📰 AI-Powered News Classification & Indexing

```
import os
os.environ["OPENAI_API_KEY"] = "sk-proj-oSqbiX-zZAtEAsFS0PMsLBCnK8AaTBtaQ-lOOlvNqjy9MG1v2v19C8tI8TC7CFkyI1jLP3egMvT3Blbk
```

```
# Install Dependencies
!pip install --upgrade pip
!pip install pandas openpyxl chromadb langchain umap-learn hdbscan tiktoken -q
```

Show hidden output

1. Imports & Configuration

```
!pip install chromadb langchain-community -q
import os
import pandas as pd
from langchain.embeddings import OpenAIEmbeddings
from chromadb import Client
from chromadb.config import Settings
import umap
import hdbscan
```

2. Load & Prepare CSV

```
# 2.1 Load CSV
df = pd.read_csv('Aggregated News Dataset - Sheet1.csv')

# Combine title + summary/content into one field
df['text'] = (df['Title'].fillna('') + ' \n\n ' +
              df['news_summary'].fillna(''))

# 2.3 Keep only needed columns
# Check if 'id' column exists, if not, use a different unique identifier or create one
if 'id' not in df.columns:
  df['id'] = df.index

df = df[['id', 'Title', 'text']]
df
```

Show hidden output

Next steps: Explain error

3. Define Category Prototypes

```
CATEGORIES = {
    "sports": "Sports news about matches, athletes, teams and sporting events",
    "finance": "Financial markets, stocks, economy and business news",
    "politics": "Political news, elections, government policies",
    "lifestyle": "Lifestyle, health, travel, food and culture",
    "music": "Music artists, albums, concerts and industry news"
}
```

4. Spin Up ChromaDB – Categories Collection

```
import chromadb
from chromadb.config import Settings
from langchain.embeddings import OpenAIEmbeddings
from chromadb.utils import embedding_functions

# 4.1 Initialize Chroma with persistant storage
# PersistentClient will automatically use DuckDB with Parquet serialization
chroma_client = chromadb.PersistentClient(path="./chroma_db")

# 4.2 Create (or get) collection for categories,
#     using Chroma's own OpenAI wrapper
```

```
openai_ef = embedding_functions.OpenAIEmbeddingFunction(
    api_key=os.getenv("OPENAI_API_KEY"),
    model_name="text-embedding-ada-002"
)

cat_coll = chroma_client.get_or_create_collection(
    name="news_categories",
    embedding_function=openai_ef,
    metadata={"description": "Prototype embeddings for news categories"}
)

# 4.3 Upsert one “document” per category — Chroma will call OpenAIEmbeddingFunction internally
for cat_name, cat_prompt in CATEGORIES.items():
    cat_coll.upsert(
        ids=[cat_name],
        documents=[cat_prompt],
        metadatas=[{"category": cat_name}]
    )
# No need to call persist()—PersistentClient writes every upsert to disk automatically.
```

5. Classify Articles via Nearest-Neighbor

```
# 5.1 Helper to classify one text
!pip install tiktoken
import tiktoken
embeddings = OpenAIEmbeddings()
def classify(text: str):
    emb = embeddings.embed_query(text)
    results = cat_coll.query(query_embeddings=[emb], n_results=1)
    return results['ids'][0][0], results['distances'][0][0]

# 5.2 Run classification
df[['predicted_category', 'similarity']] = df['text'].apply(
    lambda t: pd.Series(classify(t))
)

# Inspect
df.head()

```

Show hidden output

Next steps: Generate code with df View recommended plots New interactive sheet

6. Article Clustering (for RAG-friendly groups)

```
# 6.1 Compute all article embeddings
embs = embeddings.embed_documents(df['text'].tolist())

# 6.2 UMAP reduction to 5D (speeds HDBSCAN)
umap_model = umap.UMAP(n_components=5, random_state=42)
reduced = umap_model.fit_transform(embs)
# --------------------------
# STEP 6: Deduplication & Prioritization
# --------------------------
# Clustering for deduplication
print("🔄 Detecting duplicates...")
# 6.3 HDBSCAN clustering
clusterer = hdbscan.HDBSCAN(min_cluster_size=15, prediction_data=True)
clusters = clusterer.fit_predict(reduced)
df['cluster'] = clusters

```

Show hidden output

7. Priority Tagging & Highlight Extraction

```
import re

# 5.1 Cluster sizes → proxy for frequency across sources
df['cluster_size'] = df.groupby('cluster')['id'].transform('count')

# 5.2 Compile per-category keyword patterns once
PRIORITY_KEYWORDS = {
    "sports": ["breaking", "championship", "cup final", "olympics", "transfer",
               "injury update", "record broken", "doping scandal", "last minute",
               "trophy", "victory", "defeat", "comeback"],
    "finance": ["alert", "breaking", "market crash", "rate hike", "earnings report",
                "recession", "inflation", "fed decision", "stock plunge", "merger",
                "bankruptcy", "crypto crash", "dividend", "short squeeze", "ipo"],
    "politics": ["election", "scandal", "resignation", "law passed", "protest",
                 "breaking", "impeachment", "summit", "sanctions", "diplomatic crisis",
                 "war", "treaty", "vote result", "corruption", "speech"],
    "lifestyle": ["viral", "trending", "recipe", "hack", "celebrity",
                  "wedding", "divorce", "pregnancy", "royal family", "makeover",
                  "controversial", "banned", "recall", "study finds"],
    "music": ["tour announced", "new album", "grammy", "controversy", "feud",
              "concert disaster", "chart-topping", "streaming record", "comeback",
              "breakup", "collaboration", "lyrics decoded", "canceled", "viral hit"]
}

# lower-case titles once
df['title_lc'] = df['Title'].str.lower()

# build regex dict
keyword_patterns = {
    cat: re.compile(r'\b(' + '|'.join(map(re.escape, kws)) + r')\b')
    for cat, kws in PRIORITY_KEYWORDS.items()
}

# 5.3 Flag priority articles
def _is_priority(row):
    pat = keyword_patterns.get(row['predicted_category'])
    return bool(pat.search(row['title_lc'])) if pat else False

df['is_priority'] = df.apply(_is_priority, axis=1)

# 5.4 Compute a single highlight_score
#    → boost keyword hits heavily, then cluster size
df['highlight_score'] = df['is_priority'].astype(int) * 1000 + df['cluster_size']

# 5.5 Extract top-5 highlights per category
highlights = (
    df.sort_values(['predicted_category','highlight_score'], ascending=[True, False])
      .groupby('predicted_category')
      .head(5)
      .reset_index(drop=True)
)

# 5.6 Preview or export to CSV
print(highlights[['predicted_category','Title','cluster_size','is_priority','highlight_score']])
highlights.to_csv('daily_highlights.csv', index=False)
```

Show hidden output

7. Index All Articles for Retrieval

```
# 7.1 Create/get an "articles" collection
art_coll = chroma_client.get_or_create_collection(
    name="news_articles",
    metadata={"description": "All articles indexed for retrieval"}
)

# 7.1.1 Ensure IDs are strings
if 'id' not in df.columns:
    df['id'] = df.index
df['id'] = df['id'].astype(str)

# 7.1.2 Prepare payload
docs = df['text'].tolist()
ids  = df['id'].tolist()
metas = df[['Title', 'predicted_category', 'cluster']].to_dict(orient='records')
# assume `embs` is your precomputed list of embeddings for df['text']

# 7.2 Upsert with retry and logging
from tenacity import retry, stop_after_attempt, wait_exponential, retry_if_exception_type
import logging

logger = logging.getLogger(__name__)
logging.basicConfig(level=logging.INFO)

@retry(
    reraise=True,
    stop=stop_after_attempt(3),
    wait=wait_exponential(multiplier=1, min=2, max=10),
    retry=retry_if_exception_type(Exception)
)
def _upsert_all(docs, embs, ids, metas):
    art_coll.upsert(
        documents=docs,
        embeddings=embs,
        ids=ids,
        metadatas=metas
    )

try:
    _upsert_all(docs, embs, ids, metas)
    logger.info(f"Successfully upserted {len(ids)} articles to 'news_articles'.")
except Exception as err:
    logger.error(f"Failed to upsert articles after retries: {err}")
    # fallback or raise, as needed

# No explicit persist()—PersistentClient writes every upsert to disk automatically.
```

8. Export Classified Articles Results

```
# 9.1 To CSV
df.to_csv('classified_articles.csv', index=False)
```