Information Retrieval and Web Analytics - IT3041

End-to-End Text Summarization and Analysis System

Group Details

Name-Mining Masters

Name with Initials		Registration Number
1.	Deemantha P.H.H.C	IT22560162
2.	Jayarathna J.V.D	IT22577610
3.	Thrimanna A	IT22585530
4.	Pallewela S. M	IT22594136

Contents

Abstract

Methodology

Acknowledgement

Introduction

System Design and Architecture

Implementation

Evaluation and Testing

Conclusion

Abstract

The goal of this project is to develop an End-to-End Text Summarization and Analysis System capable of processing large corpus of text (e.g., news articles, research papers) to automatically generate concise summaries. Additionally, the system will integrate advanced language processing techniques to perform sentiment analysis, keyword extraction, and topic modeling, providing a comprehensive overview and insights from the text.

Methodology

1. Data Ingestion: The system will first ingest a large corpus of text from various sources, preparing it for analysis.
2. Text Summarization: Using pretrained NLP (Natural Language Processing) model BART (Bidirectional and Auto-Regressive transformers), the system will create concise and coherent summaries of the text.
3. Sentiment Analysis: A sentiment analysis module will evaluate the emotional tone of the text, categorizing it as positive, negative, or neutral using the DistilBERT pretrain model.
4. Keyword Extraction: The system will identify the most relevant keywords from the text using KeyBERT pretrain model.
5. Topic Modeling: Using techniques such as Genism Latent Dirichlet Allocation (LDA), the system will extract the underlying topics discussed in the text corpus.
6. Integration and Visualization: The system will present the summarized content, sentiment insights, keywords, and topic clusters in an interactive and user-friendly dashboard.

This project aims to streamline the process of analyzing large amounts of text, saving time while delivering actionable insights.

Acknowledgements

We would like to express my sincere gratitude to Mr. Samadhi Rathnayake for his invaluable guidance, support, and encouragement throughout the course of this project. His expertise and insights have been instrumental in shaping the direction of this work. I deeply appreciate the time and effort he has devoted to mentoring me and providing the necessary resources to ensure the successful completion of this project.

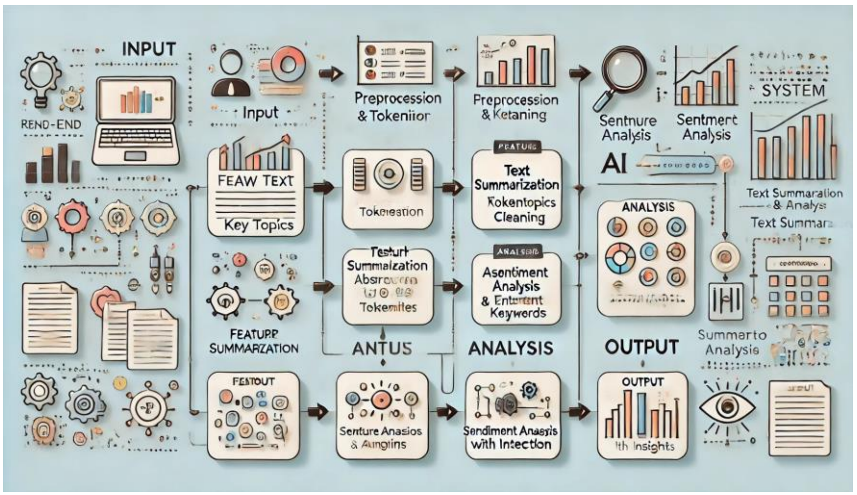
Introduction

The exponential growth of digital content, particularly in the form of news articles, research papers, and blogs, has made it increasingly challenging to process and extract relevant information manually. As a result, text summarization has become a vital tool in natural language processing (NLP), enabling users to condense large bodies of text into concise summaries without losing essential information. In this project, we utilize the "Newspaper Text Summarization (CNN/DailyMail)" dataset from Kaggle, which is widely used for training models to generate summaries from news articles. This dataset provides an ideal foundation for developing and testing an automated summarization system due to its size and relevance in real-world applications.

The primary objective of this project is to build an End-to-End Text Summarization and Analysis System capable of processing large amounts of text, generating concise summaries, and offering additional insights through sentiment analysis, keyword extraction, and topic modeling. The project is significant as it addresses the growing demand for automated tools that can simplify content consumption and analysis, especially for researchers, journalists, and data analysts. By reducing the time spent on reading large documents, this system will enhance productivity and improve decision-making processes.

The scope of this project includes the implementation of a text summarization model, coupled with sentiment analysis, keyword extraction, and topic modeling functionalities. This system is expected to handle large corpus of text, such as news articles and research papers, and provide users with an efficient way to extract critical information. However, the project has certain limitations, such as the dependency on the quality and structure of the input data. Furthermore, while the system aims to generalize across different text domains, its performance may vary depending on the type and complexity of the content.

Dataset Link: Newspaper Text Summarization (CNN/DailyMail) dataset.



System Design and Architecture

1.frontend

Framework: HTML, CSS, and JavaScript

Pages and Components: ***Home Page (index.html)* -** Allows users to enter text or upload a PDF file for analysis. Displays results including summary, keywords, topics, and sentiment.

Client-Side Interactions: script.js: Manages user interactions, like pasting text and triggering text analysis and file upload.

2.Backend

Framework: Flask (Python)

APIs: ***Analyze Text Endpoint (/analyze)****:* Receives text, processes it for summarization, keyword extraction, topic modeling, and sentiment analysis, then returns the results as JSON.

3. Data and Models

NLP Models: Pre-trained models from Hugging Face (distilbart-cnn-12-6 for summarization, distilbert-base-uncased-finetuned-sst-2-english for sentiment analysis, KeyBERT for keyword extraction and Gensim LDA for topic modeling).

Data Storage: Temporary file storage (. /uploads) for handling uploaded PDFs and storing summarized PDFs for download.

4. Deployment Considerations

Deploy: use Flask.

Dependencies: Managed via requirements.txt, ensuring that necessary packages like Flask, Transformers, and KeyBERT are installed.

5.System Flow

User Interaction: A user inputs text or uploads a PDF via the web interface.

Backend Processing: For text input, the text is processed and analyzed in real-time. For file upload, the PDF is processed, summarized, and a new PDF is generated for download.

Output Delivery: The summarized results, including keywords, topics, and sentiment, are displayed on the page. If a PDF was uploaded, a summarized PDF is made available for download.

Implementation

The backend will handle the ingestion of large corpus of text, processing through various natural language processing (NLP) modules, and responding to user requests such as summarization, sentiment analysis, keyword extraction, and topic modeling.

Backend Development

1.Backend Architecture

Natural Language Processing (NLP) Pipeline:

- **Text Preprocessing:** Tokenization, cleaning, and normalization of the text.
- **Summarization Engine:** NLP model (BART) that will summarize long text into concise summaries.
- **Sentiment Analysis Engine:** Identifying the sentiment (positive, negative, neutral) of the text using DistilBERT pre trained model.
- **Keyword Extraction Module:** Identifies key terms that summarize the content using KeyBERT model.
- **Topic Modeling:** Use technique of Gensim LDA (Latent Dirichlet Allocation).

2.Implemented Key Features

Indexing: Improves the speed and performance of data retrieval.

Text Summarization Algorithms: this pipeline typically uses the BART model, which is designed for *abstractive summarization.*

NLP Integration: *Natural Language Processing (NLP)* - This feature allows understanding, analyzing, and generating human language.

The system will rely on pretrained NLP models or libraries like:

- **Transformers** from Hugging Face for state-of-the-art models.
- **Spacy** and **NLTK** for tokenization, parsing, and basic NLP tasks.
- **Gensim LDA** for topic modeling.
- **BERT**, **DistilBERT** for sentiment analysis.
- **KeyBERT** for keyword extraction.

Sentiment Analysis: Use pretrained sentiment analysis models like BERT-based sentiment classifiers. Fine-tuning is needed for specific text types such as news.

Keyword Extraction: keyBERT model is used to extract the most important keywords from a document.

Topic Modeling: Gensim LDA (Latent Dirichlet Allocation) can cluster documents into topics, giving insights into the themes present in the corpus.

3.Implementation Technologies

Programming Languages: Python (due to extensive NLP library support).
Frameworks/Libraries:

- **Hugging Face Transformers** for summarization and sentiment analysis.
- **Gensim LDA** for topic modeling.
- **Spacy** or **NLTK** for preprocessing and named entity recognition (NER).
- **API Framework: Flask** for building the REST API.

4.Flow of Operations

- **Preprocessing:** Tokenization, sentence splitting, removing stopwords, stemming and lemmatization.
- **Summarization:** The text is passed to the summarization engine, which could use BART model.
- **Sentiment Analysis:** After summarization, by using DistilBERT the system analyzes the sentiment of the summary or the full text.
- **Keyword Extraction:** Extracts important keywords using KeyBERT pre trained model from the summary or the full text.
- **Topic Modeling:** Topics are identified for the corpus based on Genism LDA.
- **Retrieval:** The summary, keywords, sentiment, and topics are indexed for efficient retrieval in the future.

Frontend Development

Frontend development focuses on the **client side** of the application - the part that users interact with directly. It involves designing and implementing the **user interface (UI)**, handling the **user experience (UX)**, and ensuring that the frontend works seamlessly with the backend through API integration.

1.Design and Functionality of the User Interface (UI)

User Interface (UI) Overview: The UI should be designed with simplicity, usability, and clarity in mind, given that users will likely upload or input large amounts of text.

- **Responsive Design:** Interface is accessible on both desktop and mobile devices.
- **Accessibility**: Ensuring the design is inclusive, allowing users with disabilities to interact with the site.

- **Clean and Minimalist Layout:** The interface should focus on readability, minimizing clutter while emphasizing text content and results.
- **Sensitive Navigation:** Users should be able to quickly access different functionalities like uploading documents, viewing summaries, analyzing text without confusion.
- **Real-Time Interaction:** Provide users with a dynamic experience where operations such as text summarization and analysis happen quickly.

Frontend Architecture: The frontend will likely be a single-page application (SPA) for real-time interaction and responsiveness. It can be built using modern frontend frameworks like **HTML**, **CSS**, or **JS**. These frameworks ensure scalability, maintainability, and reactivity in the UI.

*2.*Key UI Features and User Experience (UX) Considerations

Simplicity: The interface should be easy to understand and navigate. Users should easily understand how to interact with the system. For instance, clickable items.

Text Input and Uploading: Users should have the ability to directly paste text (news articles) into a large text area as well as the upload pdf documents.

Summarization Results Display: After the user submits the text, the summary should be dynamically displayed in a visually distinct section.

Fast Load Times: Ensure that the interface loads quickly for all users.

Topic Modeling Display: Show the list identified topics.

Error Handling: Display friendly error messages when issues occur (e.g.- text too long).

File Upload Security: File types are restricted to PDF by validating file extensions

Export results: Allow users to export results (summarized text) in different format as PDF.

Responsiveness and Real-Time Interaction: UI must respond swiftly to user actions and provide immediate output to prevent user frustration.

Accessibility: Make the interface accessible to all users, including those with disabilities.

4.Technology Stack for Frontend Development

- **Frontend Framework:** HTML/CSS/JavaScript for creating a responsive, component-driven single-page application (SPA).
- **Pages and Components:** index.html (Home Page)
- **Deployment:** Flask

Key Features and Algorithms

Below is how the detailed explanation of the key algorithms and models implemented.

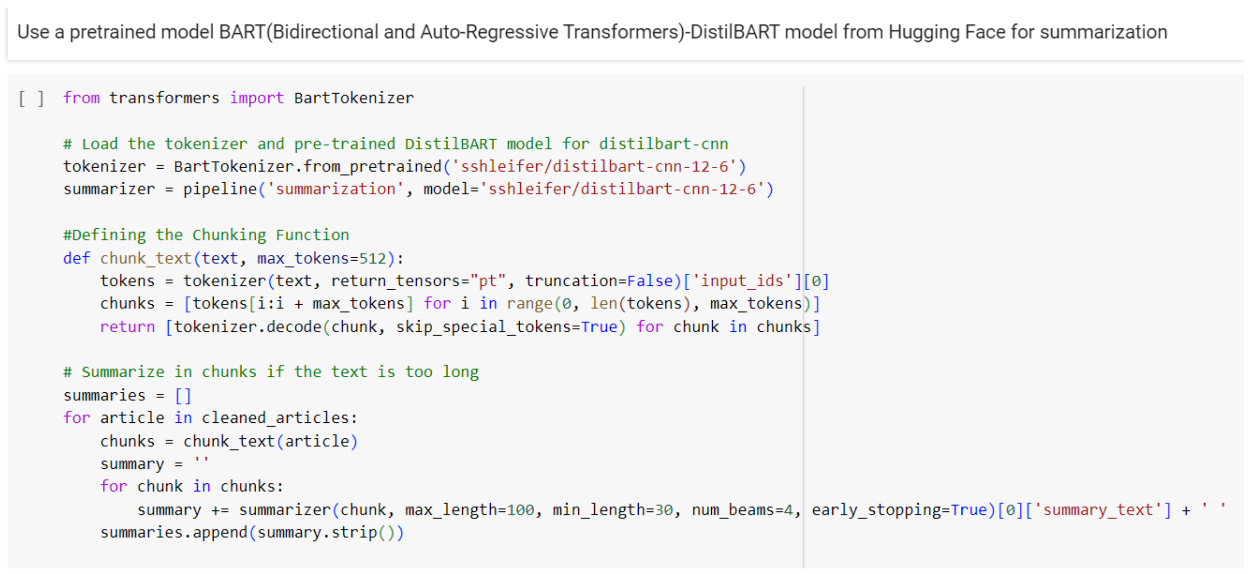
1. Text Preprocessing:

The preprocessing step involves:

- **Tokenization:** Breaking down the text into individual words or tokens.
- **Cleaning:** Removing references, URLs, non-alphabetic characters, and extra spaces.
- **Normalization:** Converting text to lowercase and removing stop words (commonly used words that do not contribute significant meaning).
- **Lemmatization and Stemming:** Using spaCy for lemmatization (converting words to their base form) and NLTK's Porter Stemmer for stemming (reducing words to their root form).

2. Summarization (BART Model):

The **Bidirectional and Auto-Regressive Transformers (BART)** model, specifically **DistilBART**, is used for abstractive text summarization. This model generates concise summaries by understanding the meaning of the entire input text. It is pretrained on summarization tasks and fine-tuned on specific datasets (e.g., CNN/Daily Mail). Beam search is used to optimize summary generation by considering multiple output paths to find the most likely one.

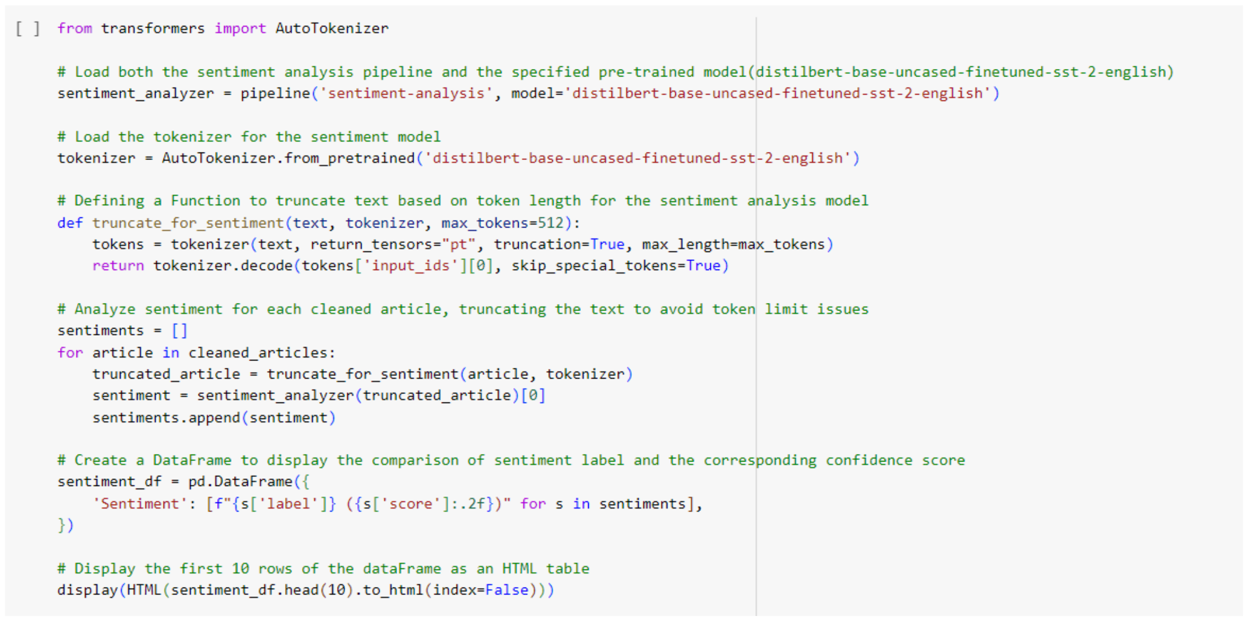

Use a pretrained model BART(Bidirectional and Auto-Regressive Transformers)-DistilBART model from Hugging Face for summarization

```
[ ]  from transformers import BartTokenizer

     # Load the tokenizer and pre-trained DistilBART model for distilbart-cnn
     tokenizer = BartTokenizer.from_pretrained('sshleifer/distilbart-cnn-12-6')
     summarizer = pipeline('summarization', model='sshleifer/distilbart-cnn-12-6')

     #Defining the Chunking Function
     def chunk_text(text, max_tokens=512):
         tokens = tokenizer(text, return_tensors="pt", truncation=False)['input_ids'][0]
         chunks = [tokens[i:i + max_tokens] for i in range(0, len(tokens), max_tokens)]
         return [tokenizer.decode(chunk, skip_special_tokens=True) for chunk in chunks]

     # Summarize in chunks if the text is too long
     summaries = []
     for article in cleaned_articles:
         chunks = chunk_text(article)
         summary = ''
         for chunk in chunks:
             summary += summarizer(chunk, max_length=100, min_length=30, num_beams=4, early_stopping=True)[0]['summary_text'] + ' '
         summaries.append(summary.strip())
```

3. Sentiment Analysis (DistilBERT Model):

For sentiment analysis, **DistilBERT**, a lighter version of BERT (Bidirectional Encoder Representations from Transformers), is employed. DistilBERT retains 97% of BERT's performance while being 60% faster. The sentiment analyzer classifies text into categories like positive, negative, or neutral based on patterns learned from a large corpus of labeled data.

arı Sentiment Analysis (Hugging Face) - using DistilBERT predefined model used

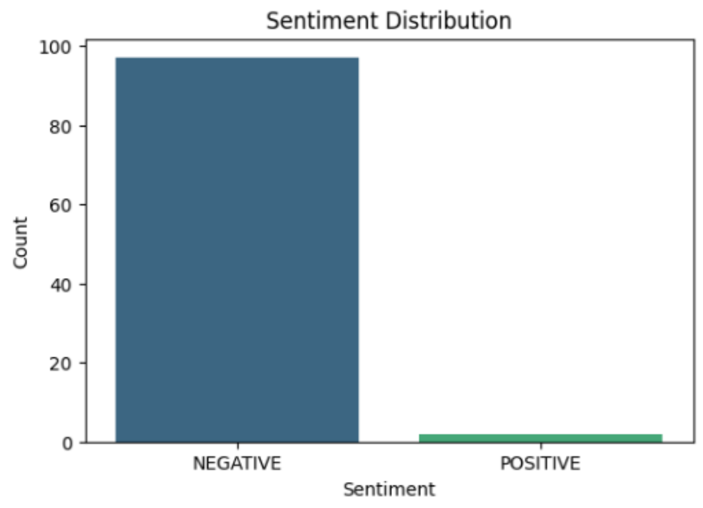


```
[ ] from transformers import AutoTokenizer

    # Load both the sentiment analysis pipeline and the specified pre-trained model(distilbert-base-uncased-finetuned-sst-2-english)
    sentiment_analyzer = pipeline('sentiment-analysis', model='distilbert-base-uncased-finetuned-sst-2-english')

    # Load the tokenizer for the sentiment model
    tokenizer = AutoTokenizer.from_pretrained('distilbert-base-uncased-finetuned-sst-2-english')

    # Defining a Function to truncate text based on token length for the sentiment analysis model
    def truncate_for_sentiment(text, tokenizer, max_tokens=512):
        tokens = tokenizer(text, return_tensors="pt", truncation=True, max_length=max_tokens)
        return tokenizer.decode(tokens['input_ids'][0], skip_special_tokens=True)

    # Analyze sentiment for each cleaned article, truncating the text to avoid token limit issues
    sentiments = []
    for article in cleaned_articles:
        truncated_article = truncate_for_sentiment(article, tokenizer)
        sentiment = sentiment_analyzer(truncated_article)[0]
        sentiments.append(sentiment)

    # Create a DataFrame to display the comparison of sentiment label and the corresponding confidence score
    sentiment_df = pd.DataFrame({
        'Sentiment': [f"{s['label']} ({s['score']:.2f})" for s in sentiments],
    })

    # Display the first 10 rows of the dataFrame as an HTML table
    display(HTML(sentiment_df.head(10).to_html(index=False)))
```

Visualization:

Sentiment Analysis Visualization

```
[ ] import matplotlib.pyplot as plt
    import seaborn as sns

    # Create a count plot for sentiment labels
    sentiment_labels = [s['label'] for s in sentiments]
    plt.figure(figsize=(6,4))
    sns.countplot(x=sentiment_labels, palette='viridis')
    plt.title('Sentiment Distribution')
    plt.xlabel('Sentiment')
    plt.ylabel('Count')
    plt.show()
```



4. Keyword Extraction (KeyBERT):

KeyBERT is used for extracting the most important keywords from a document. It leverages BERT embeddings to find the keywords by computing word similarities and identifying the ones most closely related to the document's content. This helps in summarizing the key topics and concepts of the text.

In additionally we can use RAKE also.

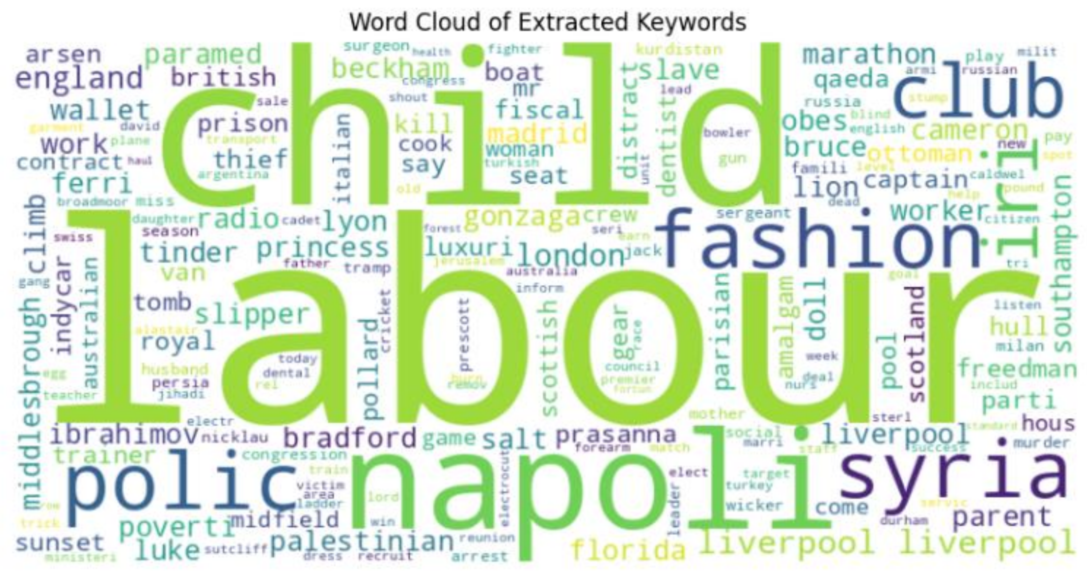
Keyword Extraction (using KeyBERT predefined model)

```
] # initializes an instance of the KeyBERT model
  kw_model = KeyBERT()

  # Extract keywords for each cleaned article(extracts the top 5 keywords from each)
  keywords = [kw_model.extract_keywords(article, keyphrase_ngram_range=(1, 2), stop_words='english', top_n=5)
              for article in cleaned_articles]

  # Create a DataFrame to display the keywords
  keywords_df = pd.DataFrame({
      'Keywords': ['; '.join([kw[0] for kw in kw_list]) for kw_list in keywords],
  })

  # Display the first 10 rows as an HTML table
  display(HTML(keywords_df.head(10).to_html(index=False)))
```

Visualization:

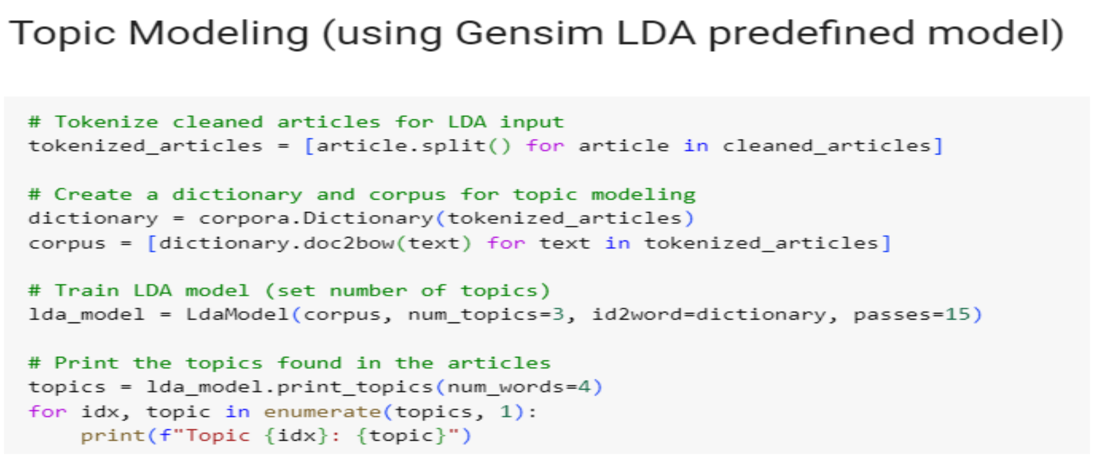
Here we used Word Clouds visualization for keyword extraction visualization

```
[ ]  from wordcloud import WordCloud

     # Combine all extracted keywords into one string
     all_keywords = ' '.join([kw[0] for kw_list in keywords for kw in kw_list])

     # Generate the word cloud
     wordcloud = WordCloud(width=800, height=400, background_color='white').generate(all_keywords)

     # Display the word cloud
     plt.figure(figsize=(10,5))
     plt.imshow(wordcloud, interpolation='bilinear')
     plt.axis('off')
     plt.title('Word Cloud of Extracted Keywords')
     plt.show()
```



5. Topic Modeling (Gensim LDA):

The **Latent Dirichlet Allocation (LDA)** algorithm, implemented using Gensim, is applied for topic modeling. It is a generative statistical model that clusters documents into multiple topics by identifying patterns of word co-occurrence. Each document is represented as a mixture of topics, and each topic is represented by a mixture of words. This technique is useful for discovering hidden themes in large text corpora.

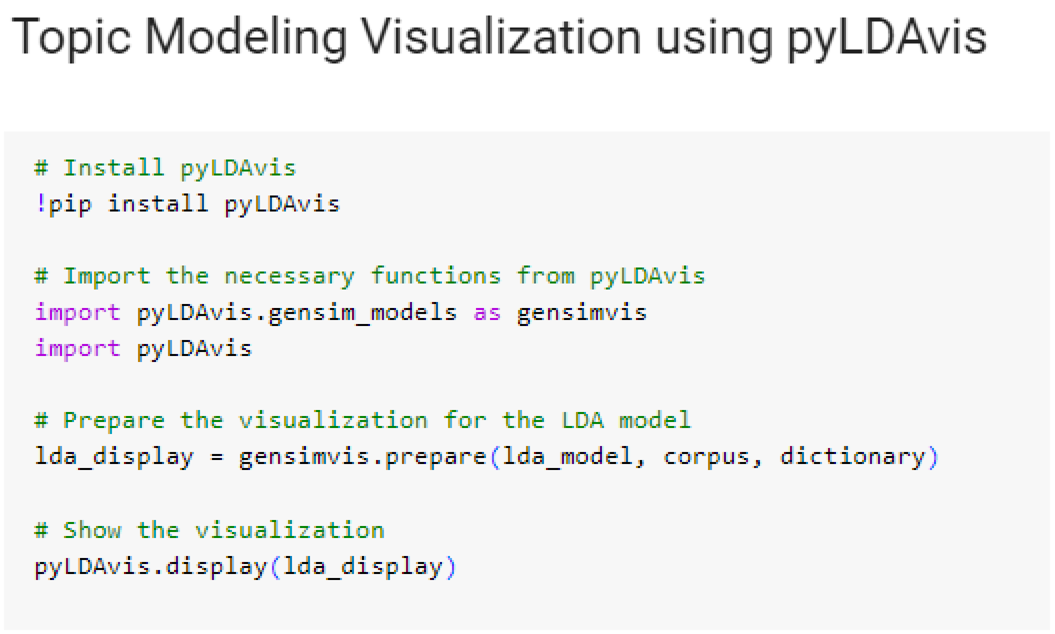


Topic Modeling (using Gensim LDA predefined model)

```python
# Tokenize cleaned articles for LDA input
tokenized_articles = [article.split() for article in cleaned_articles]

# Create a dictionary and corpus for topic modeling
dictionary = corpora.Dictionary(tokenized_articles)
corpus = [dictionary.doc2bow(text) for text in tokenized_articles]

# Train LDA model (set number of topics)
lda_model = LdaModel(corpus, num_topics=3, id2word=dictionary, passes=15)

# Print the topics found in the articles
topics = lda_model.print_topics(num_words=4)
for idx, topic in enumerate(topics, 1):
    print(f"Topic {idx}: {topic}")
```

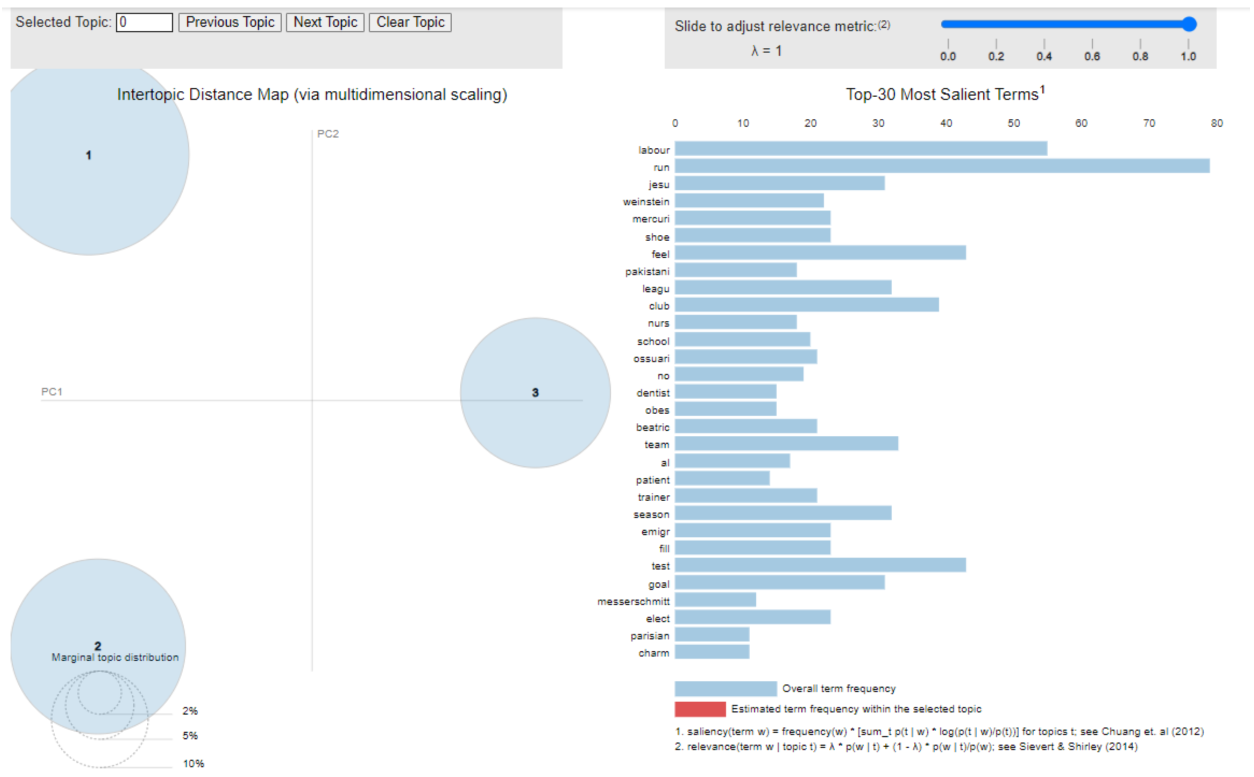
Visualization:

Here we used pyLDAvis for topic handling visualization



Topic Modeling Visualization using pyLDAvis

```python
# Install pyLDAvis
!pip install pyLDAvis

# Import the necessary functions from pyLDAvis
import pyLDAvis.gensim_models as gensimvis
import pyLDAvis

# Prepare the visualization for the LDA model
lda_display = gensimvis.prepare(lda_model, corpus, dictionary)

# Show the visualization
pyLDAvis.display(lda_display)
```


Selected Topic: 0
Previous Topic
Next Topic
Clear Topic
Slide to adjust relevance metric:(2)
λ = 1
0.0
0.2
0.4
0.6
0.8
1.0
Intertopic Distance Map (via multidimensional scaling)
PC2
PC1
1
2
3
Marginal topic distribution
2%
5%
10%
Top-30 Most Salient Terms1
0
10
20
30
40
50
60
70
80
labour
run
jesu
weinstein
mercuri
shoe
feel
pakistani
leagu
club
nurs
school
ossuari
no
dentist
obes
beatric
team
al
patient
trainer
season
emigr
fill
test
goal
messerschmitt
elect
parisian
charm
Overall term frequency
Estimated term frequency within the selected topic
1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

6. NLP Tools and Libraries:

- **spaCy**: Used for tokenization, lemmatization, and other preprocessing tasks.
- **NLTK**: Supports stopword removal and stemming.
- **Hugging Face Transformers**: Provides state-of-the-art NLP models like BART and DistilBERT for summarization and sentiment analysis.
- **Gensim**: Implements the LDA model for topic modeling.
- **KeyBERT**: For extracting keywords from the text.

Evaluation and Testing

ROUGE Score:

The **ROUGE (Recall-Oriented Understudy for Gisting Evaluation)** metric evaluates the quality of summaries by comparing the generated summaries to reference summaries (highlights). ROUGE-1, ROUGE-2, and ROUGE-L scores measure the overlap of unigrams, bigrams, and longest common subsequences between the two sets of summaries. These scores help assess how well the summarization model performs.

The following ROUGE variants are typically employed to assess the quality of a generated summary:

1. **ROUGE-1 (Unigram Overlap):**
   - **Purpose**: ROUGE-1 calculates the overlap of single words (unigrams) between the generated summary and the reference summary.
   - **Usage**: It gives a sense of how much content (individual words) from the reference summary has been captured by the generated summary.
   - **Typical Range**: 0.2 to 0.4 for well-performing systems.

2. **ROUGE-2 (Bigram Overlap):**
   - **Purpose**: ROUGE-2 measures the overlap of two-word sequences (bigrams) between the generated and reference summaries.
   - **Usage**: This helps evaluate how well the system captures relationships between adjacent words and reflects the fluency and coherence of the summary.
   - **Typical Range**: Usually lower than ROUGE-1, around 0.1 to 0.2.

3. **ROUGE-L (Longest Common Subsequence - LCS):**
   - **Purpose**: ROUGE-L focuses on the longest common subsequence (LCS) between the generated and reference summaries, which provides insight into the syntactic and sequential alignment.
   - **Usage**: This metric assesses the structure and order of words in the summary, making sure it follows a similar pattern to the reference text.
   - **Typical Range**: Around 0.2 to 0.4.

How ROUGE is Calculated:

- **Precision**: Measures how many of the words/phrases in the generated summary are also present in the reference summary.

$$Precision = \frac{\text{Overlap Count}}{\text{Total Count in Generated Summary}}$$

- **Recall**: Measures how many of the words/phrases in the reference summary are present in the generated summary.

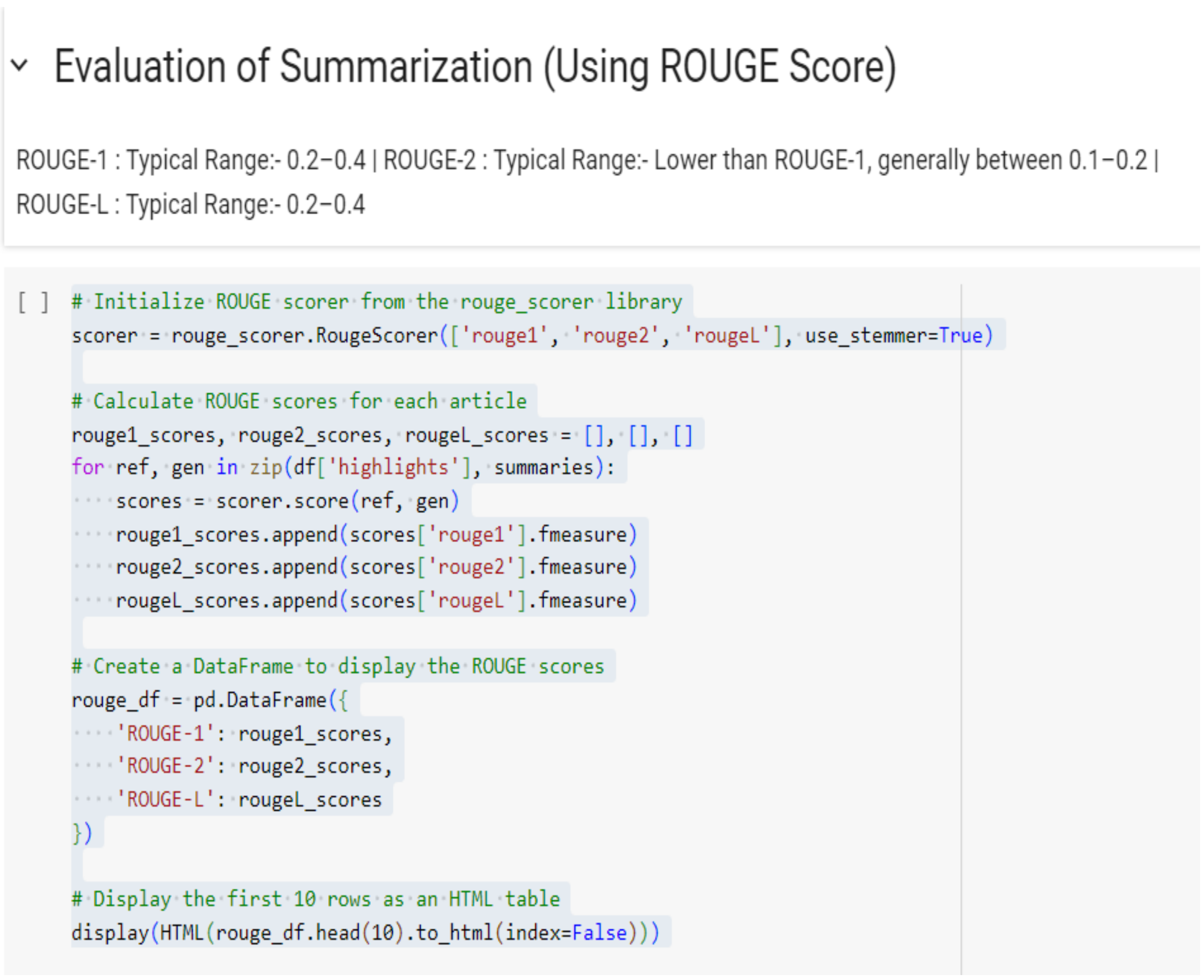
$$Recall = \frac{\text{Overlap Count}}{\text{Total Count in Reference Summary}}$$

- **F1-Score**: A balanced measure of both precision and recall.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Higher ROUGE scores indicate better summary quality, as more words and phrases from the reference summaries are captured in the generated summaries.

The results are typically visualized in boxplots or other graphical representations to understand the distribution and spread of ROUGE scores across multiple generated summaries.



Evaluation of Summarization (Using ROUGE Score)

ROUGE-1 : Typical Range:- 0.2–0.4 | ROUGE-2 : Typical Range:- Lower than ROUGE-1, generally between 0.1–0.2 | ROUGE-L : Typical Range:- 0.2–0.4

```
[ ]  # Initialize ROUGE scorer from the rouge_scorer library
     scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)

     # Calculate ROUGE scores for each article
     rouge1_scores, rouge2_scores, rougeL_scores = [], [], []
     for ref, gen in zip(df['highlights'], summaries):
         scores = scorer.score(ref, gen)
         rouge1_scores.append(scores['rouge1'].fmeasure)
         rouge2_scores.append(scores['rouge2'].fmeasure)
         rougeL_scores.append(scores['rougeL'].fmeasure)

     # Create a DataFrame to display the ROUGE scores
     rouge_df = pd.DataFrame({
         'ROUGE-1': rouge1_scores,
         'ROUGE-2': rouge2_scores,
         'ROUGE-L': rougeL_scores
     })

     # Display the first 10 rows as an HTML table
     display(HTML(rouge_df.head(10).to_html(index=False)))
```

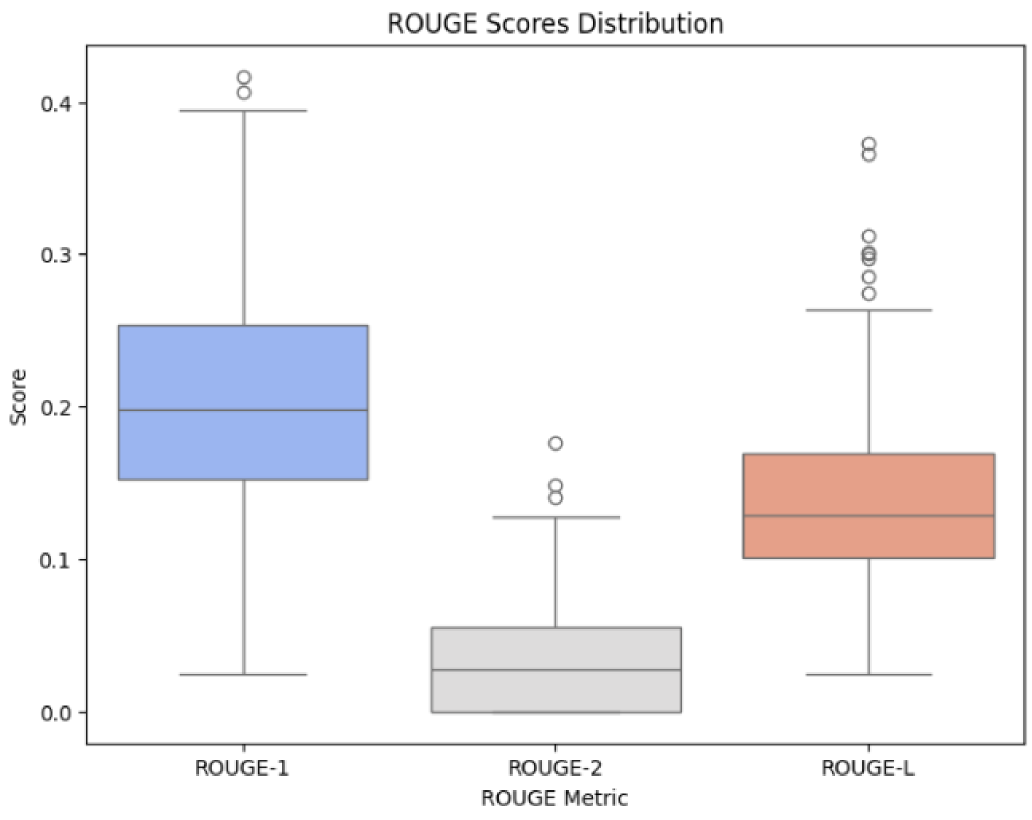
Display the rouge score

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not cal
  and should_run_async(code)
```

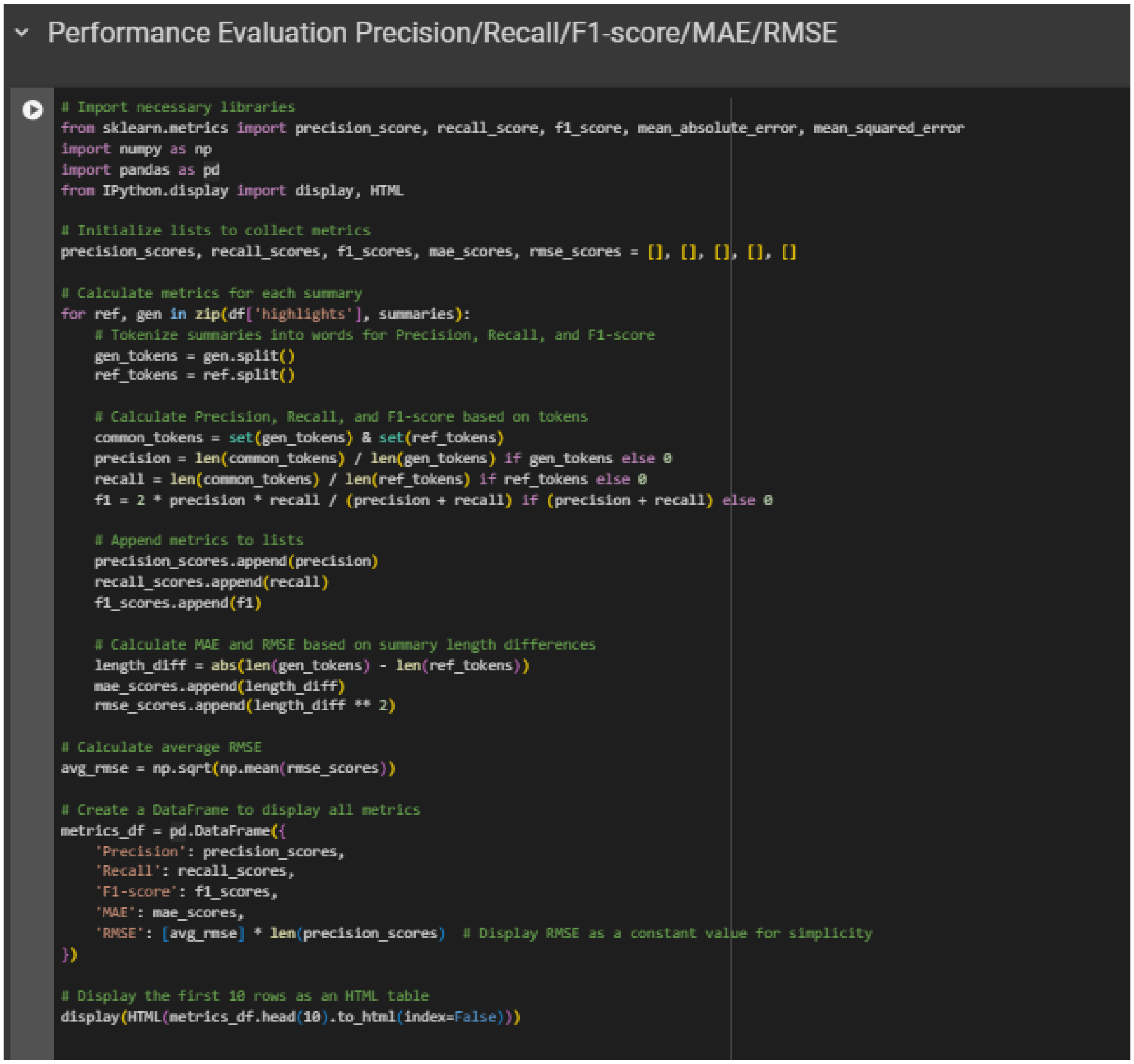
ROUGE-1	ROUGE-2	ROUGE-L
0.301370	0.084507	0.301370
0.253521	0.115942	0.140845
0.245614	0.036364	0.210526
0.152381	0.000000	0.095238
0.181818	0.018519	0.127273
0.246575	0.056338	0.164384
0.157480	0.000000	0.110236
0.175000	0.051282	0.075000
0.109589	0.000000	0.109589
0.079365	0.000000	0.047619

```
[ ] # Create boxplots for ROUGE scores
    plt.figure(figsize=(8,6))
    sns.boxplot(data=rouge_df, palette='coolwarm')
    plt.title('ROUGE Scores Distribution')
    plt.xlabel('ROUGE Metric')
    plt.ylabel('Score')
    plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_a
  and should_run_async(code)
```

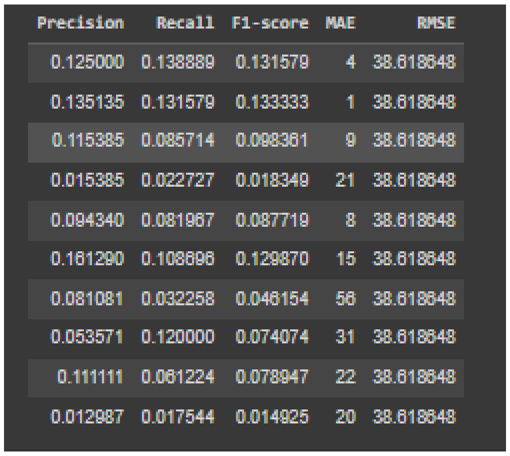


Precision/Recall/F1-Score/MAE/RMSE

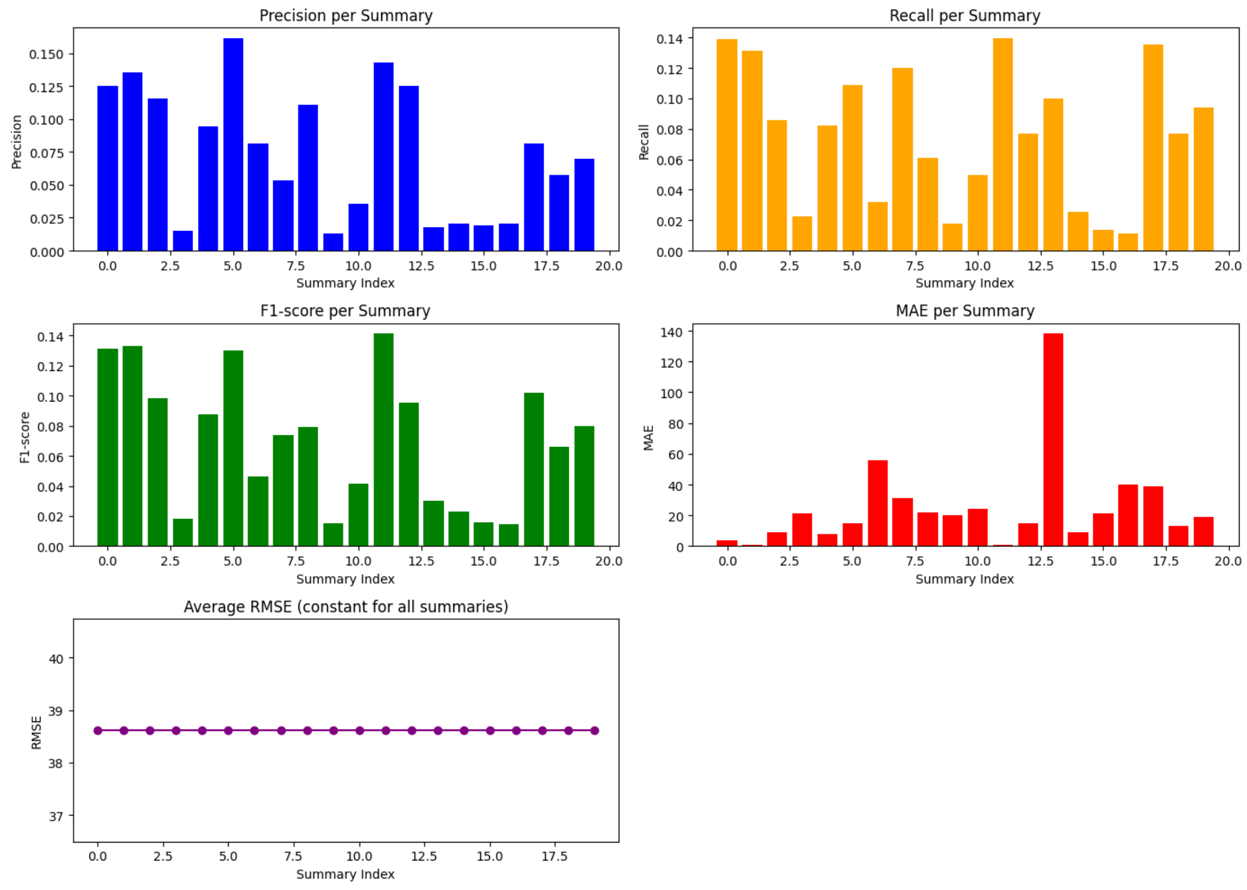


Performance Evaluation Precision/Recall/F1-score/MAE/RMSE

```python
# Import necessary libraries
from sklearn.metrics import precision_score, recall_score, f1_score, mean_absolute_error, mean_squared_error
import numpy as np
import pandas as pd
from IPython.display import display, HTML

# Initialize lists to collect metrics
precision_scores, recall_scores, f1_scores, mae_scores, rmse_scores = [], [], [], [], []

# Calculate metrics for each summary
for ref, gen in zip(df['highlights'], summaries):
    # Tokenize summaries into words for Precision, Recall, and F1-score
    gen_tokens = gen.split()
    ref_tokens = ref.split()

    # Calculate Precision, Recall, and F1-score based on tokens
    common_tokens = set(gen_tokens) & set(ref_tokens)
    precision = len(common_tokens) / len(gen_tokens) if gen_tokens else 0
    recall = len(common_tokens) / len(ref_tokens) if ref_tokens else 0
    f1 = 2 * precision * recall / (precision + recall) if (precision + recall) else 0

    # Append metrics to lists
    precision_scores.append(precision)
    recall_scores.append(recall)
    f1_scores.append(f1)

    # Calculate MAE and RMSE based on summary length differences
    length_diff = abs(len(gen_tokens) - len(ref_tokens))
    mae_scores.append(length_diff)
    rmse_scores.append(length_diff ** 2)

# Calculate average RMSE
avg_rmse = np.sqrt(np.mean(rmse_scores))

# Create a DataFrame to display all metrics
metrics_df = pd.DataFrame({
    'Precision': precision_scores,
    'Recall': recall_scores,
    'F1-score': f1_scores,
    'MAE': mae_scores,
    'RMSE': [avg_rmse] * len(precision_scores)  # Display RMSE as a constant value for simplicity
})

# Display the first 10 rows as an HTML table
display(HTML(metrics_df.head(10).to_html(index=False)))
```



Precision	Recall	F1-score	MAE	RMSE
0.125000	0.138889	0.131579	4	38.618848
0.135135	0.131579	0.133333	1	38.618848
0.115385	0.085714	0.098361	9	38.618848
0.015385	0.022727	0.018349	21	38.618848
0.094340	0.081967	0.087719	8	38.618848
0.161290	0.108696	0.129870	15	38.618848
0.081081	0.032258	0.046154	56	38.618848
0.053571	0.120000	0.074074	31	38.618848
0.111111	0.061224	0.078947	22	38.618848
0.012987	0.017544	0.014925	20	38.618848

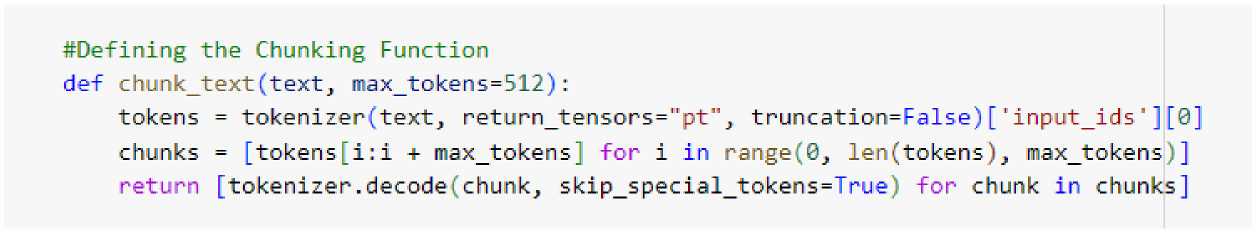

Precision per Summary
Precision
Summary Index
Recall per Summary
Recall
Summary Index
F1-score per Summary
F1-score
Summary Index
MAE per Summary
MAE
Summary Index
Average RMSE (constant for all summaries)
RMSE
Summary Index

Challenges Faced and Solutions

1.Handling Long Texts in Summarization:

- **Challenge:** Some texts may exceed the input size limit of the **BART model,** which could lead to truncation and loss of important content.

- **Solution:** A **chunking function** was implemented to divide long texts into smaller chunks that fit within the token limit of the summarization model. Each chunk was summarized individually and then concatenated into a final summary.

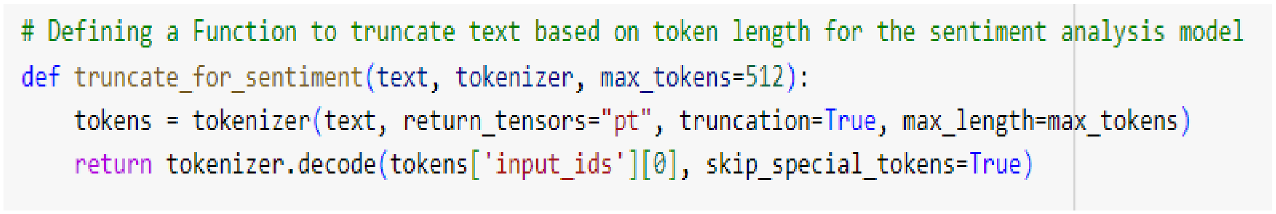


```python
#Defining the Chunking Function
def chunk_text(text, max_tokens=512):
    tokens = tokenizer(text, return_tensors="pt", truncation=False)['input_ids'][0]
    chunks = [tokens[i:i + max_tokens] for i in range(0, len(tokens), max_tokens)]
    return [tokenizer.decode(chunk, skip_special_tokens=True) for chunk in chunks]
```

2. Token Limit in Sentiment Analysis:

- **Challenge:** The **DistilBERT** model for sentiment analysis also has a token limit, and texts longer than this limit may be cut off.

- **Solution:** A **text truncation function** was used to ensure that input texts were truncated to fit within the token length limit of DistilBERT without losing essential meaning. The tokenizer was used to manage this truncation effectively.



```python
# Defining a Function to truncate text based on token length for the sentiment analysis model
def truncate_for_sentiment(text, tokenizer, max_tokens=512):
    tokens = tokenizer(text, return_tensors="pt", truncation=True, max_length=max_tokens)
    return tokenizer.decode(tokens['input_ids'][0], skip_special_tokens=True)
```

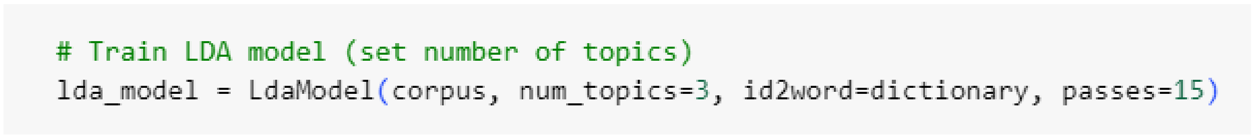
3. Overfitting in Topic Modeling:

- **Challenge:** The **LDA topic modeling** algorithm might overfit, especially when there are too many topics or insufficient tuning.

- **Solution: Hyperparameter tuning** was done by adjusting the number of topics and the number of passes through the data. This helped balance model complexity and performance, ensuring coherent and interpretable topics.

  Number of Topics (num_topics): This defines how many topics the LDA model should identify. Selecting the right number of topics is crucial to ensure the topics are coherent and interpretable.

  Number of Passes (passes): This refers to how many times the model will iterate over the entire corpus. More passes generally improve the accuracy of topic assignments but may also increase computation time and the risk of overfitting.

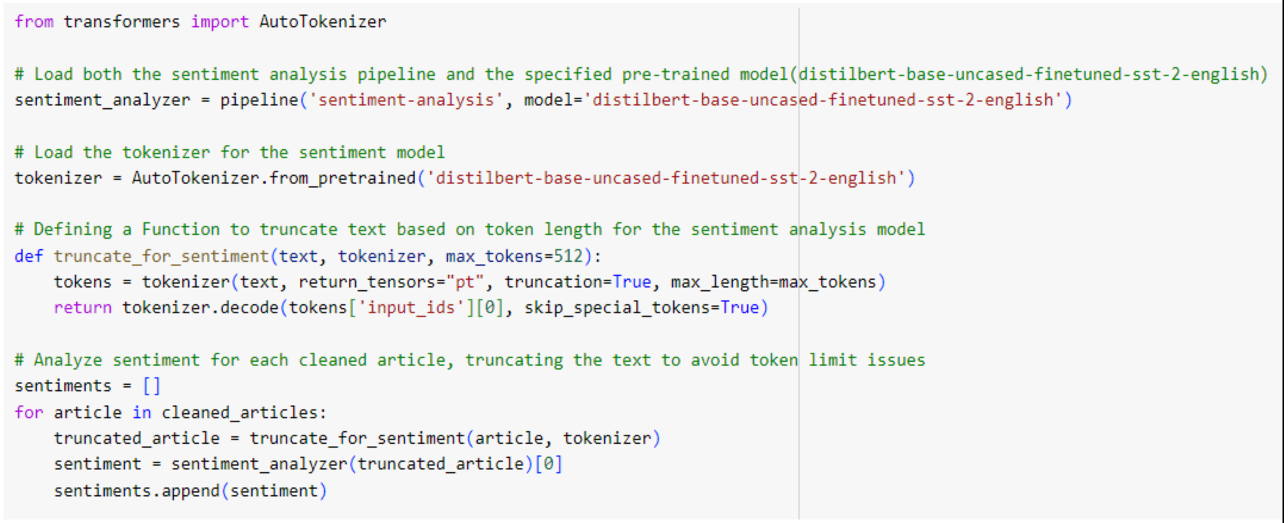


```python
# Train LDA model (set number of topics)
lda_model = LdaModel(corpus, num_topics=3, id2word=dictionary, passes=15)
```

4. Performance Optimization of NLP Models:

- **Challenge:** Large NLP models like **BART** and **DistilBERT** are computationally expensive, which may slow down the processing of large corpora.

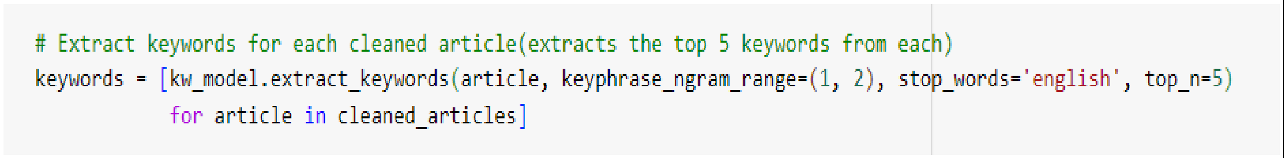
- **Solution:** Using optimized versions of models such as **DistilBERT** (which is a distilled, faster version of BERT), along with batch processing, improved performance. Additionally, leveraging GPU acceleration in environments like Google Colab sped up computations.



```python
from transformers import AutoTokenizer

# Load both the sentiment analysis pipeline and the specified pre-trained model(distilbert-base-uncased-finetuned-sst-2-english)
sentiment_analyzer = pipeline('sentiment-analysis', model='distilbert-base-uncased-finetuned-sst-2-english')

# Load the tokenizer for the sentiment model
tokenizer = AutoTokenizer.from_pretrained('distilbert-base-uncased-finetuned-sst-2-english')

# Defining a Function to truncate text based on token length for the sentiment analysis model
def truncate_for_sentiment(text, tokenizer, max_tokens=512):
    tokens = tokenizer(text, return_tensors="pt", truncation=True, max_length=max_tokens)
    return tokenizer.decode(tokens['input_ids'][0], skip_special_tokens=True)

# Analyze sentiment for each cleaned article, truncating the text to avoid token limit issues
sentiments = []
for article in cleaned_articles:
    truncated_article = truncate_for_sentiment(article, tokenizer)
    sentiment = sentiment_analyzer(truncated_article)[0]
    sentiments.append(sentiment)
```

5. Keyword Extraction Fails for Certain Texts:

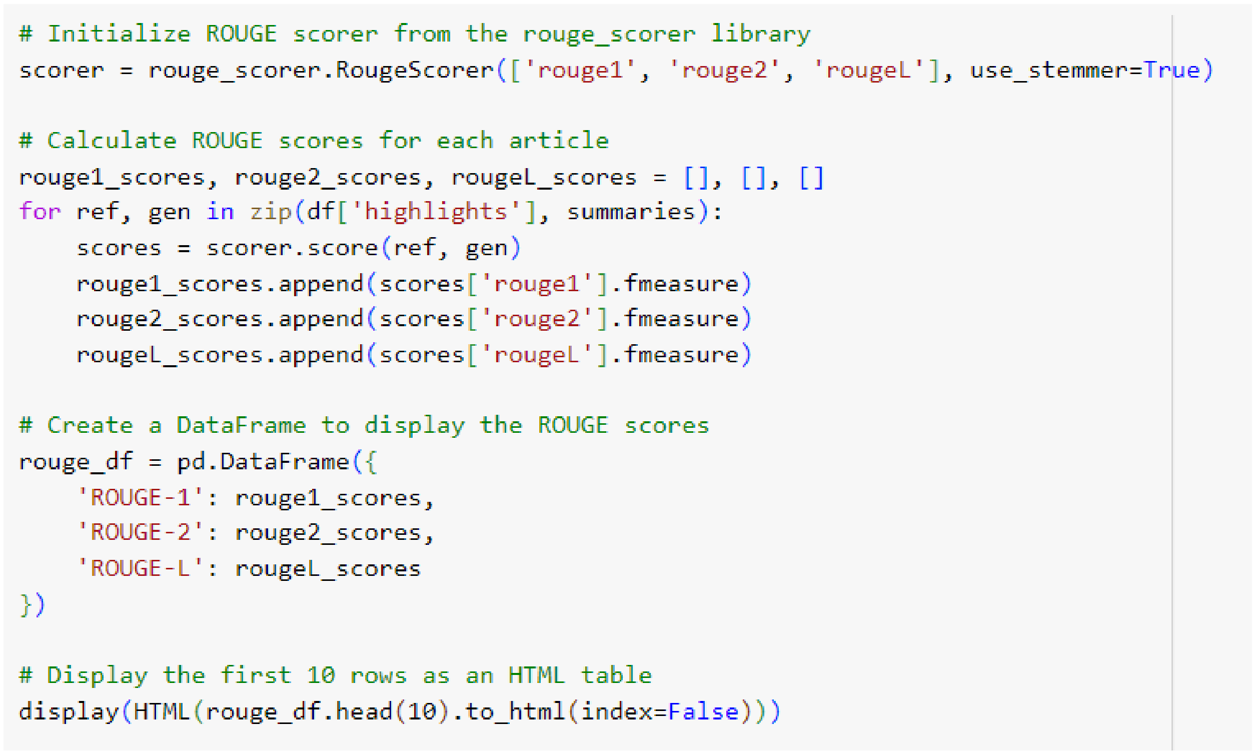
- **Challenge:** In some cases, **KeyBERT** might extract irrelevant or redundant keywords due to limited contextual understanding of the text.
- **Solution:** Fine-tuning of the model's hyperparameters (such as the number of keywords or the n-gram range) was done to improve relevance. Filtering stopwords and applying domain-specific constraints also enhanced keyword quality.



```
# Extract keywords for each cleaned article(extracts the top 5 keywords from each)
keywords = [kw_model.extract_keywords(article, keyphrase_ngram_range=(1, 2), stop_words='english', top_n=5)
            for article in cleaned_articles]
```

6. Evaluating Summarization Accuracy:

- **Challenge:** Accurately evaluating the summaries generated by the model was difficult due to the subjectivity involved in text summarization.
- **Solution:** The **ROUGE metric** was applied to provide an objective measure of how well the generated summaries matched reference summaries (highlights). ROUGE scores for unigram, bigram, and longest common subsequence overlaps helped quantify summarization performance.



```
# Initialize ROUGE scorer from the rouge_scorer library
scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)

# Calculate ROUGE scores for each article
rouge1_scores, rouge2_scores, rougeL_scores = [], [], []
for ref, gen in zip(df['highlights'], summaries):
    scores = scorer.score(ref, gen)
    rouge1_scores.append(scores['rouge1'].fmeasure)
    rouge2_scores.append(scores['rouge2'].fmeasure)
    rougeL_scores.append(scores['rougeL'].fmeasure)

# Create a DataFrame to display the ROUGE scores
rouge_df = pd.DataFrame({
    'ROUGE-1': rouge1_scores,
    'ROUGE-2': rouge2_scores,
    'ROUGE-L': rougeL_scores
})

# Display the first 10 rows as an HTML table
display(HTML(rouge_df.head(10).to_html(index=False)))
```

Conclusion

1. Summary of the project’s achievements

The **End-to-End Text Summarization and Analysis System** has successfully achieved several significant milestones across multiple Natural Language Processing (NLP) tasks.

- **NLP Pipeline Implementation:** A robust NLP pipeline was created, comprising various components such as text preprocessing, summarization, sentiment analysis, keyword extraction, and topic modeling.

- **Effective Text Summarization**: The system efficiently generates concise summaries of large corpus using state-of-the-art models like **BART.** In there the system delivers accurate and coherent summaries, making it applicable to various domains like news articles.

- **Sentiment Analysis and Keyword Extraction**: The sentiment analysis component accurately classifies text sentiment (positive, negative, or neutral) and provides actionable insights. The system also performs robust **keyword extraction using BERT-based techniques** to identify essential terms, enhancing information retrieval.

- **Topic Modeling**: Using advanced models like *Genism LDA*, the system effectively groups texts into meaningful topics, ensuring better organization and understanding of the content.

- **Evaluation Metrics:** The implementation of the ROUGE metric provided an objective evaluation framework to assess the quality of generated summaries against reference highlights, demonstrating the system's performance and effectiveness in summarization tasks.

- **User-Friendly Interface:** The integration of APIs facilitated seamless communication between the backend and other system components, enabling users to interact with the summarization features easily.

2. Limitations of the current system

- **Processing Time:** The use of large pre-trained models may result in longer processing times, especially for larger datasets, which could affect user experience in real-time applications.

- **Dependency on Quality of Input Data:** The performance of the summarization and analysis heavily relies on the quality of input text. Poorly written or unclear articles can lead to suboptimal results.

- **Limited Abstractive Summarization Quality**: While the abstractive summarization model performs well, it still struggles with certain edge cases where the summaries may lose essential details or produce slightly incoherent sentences. Handling complex, highly technical, or niche language can further affect output quality.

- **Topic Model Interpretability**: Although Genism provides coherent topics, interpretability in some complex domains still requires human intervention. Automated grouping may occasionally yield topics that are too broad or vague.

- **Keyword Extraction Limitations**: Keyword extraction methods struggle with multi-word phrases and do not always capture the full context of domain-specific terminology.

- **Dependency on Pre-Trained Models**: The system's reliance on large, pre-trained models like *BERT* and *BART* poses challenges in terms of computational costs, especially for real-time applications or when handling very large datasets.

- **Real-Time Processing**: While the system performs well with batch processing, it may face latency issues when deployed for real-time summarization, analysis, or recommendation in a high-traffic environment.

Suggestions for future enhancements or research directions

- **Model Improvement with Domain-Specific Fine-Tuning**:
  Future Research: Further *fine-tuning* of models (e.g., BERT, T5) on domain-specific datasets can significantly improve both summarization and sentiment analysis tasks. Models pre-trained on specialized corpora (e.g., legal, medical, or financial data) can better handle complex, technical language.

- **Introduction of Explainability Features**:
  Enhancement: Implement **explainable AI** (XAI) techniques to help users understand the decisions behind sentiment classifications, keyword extractions, or topic assignments. This could include showing why specific keywords were selected or how a particular sentiment was determined.

- **Multi-Modal Summarization**:
  Future Research: Extend the system to handle multi-modal content, such as integrating text and images, or summarizing video transcripts alongside articles. This would involve

combining image recognition models with NLP models to provide richer, more diverse summaries.

- **Incorporating User Feedback for Personalized Summaries**:
  Enhancement: Introduce a *user feedback loop* that allows users to rate the quality of summaries, recommendations, or keyword extractions. The system can then learn from these inputs and *adapt to individual user preferences*, delivering more personalized results.

- **Real-Time Processing and Optimization**:
  Enhancement: Focus on optimizing the system for *real-time processing* by using model compression techniques like *quantization, pruning*, and *knowledge distillation.* Implementing more efficient inference methods can reduce latency and enable the system to scale for real-time applications.

- **Exploration of Few-Shot or Zero-Shot Learning**:
  Future Research: Investigate *few-shot or zero-shot learning* models (e.g., GPT-3, T0) that require minimal training data for specific tasks, improving the system's ability to adapt to new domains with little labeled data.

- **Improved Topic Coherence with Hybrid Models**:
  Future Research: Explore hybrid models that combine *BERTopic* with *neural topic models* like *NVDM* (Neural Variational Document Model) for improved topic coherence and interpretability in more complex domains.

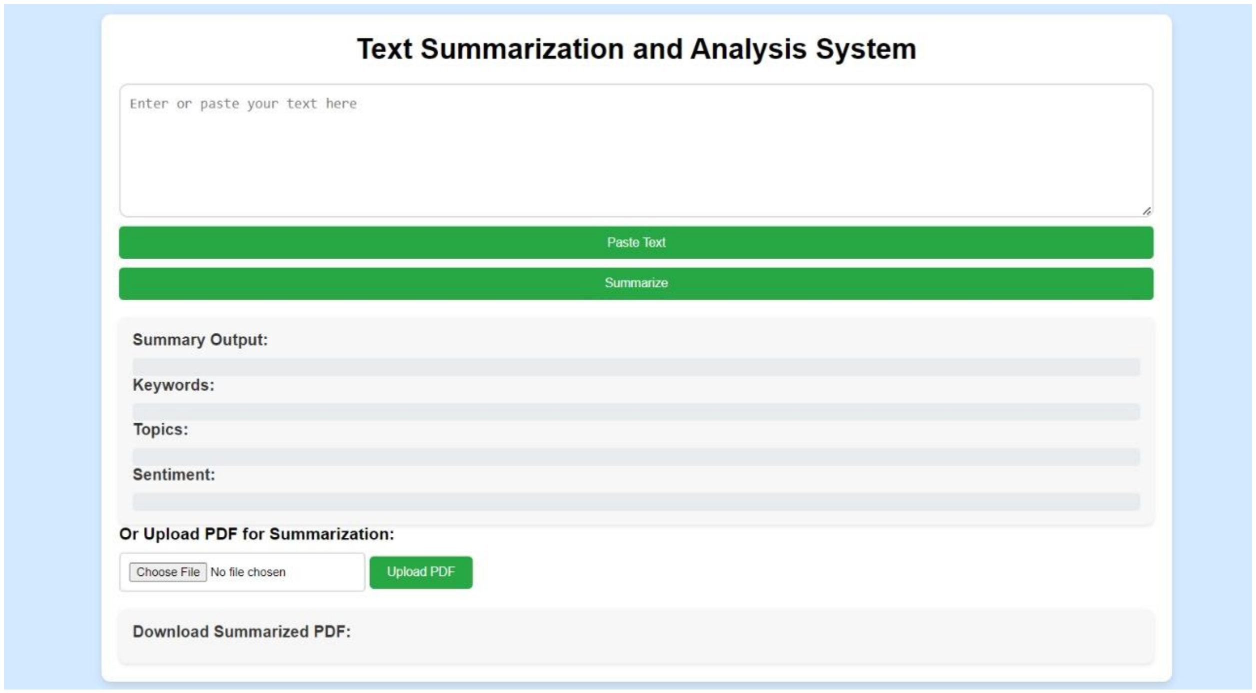
- **Scaling the Recommendation Engine**:
  Enhancement: Optimize the recommendation engine by leveraging *approximate nearest neighbor (ANN)* techniques such as *FAISS* or *HNSW* to improve the scalability of document similarity searches, allowing for fast retrieval even with millions of documents.

- **Ethical Considerations and Bias Mitigation**:
  Future Research: Incorporate mechanisms to detect and mitigate potential biases in NLP models, especially when summarizing or analyzing sensitive content. Ethical considerations should be a priority when working with biased data, ensuring that outputs remain fair and neutral.

- **Integration with External APIs for Contextual Enhancements**:
  Enhancement: Extend the system by integrating *external APIs* (e.g., news or academic databases) for providing real-time data, expanding its use cases. Additionally, integrating *Wikipedia APIs* or other knowledge bases could provide contextual information to enhance summaries or keyword extractions.

In summary, the **End-to-End Text Summarization and Analysis System** has delivered a highly functional and scalable solution for multiple NLP tasks such as summarization, sentiment analysis, keyword extraction, and document recommendations. While it has achieved remarkable accuracy and performance, there are still opportunities for further refinement, especially in handling complex domains, real-time processing, and providing more personalized, explainable outputs. With future enhancements such as domain-specific model fine-tuning, improved scalability for real-time usage, and exploration of explainability and bias mitigation, the system can evolve into a more robust, versatile tool suitable for a wider range of applications.

Final User Interface


Text Summarization and Analysis System
Enter or paste your text here
Paste Text
Summarize
Summary Output:
Keywords:
Topics:
Sentiment:
Or Upload PDF for Summarization:
Choose File No file chosen
Upload PDF
Download Summarized PDF: