# Text Summarization using Python & NLTK: TF-IDF Algorithm

|  |  |  |
| --- | --- | --- |
| **Name** | **BITS ID** | **Contribution** |
| SUBHRANSU MISHRA | 2023AC05489 | 100% |

## Solution Overview & Various Approaches Considered:

## The focus of this assignment is to create summaries for long text passages while preserving their key information. In **Approach 1**, I used TF-IDF combined with Part-of-Speech (PoS) tagging to rank sentences based on the importance of their words. This involved processing the text by tokenizing it, removing unnecessary words, and assigning higher scores to nouns and proper nouns to emphasize context. In **Approach 2**, I explored a graph-based method where sentences were represented as vectors using TF-IDF, and cosine similarity was used to measure their relationships. PageRank was then applied to rank the sentences based on their importance. Both methods aim to ensure the summaries are accurate and concise, highlighting the main points effectively.

I will be discussing about both the approaches in the following sections in more detail. *I could not attach the complete code notebook file in the submission documents as the limit of files is 1. I have added code snippets and if require I can share the file separately.*

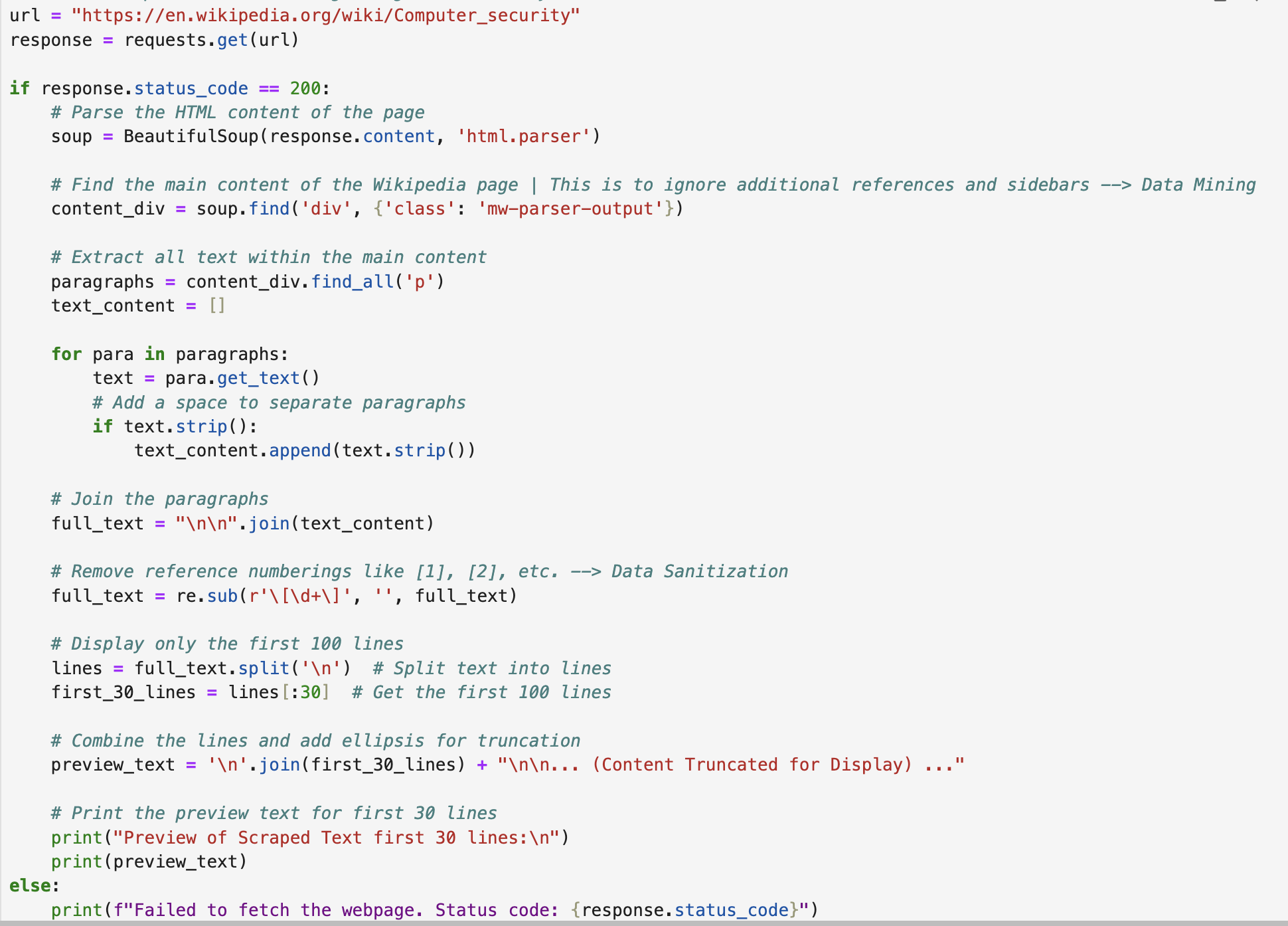
## Data Extraction for this Project:

For this project, I used data **scraped** from a publicly available source (**www.wikipedia.com**) to ensure sufficient text for processing and summarization. The text was extracted programmatically using Python libraries such as requests and “BeautifulSoup”. The process involved sending a GET request to the webpage and parsing the HTML content to retrieve relevant text sections. I focused specifically on the main content area, filtering out navigation links, advertisements, and irrelevant elements. Additionally, the extracted text was sanitized by removing reference numbers, special characters, and unnecessary formatting. This cleaned data served as the input for implementing various summarization techniques used in the assignment.

I have selected the topic as Cyber Security and the page reference is <https://en.wikipedia.org/wiki/Computer_security>

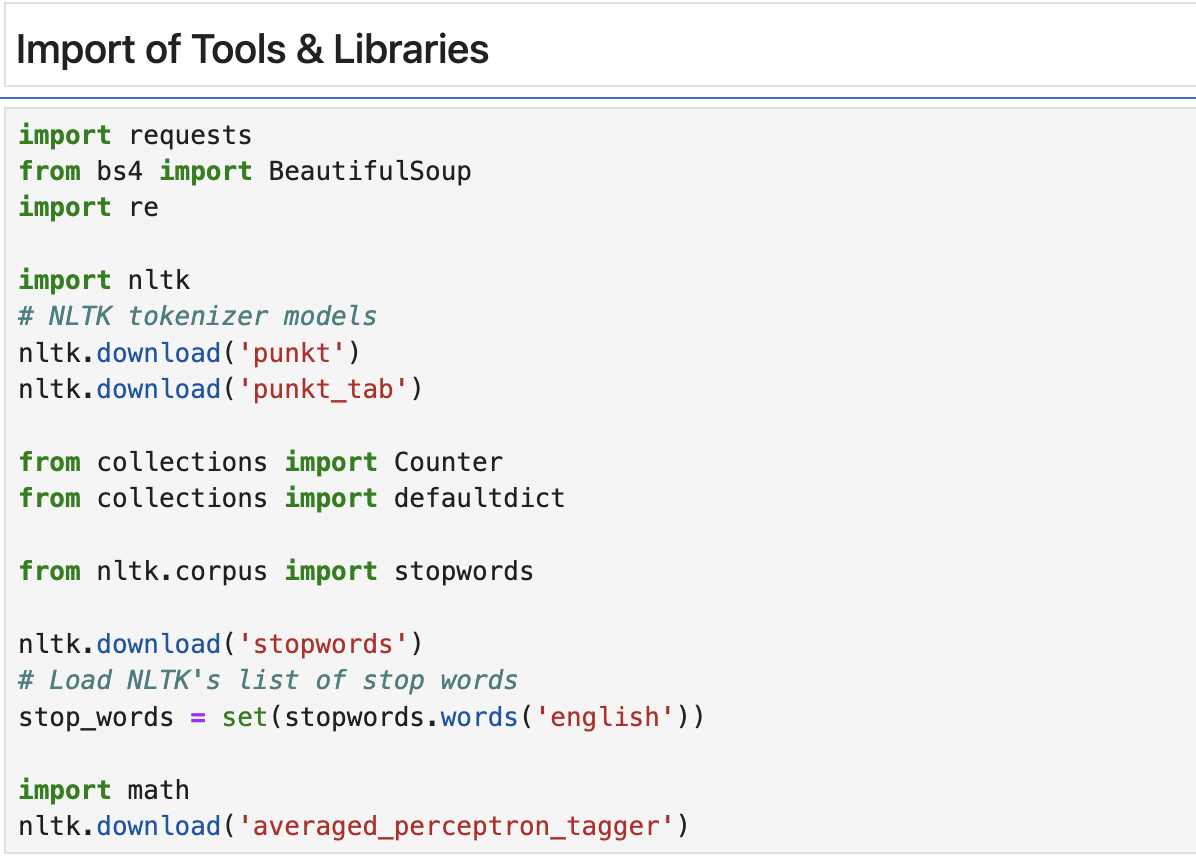
The interesting part of this process of data scrapping is, one can switch to any other supported WIKI url and the developed model can generate a summary for that page giving a good flexibility to test the concepts uses.

Code Snippet from Notebook:



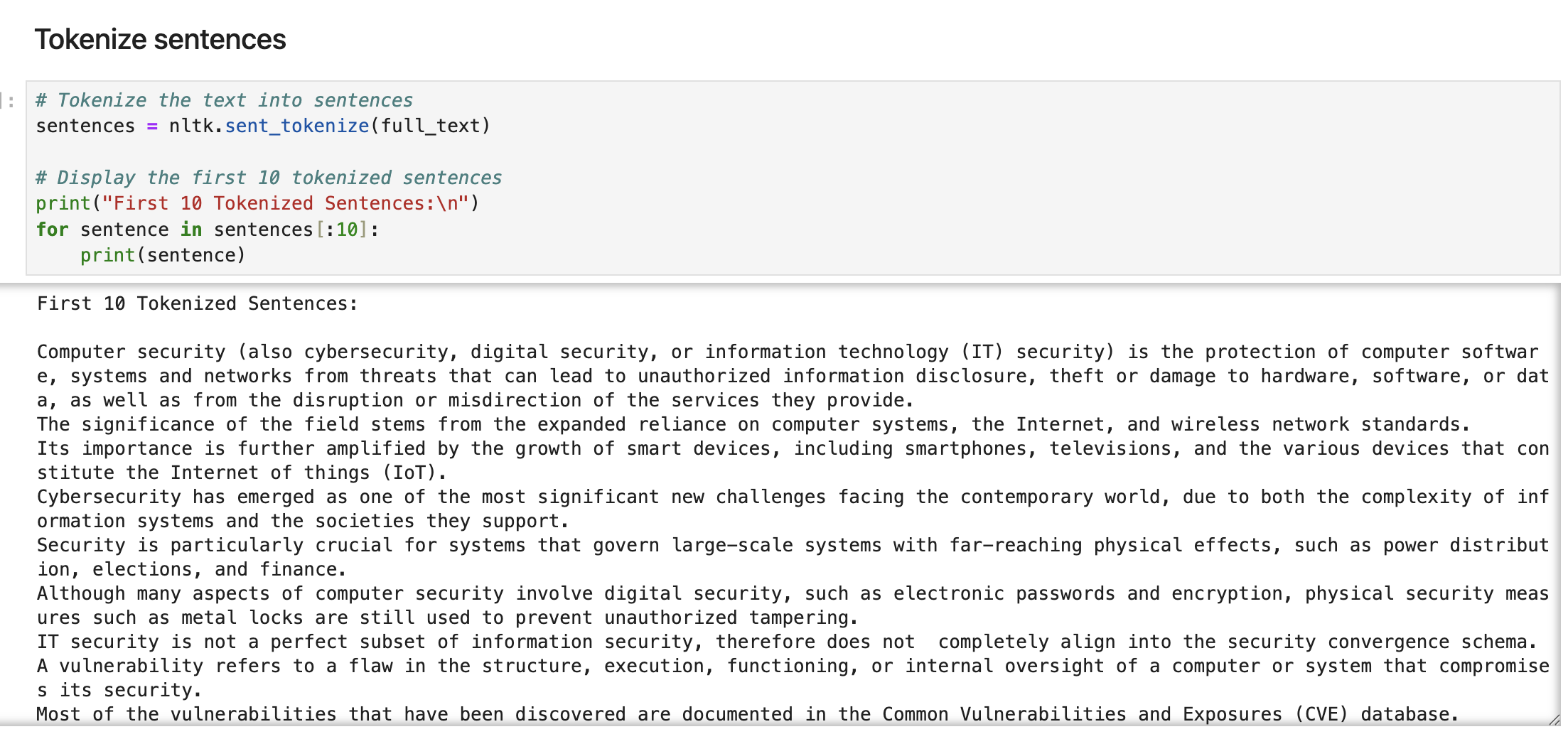
## Tools & Library Used and Purpose Of them:

* **JupyterLab**: Chosen for its interactive environment, which allows step-by-step execution and visualization of results, making it ideal for experimenting with summarization techniques.
* **Python 3**: Provides extensive support for data manipulation, text processing, and integration with various ML libraries, ensuring flexibility and scalability.
* **nltk**: Used for tokenization, stop word removal, and Part-of-Speech (PoS) tagging, which are essential for preprocessing text and identifying key words or phrases.
* **requests**: Facilitates easy web scraping by sending HTTP requests to extract data from online sources for use in the project.
* **BeautifulSoup**: Employed for parsing and navigating HTML documents to extract clean text from the webpage content.
* **collections**: Provides the Counter and defaultdict tools to efficiently calculate word frequencies and manage dictionary-based data structures.
* **math**: Used to compute logarithmic functions for calculating Inverse Document Frequency (IDF), a critical component of the TF-IDF model.
* **scikit-learn**: Supports vectorization and cosine similarity computations for sentence comparison, enabling advanced summarization techniques like PageRank. (Approach 2)
* **networkx**: Implements graph-based methods, such as PageRank, to rank sentences based on their importance. (Approach 2)
* **re**: Used for regular expressions to sanitize the text by removing unwanted characters, such as reference numbers.



## Process Steps: **Approach 1 - Using IDF TF-IDF Along with POS (Part-of-Speech) Tagging**

### Tokenize sentences In this assignment, I used sentence tokenization to break down the text into individual sentences. This step was done using the **nltk.sent\_tokenize** function, which is part of the nltk library. Sentence tokenization is important because it allows us to treat each sentence as a separate unit, making it easier to analyze, rank, and eventually use for summarization. The function is capable of identifying sentence boundaries based on punctuation and language rules, ensuring that the text is split accurately. By tokenizing the text into sentences, I created a foundation for the later stages of the assignment, like calculating word frequencies, TF-IDF scores, and ranking sentences for the final summary.



Create frequency matrix of words in each sentence

For this assignment, I created a frequency matrix to calculate how often each word appears in each sentence. This was done by tokenizing each sentence into words using the nltk.word\_tokenize function and then using the Counter class from the collections library to count word occurrences. I also ensured that all words were converted to lowercase to avoid duplication caused by case sensitivity and removed non-alphanumeric tokens to focus only on meaningful words.

The frequency matrix serves as the foundation for calculating term frequency (TF) in later steps. By keeping track of word occurrences in each sentence, this step helps identify the most important words within the context of individual sentences, which is crucial for building a meaningful summary.



Calculate Term Frequency and Generate matrix

In this step, I calculated the **Term Frequency (TF)** for each word in every sentence. TF measures how often a word appears in a sentence relative to the total number of words in that sentence. This was done using the frequency matrix created earlier, where I divided the frequency of each word in a sentence by the total number of words in that sentence.

The resulting TF matrix provides normalized values, making it easier to compare the importance of words across sentences of varying lengths. This normalization ensures that longer sentences do not unfairly dominate the analysis. The TF matrix plays a critical role in determining the importance of words within each sentence and is a key component for calculating the TF-IDF scores used later in the assignment.

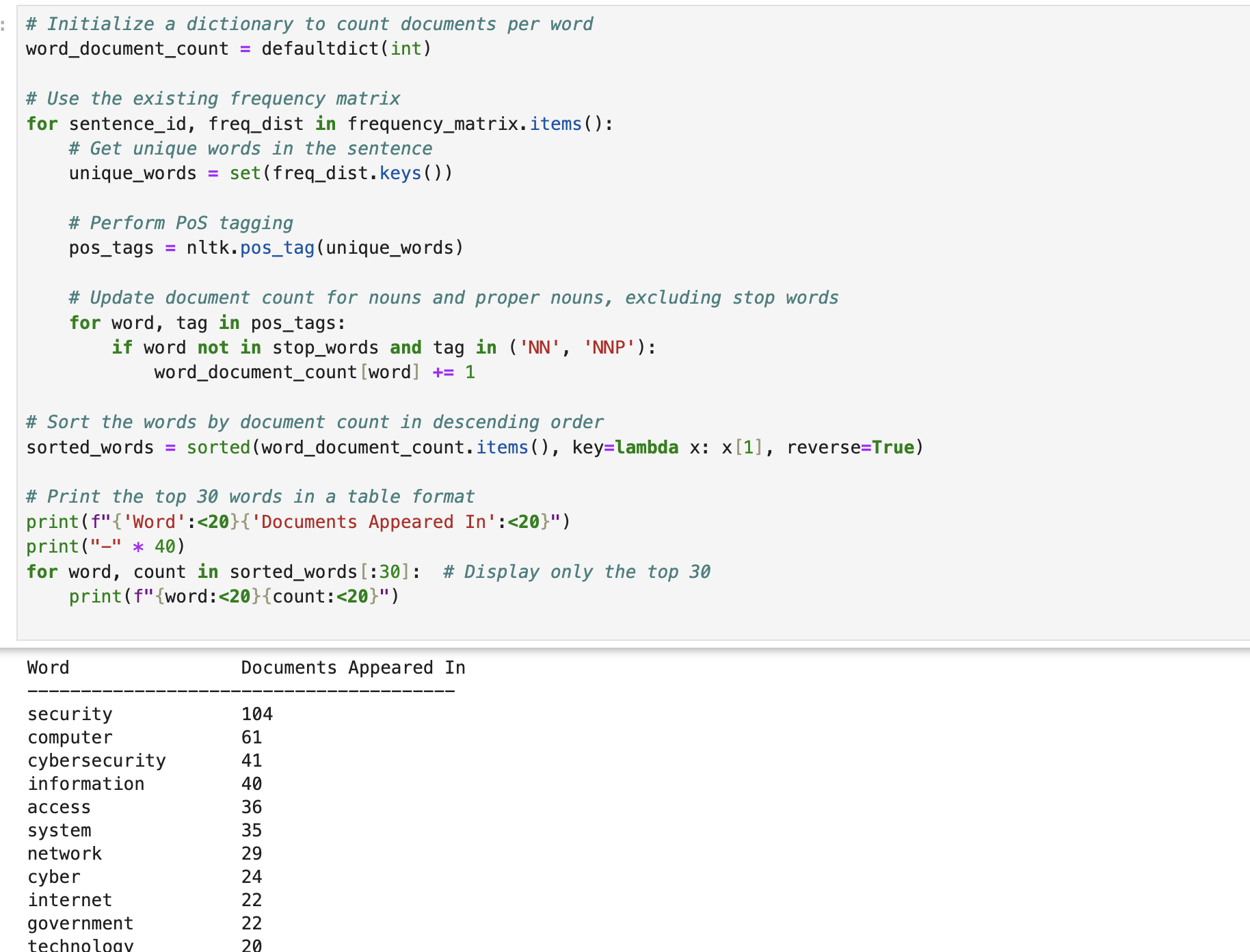


Create a table for documents per words

In this step, I created a table that tracks how many sentences (documents) each word appears in. Using the frequency matrix from the earlier steps, I identified the unique words in each sentence and counted the number of sentences in which each word appeared. This was achieved using a **defaultdict** from the collections library.

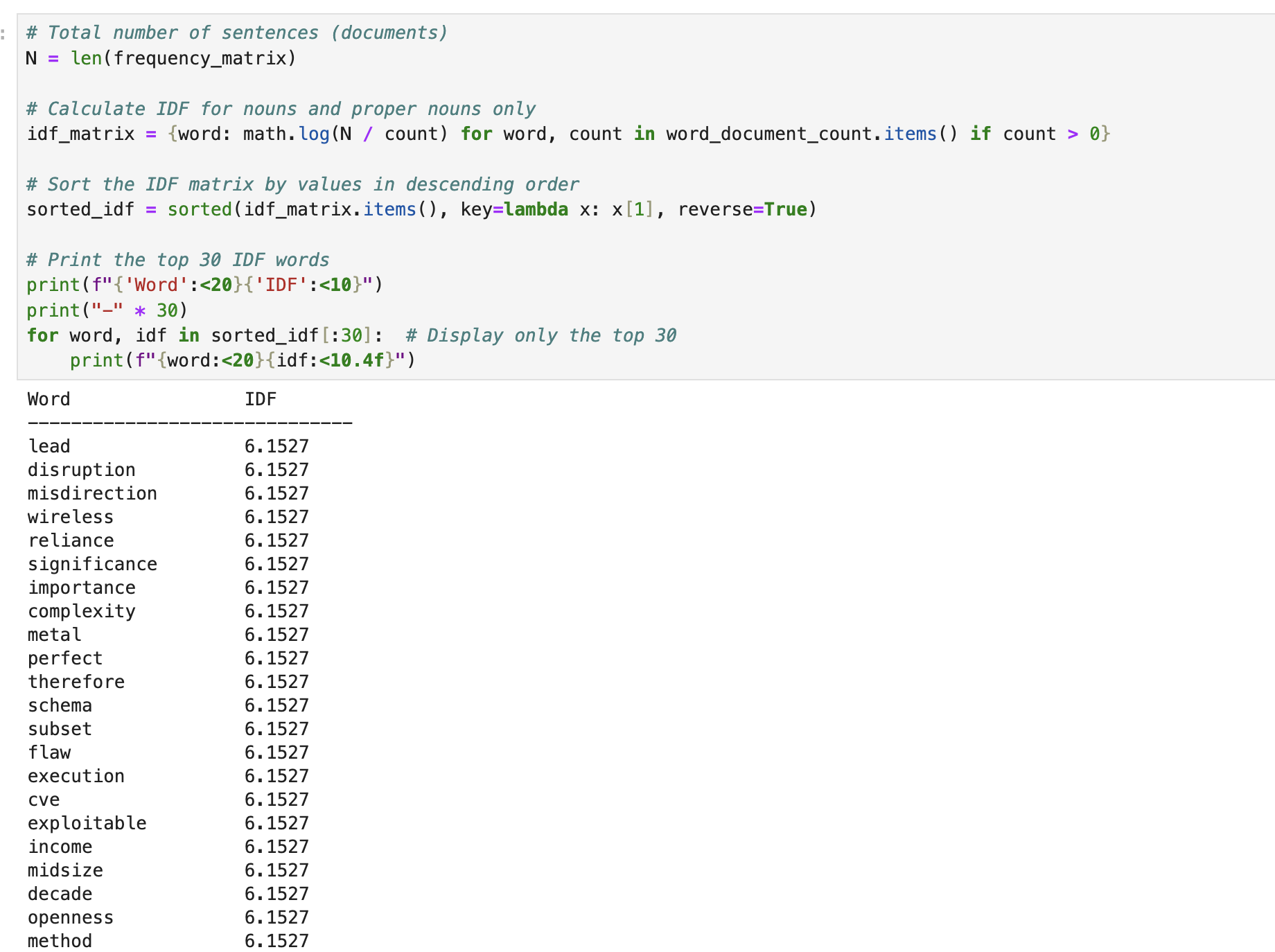
To ensure the results were meaningful, I excluded stop words (common words like "and" or "the") during this process. The resulting table provides a count of sentences for each word, which is essential for calculating the **Inverse Document Frequency (IDF)** in the next step. By identifying the spread of words across sentences, this table helps determine which words are specific to certain parts of the text and which ones are more generic, contributing to the overall summarization process.

Here I have also used the PoS(Part of speech) technique to identify more relevant words by adding weightage.



Calculate IDF and generate matrix

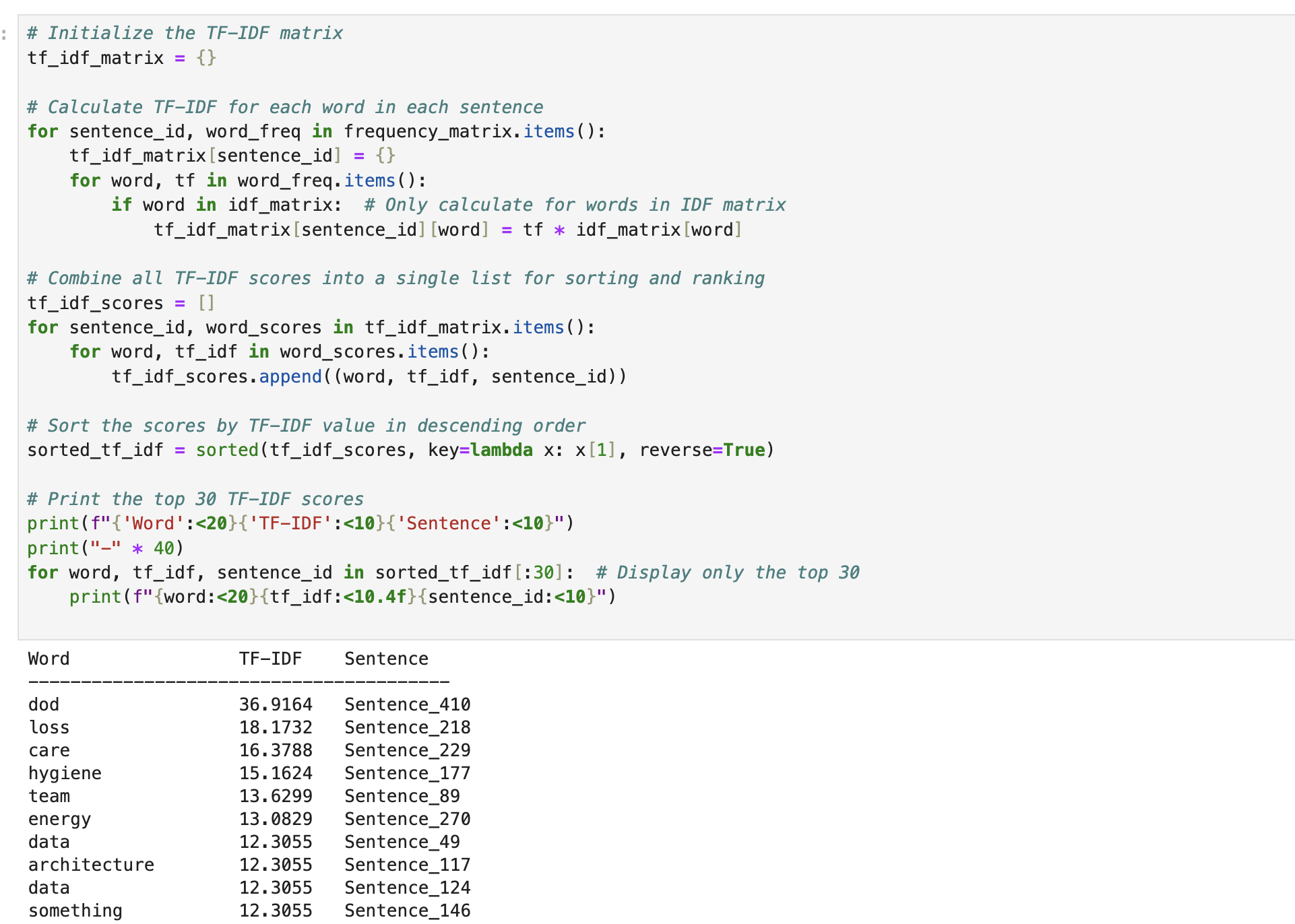
In this step, I calculated the **Inverse Document Frequency (IDF)** for each word to measure its uniqueness across all sentences (documents). Words that appear in many sentences are considered less important and are given lower IDF values, while words that appear in only a few sentences are assigned higher IDF values. This step helps differentiate between generic words and contextually significant ones. Using the total number of sentences and the count of sentences each word appears in, I generated an IDF matrix. This matrix assigns a weight to each word, which is crucial for the subsequent TF-IDF calculation.



Calculate TF-IDF and generate matrix

In this step, I combined the **Term Frequency (TF)** and **Inverse Document Frequency (IDF)** values to calculate the **TF-IDF score** for each word in every sentence. TF-IDF highlights the importance of a word in a specific sentence while accounting for its rarity across all sentences. This ensures that commonly used words, even if frequent within a sentence, do not dominate the analysis.

To compute the TF-IDF, I multiplied the TF value of a word in a sentence by its corresponding IDF value from the matrix generated earlier. The resulting TF-IDF matrix provided a detailed representation of the significance of each word in each sentence, serving as the foundation for scoring and ranking sentences in the summarization process.



Score the sentences

In this step, I calculated a score for each sentence based on the **TF-IDF scores** of the words it contains. For every sentence, the scores of all its words were summed up to determine the overall importance of that sentence. Sentences with higher scores were considered more significant as they contained words that were both frequent within the sentence and unique across the document.

This scoring process allowed me to rank the sentences based on their relevance to the overall content, providing a quantitative basis for selecting sentences to include in the summary. The sentence scores formed a key part of the thresholding and selection process in the final stages of summarization.

A screenshot of a computer program

Description automatically generated

Find the threshold

In this step, I determined the threshold scores to decide which sentences should be included in the summary. Two thresholds were calculated:

1. **Mean Threshold**:
   * The average score of all sentences was computed. Sentences with scores above this mean were considered significant and eligible for inclusion.
2. **Max Threshold**:
   * A percentage of the highest sentence score (e.g., 70%) was used as a second threshold. This ensured that only the most impactful sentences, relative to the highest scoring one, were considered.

By applying these thresholds, I filtered out less relevant sentences while focusing on those that were more meaningful. This dynamic approach ensures that the summary adapts to the distribution of sentence scores in the document, balancing quality and conciseness.

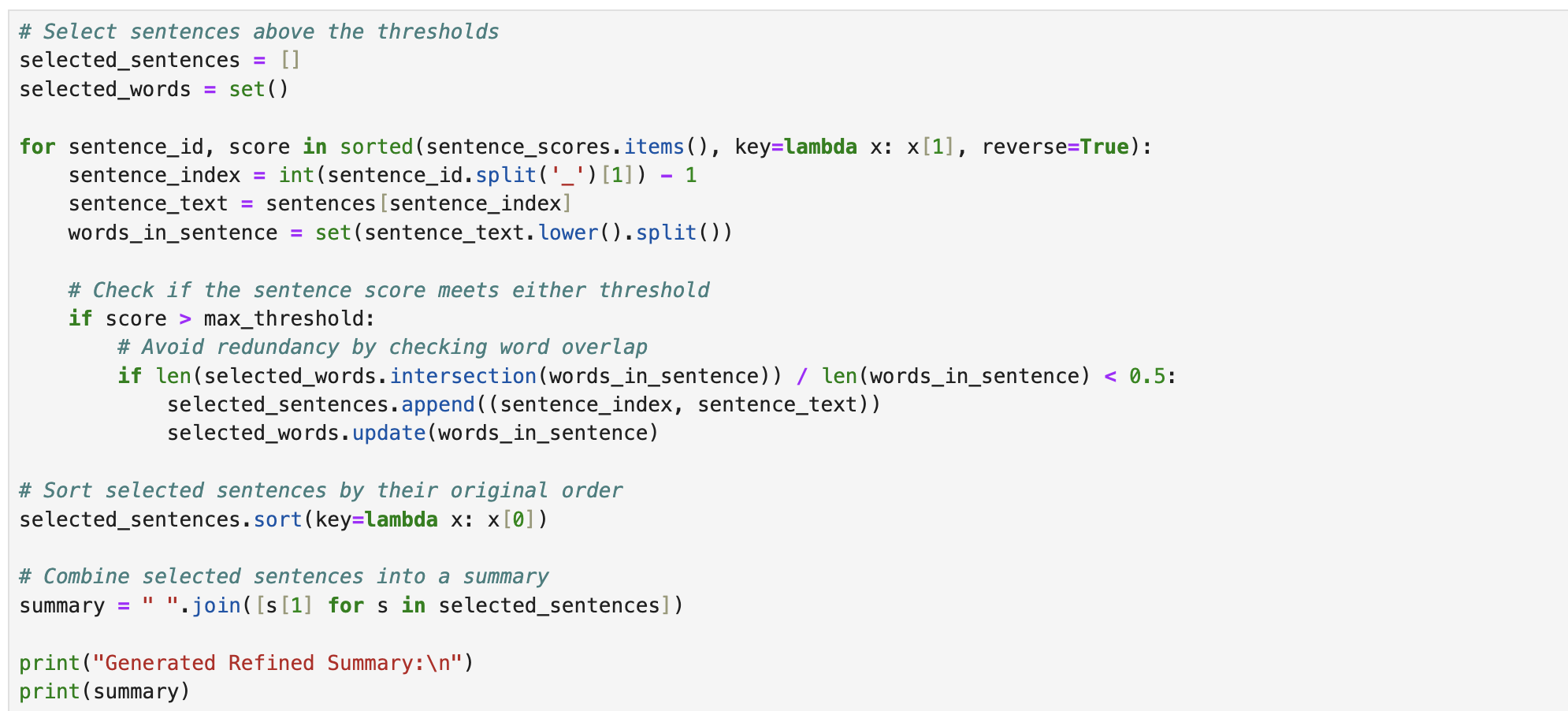
A screenshot of a computer code

Description automatically generated

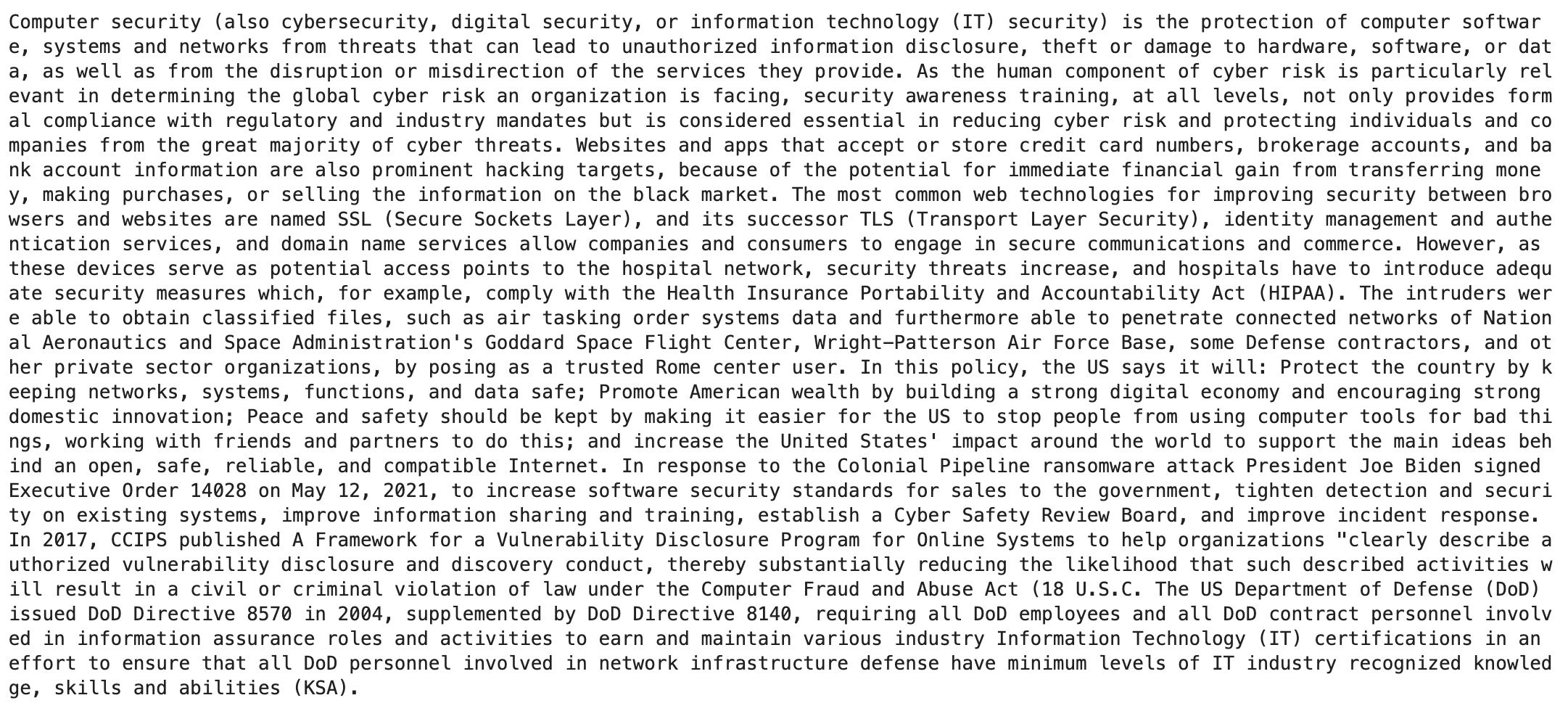
Generate the summary

In the final step, I generated the summary by selecting sentences that met the thresholds determined earlier. Sentences with scores above the **max threshold** were included in the summary. To avoid redundancy, a word overlap check was performed, ensuring that selected sentences contained minimal repetition of content.

The selected sentences were then sorted in the order they appeared in the original text to maintain coherence and logical flow. Finally, these sentences were combined into a concise and meaningful summary that effectively retained the key information from the original document. This step completed the summarization process, providing a refined output that highlights the most important aspects of the text.



**Generated Summary : (WiKi Had Around 40 to 50 Pages)**



## Process Steps: **Approach 2 - Using TF-IDF Vectorization Along with Page Ranking Algorithm and Cosine Similarity**

After implementing Approach 1, which relied on TF-IDF and word-based scoring, I realized that while it effectively identified key sentences, it didn't fully capture the relationships between sentences or their contextual relevance. To address this, I introduced **Approach 2**, which leverages TF-IDF with **cosine similarity** and the **PageRank algorithm**.

In Approach 2, sentences are represented as vectors using TF-IDF, and their semantic similarity is calculated to build a graph where each sentence is a node, and edges represent their similarity. Using the PageRank algorithm, I ranked sentences based on their connections and overall importance within the graph. This approach was taken to improve the coherence of the summary and ensure that the selected sentences not only contained high-value words but also complemented each other in terms of context.

The input remains same from the previous process of extracted full text.



**Generated Summary :**

## 

Conclusion:

In this assignment, I explored two approaches for text summarization, each leveraging different techniques to generate concise and meaningful summaries. Approach 1 focused on TF-IDF and Part-of-Speech (PoS) tagging to rank sentences based on word importance and contextual relevance. This method provided a solid foundation by identifying high-value sentences through frequency analysis. However, it had limitations in capturing the relationships between sentences, which sometimes affected the coherence of the summary.

To overcome this, Approach 2 introduced a graph-based technique using cosine similarity and the PageRank algorithm. By representing sentences as nodes in a graph and ranking them based on their interconnections, this approach ensured a more cohesive summary. The graph-based ranking captured both sentence importance and their semantic relationships, resulting in summaries that were not only relevant but also well-structured.

Upon comparing the results, Approach 2 consistently produced summaries that were more contextually connected and easier to read. While Approach 1 excelled in identifying key sentences based on individual word significance, it occasionally lacked the coherence achieved by Approach 2. The data-driven insights from both approaches highlight the importance of combining word-level analysis with sentence-level relationships to achieve high-quality text summarization. This iterative exploration underscores the value of experimenting with different techniques to refine and improve outcomes.