Data Ingestion & Profiling

Objective: Collect and preprocess diverse text data from "Human" and "AI" sources to augment the training set. This notebook handles web scraping, API fetching, and parsing personal data dumps.

Data Sources Processed:

- **Wikipedia (Human):** Scraped articles across STEM, History, and Culture topics.
- **arXiv (Human):** Abstract summaries from ML, Stats, and Physics papers.
- **News Feeds (Human):** RSS articles from BBC, Reuters, and AP.
- **ChatGPT History (AI):** Personal conversations exported (with consent) from OpenAI (Label 1).

Output:

- Generates `scraped_data_combined.csv` (saved to Drive), which will be merged with the larger Kaggle dataset in Notebook 2.

```
!pip -q install pandas numpy scikit-learn tqdm wikipedia feedparser beautifulsoup4 readability-lxml requests
```

```
  Preparing metadata (setup.py) ... done
  Preparing metadata (setup.py) ... done
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.5/81.5 kB 2.0 MB/s eta 0:00:00
  Building wheel for wikipedia (setup.py) ... done
  Building wheel for sgmllib3k (setup.py) ... done
```

```
import os, re, json, time, hashlib
import numpy as np
import pandas as pd
from tqdm.notebook import tqdm
from sklearn.model_selection import train_test_split

import requests
from bs4 import BeautifulSoup
from readability import Document
import feedparser
import wikipedia

tqdm.pandas()

RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)

TEXT_COL = "text"
LABEL_COL = "label"     # 0=human, 1=AI

def clean_text(t: str) -> str:
    if t is None:
        return ""
    t = re.sub(r"http\S+", " ", str(t))
    t = re.sub(r"\S+@\S+", " ", t)
    t = re.sub(r"\s+", " ", t).strip()
    return t
```

```
def stable_hash(s: str) -> str:
    return hashlib.sha1(s.encode("utf-8", errors="ignore")).hexdigest()

def chunk_text_words(text: str, chunk_words=200, overlap=40, min_words=40):
    words = re.findall(r"\S+", text)
    if len(words) < min_words:
        return []
    step = max(1, chunk_words - overlap)
    chunks = []
    for i in range(0, len(words), step):
        ch = " ".join(words[i:i+chunk_words]).strip()
        if len(ch.split()) >= min_words:
            chunks.append(ch)
    return chunks

def add_len_bins(df: pd.DataFrame):
    df["len_words"] = df[TEXT_COL].str.split().str.len()
    df["len_bin"] = pd.cut(df["len_words"], bins=[0,10,25,50,100,200,400,1000,10_000],
                           labels=False, include_lowest=True)
    return df

def dedup_by_text(df: pd.DataFrame):
    df["text_hash"] = df[TEXT_COL].map(stable_hash)
    df = df.drop_duplicates(subset=["text_hash"]).drop(columns=["text_hash"])
    return df
```

```
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

```
import pandas as pd
import re
```

```
CHATGPT_JSON = "/content/drive/MyDrive/conversations.json"  # upload this

with open(CHATGPT_JSON, "r", encoding="utf-8") as f:
    convos = json.load(f)

rows = []
for convo in convos:
    convo_id = convo.get("id", "unknown_convo")
    mapping = convo.get("mapping", {})
    for node in mapping.values():
        msg = node.get("message")
        if not msg:
            continue
        role = msg.get("author", {}).get("role")
        parts = (msg.get("content", {}) or {}).get("parts", []) or []
        if role == "assistant" and parts:
            text = clean_text(" ".join([p for p in parts if isinstance(p, str)]))
            if len(text.split()) >= 20:
                rows.append({
                    "doc_id": f"chatgpt_{convo_id}",
                "source": "chatgpt",
                "text": text,
                "label": 1
            })

chatgpt_df = pd.DataFrame(rows)
print("ChatGPT raw samples:", chatgpt_df.shape)
chatgpt_df.head()
```

```
ChatGPT raw samples: (4178, 4)
```

	doc_id	source	text	label
0	chatgpt_69351ea1-82b0-8325-8e60-8b957ba06590	chatgpt	Short answer: treat this like *two* binary cla...	1
1	chatgpt_69351ea1-82b0-8325-8e60-8b957ba06590	chatgpt	Text. 100%. Given your background and what you...	1
2	chatgpt_69351ea1-82b0-8325-8e60-8b957ba06590	chatgpt	Perfect — here is a **clean, actionable list**...	1
3	chatgpt_69351ea1-82b0-8325-8e60-8b957ba06590	chatgpt	Nice, that Kaggle dataset is a good choice. Le...	1
4	chatgpt_69351ea1-82b0-8325-8e60-8b957ba06590	chatgpt	Yes — you **already have a proper validation s...	1

Next steps: Generate code with chatgpt_df New interactive sheet

```
chat_chunks = []
for _, r in chatgpt_df.iterrows():
    for ch in chunk_text_words(r["text"], chunk_words=180, overlap=40, min_words=40):
        chat_chunks.append({"doc_id": r["doc_id"], "source": r["source"], "text": ch, "label": r["label"]})

chatgpt_df = pd.DataFrame(chat_chunks)
print("ChatGPT chunks:", chatgpt_df.shape)

#So that long outputs don't dominate
```

```
ChatGPT chunks: (12179, 4)
```

```
WIKI_TOPICS = [
    # --- Core AI / CS / Tech ---
    "Artificial intelligence",
    "Machine learning",
    "Deep learning",
    "Natural language processing",
    "Computer vision",
    "Algorithms",
    "Data science",
    "Cloud computing",
    "Cybersecurity",
    "Blockchain",

    # --- Physical & Life Sciences ---
    "Quantum mechanics",
    "Relativity",
    "Thermodynamics",
    "Biochemistry",
    "Genetics",
    "Molecular biology",
    "Neuroscience",
    "Climate change",
        "Evolution",
        "Epidemiology",

        # --- Mathematics & Logic ---
        "Calculus",
        "Linear algebra",
        "Probability theory",
        "Statistics",
        "Graph theory",
        "Number theory",
        "Mathematical logic",

        # --- Law, Policy & Society ---
        "Indian Contract Act",
        "Constitution of India",
        "International law",
        "Human rights",
        "Intellectual property",
        "Privacy law",
        "Cyber law",
        "Regulation of artificial intelligence",

        # --- Economics, Business & Finance ---
        "Macroeconomics",
        "Microeconomics",
        "Behavioral economics",
        "Game theory",
        "Financial markets",
        "Inflation",
        "Globalization",

        # --- History & Politics ---
        "World War I",
        "World War II",
        "Cold War",
        "French Revolution",
        "Indian independence movement",
        "United States presidential election",
        "Geopolitics",

        # --- Literature, Language & Arts ---
        "William Shakespeare",
        "English literature",
        "Poetry",
        "Drama",
        "Literary criticism",
        "Rhetoric",
        "Comparative literature",

        # --- Philosophy & Social Thought ---
        "Philosophy",
        "Ethics",
        "Epistemology",
        "Metaphysics",
        "Existentialism",
        "Philosophy of science",

        # --- Media, Sports & Culture ---
```

```
    "Formula One",
    "Olympic Games",
    "Association football",
    "Cricket",
    "Film theory",
    "Music theory",
    "Popular culture",

    # --- Current / General ---
    "Current events",
    "Global health",
    "Sustainability",
    "Energy transition",
    "Artificial general intelligence"
]


wiki_docs = []
for topic in WIKI_TOPICS:
    try:
        page = wikipedia.page(topic, auto_suggest=False)
        text = clean_text(page.content)
        wiki_docs.append({
            "doc_id": f"wiki_{page.pageid}",
            "source": "wikipedia",
            "text": text,
            "label": 0
        })
    except Exception as e:
        print("Skip wiki:", topic, "|", str(e)[:120])

wiki_docs = pd.DataFrame(wiki_docs)

wiki_chunks = []
for _, r in wiki_docs.iterrows():
    for ch in chunk_text_words(r["text"], chunk_words=220, overlap=50, min_words=60):
        wiki_chunks.append({"doc_id": r["doc_id"], "source": r["source"], "text": ch, "label": r["label"]})

wiki_df = pd.DataFrame(wiki_chunks)
print("Wikipedia chunks:", wiki_df.shape)
```

```
/usr/local/lib/python3.12/dist-packages/wikipedia/wikipedia.py:389: GuessedAtParserWarning: No parser was explicitly specified, so I'm using the best available

The code that caused this warning is on line 389 of the file /usr/local/lib/python3.12/dist-packages/wikipedia/wikipedia.py. To get rid of this warning, pass th

  lis = BeautifulSoup(html).find_all('li')
Skip wiki: Relativity | "Relativity" may refer to:
Galilean relativity
Numerical relativity
Principle of relativity
Theory of relativity
Genera
Wikipedia chunks: (3355, 4)
```

Start coding or generate with AI.

```
import time
import urllib.parse
import feedparser
import pandas as pd

# import re
# def clean_text(t: str) -> str:
#     t = re.sub(r"http\S+", " ", str(t))
#     t = re.sub(r"\S+@\S+", " ", t)
#     t = re.sub(r"\s+", " ", t).strip()
#     return t

ARXIV_QUERIES = [
    "cat:stat.ML",        # Stats ML
    "cat:math.ST",        # Statistics
    "cat:physics.optics", # Physics
    "cat:q-bio.BM"        # Bioinformatics
]

MAX_RESULTS_PER_QUERY = 200     # per category
ARXIV_START = 0
MIN_WORDS = 60                  # keep decent-length abstracts
SLEEP_SECONDS = 3               # be polite to arXiv

paper_rows = []
seen_ids = set()

for q in ARXIV_QUERIES:
    encoded_q = urllib.parse.quote(q)
    url = (
        "http://export.arxiv.org/api/query"
        f"?search_query={encoded_q}"
        f"&start={ARXIV_START}"
        f"&max_results={MAX_RESULTS_PER_QUERY}"
    )

    feed = feedparser.parse(url)
    print(f"Query={q} | fetched entries={len(feed.entries)}")

    for entry in feed.entries:
        arxiv_id = entry.id.split("/")[-1]
        if arxiv_id in seen_ids:
            continue
        seen_ids.add(arxiv_id)

        title = clean_text(getattr(entry, "title", ""))
        abstract = clean_text(getattr(entry, "summary", ""))

        text = f"{title}. {abstract}".strip()
        if len(text.split()) < MIN_WORDS:
            continue

        paper_rows.append({
            "doc_id": f"arxiv_{arxiv_id}",
            "source": "arxiv",
            "text": text,
            "label": 0
        })
    time.sleep(SLEEP_SECONDS)

papers_df = pd.DataFrame(paper_rows).drop_duplicates(subset=["doc_id"]).reset_index(drop=True)

print("\nDone")
print("Total arXiv samples:", len(papers_df))
print(papers_df.head(3))
```

```
Query=cat:stat.ML | fetched entries=200
Query=cat:math.ST | fetched entries=200
Query=cat:physics.optics | fetched entries=200
Query=cat:q-bio.BM | fetched entries=200

Done
Total arXiv samples: 780
               doc_id source  \
0  arxiv_2012.12056v1  arxiv
1  arxiv_2012.13115v1  arxiv
2  arxiv_2012.13190v2  arxiv

                                                text  label
0  Data Assimilation in the Latent Space of a Neu...      0
1  Upper Confidence Bounds for Combining Stochast...      0
2  QUACKIE: A NLP Classification Task With Ground...      0
```

```
paper_chunks = []
for _, r in papers_df.iterrows():
    for ch in chunk_text_words(r["text"], chunk_words=200, overlap=40, min_words=60):
        paper_chunks.append({"doc_id": r["doc_id"], "source": r["source"], "text": ch, "label": r["label"]})

papers_df = pd.DataFrame(paper_chunks)
print("arXiv chunks:", papers_df.shape)


#So long outputs don't dominate
```

```
arXiv chunks: (893, 4)
```

```
NEWS_RSS = [
    # --- BBC ---
    "https://feeds.bbci.co.uk/news/world/rss.xml",
    "https://feeds.bbci.co.uk/news/technology/rss.xml",
    "https://feeds.bbci.co.uk/news/business/rss.xml",
    "https://feeds.bbci.co.uk/news/science_and_environment/rss.xml",

    # --- NPR (US, high editorial quality) ---
    "https://www.npr.org/rss/rss.php?id=1001",   # News
    "https://www.npr.org/rss/rss.php?id=1019",   # Technology
    "https://www.npr.org/rss/rss.php?id=1007",   # Science
    "https://www.npr.org/rss/rss.php?id=1014",   # World

    # --- Reuters (excellent for neutral tone) ---
    "https://feeds.reuters.com/reuters/worldNews",
    "https://feeds.reuters.com/reuters/businessNews",
    "https://feeds.reuters.com/reuters/technologyNews",
    "https://feeds.reuters.com/reuters/scienceNews",

    # --- Associated Press ---
    "https://apnews.com/rss",
    "https://apnews.com/hub/technology/rss",
    "https://apnews.com/hub/science/rss",

    # --- The Guardian ---
    "https://www.theguardian.com/world/rss",
    "https://www.theguardian.com/technology/rss",
    "https://www.theguardian.com/science/rss",
    "https://www.theguardian.com/business/rss",

    # --- Financial / Economics ---
    "https://www.ft.com/rss/home",            # Financial Times (some paywall, still usable)
    "https://www.economist.com/rss"            # Economist (often truncated)
]


def fetch_article_text(url: str, timeout=10) -> str:
    try:
        r = requests.get(url, timeout=timeout, headers={"User-Agent":"Mozilla/5.0"})
        if r.status_code != 200:
            return ""
        doc = Document(r.text)
        html = doc.summary()
        soup = BeautifulSoup(html, "html.parser")
        text = clean_text(soup.get_text(" "))
        return text
    except Exception:
        return ""

news_rows = []
MAX_ARTICLES_PER_FEED = 50

for feed_url in NEWS_RSS:
    f = feedparser.parse(feed_url)
    for entry in f.entries[:MAX_ARTICLES_PER_FEED]:
        url = entry.get("link", "")
        title = clean_text(entry.get("title", ""))
        body = fetch_article_text(url)
        text = f"{title}. {body}".strip()
        if len(text.split()) >= 80:
            news_rows.append({
                "doc_id": f"news_{stable_hash(url)[:12]}",
                "source": "news",
                "text": text,
                "label": 0
            })
        time.sleep(0.2)  # be polite

news_df = pd.DataFrame(news_rows)
print("News samples:", news_df.shape)
```

```
News samples: (334, 4)
```

```
news_chunks = []
for _, r in news_df.iterrows():
    for ch in chunk_text_words(r["text"], chunk_words=220, overlap=50, min_words=80):
        news_chunks.append({"doc_id": r["doc_id"], "source": r["source"], "text": ch, "label": r["label"]})
```

```
news_df = pd.DataFrame(news_chunks)
print("News chunks:", news_df.shape)
```

```
News chunks: (1873, 4)
```

```
dfs_to_merge = []

if 'chatgpt_df' in locals(): dfs_to_merge.append(chatgpt_df)
if 'wiki_df' in locals(): dfs_to_merge.append(wiki_df)
if 'papers_df' in locals(): dfs_to_merge.append(papers_df)
if 'news_df' in locals(): dfs_to_merge.append(news_df)

if dfs_to_merge:
    scraped_combined = pd.concat(dfs_to_merge, ignore_index=True)
    scraped_combined["text"] = scraped_combined["text"].astype(str).str.strip()
    output_path = "/content/drive/MyDrive/scraped_data_combined.csv"
    print(f"Merging {len(dfs_to_merge)} data sources...")
    scraped_combined.to_csv(output_path, index=False)
    print("-" * 30)
    print(f"SUCCESS: Scraped data merged and saved to: {output_path}")
    print(f"Total Combined Shape: {scraped_combined.shape}")
    print("Source Distribution:")
    print(scraped_combined["source"].value_counts())
else:
    print("ERROR: No dataframes found to merge. Please run the scraping cells first.")
```

```
Merging 4 data sources...
------------------------------
SUCCESS: Scraped data merged and saved to: /content/drive/MyDrive/scraped_data_combined.csv
Total Combined Shape: (18300, 4)
Source Distribution:
source
chatgpt      12179
wikipedia     3355
news          1873
arxiv          893
Name: count, dtype: int64
```

```
#Final Completed
```