Graphical user interface

Description automatically generated

Course

**ITC60008 Search Engines and Web Mining**

FINAL PROJECT

*Information Retrieval System*

*Climate Change Articles from United Nations Website*

**Professor**: Dr. Polymenakos Lazaros

**Students**

Gazepidis Frixos - 269702

Karatsiolis Christos – 270254

Moschoviti Nikolais – 270460

Zisimopoulos Theodoros – 268278

The American College of Greece

*Fall Term 2023*

Table of Contents

[Introduction 3](#_Toc153564783)

[Web Scraping 6](#_Toc153564784)

[Building the model 6](#_Toc153564785)

[**Importing the libraries** 6](#_Toc153564786)

[**Finding the URLs** 8](#_Toc153564787)

[Retrieving the main information 9](#_Toc153564788)

[EDA( Exploratory Data Analysis) 11](#_Toc153564789)

[Preprocessing 14](#_Toc153564790)

[Creating Embeddings 18](#_Toc153564791)

[Bert model 18](#_Toc153564792)

[Sec2Vec Model 20](#_Toc153564793)

[SIMILARITIES & RESULTS 21](#_Toc153564794)

[Bert Model 21](#_Toc153564795)

[Conclusion & Final Thoughts 22](#_Toc153564796)

[Bibliography 23](#_Toc153564797)

# Introduction

In an era where climate change poses one of the greatest challenges to our planet, the dissemination of accurate and timely information is crucial. As the world grapples with the complexities of global warming, rising sea levels, and extreme weather events, the need for accessible, reliable information has never been more critical. This project, aimed at developing an information retrieval system for the United Nations website, is designed to meet this need by providing streamlined access to articles and resources on climate change, thereby empowering decision-makers, researchers, and the global community.

Climate change, a defining issue of our time, requires informed action based on up-to-date and accurate information. The United Nations, at the forefront of global climate initiatives, plays a pivotal role in disseminating information and fostering international cooperation. However, with the vast amount of content available on its website, locating specific information on climate change can be daunting. This challenge underscores the need for an efficient information retrieval system that not only navigates the wealth of data but also ensures that users can easily find relevant and recent information. This system becomes particularly vital as the world seeks to understand the multifaceted impacts of climate change and the diverse strategies required to mitigate and adapt to these changes.

The rationale behind this project stems from the observed difficulty in accessing targeted information on climate change from the United Nations’ extensive online resources. The current search mechanisms can be overwhelming, often yielding a plethora of results that may not be immediately relevant. This project aims to refine this process, providing a tailored search experience that prioritizes relevancy, timeliness, and accuracy. By enhancing the accessibility of information, the system will play a key role in informing policy decisions, academic research, and public understanding of climate change, aligning with the United Nations' commitment to transparency and information dissemination.

The primary objective of this project is to develop a user-centric information retrieval system that simplifies the process of accessing climate change articles on the United Nations website <https://www.un.org/en/> . Key goals include improving the precision and relevance of search results, reducing the time taken to find pertinent information, and ensuring the system is intuitive and accessible to a diverse audience. Furthermore, the project aims to incorporate advanced search functionalities, such as filtering by date, region, or topic, to cater to the specific needs of users. Ultimately, the system is intended to become a valuable tool in the global discourse on climate change.

The development of this information retrieval system employs a blend of advanced technologies and innovative methodologies. At its core, the system uses natural language processing (NLP) algorithms to understand and interpret user queries accurately. It also incorporates machine learning techniques to continually improve search result relevance based on user interactions and feedback. Data organization is another critical aspect, with a structured approach to categorizing articles by themes such as impact, mitigation, and adaptation strategies. The system is designed with a user-friendly interface, ensuring ease of use for individuals with varying levels of technical expertise. Additionally, a key focus is on scalability and adaptability, allowing the system to evolve with the growing body of knowledge on climate change and the changing needs of its users.

Anticipated challenges in this project include managing the extensive and diverse content on climate change, ensuring the accuracy and timeliness of information, and addressing potential language barriers for a global audience. To overcome these challenges, the system will employ robust data management strategies, regular updates, and quality checks. Utilizing multilingual support and translation features will also be essential to cater to non-English speaking users. Collaborations with climate experts and continuous user feedback will play a crucial role in refining the system, ensuring that it remains a reliable and effective tool for accessing climate change information.

The expected impact of this project is significant, with the potential to enhance the global understanding of climate change issues. By providing efficient access to relevant information, the system will aid policymakers in making informed decisions, assist researchers in their studies, and empower the public with knowledge. This project aligns with the United Nations' efforts to combat climate change by fostering an informed and engaged global community. The contribution of this information retrieval system to the broader dialogue on climate change cannot be overstated, as it symbolizes a step towards a more informed, proactive, and collaborative approach to tackling one of humanity's most pressing challenges.

# Web Scraping

## Building the model

### **Importing the libraries**

import requestsfrom bs4

import BeautifulSoup

from tqdm.notebook import tqdm as tqdm

import pandas as pd

from time import sleep

import random

import numpy as np

In this web scraping project, we start by using *requests* to fetch the content of a webpage, then pass that content to BeautifulSoup to parse it and extract the needed data. The *tqdm* library used to track the progress of a loop or iteration, especially if you're scraping a large number of pages. *Pandas* helped us to store the scraped data in a structured form, and time.sleep along with random can help in managing request timing to avoid being flagged as a bot by the website's security systems. *Numpy* is used for any numerical operations on the data you've collected.

**requests**: This library is used to send HTTP requests in Python. It allows you to send all kinds of HTTP requests (e.g., GET, POST, PUT, DELETE) and handle the responses. In the context of web scraping, it's often used to fetch the content of web pages.

**bs4 (BeautifulSoup):** BeautifulSoup is a Python library for parsing HTML and XML documents. It creates parse trees that is helpful to extract the data easily. It's commonly used in web scraping to extract information from web pages, as it allows you to navigate the HTML structure and retrieve the elements you're interested in.

**tqdm.notebook (tqdm):** tqdm is a library that provides a fast, extensible progress bar for loops and iterations. tqdm.notebook is a special submodule designed for use in Jupyter Notebooks to display the progress bar in a way that works well within the notebook environment.

**pandas (pd):** Pandas is a fast, powerful, flexible, and easy-to-use open-source data analysis and manipulation tool built on top of the Python programming language. It's widely used for data manipulation and analysis. In web scraping, you might use it to store and manipulate the data you extract from web pages, such as putting it into a DataFrame (a table-like data structure).

**time (sleep):** The sleep function from the time module is used to pause the execution of the program for a given number of seconds. In web scraping, it's often used to throttle the rate of requests to avoid overwhelming the server or getting blocked.

**random:** This module implements pseudo-random number generators for various distributions. In web scraping, it can be used to generate random delays between requests to mimic human behavior and reduce the chance of getting blocked by the server.

**numpy (np):** NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. While not directly related to web scraping, it's often used alongside Pandas for numerical computations.

### **Finding the URLs**

A screenshot of a computer program

Description automatically generated

Our team initiated a web scraping sequence to collect URLs from the United Nations news website, with a focus on articles about climate change. We started by initializing an empty list named url\_list to store the desired URLs. We set the main\_link variable to the base URL of the United Nations news site.

We then defined a file\_path variable, intending to specify the location and name of a text file to save the collected URLs. The max\_page variable was set to 1, indicating that we planned to scrape only the first page of the climate change news section.

We employed a for loop to iterate over a range up to the max\_page value, constructing the full URL by appending the page number to the base URL of the climate change news section. Inside the loop, we printed each constructed URL to the console for verification.

We sent an HTTP GET request to the constructed URL using the requests library, with a timeout parameter set to 25 seconds to avoid hanging indefinitely if the server did not respond. After making the request, we paused the execution for a random duration between 3 and 4 seconds using sleep from the time module and random.uniform to simulate more natural browsing behavior and reduce the risk of being blocked by the server.

Upon receiving a successful response (HTTP status code 200), we parsed the content of the page with BeautifulSoup, specifying 'html.parser' as the parser. We then created a list of <h2> elements that had a class attribute of 'node\_\_title', which typically contain the headlines and corresponding links to articles.

For each headline in the list, we attempted to find an <a> tag within the <h2> element to extract the 'href' attribute, which contains the slug of the article URL. We appended the full URL, constructed by combining the main\_link with the 'href' value, to our url\_list.

In cases where the 'href' could not be found, we printed a message stating 'href not found.' to the console. If the response status code was not 200, which would indicate an unsuccessful request, we printed "Response code not 200."

Finally, we opened the file at the specified file\_path in write mode and wrote each URL in our url\_list to the file, each on a new line. This would save the collected URLs for subsequent use, such as retrieving the articles' content or further analysis.

### Retrieving the main information

A screenshot of a computer program

Description automatically generated

The team continued the web scraping project by implementing a Python script to retrieve detailed content from the list of URLs that had been previously collected and saved. The content aimed to include various pieces of information from each article, such as the title, date, category, summary, main text, and headers.

Here is a step-by-step explanation of the code:

1. The **urls** variable was populated with a list of URLs to process. These URLs were obtained by splitting the multiline string **text**, which presumably contained the raw text with URLs separated by newline characters.
2. Instead of reading from an existing CSV file, a new empty DataFrame **df** was created using pandas, a powerful data manipulation library.
3. A **counter** variable was initialized to zero. This served as an index and a way to monitor the progress of the scraping loop.
4. A **for** loop was initiated to iterate over each URL in the **urls** list. For each URL, the following steps were taken:
   * The URL and the counter were printed to the console, providing a visual reference of the progress.
   * A **try** block was used to handle potential errors that might occur during the HTTP request or data extraction process. Inside this block:
     + An HTTP GET request was sent to the current URL with a timeout of 25 seconds to prevent hanging if the server response was delayed. A delay of 3 to 4 seconds was introduced between requests to mimic human behavior and avoid being blocked by the website's security mechanisms.
     + The content of a successful response (HTTP status code 200) was passed to BeautifulSoup for HTML parsing.
     + Several **try** and **except** blocks were used to attempt to extract the **title**, **date**, **category**, **summary**, and the main text (**temp\_text**) as well as headers (**h3\_text**) from the HTML content. If any of these elements were not found, a NumPy NaN value was assigned to their variables, indicating a missing value.
     + Text and headers were extracted from their respective HTML containers, converted to text, and concatenated into strings separated by a special character **\a**. Any non-breaking space characters (**\xa0**) were removed from the text.
     + A temporary dictionary **temp\_dict** was created to store the extracted data.
     + This dictionary was then converted into a DataFrame **temp\_df**, which was transposed (to convert from a row into a column format) and the columns were named after the keys of **temp\_dict**.
     + The newly created **temp\_df** was concatenated to the main DataFrame **df**, which collected all the extracted information, and **ignore\_index=True** was used to reindex the DataFrame properly.
   * In case of an exception (such as a connection error or a parsing issue), an error message was printed to the console.
   * After processing each URL, the **counter** was incremented by 1.
5. After each iteration and at the end of the **try** block, the DataFrame **df** was saved to a CSV file called **articles\_V2.csv**, ensuring that the data was not lost even if the program encountered an error on subsequent URLs.

The code reflects a thorough scraping process, designed to extract specific pieces of information from each article, handle exceptions gracefully, and save the progress iteratively to a CSV file for further analysis or record-keeping.

### EDA( Exploratory Data Analysis)

* **Assessing the Volume Over Time:** We calculated the number of articles published each year to understand the annual volume and identify any trends over time. This helped us determine whether interest or reporting on climate change has been increasing, decreasing, or remaining steady in recent years.
* **Monthly Analysis:** We further broke down the data to analyze the monthly distribution of articles. This step allowed us to see if there were any seasonal patterns in the reporting of climate change topics, such as a surge in articles around significant events like climate summits or environmental disasters.
* **Word Cloud Creation:** To visualize the most prominent themes and terms used in the climate change articles, we generated a word cloud. This graphical representation highlighted the most frequently occurring words, giving us a visual sense of the main topics and concerns within the body of collected articles.

These steps in EDA served distinct purposes:

* The yearly and monthly analysis provided a quantitative look at the dataset, allowing us to track reporting intensity and potentially correlate it with external events or policy changes.
* The word cloud offered a qualitative angle, presenting an immediate visual snapshot of the content. This could help in identifying key areas of focus for the United Nations in their climate change communications, as well as any gaps that might exist in the coverage.

Together, these EDA steps gave us a comprehensive overview of the dataset, which is crucial for informing subsequent data analysis, predictive modeling, or reporting to stakeholders. Through this process, we were able to establish a foundation for understanding the UN’s historical and current emphasis on climate change within their news content.

A graph of a number of articles written in green and green

Description automatically generated

A graph showing the number of articles written each month

Description automatically generated

**Word Cloud Creation:** To visualize the most prominent themes and terms used in the climate change articles, we generated a word cloud. This graphical representation highlighted the most frequently occurring words, giving us a visual sense of the main topics and concerns within the body of collected articles.

A close-up of words

Description automatically generated

## Preprocessing

import nltk

import re

import nltk.corpus

import stopwords

from nltk.tokenize

import word\_tokenizefrom nltk.tag

import pos\_tagnltk.download('stopwords')nltk.download('punkt')

Following the initial exploratory data analysis (EDA), we moved on to preprocessing the textual data collected from the United Nations news website using various natural language processing (NLP) tools from the Natural Language Toolkit (NLTK) library, alongside regular expressions (regex) provided by the **re** module in Python. Preprocessing is a crucial step in preparing raw text for NLP tasks, such as sentiment analysis, topic modeling, or machine learning.

* **NLTK (Natural Language Toolkit):** This is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning.
* **re (Regular Expression):** This module provides regular expression matching operations similar to those found in Perl. Regular expressions are a powerful language for matching text patterns. This can be used for data cleaning by finding and removing patterns of text that are not needed for analysis, such as HTML tags or special characters.
* **nltk.corpus (stopwords):** Stopwords are words that do not contribute much meaning to a sentence and are usually removed from texts. They can include common words such as 'is', 'and', 'the', etc. The **stopwords** corpus in NLTK contains a list of such words in various languages.
* **nltk.tokenize (word\_tokenize):** This function is used for splitting a sentence into words or tokens. It’s a critical step in text preprocessing as it breaks down the text into manageable pieces for further analysis.
* **nltk.tag (pos\_tag):** Part-of-speech tagging is the process of marking up a word in a text as corresponding to a particular part of speech, based on both its definition and its context. This can be useful for subsequent analysis like named entity recognition or extracting nouns for a word cloud.

Before these tools could be used, the team needed to download the necessary NLTK resources using **nltk.download()**. Specifically:

* **nltk.download('stopwords')**: This downloads the set of stopwords from NLTK's collection.
* **nltk.download('punkt')**: This downloads the Punkt tokenizer models, which is a pre-trained model that NLTK uses to tokenize text into sentences.

By using these libraries and tools, the team was able to clean and tokenize the text data, remove stopwords, and perform part-of-speech tagging, which would allow for a more focused and meaningful analysis of the text content from the climate change articles. This preprocessing stage is fundamental to ensure that the dataset is clean and standardized before any advanced NLP tasks can be performed.

We continued with the preprocessing of the dataset containing articles on climate change. They defined functions and applied transformations to the data to facilitate further analysis:

**Defining a Function to Split Text**:

A function split\_main\_text was created to handle the splitting of the main\_text field in the dataset. If main\_text contained data (was not NA), the function split the text into a list using a special delimiter \a. If main\_text was NA, it returned an empty list. This function was crucial for breaking down the text into individual segments for analysis.

Defining a Function to Concatenate Columns:

Another function concat\_cols was defined to concatenate various pieces of information into a single string. It attempted to combine title, date, category, headers, and the newly created main\_text\_splitted into one text. If there was an exception (likely due to missing data), it concatenated all fields except headers.

**Applying Functions and Filtering the DataFrame**:

Then, we loaded the data from a CSV file into a DataFrame df using pandas.

They applied the split\_main\_text function to the main\_text column and stored the result in a new column main\_text\_splitted.

We continued by dropping the summary and main\_text columns from df to clean up the dataset.

They filtered the DataFrame to keep only the rows where the category was "Climate and Environment".

The DataFrame index was reset to ensure it was in sequential order after filtering.

Expanding the main\_text\_splitted List into Rows:

The explode method was used on main\_text\_splitted to create a new row in the DataFrame for each element in the list. This operation was followed by another index reset.

Cleaning Null Values and Analyzing Text Length:

We removed rows where main\_text\_splitted was null and reset the index once more.

A new column main\_text\_len was created to store the length of each segment of main\_text\_splitted.

We plotted the distribution of main\_text\_len to visualize the length of the text segments.

The team filtered the DataFrame to keep only the rows where the text segments were longer than 100 characters, then plotted the distribution again for the filtered data.

1. **Exploding Text Segments:**
   * The **explode** method was applied to the **main\_text\_splitted** column of the DataFrame **df**. This operation separated each element in the lists of **main\_text\_splitted** into its own row, effectively increasing the granularity of the data for analysis.
   * The DataFrame index was reset using **reset\_index(drop=True)** to maintain a consecutive index without retaining the old index.
2. **Removing Null Values:**
   * The team identified rows where **main\_text\_splitted** was null and captured their indices in the variable **ind**.
   * These rows were then dropped from **df** using **df.drop(ind, axis=0)**, as they wouldn't contribute to the text analysis.
   * The DataFrame index was reset again to account for the dropped rows.
3. **Calculating and Filtering Text Length:**
   * A new column **main\_text\_len** was created by applying a lambda function to calculate the length of each text segment in **main\_text\_splitted**.
   * They visualized the distribution of text lengths using a plot to understand the spread of the data.
   * The team filtered the DataFrame to include only those rows where the text segment was greater than 100 characters, under the assumption that longer texts would provide more substantial information for analysis.
   * After filtering, they plotted the distribution of text lengths again, likely to confirm the filtering and check the new distribution.
4. **Concatenating Columns for Aggregate Text:**
   * They applied the **concat\_cols** function to each row of the DataFrame to concatenate text data from multiple columns (**title**, **date**, **category**, **headers**, and **main\_text\_splitted**) into a single column **text**.
   * This step was designed to create a unified text field for each article, potentially for use in subsequent text analysis tasks such as creating a word cloud or input into a machine learning model.
5. **Final Data Cleanup:**
   * After creating the concatenated **text** column, the team dropped several columns (**title**, **date**, **category**, **headers**, **main\_text\_splitted**, **main\_text\_len**, **link**) that were no longer needed for the analysis, as their information had been merged into the **text** column.
   * They reset the DataFrame index one last time to ensure it was in sequential order after the data manipulation.

# Creating Embeddings

## Bert model

from sentence\_transformers import SentenceTransformer

df = pd.read\_csv("../files/preprocessed\_articles.csv")

paragraphs = list(df["text"])

model\_bert = SentenceTransformer('bert-base-nli-max-tokens') embedding\_bert = np.array(model\_bert.encode(paragraphs, show\_progress\_bar=True)) *#Bert embeddings are shape of 768*print("Bert Embedding shape", embedding\_bert.shape)print("Bert Embedding sample", embedding\_bert[0][0:50])

with open('../files/embeddings.pickle.pkl', 'wb') as f:     pkl.dump(embedding\_bert, f)

We advanced our project by incorporating one of the cutting-edge technologies in natural language processing—BERT (Bidirectional Encoder Representations from Transformers). BERT is a transformer-based machine learning technique for natural language understanding, developed by Google.

Here's an explanation of the code and the process we followed to create embeddings using BERT:

* **Importing Required Libraries:**
  + We imported **SentenceTransformer** from the **sentence\_transformers** library. This library is specifically designed for generating sentence embeddings, which are dense vector representations of sentences capable of capturing semantic information.
* **Loading the Preprocessed Data:**
  + We loaded a CSV file containing the preprocessed text of the articles into a pandas DataFrame **df**. This data had already undergone cleaning and consolidation to ensure each text entry was correctly formatted for processing.
* **Preparing the Data for Embedding:**
  + A list named **paragraphs** was created from the **text** column of the DataFrame. This list was intended to be passed to the BERT model for embedding.
* **Loading the BERT Model:**
  + We instantiated a BERT model by calling **SentenceTransformer** with the specific model name 'bert-base-nli-max-tokens'. This model is pre-trained on a large corpus of text and can understand a wide range of language contexts and nuances.
* **Generating Embeddings:**
  + We used the **model\_bert.encode** method to generate embeddings for each paragraph in the **paragraphs** list. The argument **show\_progress\_bar=True** provided a visual cue of the encoding process's progress.
  + The embeddings were stored in a NumPy array **embedding\_bert**, each having a dimensionality of 768. This fixed-size vector is characteristic of the BERT-base models.
* **Inspecting Embeddings:**
  + We printed the shape of the **embedding\_bert** array to verify the number of embeddings created and their dimensionality.
  + We printed a sample of the first 50 dimensions of the first embedding. This was likely to ensure the embeddings were correctly generated and to understand their composition.
* **Saving the Embeddings:**
  + Finally, we serialized the embeddings using the **pickle** module and saved them to a file named 'embeddings.pickle.pkl'. This allows us to easily load the embeddings for future use without the need to recompute them, thus saving time and computational resources.

Creating BERT embeddings is a pivotal step in our project, as it transforms the text data into a form suitable for various machine learning tasks, such as clustering, similarity analysis, or as input features for predictive models. These embeddings capture the contextual relationships between words, providing a rich representation of the text's semantic meaning.

## Sec2Vec Model

We further enhanced our text analysis capabilities by implementing the Doc2Vec model, another advanced technique in natural language processing. Doc2Vec, a part of the Gensim library, extends the Word2Vec methodology to work with sentences, paragraphs, and documents, enabling the generation of more context-aware embeddings.

Here's how we proceeded with the Doc2Vec model implementation:

1. **Importing Necessary Libraries and Modules:**
   * We imported various NLP-related libraries and functions, including NLTK for text processing, and Gensim’s **Doc2Vec** for document vectorization.
   * The **nltk.download('averaged\_perceptron\_tagger')** was used to download the necessary data for part-of-speech tagging.
2. **Preparing Text Processing Tools:**
   * We created instances of **WordNetLemmatizer** and **PorterStemmer** from NLTK. Lemmatization and stemming are techniques used to reduce words to their base or root form.
   * A list of English stopwords was also created. Stopwords are common words that are often removed in text processing to reduce noise.
3. **Defining a Text Cleaning Function:**
   * We defined a function **cleantxt** to preprocess the text. This function removed whitespace characters, converted text to lowercase, and eliminated punctuation. It then tokenized the text, removed stopwords, and applied lemmatization to each token.
   * This preprocessing step is crucial for cleaning and standardizing the text, which enhances the quality of the embeddings generated by Doc2Vec.
4. **Defining a Cosine Similarity Function:**
   * A function **cosine** was defined to calculate the cosine similarity between two vectors. Cosine similarity is a measure used to determine how similar two vectors are, often used in text analysis to compare documents or sentences.
5. **Loading the Preprocessed Data:**
   * We loaded the preprocessed articles from a CSV file into a pandas DataFrame. This data contained the text that was to be vectorized.
6. **Tokenizing and Tagging Data for Doc2Vec:**
   * We created a list **sents** from the **text** column of the DataFrame.
   * Each sentence in **sents** was cleaned using the **cleantxt** function, resulting in a list of tokens.
   * The cleaned, tokenized sentences were then tagged with unique identifiers using **TaggedDocument** from Gensim. This step is essential for Doc2Vec, as it requires tagged data to learn the representations of sentences or documents.

By preparing our data and applying the Doc2Vec model, we were setting the stage for generating document embeddings. These embeddings are valuable for various NLP tasks, such as semantic analysis, clustering, or information retrieval, as they capture the context and meaning of larger text units (like paragraphs or entire documents) more effectively than word-level embeddings. The addition of the cosine similarity function suggested that we were also interested in comparing the similarity between different text segments or articles.

# SIMILARITIES & RESULTS

## Cosine Similarity

We further integrated our project with a feature to search and retrieve articles based on their semantic similarity to a user-input query. This was accomplished using BERT embeddings and cosine similarity, a standard approach in natural language processing for measuring similarity between two text documents.

Here's a breakdown of the steps we followed:

* **Importing Cosine Similarity Function:**
  + We imported the **cosine\_similarity** function from Scikit-learn's **metrics.pairwise** module. This function computes the cosine similarity between samples in two arrays.
* **Loading Preprocessed Data and BERT Embeddings:**
  + We loaded the preprocessed articles from a CSV file into a pandas DataFrame.
  + Using Python's **pickle** module, we loaded precomputed BERT embeddings from a file. These embeddings represent the articles in our dataset.
* **Query Input and Embedding Generation:**
  + We prompted the user to enter a search query.
  + We instantiated the **SentenceTransformer** model again with the same BERT model ('bert-base-nli-max-tokens') used for generating article embeddings.
  + We then generated a BERT embedding for the user's query. This embedding was necessary to compare the query with our dataset articles.
* **Calculating Cosine Similarities:**
  + Using the **cosine\_similarity** function, we computed the similarity scores between the query embedding and all article embeddings. This process resulted in an array of similarity scores, reflecting how semantically similar each article was to the query.
* **Assigning Scores and Sorting Articles:**
  + We added the cosine similarity scores to our DataFrame as a new column **cosine\_similarities**.
  + We sorted the DataFrame based on the **cosine\_similarities** column in descending order. This sorting brought the articles most relevant to the query to the top of the DataFrame.
* **Selecting Top Articles:**
  + We selected the top 10 articles with the highest cosine similarity scores. This subset was stored in **temp\_df**.
  + We reset the index of **temp\_df** to make it easier to read and interpret the results.

The outcome of this process was a DataFrame containing the ten articles most relevant to the user's query, ranked by their semantic similarity. This functionality provided a powerful tool for users to search the dataset using natural language queries, making the retrieval of information both efficient and intuitive. By leveraging BERT's ability to understand the context and meaning behind words and sentences, we were able to create a sophisticated search mechanism that goes beyond keyword matching, offering results based on the semantic content of the articles.

# Conclusion & Final Thoughts

**Conclusion**

Our project embarked on the ambitious task of enhancing the accessibility and analysis of climate change articles from the United Nations website. We successfully navigated the complexities of web scraping, data preprocessing, exploratory data analysis (EDA), and advanced natural language processing (NLP) techniques to derive meaningful insights from the collected data.

Through meticulous web scraping, we gathered a substantial corpus of articles, meticulously focusing on those relevant to climate change and environmental issues. Our EDA phase was instrumental in understanding the distribution and frequency of these articles over time, revealing both seasonal and annual trends in climate change reporting. The creation of a word cloud offered a visually compelling glimpse into the most prominent themes and terms within the articles, highlighting key focus areas in climate change discourse.

The adoption of cutting-edge NLP techniques, specifically BERT and Doc2Vec embeddings, marked a significant advancement in our project. By transforming textual data into semantically rich vector representations, we opened the door to sophisticated text analyses that traditional methods couldn't achieve. The BERT embeddings, in particular, allowed us to capture the nuanced contextual relationships between words, leading to more effective and insightful clustering, similarity analysis, and predictive modeling.

The implementation of a cosine similarity-based search feature was a testament to our commitment to making climate change information more accessible and user-friendly. By allowing users to input natural language queries and retrieve the most relevant articles, we significantly enhanced the utility and reach of our project.

**Future Work**

Looking ahead, there are several avenues for future work that can further expand the scope and impact of our project:

1. **Dynamic Web Scraping:** To maintain the relevance and freshness of our dataset, we plan to implement a dynamic web scraping mechanism that periodically updates our corpus with the latest articles. This will ensure that our analysis remains current and comprehensive.
2. **Multilingual Support:** Recognizing the global impact of climate change, incorporating multilingual support to our text analysis and search functionalities is a priority. This will make our project accessible to a broader audience and provide insights into climate change discourse in different linguistic contexts.
3. **Sentiment Analysis and Topic Modeling:** Leveraging advanced NLP techniques to perform sentiment analysis and topic modeling on the articles will offer deeper insights into the tone and evolving themes of climate change reporting. This can aid in understanding public sentiment and the effectiveness of communication strategies.
4. **Interactive Data Visualization Dashboard:** Developing an interactive dashboard that presents our findings in a user-friendly, visual format will make our analyses more accessible to policymakers, researchers, and the general public. This tool can include features like trend analysis, geographic heatmaps, and real-time data updates.
5. **Machine Learning for Predictive Analysis:** Employing machine learning algorithms to predict future trends in climate change reporting and its impact on public awareness and policy can provide valuable foresight for decision-makers.
6. **Collaborative Partnerships:** Forming partnerships with academic institutions, environmental organizations, and media outlets can provide new datasets, resources, and perspectives. These collaborations can enhance the quality and scope of our analyses.
7. **User Feedback and Iterative Improvement:** Incorporating user feedback to continuously improve the user experience and relevance of our search and analysis tools will be crucial. Regular updates and iterations based on user input will ensure our project remains aligned with the needs of our audience.

In conclusion, our project represents a significant step towards leveraging technology in the service of environmental awareness and action. By continually evolving and expanding our methodologies and scope, we aim to remain at the forefront of climate change information analysis and dissemination, contributing effectively to global efforts in combating this existential challenge.

# Bibliography

Introduction