# 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:

```
url = "https://en.wikipedia.org/wiki/Computer_security"
response = requests.get(url)

if response.status_code == 200:
    # Parse the HTML content of the page
    soup = BeautifulSoup(response.content, 'html.parser')

    # Find the main content of the Wikipedia page | This is to ignore additional references and sidebars --> Data Mining
    content_div = soup.find('div', {'class': 'mw-parser-output'})

    # Extract all text within the main content
    paragraphs = content_div.find_all('p')
    text_content = []

    for para in paragraphs:
        text = para.get_text()
        # Add a space to separate paragraphs
        if text.strip():
            text_content.append(text.strip())

    # Join the paragraphs
    full_text = "\n\n".join(text_content)

    # Remove reference numberings like [1], [2], etc. --> Data Sanitization
    full_text = re.sub(r'\[\d+\]', '', full_text)

    # Display only the first 100 lines
    lines = full_text.split('\n')  # Split text into lines
    first_30_lines = lines[:30]  # Get the first 100 lines

    # Combine the lines and add ellipsis for truncation
    preview_text = '\n'.join(first_30_lines) + "\n\n... (Content Truncated for Display) ..."

    # Print the preview text for first 30 lines
    print("Preview of Scraped Text first 30 lines:\n")
    print(preview_text)
else:
    print(f"Failed to fetch the webpage. Status code: {response.status_code}")
```

## 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.

### Import of Tools & Libraries

```
import requests
from bs4 import BeautifulSoup
import re

import nltk
# NLTK tokenizer models
nltk.download('punkt')
nltk.download('punkt_tab')

from collections import Counter
from collections import defaultdict

from nltk.corpus import stopwords

nltk.download('stopwords')
# Load NLTK's list of stop words
stop_words = set(stopwords.words('english'))

import math
nltk.download('averaged_perceptron_tagger')
```

## 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.

Tokenize sentences

```
# Tokenize the text into sentences
sentences = nltk.sent_tokenize(full_text)

# Display the first 10 tokenized sentences
print("First 10 Tokenized Sentences:\n")
for sentence in sentences[:10]:
    print(sentence)
```

```
First 10 Tokenized Sentences:

Computer security (also cybersecurity, digital security, or information technology (IT) security) is the protection of computer softwar
e, systems and networks from threats that can lead to unauthorized information disclosure, theft or damage to hardware, software, or dat
a, as well as from the disruption or misdirection of the services they provide.
The significance of the field stems from the expanded reliance on computer systems, the Internet, and wireless network standards.
Its importance is further amplified by the growth of smart devices, including smartphones, televisions, and the various devices that con
stitute the Internet of things (IoT).
Cybersecurity has emerged as one of the most significant new challenges facing the contemporary world, due to both the complexity of inf
ormation systems and the societies they support.
Security is particularly crucial for systems that govern large-scale systems with far-reaching physical effects, such as power distribut
ion, elections, and finance.
Although many aspects of computer security involve digital security, such as electronic passwords and encryption, physical security meas
ures such as metal locks are still used to prevent unauthorized tampering.
IT security is not a perfect subset of information security, therefore does not  completely align into the security convergence schema.
A vulnerability refers to a flaw in the structure, execution, functioning, or internal oversight of a computer or system that compromise
s its security.
Most of the vulnerabilities that have been discovered are documented in the Common Vulnerabilities and Exposures (CVE) database.
```

## 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.

```
# Initialize the frequency matrix
frequency_matrix = {}

for idx, sentence in enumerate(sentences):
    # Tokenize the sentence into words
    words = nltk.word_tokenize(sentence)

    # Convert words to lowercase and filter out non-alphanumeric tokens
    words = [word.lower() for word in words if word.isalnum()]

    # Create a frequency distribution for the sentence
    frequency_matrix[f"Sentence_{idx+1}"] = Counter(words)

print("Preview of Frequency Matrix:\n")
preview_sentences = list(frequency_matrix.items())[:5]  # Limit display to the first 5 sentences
for sentence_id, freq_dist in preview_sentences:
    print(f"{sentence_id}: {dict(freq_dist)}")

# Add truncation message
print("\n... (Content Truncated for Display) ...")
```

```
Preview of Frequency Matrix:

Sentence_1: {'computer': 2, 'security': 3, 'also': 1, 'cybersecurity': 1, 'digital': 1, 'or': 4, 'information': 2, 'technology': 1, 'i
t': 1, 'is': 1, 'the': 3, 'protection': 1, 'of': 2, 'software': 2, 'systems': 1, 'and': 1, 'networks': 1, 'from': 2, 'threats': 1, 'tha
t': 1, 'can': 1, 'lead': 1, 'to': 2, 'unauthorized': 1, 'disclosure': 1, 'theft': 1, 'damage': 1, 'hardware': 1, 'data': 1, 'as': 2, 'we
ll': 1, 'disruption': 1, 'misdirection': 1, 'services': 1, 'they': 1, 'provide': 1}
Sentence_2: {'the': 4, 'significance': 1, 'of': 1, 'field': 1, 'stems': 1, 'from': 1, 'expanded': 1, 'reliance': 1, 'on': 1, 'computer':
1, 'systems': 1, 'internet': 1, 'and': 1, 'wireless': 1, 'network': 1, 'standards': 1}
Sentence_3: {'its': 1, 'importance': 1, 'is': 1, 'further': 1, 'amplified': 1, 'by': 1, 'the': 3, 'growth': 1, 'of': 2, 'smart': 1, 'dev
ices': 2, 'including': 1, 'smartphones': 1, 'televisions': 1, 'and': 1, 'various': 1, 'that': 1, 'constitute': 1, 'internet': 1, 'thing
s': 1, 'iot': 1}
Sentence_4: {'cybersecurity': 1, 'has': 1, 'emerged': 1, 'as': 1, 'one': 1, 'of': 2, 'the': 4, 'most': 1, 'significant': 1, 'new': 1, 'c
hallenges': 1, 'facing': 1, 'contemporary': 1, 'world': 1, 'due': 1, 'to': 1, 'both': 1, 'complexity': 1, 'information': 1, 'systems':
1, 'and': 1, 'societies': 1, 'they': 1, 'support': 1}
```

## 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.

```
# Initialize the TF matrix
tf_matrix = {}

for idx, sentence in enumerate(sentences):
    # Tokenize the sentence into words
    words = nltk.word_tokenize(sentence)

    # Convert words to lowercase and filter out non-alphanumeric tokens
    words = [word.lower() for word in words if word.isalnum()]

    # Calculate word frequencies in the sentence
    word_freq = Counter(words)
    total_words = len(words)  # Total words in the sentence

    # Calculate term frequency (TF) for each word
    tf_matrix[f"Sentence_{idx+1}"] = {word: freq / total_words for word, freq in word_freq.items()}

print("Preview of Term Frequency (TF) Matrix:\n")
preview_tf = list(tf_matrix.items())[:5]  # Limit to the first 5 sentences
for sentence_id, tf_dict in preview_tf:
    print(f"{sentence_id}: {tf_dict}")

print("\n... (Content Truncated for Display) ...")
```

```
Preview of Term Frequency (TF) Matrix:

Sentence_1: {'computer': 0.04, 'security': 0.06, 'also': 0.02, 'cybersecurity': 0.02, 'digital': 0.02, 'or': 0.08, 'information': 0.04,
'technology': 0.02, 'it': 0.02, 'is': 0.02, 'the': 0.06, 'protection': 0.02, 'of': 0.04, 'software': 0.04, 'systems': 0.02, 'and': 0.02,
'networks': 0.02, 'from': 0.04, 'threats': 0.02, 'that': 0.02, 'can': 0.02, 'lead': 0.02, 'to': 0.04, 'unauthorized': 0.02, 'disclosur
e': 0.02, 'theft': 0.02, 'damage': 0.02, 'hardware': 0.02, 'data': 0.02, 'as': 0.04, 'well': 0.02, 'disruption': 0.02, 'misdirection':
0.02, 'services': 0.02, 'they': 0.02, 'provide': 0.02}
Sentence_2: {'the': 0.21052631578947367, 'significance': 0.05263157894736842, 'of': 0.05263157894736842, 'field': 0.05263157894736842,
'stems': 0.05263157894736842, 'from': 0.05263157894736842, 'expanded': 0.05263157894736842, 'reliance': 0.05263157894736842, 'on': 0.052
63157894736842, 'computer': 0.05263157894736842, 'systems': 0.05263157894736842, 'internet': 0.05263157894736842, 'and': 0.0526315789473
6842, 'wireless': 0.05263157894736842, 'network': 0.05263157894736842, 'standards': 0.05263157894736842}
Sentence_3: {'its': 0.04, 'importance': 0.04, 'is': 0.04, 'further': 0.04, 'amplified': 0.04, 'by': 0.04, 'the': 0.12, 'growth': 0.04,
'of': 0.08, 'smart': 0.04, 'devices': 0.08, 'including': 0.04, 'smartphones': 0.04, 'televisions': 0.04, 'and': 0.04, 'various': 0.04,
'that': 0.04, 'constitute': 0.04, 'internet': 0.04, 'things': 0.04, 'iot': 0.04}
```

## 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.

```
# Initialize a dictionary to count documents per word
word_document_count = defaultdict(int)

# Use the existing frequency matrix
for sentence_id, freq_dist in frequency_matrix.items():
    # Get unique words in the sentence
    unique_words = set(freq_dist.keys())

    # Perform PoS tagging
    pos_tags = nltk.pos_tag(unique_words)

    # Update document count for nouns and proper nouns, excluding stop words
    for word, tag in pos_tags:
        if word not in stop_words and tag in ('NN', 'NNP'):
            word_document_count[word] += 1

# Sort the words by document count in descending order
sorted_words = sorted(word_document_count.items(), key=lambda x: x[1], reverse=True)

# Print the top 30 words in a table format
print(f"{'Word':<20}{'Documents Appeared In':<20}")
print("-" * 40)
for word, count in sorted_words[:30]:  # Display only the top 30
    print(f"{word:<20}{count:<20}")
```

```
Word                Documents Appeared In
----------------------------------------
security            104
computer            61
cybersecurity       41
information         40
access              36
system              35
network             29
cyber               24
internet            22
government          22
technology          20
```

## 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.

```
# Total number of sentences (documents)
N = len(frequency_matrix)

# Calculate IDF for nouns and proper nouns only
idf_matrix = {word: math.log(N / count) for word, count in word_document_count.items() if count > 0}

# Sort the IDF matrix by values in descending order
sorted_idf = sorted(idf_matrix.items(), key=lambda x: x[1], reverse=True)

# Print the top 30 IDF words
print(f"{'Word':<20}{'IDF':<10}")
print("-" * 30)
for word, idf in sorted_idf[:30]:  # Display only the top 30
    print(f"{word:<20}{idf:<10.4f}")
```

```
Word                IDF
------------------------------
lead                6.1527
disruption          6.1527
misdirection        6.1527
wireless            6.1527
reliance            6.1527
significance        6.1527
importance          6.1527
complexity          6.1527
metal               6.1527
perfect             6.1527
therefore           6.1527
schema              6.1527
subset              6.1527
flaw                6.1527
execution           6.1527
cve                 6.1527
exploitable         6.1527
income              6.1527
midsize             6.1527
decade              6.1527
openness            6.1527
method              6.1527
```

## 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.

```
: # Initialize the TF-IDF matrix
tf_idf_matrix = {}

# Calculate TF-IDF for each word in each sentence
for sentence_id, word_freq in frequency_matrix.items():
    tf_idf_matrix[sentence_id] = {}
    for word, tf in word_freq.items():
        if word in idf_matrix:  # Only calculate for words in IDF matrix
            tf_idf_matrix[sentence_id][word] = tf * idf_matrix[word]

# Combine all TF-IDF scores into a single list for sorting and ranking
tf_idf_scores = []
for sentence_id, word_scores in tf_idf_matrix.items():
    for word, tf_idf in word_scores.items():
        tf_idf_scores.append((word, tf_idf, sentence_id))

# Sort the scores by TF-IDF value in descending order
sorted_tf_idf = sorted(tf_idf_scores, key=lambda x: x[1], reverse=True)

# Print the top 30 TF-IDF scores
print(f"{'Word':<20}{'TF-IDF':<10}{'Sentence':<10}")
print("-" * 40)
for word, tf_idf, sentence_id in sorted_tf_idf[:30]:  # Display only the top 30
    print(f"{word:<20}{tf_idf:<10.4f}{sentence_id:<10}")
```

```
Word                TF-IDF    Sentence
----------------------------------------
dod                 36.9164   Sentence_410
loss                18.1732   Sentence_218
care                16.3788   Sentence_229
hygiene             15.1624   Sentence_177
team                13.6299   Sentence_89
energy              13.0829   Sentence_270
data                12.3055   Sentence_49
architecture        12.3055   Sentence_117
data                12.3055   Sentence_124
something           12.3055   Sentence_146
```

## 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.

```
sentence_scores = {}

# Calculate the score for each sentence
for sentence_id, word_scores in tf_idf_matrix.items():
    # Sum all TF-IDF scores of words in the sentence
    sentence_scores[sentence_id] = sum(word_scores.values())

# Sort the sentences by their scores in descending order
sorted_sentences = sorted(sentence_scores.items(), key=lambda x: x[1], reverse=True)

# Print the top 5 scored sentences
print(f"{'Sentence':<15}{'Score':<10}")
print("-" * 30)
for sentence_id, score in sorted_sentences[:5]:  # Display top 5 sentences
    print(f"{sentence_id:<15}{score:<10.4f}")
```

```
Sentence       Score
------------------------------
Sentence_410   106.8150
Sentence_382   102.0011
Sentence_402   96.2920
Sentence_313   90.6078
Sentence_391   90.3172
```

## 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.

```
# Calculate thresholds
mean_threshold = sum(sentence_scores.values()) / len(sentence_scores)
max_threshold = max(sentence_scores.values()) * 0.7

print(f"Mean Threshold: {mean_threshold:.4f}")
print(f"70% of Max Threshold: {max_threshold:.4f}")
```

```
Mean Threshold: 29.0885
70% of Max Threshold: 74.7705
```

## 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.

```
# Select sentences above the thresholds
selected_sentences = []
selected_words = set()

for sentence_id, score in sorted(sentence_scores.items(), key=lambda x: x[1], reverse=True):
    sentence_index = int(sentence_id.split('_')[1]) - 1
    sentence_text = sentences[sentence_index]
    words_in_sentence = set(sentence_text.lower().split())

    # Check if the sentence score meets either threshold
    if score > max_threshold:
        # Avoid redundancy by checking word overlap
        if len(selected_words.intersection(words_in_sentence)) / len(words_in_sentence) < 0.5:
            selected_sentences.append((sentence_index, sentence_text))
            selected_words.update(words_in_sentence)

# Sort selected sentences by their original order
selected_sentences.sort(key=lambda x: x[0])

# Combine selected sentences into a summary
summary = " ".join([s[1] for s in selected_sentences])

print("Generated Refined Summary:\n")
print(summary)
```

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

```
Computer security (also cybersecurity, digital security, or information technology (IT) security) is the protection of computer softwar
e, systems and networks from threats that can lead to unauthorized information disclosure, theft or damage to hardware, software, or dat
a, as well as from the disruption or misdirection of the services they provide. As the human component of cyber risk is particularly rel
evant in determining the global cyber risk an organization is facing, security awareness training, at all levels, not only provides form
al compliance with regulatory and industry mandates but is considered essential in reducing cyber risk and protecting individuals and co
mpanies from the great majority of cyber threats. Websites and apps that accept or store credit card numbers, brokerage accounts, and ba
nk account information are also prominent hacking targets, because of the potential for immediate financial gain from transferring mone
y, making purchases, or selling the information on the black market. The most common web technologies for improving security between bro
wsers and websites are named SSL (Secure Sockets Layer), and its successor TLS (Transport Layer Security), identity management and authe
ntication services, and domain name services allow companies and consumers to engage in secure communications and commerce. However, as
these devices serve as potential access points to the hospital network, security threats increase, and hospitals have to introduce adequ
ate security measures which, for example, comply with the Health Insurance Portability and Accountability Act (HIPAA). The intruders wer
e able to obtain classified files, such as air tasking order systems data and furthermore able to penetrate connected networks of Nation
al Aeronautics and Space Administration's Goddard Space Flight Center, Wright-Patterson Air Force Base, some Defense contractors, and ot
her private sector organizations, by posing as a trusted Rome center user. In this policy, the US says it will: Protect the country by k
eeping networks, systems, functions, and data safe; Promote American wealth by building a strong digital economy and encouraging strong
domestic innovation; Peace and safety should be kept by making it easier for the US to stop people from using computer tools for bad thi
ngs, working with friends and partners to do this; and increase the United States' impact around the world to support the main ideas beh
ind an open, safe, reliable, and compatible Internet. In response to the Colonial Pipeline ransomware attack President Joe Biden signed
Executive Order 14028 on May 12, 2021, to increase software security standards for sales to the government, tighten detection and securi
ty on existing systems, improve information sharing and training, establish a Cyber Safety Review Board, and improve incident response.
In 2017, CCIPS published A Framework for a Vulnerability Disclosure Program for Online Systems to help organizations "clearly describe a
uthorized vulnerability disclosure and discovery conduct, thereby substantially reducing the likelihood that such described activities w
ill result in a civil or criminal violation of law under the Computer Fraud and Abuse Act (18 U.S.C. The US Department of Defense (DoD)
issued DoD Directive 8570 in 2004, supplemented by DoD Directive 8140, requiring all DoD employees and all DoD contract personnel involv
ed in information assurance roles and activities to earn and maintain various industry Information Technology (IT) certifications in an
effort to ensure that all DoD personnel involved in network infrastructure defense have minimum levels of IT industry recognized knowled
ge, skills and abilities (KSA).
```

## 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.

```
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import networkx as nx

# Step 1: Tokenize the text into sentences
sentences = nltk.sent_tokenize(full_text)

# Step 2: Generate the TF-IDF matrix for sentence similarity
vectorizer = TfidfVectorizer().fit_transform(sentences)
similarity_matrix = cosine_similarity(vectorizer)

# Step 3: Build the sentence graph
sentence_graph = nx.from_numpy_array(similarity_matrix)

# Step 4: Rank sentences using the PageRank algorithm
scores = nx.pagerank(sentence_graph)

# Step 5: Sort sentences by score
ranked_sentences = sorted(((scores[i], s) for i, s in enumerate(sentences)), reverse=True)

# Step 6: Select the top N sentences for the summary
N = 10  # Number of sentences in the summary
summary = " ".join([s for _, s in ranked_sentences[:N]])

# Print the improved summary
print("Generated Summary:\n")
print(summary)
```

**Generated Summary :**

```
Computer security (also cybersecurity, digital security, or information technology (IT) security) is the protection of computer softwar
e, systems and networks from threats that can lead to unauthorized information disclosure, theft or damage to hardware, software, or dat
a, as well as from the disruption or misdirection of the services they provide. In practice, the role of a security architect would be t
o ensure the structure of a system reinforces the security of the system, and that new changes are safe and meet the security requiremen
ts of the organization. To secure a computer system, it is important to understand the attacks that can be made against it, and these th
reats can typically be classified into one of the following categories:

A backdoor in a computer system, a cryptosystem, or an algorithm is any secret method of bypassing normal authentication or security con
trols. The role of the government is to make regulations to force companies and organizations to protect their systems, infrastructure a
nd information from any cyberattacks, but also to protect its own national infrastructure such as the national power-grid. Spoofing is a
n act of pretending to be a valid entity through the falsification of data (such as an IP address or username), in order to gain access
to information or resources that one is otherwise unauthorized to obtain. The National Cyber Security Policy 2013 is a policy framework
by the Ministry of Electronics and Information Technology (MeitY) which aims to protect the public and private infrastructure from cyber
attacks, and safeguard "information, such as personal information (of web users), financial and banking information and sovereign data".
In Side-channel attack scenarios, the attacker would gather such information about a system or network to guess its internal state and a
s a result access the information which is assumed by the victim to be secure. In this policy, the US says it will: Protect the country
by keeping networks, systems, functions, and data safe; Promote American wealth by building a strong digital economy and encouraging str
ong domestic innovation; Peace and safety should be kept by making it easier for the US to stop people from using computer tools for bad
things, working with friends and partners to do this; and increase the United States' impact around the world to support the main ideas
behind an open, safe, reliable, and compatible Internet. In the United Kingdom, a nationwide set of cybersecurity forums, known as the
U.K Cyber Security Forum, were established supported by the Government's cybersecurity strategy in order to encourage start-ups and inno
vation and to address the skills gap identified by the U.K Government. Backdoors can be very hard to detect and are usually discovered b
y someone who has access to the application source code or intimate knowledge of the operating system of the computer.
```

## 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.