w, :] = img[:h, :w, :]

    return cv2.resize(mask, (size, size), interpolation)
```

```
# Amélioration du contraste
clahe = cv2.createCLAHE(clipLimit=1, tileGridSize=(8,8))

# Afficher l'image
cl = clahe.apply(imtest_square)
plt.figure(figsize=(6,6))
plt.imshow(cl, cmap='gray')
```

ANALYSE VISUELLE

2. Détection des caractéristiques (features)


```
# On utilise orb qui a l'avantage de
orb = cv2.ORB_create(nfeatures = 500)
```

- Utilisation de BRIEF (ORB) à la place de SURF (SIFT) car le premier a l'avantage d'un traitement plus rapide que le second
- Il détecte des points clés *(keypoints)*, ces derniers étants décrits par les descripteurs *(descriptors)*
- On répète cette opération pour chaque image

ANALYSE VISUELLE

3. Regroupement des descripteurs en clusters

```
# Importation de l'algorithme de clustering (MiniBatchKMeans pour sauver de la RAM)
from sklearn.cluster import MiniBatchKMeans
from sklearn.metrics import silhouette_score

# On souhaite entraîner 100 clusters (d'autres valeurs sont testés dessous) de descripteurs à partir de notre liste

n_clusters = 100

model = MiniBatchKMeans(n_clusters=n_clusters, init_size=10000, n_init=10, random_state=2)
model.fit(list_descripteurs)
```

4. Bag of Visual Words à partir des histogrammes (fréquence des mots visuels)

```
# Visualisation concrète d'un BovW de 100 clusters
BovW[200]
```

	0	1	2	3	4	5	6	7	8	9	...	190	191	192	193	194	195	196	197	198	199
0	2.0	1.0	0.0	4.0	2.0	2.0	4.0	1.0	0.0	1.0	...	1.0	4.0	3.0	5.0	3.0	1.0	0.0	8.0	3.0	2.0
1	5.0	2.0	2.0	0.0	2.0	8.0	1.0	4.0	0.0	5.0	...	8.0	5.0	0.0	2.0	2.0	0.0	0.0	0.0	4.0	0.0
2	1.0	0.0	4.0	1.0	0.0	10.0	1.0	7.0	1.0	2.0	...	1.0	3.0	4.0	4.0	1.0	8.0	1.0	2.0	4.0	0.0
3	3.0	3.0	2.0	5.0	4.0	6.0	3.0	0.0	0.0	0.0	...	3.0	0.0	1.0	3.0	2.0	7.0	2.0	5.0	0.0	2.0
4	3.0	3.0	2.0	3.0	0.0	3.0	1.0	2.0	0.0	2.0	...	3.0	3.0	0.0	6.0	1.0	0.0	3.0	3.0	8.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1045	2.0	2.0	2.0	0.0	0.0	8.0	0.0	9.0	1.0	0.0	...	5.0	3.0	2.0	0.0	0.0	0.0	1.0	2.0	4.0	1.0
1046	2.0	0.0	1.0	0.0	1.0	2.0	0.0	4.0	0.0	5.0	...	9.0	3.0	0.0	1.0	0.0	0.0	3.0	0.0	2.0	1.0
1047	1.0	0.0	7.0	0.0	1.0	5.0	4.0	3.0	3.0	1.0	...	3.0	0.0	1.0	0.0	1.0	1.0	6.0	0.0	4.0	0.0
1048	1.0	2.0	4.0	0.0	1.0	1.0	4.0	3.0	0.0	0.0	...	1.0	2.0	3.0	1.0	9.0	3.0	2.0	2.0	1.0	2.0
1049	4.0	1.0	9.0	0.0	1.0	5.0	2.0	6.0	0.0	4.0	...	5.0	4.0	3.0	0.0	2.0	1.0	2.0	0.0	3.0	2.0

```
# Et voici à quoi ressemble le dictionnaire BovW
BovW
```

```
{200:       0    1    2    3    4    5    6    7    8    9    ...  190  191  192  \
0     2.0  1.0  0.0  4.0  2.0  2.0  4.0  1.0  0.0  1.0  ...  1.0  4.0  3.0
1     5.0  2.0  2.0  0.0  2.0  8.0  1.0  4.0  0.0  5.0  ...  8.0  5.0  0.0
2     1.0  0.0  4.0  1.0  0.0 10.0  1.0  7.0  1.0  2.0  ...  1.0  3.0  4.0
3     3.0  3.0  2.0  5.0  4.0  6.0  3.0  0.0  0.0  0.0  ...  3.0  0.0  1.0
4     3.0  3.0  2.0  3.0  0.0  3.0  1.0  2.0  0.0  2.0  ...  3.0  3.0  0.0
...   ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
1045  2.0  2.0  2.0  0.0  0.0  8.0  0.0  9.0  1.0  0.0  ...  5.0  3.0  2.0
1046  2.0  0.0  1.0  0.0  1.0  2.0  0.0  4.0  0.0  5.0  ...  9.0  3.0  0.0
1047  1.0  0.0  7.0  0.0  1.0  5.0  4.0  3.0  3.0  1.0  ...  3.0  0.0  1.0
1048  1.0  2.0  4.0  0.0  1.0  1.0  4.0  3.0  0.0  0.0  ...  1.0  2.0  3.0
1049  4.0  1.0  9.0  0.0  1.0  5.0  2.0  6.0  0.0  4.0  ...  5.0  4.0  3.0

      193  194  195  196  197  198  199
0     5.0  3.0  1.0  0.0  8.0  3.0  2.0
1     2.0  2.0  0.0  0.0  0.0  4.0  0.0
2     4.0  1.0  8.0  1.0  2.0  4.0  0.0
3     3.0  2.0  7.0  2.0  5.0  0.0  2.0
4     6.0  1.0  0.0  3.0  3.0  8.0  1.0
...   ...  ...  ...  ...  ...  ...  ...
1045  0.0  0.0  0.0  1.0  2.0  4.0  1.0
1046  1.0  0.0  0.0  3.0  0.0  2.0  1.0
1047  0.0  1.0  1.0  6.0  0.0  4.0  0.0
1048  1.0  9.0  3.0  2.0  2.0  1.0  2.0
1049  0.0  2.0  1.0  2.0  0.0  3.0  2.0

[1050 rows x 200 columns],
400:       0    1    2    3    4    5    6    7    8    9    ...  390  391  392  \
0     1.0  1.0  1.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
1     0.0  1.0  0.0  1.0  0.0  0.0  1.0  0.0  1.0  0.0  ...  1.0  2.0  0.0
2     0.0  0.0  0.0  2.0  1.0  1.0  0.0  2.0  0.0  1.0  ...  3.0  1.0  0.0
```

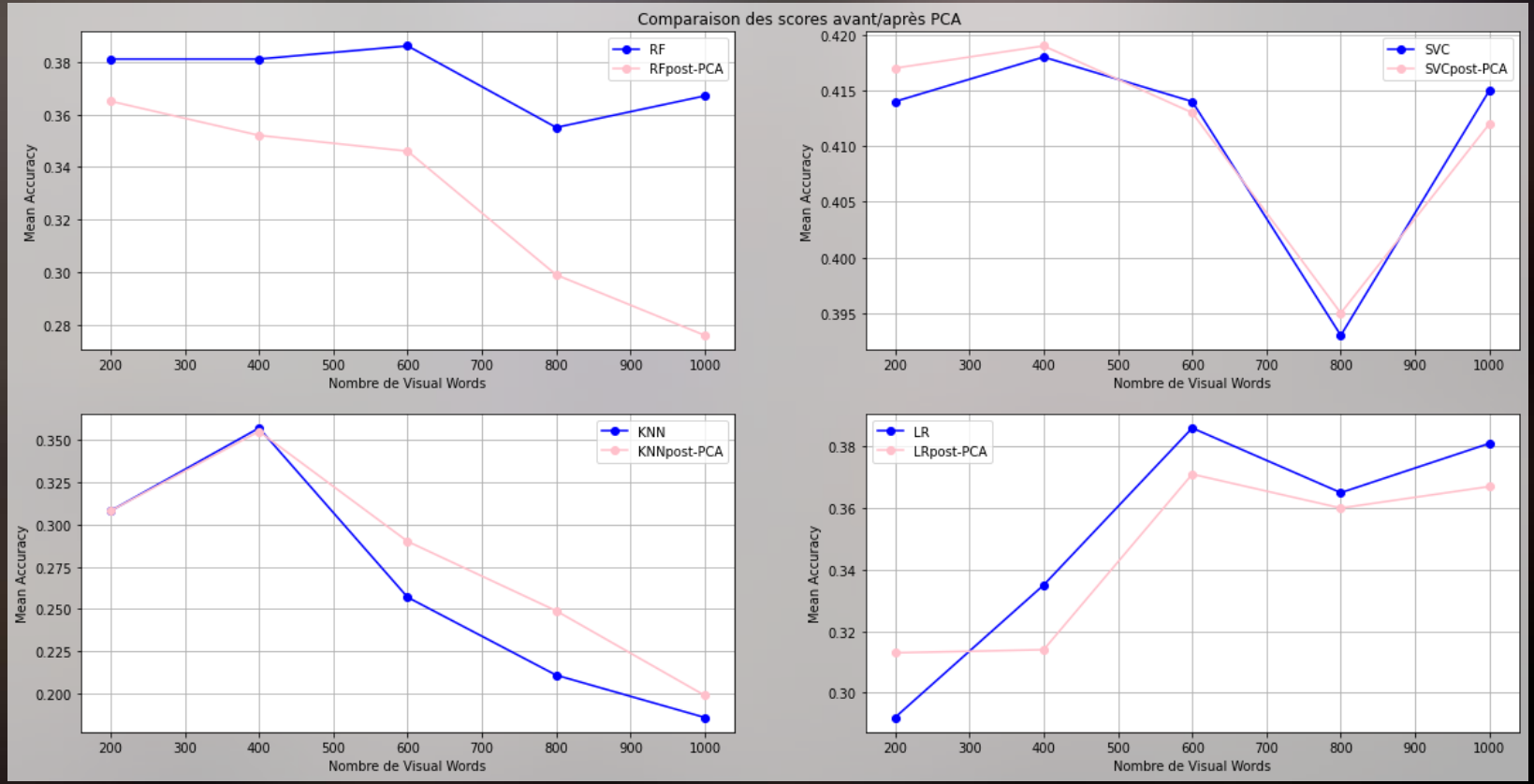
- Utilisation de MiniBatchKMeans permettant de ne pas saturer la mémoire vive
- On choisi plusieurs nombres de clusters : 200, 400, 600, 800, 1000
- Et on enregistre leurs histogrammes correspondant dans un dictionnaire nommé Bag of Visual Word *(BovW)*

ANALYSE VISUELLE

5. Classification supervisée du BoVW

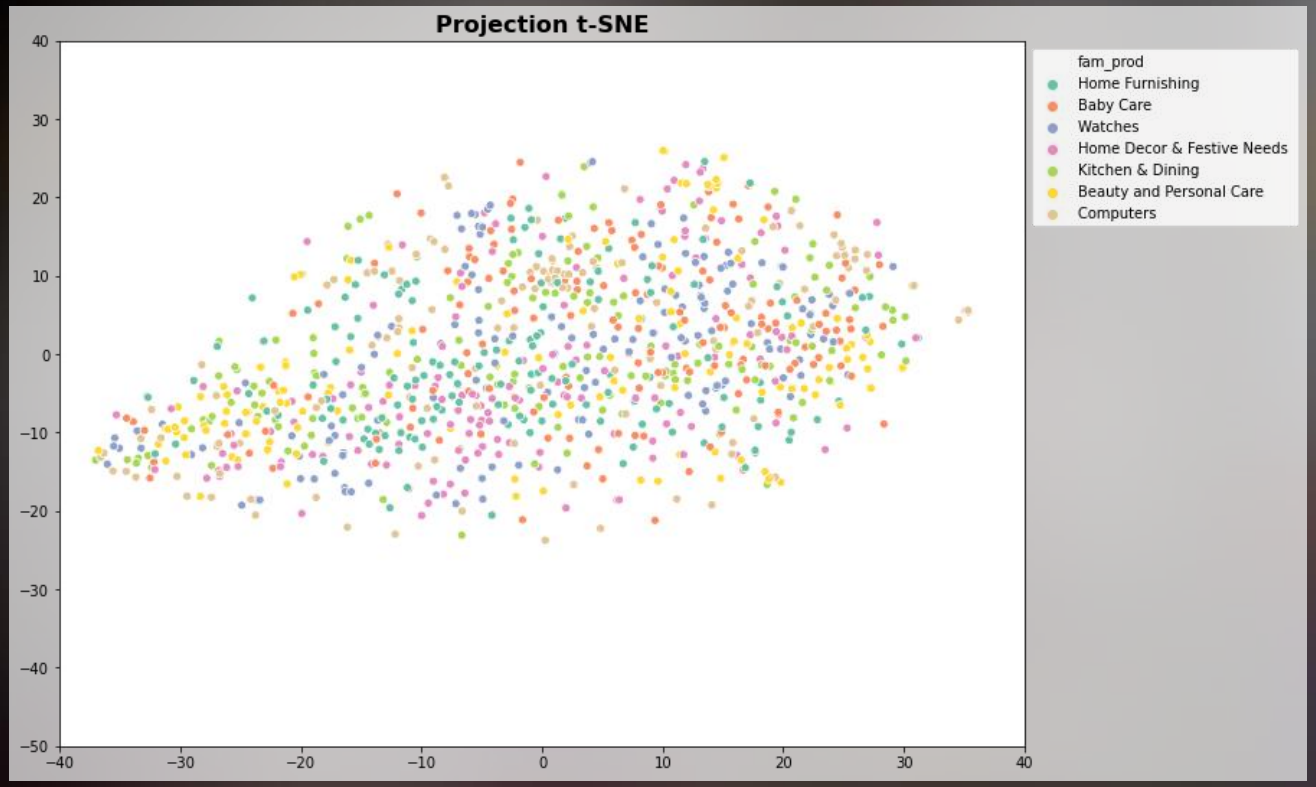
- On décide de conserver au moins 80% des composantes principales
- RF, KNN et LR atteignent un score maximal similaire
- SVC propose le meilleur score pour 400 mots par cluster
- Mais les scores restent faibles (moins de 50%)
- On remarque que la réduction de dimension par PCA n'affecte pas les scores
- Il serait par exemple possible de réduire la dimension pour faciliter les ressources

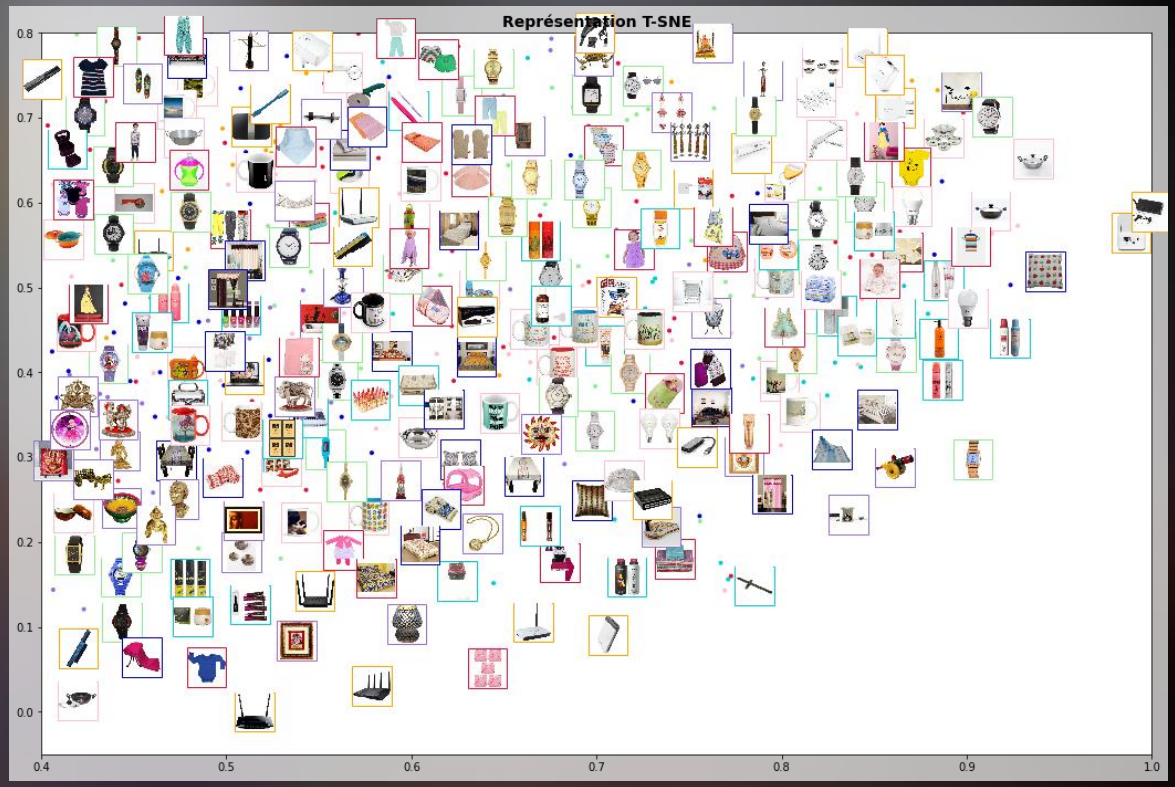
- Après avoir extrait, concaténé et sauvegardé les descripteurs de nos images, on entraîne des algorithmes pour classifier ces données en images similaires
- Calcul de l'accuracy moyenne par cross validation sur le train set pour différents modèles de classification (RF, SVC, KNN, LR)
- On calcule cet accuracy moyenne après PCA afin d'étudier l'impact sur les scores



ANALYSE VISUELLE

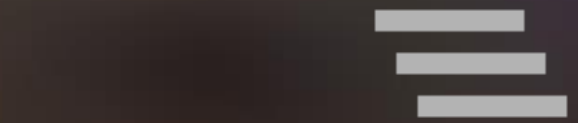
Observations de l'algorithme le plus performant (SVC – 400 mots)





- La matrice de confusion montre une meilleure performance sur les produits de beauté et les montres
- Pour 1050 images, la visualisation par projection t–SNE permet d'observer des clusters de 2 ou 3 produits similaires mélangés à d'autres clusters de manière très disparate
- Un réseau de neurones peut–il améliorer significativement cette répartition ?

TRANSFERT LEARNING : UTILISER UN CNN PRÉ–ENTRAINÉ POUR CLASSIFIER LES IMAGES



ANALYSE VISUELLE

Un réseau de neurones pré-entraîné peut-il améliorer notre classification de produits en fonction de leur image ?

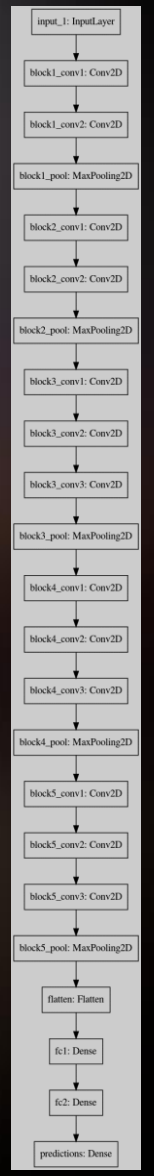
- **Proposition :** utiliser VGG-16 pré-entrainé pour la classification car l'extraction et la hiérarchisation des features s'adaptent au problème donné
- **Comment ? :**

1. Pre-processing des images

2. Entrainer le modèle pré-entrainé (via la base ImageNet) avec nos données images

3. Utilisation de VGG 16 pré-entrainé pour la classification

VGG
UNIVERSITY OF OXFORD



ANALYSE VISUELLE

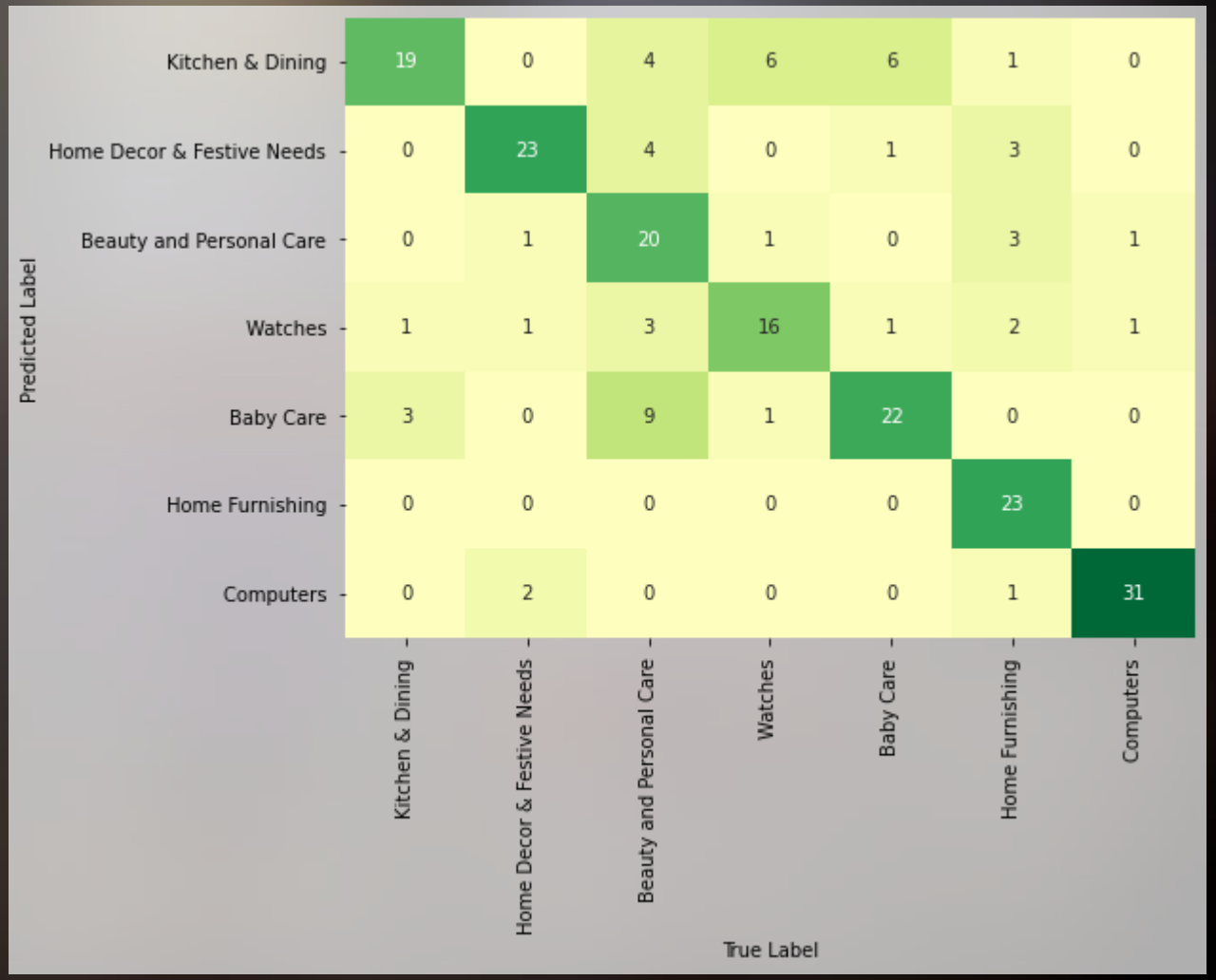
1. Pre-processing des images

2. Entrainer le modèle pré-entrainé (via la base ImageNet) avec nos données images)

```
from keras import optimizers
optimizer=optimizers.SGD(lr=0.0001, momentum=0.9)

from keras.layers import Dense, Dropout, Flatten #
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
for layer in base_model.layers:
    layer.trainable = False

# On récupère la sortie de ce réseau
x = base_model.output
x = Flatten()(x)

predictions = Dense(7, activation='softmax')(x)

# Avec ça on peut définir le nouveau modèle
from keras.models import Model
model = Model(inputs=base_model.input, outputs=predictions)

# Compilation
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])

# Affichage
model.summary()
```

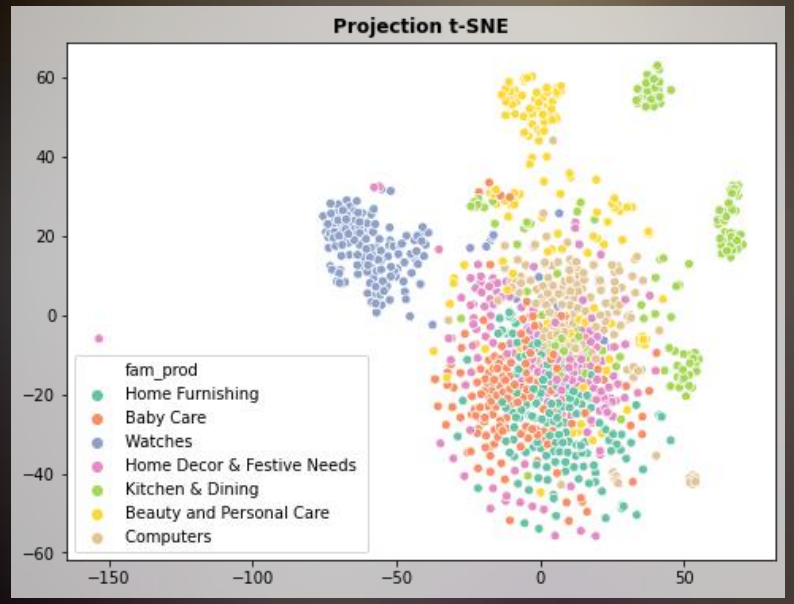
```
# On lance l'entraînement
model.fit(xtrain, ytrain, epochs=20, batch_size=135, validation_split=0.2, verbose=1, callbacks=[early_stopping])
```

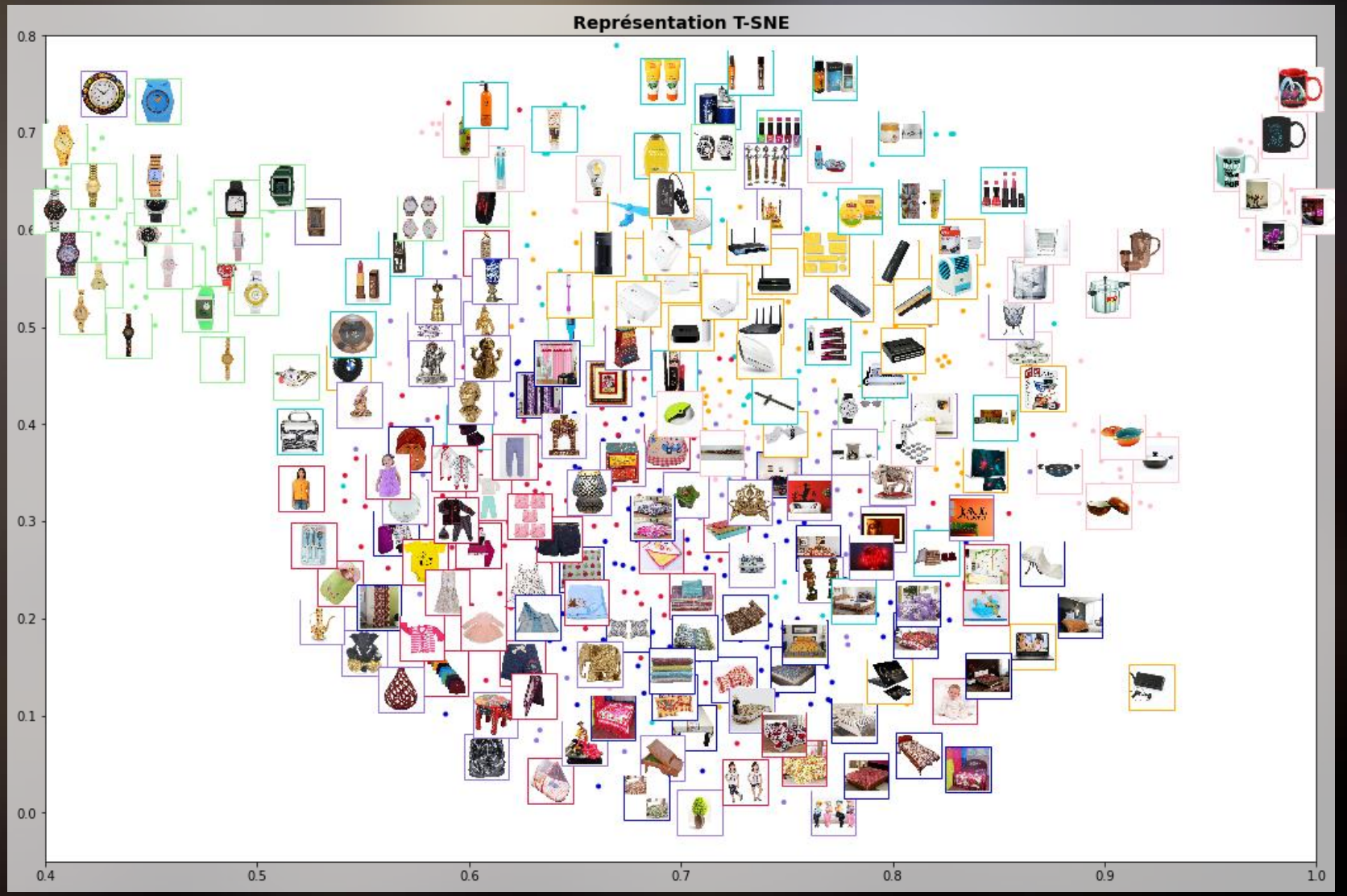
- On fige une partie des couches du modèle afin de ne pas les ré-entraîner (gain de temps de calcul), puis on y ajoute nos données d'entraînement
- La matrice de confusion fourni des données bien plus fiables et mieux réparties (zones contrastées = diagonales, zones claires = hors diagonales)



ANALYSE VISUELLE

3. Utilisation de VGG 16 pré-entrainé pour la classification





- On observe une visualisation avec des familles de produit beaucoup mieux réparties avec VGG16 qu'avec SVC-400 mots.
- Il se peut que l'algorithme SVC s'améliore avec un volume de données plus conséquent, mais pour 1050 images, l'utilisation d'un réseau de neurones pré-entraîné est recommandé.
- On observe que certains clusters (vêtements, linge de maison et meubles, informatique et cosmétique) sont très rapprochés sans être mélangés

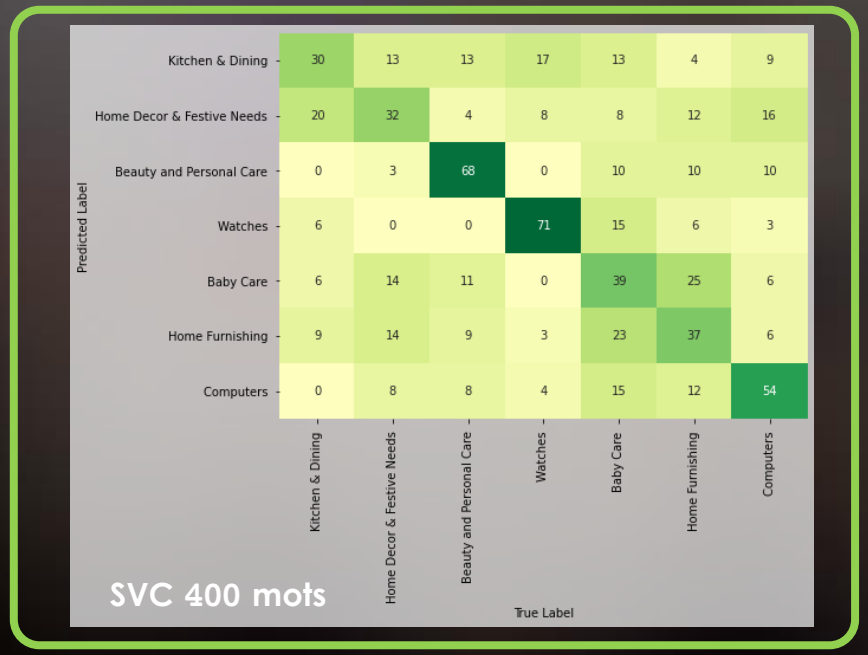
CONCLUSION ET PERSPECTIVES

CONCLUSION ET PERSPECTIVES

Il a été effectué :

- Nettoyage d'un jeu de données
- Analyse univariée de ces données (statistiques, distributions)
- Une tentative de classification de produits sur la base de leur description : résultat infructueux (clustering correct mais plusieurs thèmes par cluster)
- Une tentative de classification de produits sur la base de leur image : résultat médiocre
- Une tentative de classification de produits par Transfer Learning : résultat concluant

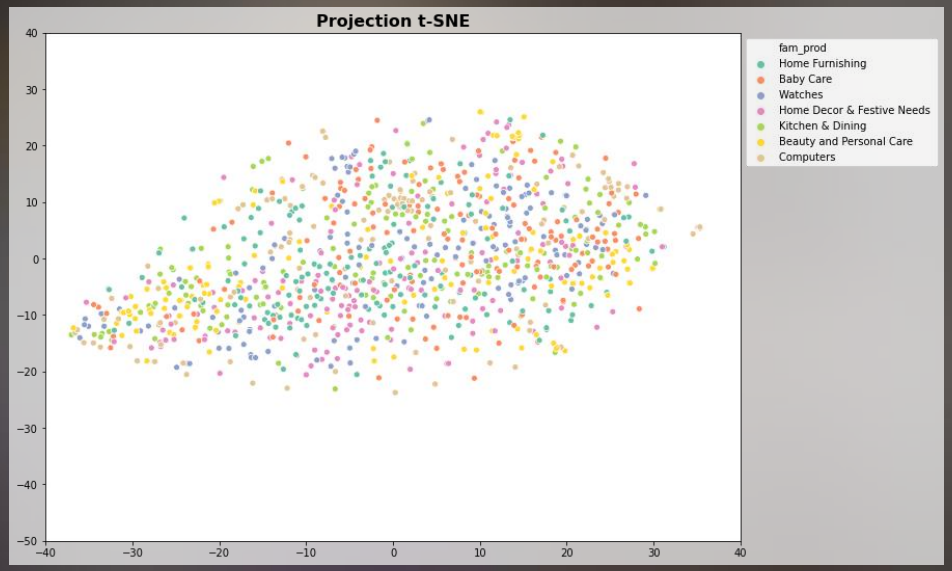
Quelques comparatifs :

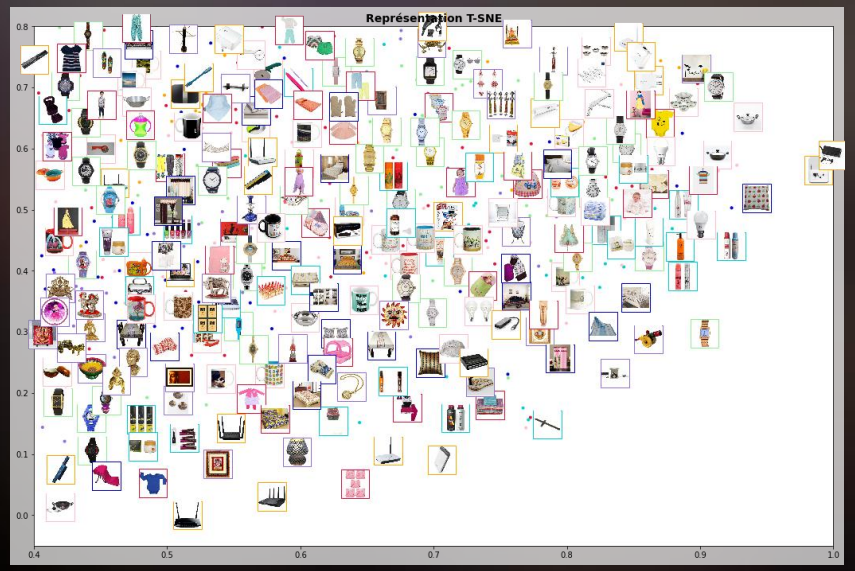


CONCLUSION ET PERSPECTIVES

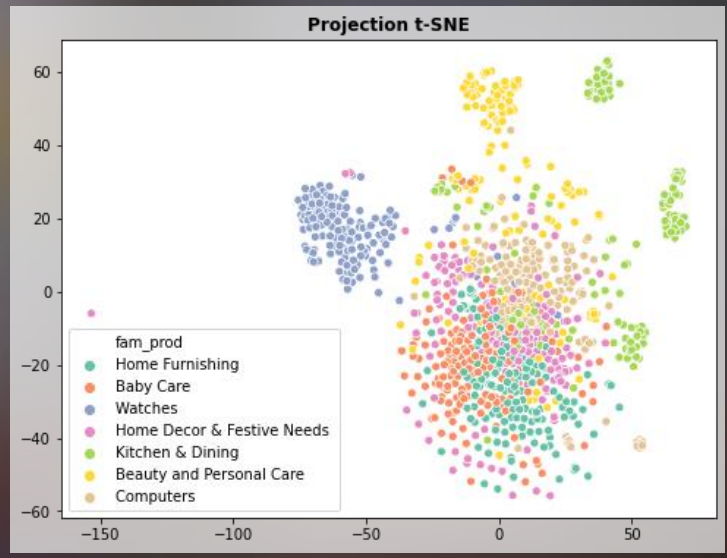
Quelques comparatifs :

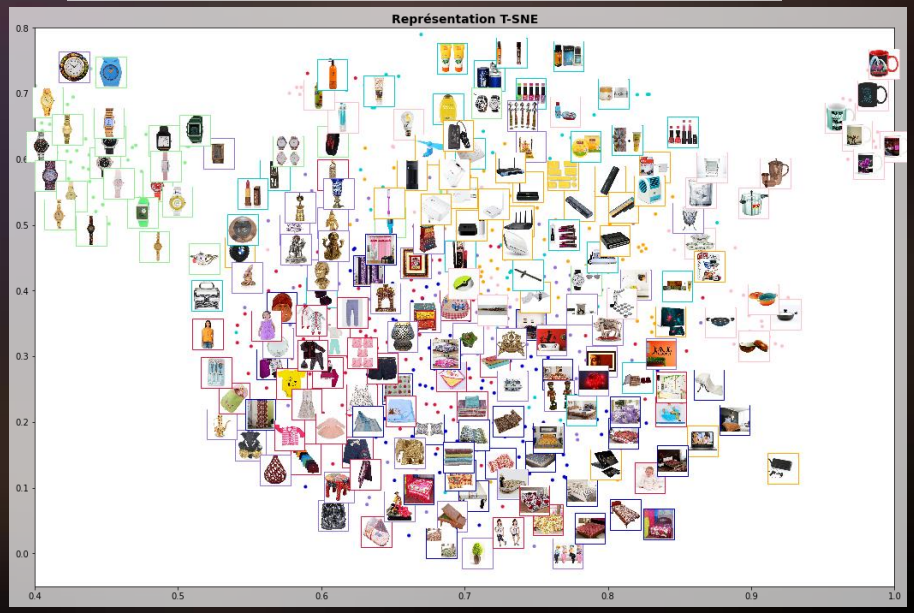
SVC 400 mots





VGG-16





CONCLUSION ET PERSPECTIVES

Perspectives pour place de marché:

- **Texte** :
  - Améliorer le dataset en volume et tenter de classer une nouvelle fois en fonction des descriptions
  - Utiliser un modèle autre que LDA pour les descriptions (NMF, *Nonnegative Matrix Factorization)*
- **Image** :
  - Etudier l'impact de la résolution d'image sur la performance de l'algorithme SVG–400 mots et via le Transfert Learning
  - Si l'impact est significatif, définir une charte de photographies (seuil de résolution) destinée aux vendeurs