Project Plan: News Deduplication

1. Problem Statement Recap:

The problem is the presence of **duplicate** or **near-duplicate** news articles covering the same event across multiple sources. The goal is to **aggregate** those articles while maintaining all critical insights and perspectives. A major challenge is ensuring minimal loss of information while removing redundant content.

---

2. Approach:

To tackle this problem, we will use **RAG** (which is an advanced retrieval-augmented model) and integrate it with **App** (a Streamlit-based web application). Here's a step-by-step approach for the solution:

Step 1: Data Collection

- **Sources**: Collect news articles from multiple sources using **web scraping** (using tools like **BeautifulSoup** or **Scrapy**) or through **news APIs**.
- **Data Ingestion**:
    - For **PDF-based news**, use **RAG's ingestion functions**.
    - For **URLs**, ingest articles using **App's `ingest_urls` function** (from RAG), which will fetch and process content from URLs.

Step 2: Preprocessing and Deduplication:

- **Text Preprocessing**:
    - Clean the articles: Remove HTML tags, stop words, and normalize the text (lowercase, punctuation handling).
    - **Text Splitting**: Use **RecursiveCharacterTextSplitter** to split large articles into smaller chunks, making them more manageable for processing.
- **Embedding**:
    - Use the **embedding model from RAG** (`sentence-transformers/all-MiniLM-L6-v2`) to generate embeddings for the news articles.
- **Similarity Calculation**:
    - Calculate the similarity between articles using **Cosine Similarity** (using the embeddings generated). This allows detection of duplicate or similar articles.
- **Clustering**:
    - Cluster articles based on similarity using a clustering algorithm (e.g., **K-Means**). Articles in the same cluster are grouped together based on content similarity.

Step 3: Content Aggregation:

- **Aggregation Strategy**:
    - After clustering, select a **representative article** for each cluster (e.g., the one with the most comprehensive information or highest editorial quality).
    - Merge content from different articles in the cluster (summarize key points, remove redundant text) while keeping unique perspectives.
- **Highlight Extraction**:
    - Extract key highlights from each article using **summarization models**. This ensures that while articles are deduplicated, all important details and perspectives are retained.

Step 4: Retrieval-Augmented Generation (RAG):

- **Search and Retrieval**:
    - Use **FAISS vector store** from RAG to enable fast, similarity-based retrieval of relevant content (articles).
    - For each query or user input, retrieve the most relevant content based on the similarity scores, making use of **RAG's chain**.
- **Answer Generation**:
    - Once relevant content is retrieved, **RAG** can generate answers (or summaries) by using a **prompt-based generation model** (like **ChatOllama**), which will generate a cohesive summary of the articles related to a particular event.

Step 5: Content Presentation (Using App):

- **User Interface**:
    - Create a simple and intuitive **Streamlit** app (App) where users can:
        - Enter a query about a news topic.
        - View aggregated and deduplicated news articles related to that query.
- **App Workflow**:
    - **Sidebars** allow the user to navigate between pages (like "About", "PDF", "URL").
    - On the **"URL" page**, users can input URLs of news articles, which will be ingested and processed by RAG to provide deduplicated and aggregated content.
    - **PDFs** and **URLs** are processed using RAG's ingestion functions to build a unified vector store of articles, which is then queried for the relevant content.

Step 6: Continuous Learning and Feedback Loop:

- **Monitoring**: Track user engagement and satisfaction to continuously refine the deduplication algorithm. This includes:
    - Monitoring click-through rates for aggregated articles.

    - Collecting feedback on the quality of generated summaries and content representation.
- **Reinforcement Learning**: Based on user feedback, refine the article similarity measures and content clustering over time to enhance the quality of the deduplication process.

---

3. Technology Choices:

- **Programming Language**: Python (for its rich ecosystem of NLP, ML, and web frameworks).
- **Libraries**:
    - **BeautifulSoup / Scrapy**: For scraping news websites.
    - **NLP Models**: **Sentence-Transformers (BERT-based)** for generating article embeddings, **ChatOllama** for generating summaries.
    - **FAISS**: For efficient similarity search.
    - **Pandas**: For data processing and manipulation.
    - **Streamlit**: For creating the user interface (App).

---

4. Solution Design:

- **Data Flow**:
    1. News articles (from PDFs and URLs) are ingested.
    2. Articles are preprocessed, tokenized, and embedded.
    3. Similarity scores are calculated, and articles are grouped via clustering.
    4. Aggregated content (deduplicated articles with merged insights) is prepared.
    5. The user inputs a query in App.
    6. The RAG model retrieves relevant, deduplicated content and generates a response.
    7. The user is presented with the summarized news, free from redundancy but rich in unique insights.

---

5. Challenges and Solutions:

- **Challenge**: Identifying duplicate articles that have different structures.
    - **Solution**: Use deep semantic similarity models (like **BERT** embeddings) to detect content-level similarity, rather than relying solely on exact keyword matching.
- **Challenge**: Scaling to handle millions of articles efficiently.

    - **Solution**: Use **distributed systems** (e.g., **Dask** or **Apache Spark**) for parallel processing and leveraging **cloud-based storage** and computation (e.g., **AWS** or **Google Cloud**).
- **Challenge**: Preserving the diversity of perspectives while removing redundancy.
    - **Solution**: Use clustering to preserve multiple perspectives, and apply intelligent content merging techniques (e.g., combining key points from multiple articles).

---

6. Assumptions:

- News articles from various sources can be accessed via APIs or scraping.
- The articles may vary in terms of length, structure, and editorial quality.
- The system should focus on topic-level aggregation rather than individual articles.

---

7. Code/Workflow/POC (Proof of Concept):

Example of Summarization using RAG:

python
Copy code

```
from langchain.prompts import PromptTemplate
from langchain_community.chat_models import ChatOllama

# Define prompt template
prompt = PromptTemplate.from_template(
    """
    <s> [INST] You are an assistant for question-answering tasks. Use
the following pieces of retrieved context
    to answer the question. If you don't know the answer, just say
that you don't know. Use three sentences
    maximum and keep the answer concise. [/INST] </s>
    [INST] Question: {question}
    Context: {context}
    Answer: [/INST]
    """
)

# Initialize model
model = ChatOllama(model="mistral")
```

```
# Sample query and context
query = "What happened in the US election?"
context = "Content from multiple articles covering the same topic."

# Generate answer
answer = model.invoke({"question": query, "context": context})

print(answer)
```

This code snippet will help generate concise answers for the news query using aggregated content.

---

8. Additional Information:

- **Success Metrics**:
  - Reduction in content redundancy by a significant percentage (e.g., 90%).
  - Improved user engagement based on the quality of content served.
- **User Testing and Validation**:
  - Run A/B tests with users to validate that the deduplication system delivers higher quality and relevance.

This project plan outlines the **News Deduplication** system, integrating **RAG** and **App** for efficient deduplication, content aggregation, and summarization, ensuring a comprehensive and unique view of news events without redundancy.