Pipeline complet — OMS vs Forbes

Ce document contient **toutes** les étapes du projet avec l'objectif expliqué pour chaque partie, le code prêt à exécuter, et des vérifications/QC. Utilise-le comme guide interactif pour exécuter le pipeline pas-à-pas.

---

0) Préparation & dépendances

Objectif : installer les bibliothèques nécessaires et préparer l'environnement.

Pourquoi : certaines étapes (spaCy, SentenceTransformers, gensim) nécessitent des paquets spécifiques et des modèles linguistiques.

Commandes (terminal) :

```
pip install -r requirements.txt
python -m spacy download fr_core_news_sm
python -m spacy download en_core_web_sm
```

---

1) Chargement & fusion

Objectif : charger `oms.csv` et `forbes.csv`, vérifier les colonnes, concaténer et sauvegarder un backup `all_raw.csv`.

Pourquoi : garantir une copie brute immuable pour traçabilité.

Code (extrait) :

```
oms = pd.read_csv('data/raw/oms.csv')
forbes = pd.read_csv('data/raw/forbes.csv')
oms['source'] = 'OMS'
forbes['source'] = 'Forbes'
df = pd.concat([oms, forbes], ignore_index=True)
df.to_csv('data/raw/all_raw.csv', index=False)
```

---

2) Nettoyage HTML & boilerplate

Objectif : retirer balises HTML, `<script>`, `<style>`, URLs, emails et phrases de cookie/JS.

Pourquoi : le texte brut issu de scraping contient souvent des fragments non textuels qui polluent les tokens.

Code (extrait) :

```
from bs4 import BeautifulSoup
import re

def clean_html_and_boilerplate(text):
    soup = BeautifulSoup(str(text), 'html.parser')
    for s in soup(['script','style']): s.decompose()
    txt = soup.get_text(separator=' ')
    txt = re.sub(r'https?://\S+|www\.\S+|\S+@\S+', ' ', txt)
    txt = re.sub(r'\s+', ' ', txt).strip()
    txt = re.sub(r'javascript.*disabled.*', ' ', txt, flags=re.I)
    txt = re.sub(r'cookie.*', ' ', txt, flags=re.I)
    return txt

df['texte_html_clean'] =
df['texte'].astype(str).apply(clean_html_and_boilerplate)
```

---

3) Détection de la langue

Objectif : ajouter une colonne `lang` (fr/en) qui guidera le prétraitement.

Pourquoi : pour appliquer la lemmatisation/stopwords adaptés.

Code (extrait) :

```
from langdetect import detect

def safe_detect(s):
    try: return detect(s)
    except: return 'unknown'

df['lang'] = df['texte_html_clean'].apply(lambda s: safe_detect(s) if
s.strip() else 'unknown')
```

---

4) Création de deux textes (BERT vs TF-IDF)

Objectif : produire `texte_clean_bert` (nettoyage léger) et `texte_clean_tfidf` (agressif : lowercase, lemmatisation, stopwords).

Pourquoi : BERT a besoin de phrases intactes pour exploiter le contexte; TF-IDF/LDA gagnent à disposer de lemmes et stopwords supprimés.

Code (extrait) :

```
import spacy
nlp_fr = spacy.load('fr_core_news_sm', disable=['parser','ner'])
nlp_en = spacy.load('en_core_web_sm', disable=['parser','ner'])

CUSTOM_STOPWORDS =
set(['forbes','oms','com','www','http','https','article','publ','publi'])

def preprocess_for_tfidf(text, lang_hint='fr', min_tok=2):
    text = clean_html_and_boilerplate(text)
    text = text.lower()
    text = re.sub(r"[^a-z0-9àâäçéèêëîïôöùûüÿœæ'\s-]", ' ', text)
    nlp = nlp_fr if str(lang_hint).startswith('fr') else nlp_en
    doc = nlp(text)
    toks = []
    for t in doc:
        if t.is_stop or t.is_punct or t.is_space or t.like_num: continue
        lemma = t.lemma_.lower().strip()
        if len(lemma) < min_tok: continue
        if lemma in CUSTOM_STOPWORDS: continue
        toks.append(lemma)
    return ' '.join(toks)

# create two columns
ndf['texte_clean_bert'] = ndf['texte_html_clean']
ndf['texte_clean_tfidf'] = ndf.apply(lambda r:
preprocess_for_tfidf(r['texte_html_clean'], lang_hint=r['lang']), axis=1)
```

---

5) Diagnostics rapides

Objectif : lister tokens les plus fréquents avant/après pour détecter du bruit résiduel.

Pourquoi : ajuster la liste `CUSTOM_STOPWORDS` et la qualité du nettoyage.

Code (extrait) :

```
from collections import Counter
before = ' '.join(df['texte_html_clean'].astype(str).tolist()).split()
after = ' '.join(df['texte_clean_tfidf'].astype(str).tolist()).split()
Counter(before).most_common(30)
Counter(after).most_common(30)
```

---

6) Représentations : TF-IDF et Embeddings

Objectif : construire `X_tfidf` (TF-IDF) et `embeddings` (SentenceTransformers) pour analyses différentes.

Pourquoi : TF-IDF = baselines, features discriminantes ; Embeddings = sémantique, clustering fin.

Code (extrait) :

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sentence_transformers import SentenceTransformer

tfidf = TfidfVectorizer(max_features=15000, ngram_range=(1,2))
X_tfidf = tfidf.fit_transform(df['texte_clean_tfidf'].fillna(''))
model = SentenceTransformer('all-mpnet-base-v2')
embeddings = model.encode(df['texte_clean_bert'].tolist(),
show_progress_bar=True)
```

---

7) Topic modeling lexical (LDA/NMF)

Objectif : extraire topics lisibles via LDA (gensim) ou NMF (sklearn) sur la représentation TF-IDF / token lists.

Pourquoi : ces topics sont interprétables et utiles pour le reporting humain.

Code (extrait) :

```
import gensim
texts = [t.split() for t in df['texte_clean_tfidf']]
dictionary = gensim.corpora.Dictionary(texts)
corpus = [dictionary.doc2bow(t) for t in texts]
lda = gensim.models.LdaModel(corpus, id2word=dictionary, num_topics=8)
```

---

8) Clustering (KMeans, HDBSCAN)

Objectif : construire clusters lexicaux et sémantiques et comparer leur répartition par source.

Pourquoi : voir si OMS et Forbes forment des groupes séparés ou partagent des thèmes.

Code (extrait) :

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=8).fit(X_tfidf_reduced)
df['cluster_tfidf'] = kmeans.predict(X_tfidf_reduced)
# HDBSCAN on embeddings
import hdbscan
clusterer = hdbscan.HDBSCAN(min_cluster_size=5).fit(embeddings)
df['cluster_embed'] = clusterer.labels_
```

9) Visualisations (UMAP)

Objectif : projeter TF-IDF (après SVD) et embeddings en 2D pour inspection visuelle.

Pourquoi : très utile pour valider clustering et separation OMS/Forbes.

Code (extrait) :

```
import umap
reducer = umap.UMAP(n_neighbors=15, min_dist=0.1)
umap_tf = reducer.fit_transform(X_tfidf_reduced)
umap_emb = reducer.fit_transform(embeddings)
# plot with matplotlib
```

---

10) Keywords & résumés

Objectif : extraire mots-clés (KeyBERT) et générer résumés (BART/T5) pour chaque article.

Pourquoi : faciliter revue manuelle et reporting.

Code (extrait) :

```
from keybert import KeyBERT
kw = KeyBERT(model='all-mpnet-base-v2')
df['keywords'] = df['texte_clean_bert'].apply(lambda t:
kw.extract_keywords(t, top_n=5))
# summarization (chunk if long)
```

---

11) Sentiment & NER

Objectif : ajouter colonnes `sentiment` et `entities` (spaCy).

Pourquoi : comparer ton, polarité et acteurs cités entre OMS et Forbes.

Code (extrait) :

```
from transformers import pipeline
sent = pipeline('sentiment-analysis')
df['sentiment'] = df['texte_clean_bert'].apply(lambda t: sent(t[:512])[0])
# entities with spaCy
```

---

12) Diagnostics & QA

Objectif : échantillonnage manuel par topic, silhouette score, coherence score pour topics.

Pourquoi : garantir qualité et interprétabilité.

Checks : lire 5 docs par topic, silhouette score for clusters, gensim coherence for LDA.

---

13) Export & livrables

Objectif : sauvegarder `all_articles_processed.csv`, modèles `tfidf.joblib`, `embeddings.npy`, figures UMAP.

Pourquoi : produire artefacts réutilisables pour reporting et déploiement.

---

14) Packaging / Docker (optionnel)

Objectif : containeriser le pipeline pour exécution répétée.

Pourquoi : faciliter déploiement et exécution sur serveur.

Exemple : `Dockerfile` de base + `requirements.txt`.

---

15) Notes pratiques & recommandations

- Garde toujours deux colonnes : une pour TF-IDF (lemmes, stopwords enlevés), une pour BERT (phrases intactes).
- Ne fais pas de stemming agressif si tu veux des résultats lisibles humainement — privilégie la lemmatisation.
- Ajoute un `CUSTOM_STOPWORDS` et ré-évalue les top tokens après nettoyage.
- Pour les gros volumes (>10k docs) : encode les embeddings par batch et utilise GPU.

Fin du notebook (version canvas).