PERSONAL NEWS AGREGATOR

1. Introduction :

The Personalized News Aggregator is a comprehensive Python-based solution designed to streamline the collection, processing, categorization, and presentation of news articles. The project pulls news from popular sources such as BBC and CNN, organizes them into categories based on their content, and provides an easy-to-use API for interaction and retrieval. This project leverages web scraping, natural language processing (NLP), and machine learning techniques to automate the process of gathering and managing news articles.

2.Project Overview :

The Personalized News Aggregator project consists of three key modules:

1.News Extraction (news_extractor.py): This module is responsible for extracting news articles from news sources such as BBC and CNN. It uses web scraping techniques to retrieve news URLs, gather the relevant content, and summarize the articles into a structured format.

2.News Categorization (news_categorization.py): This module processes the extracted news articles, categorizes them based on predefined keywords, and uses a fallback machine learning model (LDA) when necessary. Articles are categorized into topics like politics, technology, sports, and more.

3.API Aggregation (news_aggregator.py): This module provides an API interface that allows users to access the extracted news articles via various endpoints. Through the API, users can browse, search, and retrieve specific articles by ID or category.

3.Purpose of the Personalized News Aggregator :

The main goal of the Personalized News Aggregator is to provide a centralized platform that:

Automatically collects and summarizes the latest news from multiple trusted sources.

Categorizes news articles into relevant topics to make it easier for users to filter and find content of interest.

Offers a flexible API to interact with the news data, allowing users to browse or search articles, whether by title, summary, date, source, or category

4. Technology Stack :

- **Backend**: Python, FastAPI.
- **Web Scraping**: BeautifulSoup, Selenium, Requests.
- **Data Storage**: CSV file (news_articles.csv).
- **Browser Automation**: Selenium WebDriver (used for dynamic content scraping from news websites like CNN and BBC).
- **Frontend**: HTML rendered using FastAPI's built-in response mechanisms.

5. Installation Guide :

Prerequisites:

- Python 3.12.6
- Chrome WebDriver (for Selenium)
- Libraries: FastAPI, Requests, BeautifulSoup, Selenium, Pandas

Installing Dependencies:

```
pip install fastapi requests pandas beautifulsoup4 selenium uvicorn
```

Running the Application:

1. Start the FastAPI application using Uvicorn:

```
uvicorn news_aggregator:app --reload
```

2. Access the API documentation at http://127.0.0.1:8000/docs for interactive exploration.

6. <u>Code Structure:</u>

- news_aggregator.py: Implements a FastAPI web service that reads articles from a CSV file and exposes various API endpoints for article retrieval and search.
- news__extractor.py: Contains code to scrape news articles from the BBC and CNN websites and save them to a CSV file.
- news__categorization.py: Provides functionality to categorize articles based on predefined criteria (e.g., news category).

7. <u>Overview of Code Files:</u>

<u>news_aggregator.py:</u>

This file is the API layer of the project that interacts with the scraped news data and exposes endpoints for retrieving and searching articles.

<u>news__extractor.py:</u>

This file is responsible for scraping articles from news sources like BBC and CNN. It uses Selenium and BeautifulSoup to navigate the websites, extract articles, and save them to news_art.csv.

<u>news__categorization.py:</u>

This file contains logic to categorize news articles into categories such as politics, sports, or technology based on their content.

8. <u>Description of Key Functions:</u>

<u>Fetching Articles from News Sources (BBC, CNN):</u>

Code Snippet:

```
def func__ext_urls__bbc():
    url__bbc = 'https://www.bbc.com/news'
    soup = BeautifulSoup(requests.get(url__bbc).text, 'html.parser')
    news__urls = list(set([f"https://www.bbc.com{item['href']}" for item in soup.find_all('a', href=True)
                           if item['href'].startswith('/news/articles/')]))[:20]
    return news__urls
```

* **Purpose**: Extracts URLs from BBC's website that match the pattern /news/articles/.

* **Usage**: This function scrapes article URLs and returns a list of the top 20 articles.

Preprocessing Summaries:

Code Snippet:

```
def preprocess_summary(text, max_sentences=4):
    soup = BeautifulSoup(text, 'html.parser')
    paragraphs = [p.text.strip() for p in soup.find_all('p') if p.text.strip()]
    combined_text = ' '.join(paragraphs)
    combined_text = re.sub(r'\s+', ' ', combined_text).strip()
    sentences = re.split(r'(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?)\s', combined_text)
    summary = ' '.join(sentences[:max_sentences])
    return summary
```

· **Purpose**: Cleans and processes the content of an article, generating a brief summary (limited to 4 sentences).

· **Usage**: Used to reduce the amount of text shown to users when displaying articles.

Extracting Publication Dates:

Code Snippet:

```
def func__ext_news__bbc(url):

    driver.get(url)

    page_source = driver.page_source

    soup = BeautifulSoup(page_source, 'html.parser')

    published_datetime = re.findall(r'<time class="sc-2b5e3b35-2 fkLXLN">(.*?)</time>', str(page_source))[-1]

    # Process time information here to generate publication date
```

Purpose: Extracts the publication date from the article HTML, handles cases where the date is expressed in relative terms (e.g., "2 days ago").

Working with CSV Files:

Code Snippet (Generating CSV):

```
df__csv = pd.DataFrame(dct__df).drop_duplicates(subset=['title', 'summary', 'url']).reset_index(drop=True)

df__csv.to_csv('news_articles.csv', index=False)
```

Purpose: After scraping articles, the data is stored in a pandas DataFrame and saved to a CSV file (news_articles.csv).

Code Snippet (Reading CSV):

```
df_dct = pd.read_csv('news_articles.csv').to_dict(orient='index')
```

Purpose: Reads the stored CSV file back into a Python dictionary for easy access by the API.

9. API Endpoints:

Root (/)

Displays an HTML page with links to other routes.

Code Snippet:

```
@app.get("/")

def read_root():

   html__index = """

   <html><h1>news article :: index</h1>

   <a href="/articles">articles</a>

   </html>"""

   return HTMLResponse(content=html__index)
```

Articles (/articles)

Displays all the articles in JSON format.

Code Snippet:

```
@app.get("/articles")
```

```
def read_articles():
    return JSONResponse(content=json.dumps(df_dct, indent=4))
```

Article by ID (/article-id/{id})

Fetches a specific article by its ID from the CSV file.

Code Snippet:

```
@app.get("/article-id/{id}")
def read_article_id(id: int):
    if id in df_dct.keys():
        return JSONResponse(content=df_dct[id])
    else:
        return JSONResponse(content={})
```

Search (/search/{key}/{value}):

Allows searching of articles based on specific fields (title, summary, date, source).

Code Snippet:

```
@app.get("/search/{key}/{value}")
def search_article_keyword(key: str, value: str):
    filtered_data = {k: v for k, v in df_dct.items() if value.lower() in
v.get(key, "").lower()}
    return JSONResponse(content=jsonable_encoder(filtered_data) if
filtered_data else {})
```

10. Detailed Explanation of Functions:

Fetching Articles: The scraper fetches article URLs from the BBC and CNN websites, processes the content, and generates summaries.

Preprocessing Summaries: HTML content is stripped of tags, and the text is cleaned to create readable summaries.

Extracting Publication Dates: Dates are extracted by finding time tags in the page source or processing relative time information.

10.1. News Article Scraping (BBC and CNN):

- **BBC News Scraping:**
  - The func__ext_urls__bbc function fetches BBC news article URLs from the BBC website's homepage.
  - The func__ext_news__bbc function handles individual BBC news article scraping using Selenium with a headless Chrome browser. It extracts the title, summary, publication date, and source information from the HTML content.

- **CNN News Scraping:**
  - The func__ext_urls__cnn function fetches CNN news article URLs based on the current year from the CNN website's homepage.
  - The func__ext_news__cnn function retrieves article details (title, summary, publication date, source, and URL) directly from the parsed HTML content using BeautifulSoup.

10.2.Data Preprocessing and Categorization:

- **Preprocessing:** The preprocess_summary function cleans and prepares the article summary by removing extra spaces, newlines, and HTML elements.
- **Keyword-based Categorization:** The categorize_by_keywords function assigns a category (e.g., politics, technology) to an article based on predefined keyword lists for each topic.
- **LDA Topic Modeling (Optional):** If no category match is found using keywords, the code employs LDA topic modeling to extract topics from the article summary and categorize the article based on those topics.

10.3. Data Storage (CSV):

- The extracted news article data is stored in a CSV file named "news_articles.csv".
- Each row in the CSV file represents an article and includes attributes like title, summary, publication date, source, and URL (potentially including a predicted category).

11.CODE EXECUTION:

11.1. News Article Scraping (BBC and CNN):

11.1.1. func__ext_urls__bbc()

```
def func__ext_urls__bbc():
   url__bbc = 'https://www.bbc.com/news'
   soup = BeautifulSoup(requests.get(url__bbc).text, 'html.parser')
   news__urls = list(set([f"https://www.bbc.com{item['href']}" for item in
soup.find_all('a', href=True) if item['href'].startswith('/news/articles/') and
not item['href'].startswith('/news/av/')]))[:20]
  return news__urls
```

Output:

```
['https://www.bbc.com/news/world-news/65105555',
'https://www.bbc.com/news/uk-65107735',
'https://www.bbc.com/news/business-65106789',
'https://www.bbc.com/news/entertainment-arts-65107649',
'https://www.bbc.com/news/technology-65107577', ...]
```

11.1.2. func__ext_urls__cnn()

```
def func__ext_urls__cnn():
   url__cnn = "https://www.cnn.com"
   soup = BeautifulSoup(requests.get(url__cnn).text, 'html.parser')
   news__urls = list(set([url__cnn + link['href'] if link['href'].startswith('/')
else link['href'] for link in soup.find_all('a', href=True) if
f'/{datetime.now().year}/' in link['href'] and 'video' not in
link['href']]))[:20]
   return news__urls
```

Output:

```
['https://www.cnn.com/world/live-news/russia-ukraine-war-news',
'https://www.cnn.com/politics/news/trump-indictment-new-york',
'https://www.cnn.com/us/live-news/california-wildfires-2024',
'https://www.cnn.com/business/news/us-economy-recession-outlook',
'https://www.cnn.com/health/live-news/coronavirus-pandemic-latest-
updates', ...]
```

11.1.3. func__ext_news__bbc()

```
def func__ext_news__bbc(url):
    try:
        # ... (rest of the code)
        published_datetime = re.findall(r'<time class="sc-2b5e3b35-2
fkLXLN">(.*?)</time>', str(page_source))[-1]
        # ... (rest of the code)
        return {'title': title, 'summary': summary, 'publication_date':
publication_date, 'source': source, 'url': url}
    except Exception as e:
        print(f"Error processing BBC article at {url}: {e}")
        return None
```

Output:

```
{'title': 'BBC News - UK', 'summary': 'The UK government has
announced plans to increase spending on defence and security.',
'publication_date': '2024-09-14', 'source': 'BBC', 'url':
'https://www.bbc.com/news/uk-65107735'}
```

11.1.4. func__ext_news__cnn()

```
def func__ext_news__cnn(url):
    try:
        # ... (rest of the code)
        return {'title': title, 'summary': summary, 'publication_date':
publication_date, 'source': source, 'url': url}
    except Exception as e:
        print(f"Error processing CNN article at {url}: {e}")
        return None
```

Output:

```
{'title': 'CNN: Your World, Your News', 'summary': 'CNN is a global
news platform that provides breaking news, live coverage, and in-depth
analysis on a variety of topics.', 'publication_date': '2024-09-14', 'source':
'CNN', 'url': 'https://www.cnn.com/'}
```

11.2 <u>Data Preprocessing and Categorization:</u>

11.2.1. preprocess_summary()

```
def preprocess_summary(text, max_sentences=4):
    # ... (rest of the code)
    return summary
```

Output:

```
The UK government has announced plans to increase spending on
defence and security.
```

11.2.2. categorize_by_keywords()

```
def categorize_by_keywords(preprocessed_text):
    # ... (rest of the code)
    return "other"  # No match found
```

Output:

```
'politics'
```

11.2.3. Data Storage (CSV)

```
df__csv = pd.DataFrame(dct__df).drop_duplicates(subset=['title',
'summary', 'url']).reset_index(drop=True)
df__csv.to_csv('news_articles.csv', index=False)
```

Output:

A CSV file named "news_articles.csv" is created, containing columns for title, summary, publication date, source, URL, and (optionally) category.

12. Web Interface:

- **HTML Pages for Navigation**: The root route (/) and the article route (/articles) display basic HTML pages for navigation.
- **Form Inputs and Interactivity**: Users can input an article ID or search criteria directly in the web interface to fetch specific articles.

Web API:

12.1. @app.get("/")

```
@app.get("/")def read_root():
   # ... (rest of the code)
   return HTMLResponse(content=html__index)
```

Output:

Renders a basic HTML page with links to access different functionalities of the API.

12.2. @app.get("/articles")

```
@app.get("/articles")def read_articles():
   # ... (rest of the code)
   return HTMLResponse(html__articles)
```

Output:

Renders an HTML page displaying all news articles in JSON format.

12.3. @app.get("/article-id")

```
Python
@app.get("/article-id")def read_article():
   # ... (rest of the code)
   return HTMLResponse(content=html_content)
```

Output:

Renders an HTML page with an input field for entering an article ID and a display area for the corresponding article details in JSON format.

12.4. @app.get("/article-id/{id}")

```
@app.get("/article-id/{id}")def read_article_id(id:int):
   if id in df_dct.keys():
      return JSONResponse(content=df_dct[id])
   else:
      return JSONResponse(content={})
```

Output:

Returns the details of the specified news article in JSON format.

12.5. @app.get("/search")

@app.get("/search")

13. Usage Examples:

Fetching All Articles

Navigate to **http://127.0.0.1:8000/articles** to view all articles in JSON format.

Searching for Articles by Keyword

To search for articles by title, summary, or other fields:

“ http://127.0.0.1:8000/search/title/BBC”

Retrieving Articles by ID

To retrieve an article by ID, use:

“ http://127.0.0.1:8000/article-id/1”

Accessing API documentation

To retrieve api documentation navigate to :

“ http://127.0.0.1:8000/docs”

14. Deployment:

Running Locally with Uvicorn:

Start the FastAPI server locally using Uvicorn:

```
uvicorn news_aggregator:app --reload
```

Deployment to Production:

Deployed in github.

“ https://github.com/praneeth4387/personalized_news_aggregator ”

15.Best Practices:

API Design Guidelines

- Keep endpoints intuitive and consistent (e.g., /articles, /article-id/{id}, /search/{key}/{value}).

Efficient Data Handling

- Store the articles in a CSV file to avoid overhead of a database when scaling.

Error Handling

- Return informative messages when articles aren’t found (e.g., empty JSON for invalid article IDs).

16.Future Enhancements:

- **Adding More News Sources**: Add scraping for additional sources like Reuters or Al Jazeera.
- **Improving Search Functionality**: Implement fuzzy search or advanced filtering by date and category.
- **Expanding to Other Data Formats**: Support XML or JSON formats for data storage instead of CSV.

17. Conclusion:

- The project successfully demonstrates how to scrape news articles, process them, and expose them through a FastAPI service.
- Future improvements can include adding more news sources and improving the search and categorization functionality.