```python
import yt_dlp
import os

# Step 1: Download Audio from YouTube with Cookies
def download_audio(youtube_url, output_path="audio.mp3"):
    ydl_opts = {
        'format': 'bestaudio/best',
        'outtmpl': output_path.replace('.mp3', ''),  # Remove .mp3 for yt-dlp to handle extension
        'postprocessors': [{
            'key': 'FFmpegExtractAudio',
            'preferredcodec': 'mp3',
            'preferredquality': '192',
        }],
        'cookiefile': 'cookies.txt'  # Use cookies to bypass restrictions
    }

    with yt_dlp.YoutubeDL(ydl_opts) as ydl:
        ydl.download([youtube_url])

    # Fix the file name if it ends up as audio.mp3.mp3
    downloaded_file = output_path.replace('.mp3', '.mp3.mp3')
    if os.path.exists(downloaded_file):
        os.rename(downloaded_file, output_path)

# Replace with your YouTube link
youtube_url = "https://youtu.be/sK8SILOM37I"
download_audio(youtube_url)
```

```
[youtube] Extracting URL: https://youtu.be/sK8SILOM37I
[youtube] sK8SILOM37I: Downloading webpage
[youtube] sK8SILOM37I: Downloading tv client config
[youtube] sK8SILOM37I: Downloading player f6e09c70
[youtube] sK8SILOM37I: Downloading tv player API JSON
[youtube] sK8SILOM37I: Downloading ios player API JSON
[youtube] sK8SILOM37I: Downloading m3u8 information
[info] sK8SILOM37I: Downloading 1 format(s): 251
[download] Destination: audio
[download] 100% of   39.37MiB in 00:00:04 at 9.49MiB/s
[ExtractAudio] Destination: audio.mp3
Deleting original file audio (pass -k to keep)
```

```python
#!pip install yt_dlp
```

```python
from IPython.display import Audio

# Path to the downloaded audio file
audio_file = "audio.mp3"

# Play the audio
Audio(audio_file)
```

0:00 / 57:22

```python
#!pip install youtube_transcript_api
```

```python
#!pip install pydub
```

```python
#!pip install SpeechRecognition
```

```python
#!pip install pytube
```

```python
pip install deepmultilingualpunctuation
```

```
Collecting deepmultilingualpunctuation
  Downloading deepmultilingualpunctuation-1.0.1-py3-none-any.whl.metadata (4.0 kB)
Requirement already satisfied: torch>=1.8.1 in /usr/local/lib/python3.11/dist-packages (from deepmultilingualpunctuation) (2.5.1+
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (from deepmultilingualpunctuation) (4.48.3
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.1->deepmultilingualpunctuatio
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.1->deepmultil
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.1->deepmultilingualpunctuatio
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.1->deepmultilingualpunctuation
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.1->deepmultilingualpunctuation
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch>=1.8.1->deepmultilingualpunctuation)
  Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch>=1.8.1->deepmultilingualpunctuation)
  Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
```

```
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch>=1.8.1->deepmultilingualpunctuation)
  Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch>=1.8.1->deepmultilingualpunctuation)
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch>=1.8.1->deepmultilingualpunctuation)
  Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch>=1.8.1->deepmultilingualpunctuation)
  Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.5.147 (from torch>=1.8.1->deepmultilingualpunctuation)
  Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch>=1.8.1->deepmultilingualpunctuation)
  Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch>=1.8.1->deepmultilingualpunctuation)
  Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.1->deepmultil
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.1->deepmult
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch>=1.8.1->deepmultilingualpunctuation)
  Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Requirement already satisfied: triton==3.1.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.1->deepmultilingualpunct
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.1->deepmultilingualpunct
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch>=1.8.1->d
Requirement already satisfied: huggingface-hub<1.0,>=0.24.0 in /usr/local/lib/python3.11/dist-packages (from transformers->deepmu
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers->deepmultilingualpunctua
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers->deepmultilingualpun
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers->deepmultilingualpunctua
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers->deepmultilingual
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers->deepmultilingualpunctuatio
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers->deepmultilin
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.11/dist-packages (from transformers->deepmultilingual
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from transformers->deepmultilingualpunctuat
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch>=1.8.1->deepmultili
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->transformers->
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->transformers->deepmultilin
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->transformers->deepmu
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->transformers->deepmu
Downloading deepmultilingualpunctuation-1.0.1-py3-none-any.whl (5.4 kB)
Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl (363.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 3.7 MB/s eta 0:00:00
Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (13.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 22.0 MB/s eta 0:00:00
Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (24.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 20.5 MB/s eta 0:00:00
Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (883 kB)
```

```python
import re
import urllib.parse
import requests
from youtube_transcript_api import YouTubeTranscriptApi
from pytube import YouTube
import speech_recognition as sr
from pydub import AudioSegment
from deepmultilingualpunctuation import PunctuationModel
import os
# Initialize model once at the top
model = PunctuationModel()

def extract_video_id(video_url):
    """
    Extracts the YouTube video ID from various URL formats.
    """
    parsed_url = urllib.parse.urlparse(video_url)
    query_params = urllib.parse.parse_qs(parsed_url.query)

    if "v" in query_params:
        return query_params["v"][0]

    match = re.search(r"(youtu\.be/|youtube\.com/embed/|youtube\.com/shorts/)([\w-]+)", video_url)
    if match:
        return match.group(2)

    return None

def download_audio(video_url):
    """
    Downloads the audio using yt-dlp with cookies and returns the file path.
    """
    try:
        ydl_opts = {
            'format': 'bestaudio/best',
            'outtmpl': 'audio.%(ext)s',
            'cookiefile': 'cookies (1).txt',  # Use the exported cookies
            'postprocessors': [{
                'key': 'FFmpegExtractAudio',
                'preferredcodec': 'mp3',
                'preferredquality': '192',
            }],
        }
        with yt_dlp.YoutubeDL(ydl_opts) as ydl:
            info = ydl.extract_info(video_url, download=True)
            return "audio.mp3"
    except Exception as e:
        return f"Error downloading audio: {str(e)}"

def convert_audio_to_wav(audio_file):
    """
    Converts the downloaded MP3 audio to WAV format using pydub.
    """
    wav_file = "audio.wav"
    try:
        AudioSegment.from_mp3(audio_file).export(wav_file, format="wav")
        return wav_file
    except Exception as e:
        return f"Error converting to WAV: {str(e)}"

def transcribe_audio(audio_path, chunk_length=30):
    """
    Splits audio into smaller chunks, transcribes each chunk separately,
    and adds punctuation using deepmultilingualpunctuation library.
    """
    recognizer = sr.Recognizer()
    audio = AudioSegment.from_wav(audio_path)
    total_duration = len(audio) / 1000  # Convert to seconds
    transcribed_text = []

    # Load punctuation model
    model = PunctuationModel()

    print("Transcribing audio in chunks...")

    # In transcribe_audio()
    punctuated_chunks = []
    for chunk_text in transcribed_text:
        punctuated = model.restore_punctuation(chunk_text)
        punctuated_chunks.append(punctuated)
        return " ".join(punctuated_chunks)

    # Split and transcribe audio in chunks
    for start in range(0, int(total_duration), chunk_length):
        end = min(start + chunk_length, int(total_duration))
        chunk = audio[start * 1000:end * 1000]  # Extract chunk in milliseconds
        chunk.export("chunk.wav", format="wav")  # Save chunk temporarily

        with sr.AudioFile("chunk.wav") as source:
            try:
                audio_data = recognizer.record(source)
                text = recognizer.recognize_google(audio_data)
                transcribed_text.append(text)
            except sr.UnknownValueError:
                transcribed_text.append("[Unintelligible]")
            except sr.RequestError as e:
                return f"Error with the speech recognition service: {str(e)}"

    os.remove("chunk.wav")  # Clean up temporary chunk file

    # Combine chunks and add punctuation
    combined_text = " ".join(transcribed_text)
    punctuated_text = model.restore_punctuation(combined_text)

    return punctuated_text

def get_transcript_unlisted(video_url):
    """
    Tries to fetch the transcript using youtube_transcript_api first,
    then falls back to downloading and transcribing audio if necessary.
    """
    model = PunctuationModel()  # Initialize once
    video_id = extract_video_id(video_url)

    if not video_id:
        return "Invalid YouTube URL."

    # Try API path with punctuation
    try:
        transcript = YouTubeTranscriptApi.get_transcript(video_id)
        raw_text = " ".join([item['text'] for item in transcript])
        return model.restore_punctuation(raw_text)  # <-- Critical fix
    except:
        print("Transcript not available via API, attempting audio transcription...")

    # Audio fallback path (existing implementation)
    # ... rest of audio processing code ...
    # Download and transcribe audio if no transcript is available
    audio_file = download_audio(video_url)
    if "Error" in audio_file:
        return audio_file

    wav_file = convert_audio_to_wav(audio_file)
    if "Error" in wav_file:
        return wav_file

    transcription = transcribe_audio(wav_file)

    # Cleanup temporary files
    os.remove(audio_file)
    os.remove(wav_file)

    return transcription

# Example usage
# Example usage
if __name__ == "__main__":
    video_url = input("Enter the YouTube video URL: ")
    transcript = get_transcript_unlisted(video_url)

    # Save transcript to a text file
    if "Error" not in transcript and "Invalid YouTube URL." not in transcript:
        output_file = "transcript.txt"
        with open(output_file, "w", encoding="utf-8") as file:
            file.write(transcript)
        print(f"\nTranscript saved successfully to {output_file}")
    else:
        print("\n", transcript)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json: 100%                                        892/892 [00:00<00:00, 29.9kB/s]
model.safetensors: 100%                                        2.24G/2.24G [00:23<00:00, 129MB/s]
tokenizer_config.json: 100%                                        406/406 [00:00<00:00, 23.3kB/s]
sentencepiece.bpe.model: 100%                                        5.07M/5.07M [00:00<00:00, 77.4MB/s]
tokenizer.json: 100%                                        17.1M/17.1M [00:00<00:00, 154MB/s]
special_tokens_map.json: 100%                                        239/239 [00:00<00:00, 16.0kB/s]
Device set to use cpu
/usr/local/lib/python3.11/dist-packages/transformers/pipelines/token_classification.py:170: UserWarning: `grouped_entities` is depre
  warnings.warn(
Enter the YouTube video URL: https://youtu.be/sK8SILOM37I
Device set to use cpu

Transcript saved successfully to transcript.txt
```

```
import re
import os

def format_transcript_sentences(input_file, output_file=None):
    """
    Processes a transcript text file to add line breaks after sentences.

    Args:
        input_file: Path to the original transcript file
        output_file: Path for formatted file (default: adds '_formatted' suffix)

    Returns:
        Path to the formatted file or error message
    """
    try:
        # Read input file
        with open(input_file, 'r', encoding='utf-8') as f:
            raw_text = f.read().replace('\n', ' ')  # Remove existing newlines

        # Split into sentences using punctuation followed by whitespace
        sentences = re.split(r'(?<=[.!?]) +', raw_text)

        # Format with each sentence on new line and proper capitalization
        formatted_text = []
        for sentence in sentences:
            sentence = sentence.strip()
            if sentence:
                # Capitalize first letter of each sentence
                formatted_sentence = sentence[0].upper() + sentence[1:]
                formatted_text.append(formatted_sentence)

        formatted_text = '\n'.join(formatted_text)

        # Create output filename if not provided
        if not output_file:
            base, ext = os.path.splitext(input_file)
            output_file = f"{base}_formatted{ext}"

        # Write formatted text
        with open(output_file, 'w', encoding='utf-8') as f:
            f.write(formatted_text)

        return output_file

    except FileNotFoundError:
        return f"Error: File '{input_file}' not found"
    except Exception as e:
        return f"Error processing file: {str(e)}"

# Example usage
if __name__ == "__main__":
    input_path = input("Enter path to transcript file: ").strip()
    result = format_transcript_sentences(input_path)

    if "Error" in result:
        print(f"\n{result}")
    else:
        print(f"\nFormatted transcript saved to: {result}")
        print("\nFirst 5 lines of formatted text:")
        with open(result, 'r', encoding='utf-8') as f:
            print(''.join(f.readlines()[:5]))
```

```
Enter path to transcript file: /content/transcript.txt

Formatted transcript saved to: /content/transcript_formatted.txt

First 5 lines of formatted text:
So, sir, we know that India has seen a huge Revolution with digital payments.
We all thought that India is a place- at least the West thought- that India is a place where, uh, many people do not get a square me
That was the narrative some 30 years ago.
And not many are literate- people cannot read, um.
But then we have now shown that digital payments number one is India, while people thought that it wouldn't even come to top 50, le
```

```
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
from sentence_transformers import SentenceTransformer

def semantic_segmentation(input_file, output_file=None, min_length=3, threshold=0.65):
    """
    Segments text into meaningful chunks with semantic coherence and keywords.

    Args:
        input_file: Path to formatted transcript file
        output_file: Output path (default: adds '_segmented' suffix)
        min_length: Minimum sentences per segment
        threshold: Semantic similarity threshold (0-1)

    Returns:
        Path to segmented file or error message
    """
    try:
        # Load ML models
        model = SentenceTransformer('all-MiniLM-L6-v2')

        # Read and split sentences
        with open(input_file, 'r', encoding='utf-8') as f:
            sentences = [line.strip() for line in f if line.strip()]

        if len(sentences) < min_length:
            return f"Need at least {min_length} sentences for segmentation"

        # Generate sentence embeddings
        embeddings = model.encode(sentences)

        # Create segments with semantic coherence
        segments = []
        current_segment = []
        current_emb = None

        for sent, emb in zip(sentences, embeddings):
            emb = emb.reshape(1, -1)
            if not current_segment:
                current_segment.append(sent)
                current_emb = emb
                continue

            similarity = cosine_similarity(current_emb, emb)[0][0]
            if similarity >= threshold and len(current_segment) < 5:
                current_segment.append(sent)
                current_emb = (current_emb * len(current_segment) + emb) / (len(current_segment) + 1)
            else:
                if len(current_segment) >= min_length:
                    segments.append(current_segment)
                current_segment = [sent]
                current_emb = emb

        # Finalize remaining sentences
        if current_segment:
            if segments and len(current_segment) < min_length:
                segments[-1].extend(current_segment)
            else:
                segments.append(current_segment)

        # Extract keywords for each segment
        results = []
        for seg in segments:
            vectorizer = TfidfVectorizer(stop_words='english', ngram_range=(1,2))
            X = vectorizer.fit_transform([' '.join(seg)])
            features = vectorizer.get_feature_names_out()
            keywords = features[np.argsort(X.toarray())[0][-3:]][::-1]

            results.append({
                'sentences': seg,
                'keywords': keywords,
                'count': len(seg)
            })

        # Create output filename
        if not output_file:
            base, ext = os.path.splitext(input_file)
            output_file = f"{base}_segmented{ext}"

        # Write segmented output
        with open(output_file, "w") as f:
            for i, seg in enumerate(results, 1):
                f.write(f"Segment {i} ({seg['count']} sentences | Keywords: {', '.join(seg['keywords'])})\n")
                f.write('\n'.join(seg['sentences']) + '\n\n')

        return output_file

    except Exception as e:
        return f"Error during segmentation: {str(e)}"

# Example usage
if __name__ == "__main__":
    input_path = input("Enter path to formatted transcript file: ").strip()
    result = semantic_segmentation(input_path)

    if "Error" in result:
        print(f"\n{result}")
    else:
        print(f"\nSegmented transcript saved to: {result}")
```

```
Enter path to formatted transcript file: /content/transcript_formatted.txt
modules.json:  100%                                              349/349  [00:00<00:00,  7.58kB/s]

config_sentence_transformers.json:  100%                         116/116  [00:00<00:00,  1.31kB/s]

README.md:  100%                                                 10.5k/10.5k  [00:00<00:00,  117kB/s]

sentence_bert_config.json:  100%                                 53.0/53.0  [00:00<00:00,  750B/s]

config.json:  100%                                               612/612  [00:00<00:00,  10.9kB/s]

model.safetensors:  100%                                         90.9M/90.9M  [00:00<00:00,  140MB/s]

tokenizer_config.json:  100%                                     350/350  [00:00<00:00,  15.2kB/s]

vocab.txt:  100%                                                 232k/232k  [00:00<00:00,  2.71MB/s]

tokenizer.json:  100%                                            466k/466k  [00:00<00:00,  14.2MB/s]

special_tokens_map.json:  100%                                   112/112  [00:00<00:00,  6.82kB/s]

config.json:  100%                                               190/190  [00:00<00:00,  10.9kB/s]
```

```python
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
from sentence_transformers import SentenceTransformer
import os

def semantic_segmentation(input_file, output_file=None, min_length=3, threshold=0.65):
    """
    Segments text into meaningful chunks with semantic coherence and keywords.

    Args:
        input_file: Path to formatted transcript file
        output_file: Output path (default: adds '_segmented' suffix)
        min_length: Minimum sentences per segment
        threshold: Semantic similarity threshold (0-1)

    Returns:
        Path to segmented file or error message
    """
    try:
        # Load ML models
        model = SentenceTransformer('all-MiniLM-L6-v2')

        # Read and split sentences
        with open(input_file, 'r', encoding='utf-8', errors='ignore') as f:
            sentences = [line.strip() for line in f if line.strip()]

        if len(sentences) < min_length:
            return f"Need at least {min_length} sentences for segmentation"

        # Generate sentence embeddings
        embeddings = model.encode(sentences)

        # Create segments with semantic coherence
        segments = []
        current_segment = []
        current_emb = None

        for i, (sent, emb) in enumerate(zip(sentences, embeddings)):
            emb = emb.reshape(1, -1)
            if not current_segment:
                current_segment.append(sent)
                current_emb = emb
                continue

            similarity = cosine_similarity(current_emb, emb)[0][0]
            if similarity >= threshold and len(current_segment) < 5:
                current_segment.append(sent)
                current_emb = (current_emb * len(current_segment) + emb) / (len(current_segment) + 1)
            else:
                # Finalize segment if it meets minimum length
                if len(current_segment) >= min_length:
                    segments.append(current_segment)
                else:
                    # If too short, append to previous segment if possible
                    if segments:
                        segments[-1].extend(current_segment)
                    else:
                        segments.append(current_segment)
                current_segment = [sent]
                current_emb = emb

            # Force finalize segment at the end of the file
            if i == len(sentences) - 1:
                if len(current_segment) >= min_length:
                    segments.append(current_segment)
                else:
                    # If too short, append to previous segment if possible
                    if segments:
                        segments[-1].extend(current_segment)
                    else:
                        segments.append(current_segment)

        # Extract keywords for each segment
        results = []
        for seg in segments:
            vectorizer = TfidfVectorizer(stop_words='english', ngram_range=(1,2))
            X = vectorizer.fit_transform([' '.join(seg)])
            features = vectorizer.get_feature_names_out()
            keywords = features[np.argsort(X.toarray())[0][-3:]][::-1]

            results.append({
                'sentences': seg,
                'keywords': keywords,
                'count': len(seg)
            })

        # Create output filename
        if not output_file:
            base, ext = os.path.splitext(input_file)
            output_file = f"{base}_segmented{ext}"

        # Write segmented output
        with open(output_file, "w") as f:
            for i, seg in enumerate(results, 1):
                f.write(f"Segment {i} ({seg['count']} sentences | Keywords: {', '.join(seg['keywords'])})\n")
                f.write('\n'.join(seg['sentences']) + '\n\n')

        return output_file

    except Exception as e:
        return f"Error during segmentation: {str(e)}"

# Example usage
if __name__ == "__main__":
    input_path = input("Enter path to formatted transcript file: ").strip()
    result = semantic_segmentation(input_path)

    if "Error" in result:
        print(f"\n{result}")
    else:
        print(f"\nSegmented transcript saved to: {result}")
```

```
Enter path to formatted transcript file: /content/transcript_formatted.txt

Segmented transcript saved to: /content/transcript_formatted_segmented.txt
```

```
#Now doing the transcription with timestamps
```

```
import re
import urllib.parse
import requests
from youtube_transcript_api import YouTubeTranscriptApi
from pytube import YouTube
import speech_recognition as sr
from pydub import AudioSegment
from deepmultilingualpunctuation import PunctuationModel
import os
# Initialize model once at the top
model = PunctuationModel()

def extract_video_id(video_url):
    """
    Extracts the YouTube video ID from various URL formats.
    """
    parsed_url = urllib.parse.urlparse(video_url)
    query_params = urllib.parse.parse_qs(parsed_url.query)

    if "v" in query_params:
        return query_params["v"][0]
```

```python

    match = re.search(r"(youtu\.be/|youtube\.com/embed/|youtube\.com/shorts/)([\w-]+)", video_url)
    if match:
        return match.group(2)

    return None

def download_audio(video_url):
    """
    Downloads the audio using yt-dlp with cookies and returns the file path.
    """
    try:
        ydl_opts = {
            'format': 'bestaudio/best',
            'outtmpl': 'audio.%(ext)s',
            'cookiefile': 'cookies (1).txt',  # Use the exported cookies
            'postprocessors': [{
                'key': 'FFmpegExtractAudio',
                'preferredcodec': 'mp3',
                'preferredquality': '192',
            }],
        }
        with yt_dlp.YoutubeDL(ydl_opts) as ydl:
            info = ydl.extract_info(video_url, download=True)
            return "audio.mp3"
    except Exception as e:
        return f"Error downloading audio: {str(e)}"

def convert_audio_to_wav(audio_file):
    """
    Converts the downloaded MP3 audio to WAV format using pydub.
    """
    wav_file = "audio.wav"
    try:
        AudioSegment.from_mp3(audio_file).export(wav_file, format="wav")
        return wav_file
    except Exception as e:
        return f"Error converting to WAV: {str(e)}"

def transcribe_audio(audio_path, chunk_length=30):
    """
    Splits audio into smaller chunks, transcribes each chunk separately,
    and adds punctuation using deepmultilingualpunctuation library.
    """
    recognizer = sr.Recognizer()
    audio = AudioSegment.from_wav(audio_path)
    total_duration = len(audio) / 1000  # Convert to seconds
    transcribed_text = []

    # Load punctuation model
    model = PunctuationModel()

    print("Transcribing audio in chunks...")

    # In transcribe_audio()
    punctuated_chunks = []
    for chunk_text in transcribed_text:
        punctuated = model.restore_punctuation(chunk_text)
        punctuated_chunks.append(punctuated)
        return " ".join(punctuated_chunks)

    # Split and transcribe audio in chunks
    for start in range(0, int(total_duration), chunk_length):
        end = min(start + chunk_length, int(total_duration))
        chunk = audio[start * 1000:end * 1000]  # Extract chunk in milliseconds
        chunk.export("chunk.wav", format="wav")  # Save chunk temporarily

        with sr.AudioFile("chunk.wav") as source:
            try:
                audio_data = recognizer.record(source)
                text = recognizer.recognize_google(audio_data)
                transcribed_text.append(text)
            except sr.UnknownValueError:
                transcribed_text.append("[Unintelligible]")
            except sr.RequestError as e:
                return f"Error with the speech recognition service: {str(e)}"

    os.remove("chunk.wav")  # Clean up temporary chunk file

    # Combine chunks and add punctuation
    combined_text = " ".join(transcribed_text)
    punctuated_text = model.restore_punctuation(combined_text)

    return punctuated_text

def get_transcript_unlisted(video_url):
    """
    Tries to fetch the transcript using youtube_transcript_api first,
    then falls back to downloading and transcribing audio if necessary.
    """
    model = PunctuationModel()  # Initialize once
    video_id = extract_video_id(video_url)

    if not video_id:
        return "Invalid YouTube URL."

    # Try API path with punctuation and timestamps
    try:
        transcript = YouTubeTranscriptApi.get_transcript(video_id)
        formatted_transcript = []
        for item in transcript:
            start_time = convert_time(item['start'])
            end_time = convert_time(item['start'] + item['duration'])
            formatted_transcript.append(f"[{start_time}-{end_time}] {item['text']}")
        return model.restore_punctuation(" ".join(formatted_transcript))
    except:
        print("Transcript not available via API, attempting audio transcription...")

    # Audio fallback path (existing implementation)
    # ... rest of audio processing code ...
    # Download and transcribe audio if no transcript is available
    audio_file = download_audio(video_url)
    if "Error" in audio_file:
        return audio_file

    wav_file = convert_audio_to_wav(audio_file)
    if "Error" in wav_file:
        return wav_file

    transcription = transcribe_audio(wav_file)

    # Cleanup temporary files
    os.remove(audio_file)
    os.remove(wav_file)

    # For audio fallback, timestamps are not directly available
    # You might need to manually add timestamps or use a different approach
    return transcription

def convert_time(seconds):
    """Converts seconds to [hrs:mins:seconds] format."""
    hrs = int(seconds // 3600)
    mins = int((seconds % 3600) // 60)
    secs = round(seconds % 60, 2)
    return f"{hrs:02d}:{mins:02d}:{secs:05.2f}"

# Example usage
# Example usage
if __name__ == "__main__":
    video_url = input("Enter the YouTube video URL: ")
    transcript = get_transcript_unlisted(video_url)

    # Save transcript to a text file
    if "Error" not in transcript and "Invalid YouTube URL." not in transcript:
        output_file = "transcript.txt"
        with open(output_file, "w", encoding="utf-8") as file:
            file.write(transcript)
        print(f"\nTranscript saved successfully to {output_file}")
    else:
        print("\n", transcript)

# Example usage
# Example usage
if __name__ == "__main__":
    video_url = input("Enter the YouTube video URL: ")
    transcript = get_transcript_unlisted(video_url)

    # Save transcript to a text file
    if "Error" not in transcript and "Invalid YouTube URL." not in transcript:
        output_file = "transcript.txt"
        with open(output_file, "w", encoding="utf-8") as file:
            file.write(transcript)
        print(f"\nTranscript saved successfully to {output_file}")
    else:
        print("\n", transcript)
```

```
Device set to use cpu
/usr/local/lib/python3.11/dist-packages/transformers/pipelines/token_classification.py:170: UserWarning: `grouped_entities` is depre
  warnings.warn(
Enter the YouTube video URL: https://youtu.be/sK8SILOM37I
Device set to use cpu
Transcript not available via API, attempting audio transcription...
[youtube] Extracting URL: https://youtu.be/sK8SILOM37I
[youtube] sK8SILOM37I: Downloading webpage
[youtube] sK8SILOM37I: Downloading tv client config
[youtube] sK8SILOM37I: Downloading player f6e09c70
[youtube] sK8SILOM37I: Downloading tv player API JSON
[youtube] sK8SILOM37I: Downloading ios player API JSON
[youtube] sK8SILOM37I: Downloading m3u8 information
[info] sK8SILOM37I: Downloading 1 format(s): 251
[download] Destination: audio.webm
[download] 100% of   39.37MiB in 00:00:01 at 30.62MiB/s
[ExtractAudio] Destination: audio.mp3
Deleting original file audio.webm (pass -k to keep)
Device set to use cpu
/usr/local/lib/python3.11/dist-packages/transformers/pipelines/token_classification.py:170: UserWarning: `grouped_entities` is depre
  warnings.warn(
Transcribing audio in chunks...

Transcript saved successfully to transcript.txt
Enter the YouTube video URL: https://youtu.be/sK8SILOM37I
Device set to use cpu
Transcript not available via API, attempting audio transcription...
[youtube] Extracting URL: https://youtu.be/sK8SILOM37I
[youtube] sK8SILOM37I: Downloading webpage
[youtube] sK8SILOM37I: Downloading tv client config
[youtube] sK8SILOM37I: Downloading player f6e09c70
[youtube] sK8SILOM37I: Downloading tv player API JSON
[youtube] sK8SILOM37I: Downloading ios player API JSON
[youtube] sK8SILOM37I: Downloading m3u8 information
[info] sK8SILOM37I: Downloading 1 format(s): 251
[download] Destination: audio.webm
[download] 100% of   39.37MiB in 00:00:01 at 33.05MiB/s
[ExtractAudio] Destination: audio.mp3
Deleting original file audio.webm (pass -k to keep)
Device set to use cpu
/usr/local/lib/python3.11/dist-packages/transformers/pipelines/token_classification.py:170: UserWarning: `grouped_entities` is depre
  warnings.warn(
Transcribing audio in chunks...

Transcript saved successfully to transcript.txt
```

```
pip install nltk scikit-learn
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages (3.9.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk) (8.1.8)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk) (4.67.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.5.0)
```

```
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from itertools import combinations
import numpy as np
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt_tab')
# Ensure required resources are downloaded
nltk.download('punkt')

# Function to read transcript from a .txt file
def read_transcript(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        transcript = file.read()
    return transcript

# Function to split transcript into individual sentences
def split_into_sentences(transcript):
    sentences = nltk.sent_tokenize(transcript)
    return sentences
```

```python

# Function to compute cosine similarity between sentence pairs
def compute_cosine_similarity(sentences):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(sentences)
    similarity_matrix = cosine_similarity(tfidf_matrix)
    return similarity_matrix

# Function to find similar sentence triplets based on cosine similarity
def find_similar_triplets(sentences, similarity_matrix, threshold=0.5):
    triplets = []
    n = len(sentences)

    # Generate all combinations of triplets
    for comb in combinations(range(n), 3):
        i, j, k = comb
        # Check if all pairs within the triplet are similar
        if (similarity_matrix[i][j] > threshold and
            similarity_matrix[j][k] > threshold and
            similarity_matrix[i][k] > threshold):
            triplets.append([sentences[i], sentences[j], sentences[k]])

    return triplets

# Function to write the segmented sentences to a new .txt file
def write_segments_to_file(triplets, output_file):
    with open(output_file, 'w', encoding='utf-8') as file:
        for idx, triplet in enumerate(triplets, 1):
            file.write(f"Segment {idx}:\n")
            for sentence in triplet:
                file.write(sentence + "\n")
            file.write("\n")

def main():
    input_file = '/content/transcript (5).txt'  # Input .txt file path
    output_file = 'segmented_transcript.txt'  # Output .txt file path

    # Reading and processing transcript
    transcript = read_transcript(input_file)
    sentences = split_into_sentences(transcript)
    similarity_matrix = compute_cosine_similarity(sentences)

    # Finding similar sentence triplets
    triplets = find_similar_triplets(sentences, similarity_matrix, threshold=0.5)

    # Writing segments to output file
    write_segments_to_file(triplets, output_file)

    print(f"Segmented transcript saved to {output_file}")

if __name__ == "__main__":
    main()
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Segmented transcript saved to segmented_transcript.txt
```

```
pip install nltk scikit-learn numpy
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages (3.9.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (1.26.4)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk) (8.1.8)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk) (4.67.1)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.5.0)
```

```python
import nltk
from nltk.tokenize import sent_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
```

```
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
True
```

```
def read_transcript(file_path):
    with open(file_path, 'r') as file:
        transcript = file.read()
    return transcript

def split_into_sentences(transcript):
    return sent_tokenize(transcript)

def calculate_cosine_similarity(sentences):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(sentences)
    cosine_sim = cosine_similarity(tfidf_matrix)
    return cosine_sim

def group_sentences(sentences, cosine_sim, min_sentences=3, max_sentences=10):
    grouped_sentences = []
    used_indices = set()

    for i in range(len(sentences)):
        if i in used_indices:
            continue
        group = [sentences[i]]
        used_indices.add(i)
        for j in range(i + 1, len(sentences)):
            if j in used_indices:
                continue
            if cosine_sim[i][j] > 0.5:  # Adjust the threshold as needed
                group.append(sentences[j])
                used_indices.add(j)
                if len(group) >= max_sentences:
                    break
        if len(group) >= min_sentences:
            grouped_sentences.append(group)
    return grouped_sentences

def process_transcript(file_path):
    transcript = read_transcript(file_path)
    sentences = split_into_sentences(transcript)
    cosine_sim = calculate_cosine_similarity(sentences)
    grouped_sentences = group_sentences(sentences, cosine_sim)
    return grouped_sentences
```

```
file_path = '/content/transcript (5).txt'
grouped_sentences = process_transcript(file_path)

for i, group in enumerate(grouped_sentences):
    print(f"Segment {i + 1}:")
    for sentence in group:
        print(sentence)
    print("\n")
```

```
Segment 1:
how do I do the transition?
how do we do this?
how do we start?


Segment 2:
all kinds of learning are being critized, including learning of soft skills, employability skills, life skills, your hand skills, yo
and all these are skills.
okay, number one: the curricular structure now specifically provides for employability skills, soft skills and life skills.
```

```
Segment 3:
absolutely there.
absolutely, yes, absolutely.
absolutely, we can.


Segment 4:
okay.
okay.
physics, right, okay.
okay.


Segment 5:
why?
why, I see, wonderful.
why not?
so for why see why this was not not even thought about?
why?


Segment 6:
can I can.
can?
we can.


Segment 7:
school C, us mhm.
the way we teach them in school, mhm.
mhm.
```

```python
import nltk
from nltk.tokenize import sent_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Download NLTK data
nltk.download('punkt')

def read_transcript(file_path):
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            transcript = file.read()
        return transcript
    except Exception as e:
        print(f"Error reading file: {e}")
        return None

def split_into_sentences(transcript):
    return sent_tokenize(transcript)

def calculate_cosine_similarity(sentences):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(sentences)
    cosine_sim = cosine_similarity(tfidf_matrix)
    return cosine_sim

def group_sentences(sentences, cosine_sim, min_sentences=3, max_sentences=10, similarity_threshold=0.5):
    grouped_sentences = []
    used_indices = set()

    for i in range(len(sentences)):
        if i in used_indices:
            continue
        group = [sentences[i]]
        used_indices.add(i)
        for j in range(i + 1, len(sentences)):
            if j in used_indices:
                continue
            if cosine_sim[i][j] > similarity_threshold:  # Adjust the threshold as needed
                group.append(sentences[j])
                used_indices.add(j)
                if len(group) >= max_sentences:
                    break
        if len(group) >= min_sentences:
            grouped_sentences.append(group)
    return grouped_sentences

def process_transcript(file_path):
    transcript = read_transcript(file_path)
    if transcript is None:
        return []  # Return an empty list if the file couldn't be read

    sentences = split_into_sentences(transcript)
    cosine_sim = calculate_cosine_similarity(sentences)
    grouped_sentences = group_sentences(sentences, cosine_sim, similarity_threshold=0.6)  # Adjusted threshold
    return grouped_sentences

# Replace with your actual file path
file_path = '/content/transcript (5).txt'
grouped_sentences = process_transcript(file_path)

if grouped_sentences:
    for i, group in enumerate(grouped_sentences):
        print(f"Segment {i + 1}:")
        for sentence in group:
            print(sentence)
        print("\n")
else:
    print("No sentences were processed. Check the file path and file content.")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Segment 1:
okay.
okay.
okay.


Segment 2:
why?
why not?
why?


Segment 3:
can I can.
can?
we can.
```

```python
import nltk
from nltk.tokenize import sent_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Download NLTK data
nltk.download('punkt')

def read_transcript(file_path):
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            transcript = file.read()
        return transcript
    except Exception as e:
        print(f"Error reading file: {e}")
        return None

def split_into_sentences(transcript):
    return sent_tokenize(transcript)

def calculate_cosine_similarity(sentences):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(sentences)
    cosine_sim = cosine_similarity(tfidf_matrix)
    return cosine_sim

def group_sentences(sentences, cosine_sim, min_sentences=3, max_sentences=10, similarity_threshold=0.5):
    grouped_sentences = []
    used_indices = set()

    for i in range(len(sentences)):
        if i in used_indices:
            continue
        group = [sentences[i]]
        used_indices.add(i)
        for j in range(i + 1, len(sentences)):
            if j in used_indices:
                continue
            if cosine_sim[i][j] > similarity_threshold:  # Adjust the threshold as needed
```

```
                group.append(sentences[j])
                used_indices.add(j)
                if len(group) >= max_sentences:
                    break
        if len(group) >= min_sentences:
            grouped_sentences.append(group)
    return grouped_sentences

def process_transcript(file_path):
    transcript = read_transcript(file_path)
    if transcript is None:
        return []  # Return an empty list if the file couldn't be read

    sentences = split_into_sentences(transcript)
    cosine_sim = calculate_cosine_similarity(sentences)
    grouped_sentences = group_sentences(sentences, cosine_sim, similarity_threshold=0.6)  # Adjusted threshold
    return grouped_sentences

# Replace with your actual file path
file_path = '/content/transcript (5).txt'
grouped_sentences = process_transcript(file_path)

if grouped_sentences:
    for i, group in enumerate(grouped_sentences):
        print(f"Segment {i + 1}:")
        for sentence in group:
            print(sentence)
        print("\n")
else:
    print("No sentences were processed. Check the file path and file content.")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Segment 1:
okay.
okay.
okay.


Segment 2:
why?
why not?
why?


Segment 3:
can I can.
can?
we can.
```

```
#Cosine Similarity approach
```

```
import nltk
from nltk.tokenize import sent_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

# Download NLTK data
nltk.download('punkt')

def read_transcript(file_path):
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            transcript = file.read()
        return transcript
    except Exception as e:
        print(f"Error reading file: {e}")
        return None

def split_into_sentences(transcript):
    return sent_tokenize(transcript)

def calculate_cosine_similarity(sentences):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(sentences)
    cosine_sim = cosine_similarity(tfidf_matrix)
    return cosine_sim

def save_cosine_similarity_to_csv(cosine_sim, sentences, output_file):
    # Create a DataFrame for the cosine similarity matrix
    df = pd.DataFrame(cosine_sim, index=sentences, columns=sentences)

    # Save the DataFrame to a CSV file
    df.to_csv(output_file)
    print(f"Cosine similarity matrix saved to {output_file}")

def process_transcript(file_path, output_file):
    # Read the transcript
    transcript = read_transcript(file_path)
    if transcript is None:
        return

    # Split the transcript into sentences
    sentences = split_into_sentences(transcript)
    print("Sentences extracted:")
    for i, sentence in enumerate(sentences):
        print(f"{i + 1}: {sentence}")

    # Calculate cosine similarity between sentences
    cosine_sim = calculate_cosine_similarity(sentences)

    # Save the cosine similarity matrix to a CSV file
    save_cosine_similarity_to_csv(cosine_sim, sentences, output_file)

# File paths (adjust as needed)
file_path = '/content/transcript (5).txt'  # Path to your uploaded transcript file
output_file = '/content/cosine_similarity_matrix.csv'  # Output CSV file name

# Process the transcript and generate the CSV file
process_transcript(file_path, output_file)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Sentences extracted:
1: so, sir, we know that India has seen a huge Revolution with digital payments.
2: we all thought that India is a place- at least the West thought- that India is a place where, uh, many people do not get a squ
3: that was the narrative some 30 years ago.
4: and not many are literate- people cannot read, um.
5: but then we have now shown that digital payments number one is India, while people thought that it wouldn't even come to top 5
6: I think, immediately after UPA, the next big revolution, personally, I think, is in education, and the complete homework for t
7: sir, my question is: do you think ncrf plus NEP put together will be the next big revolution after UPA in India?
8: absolutely, and why I think so is because in education, the last policy came up many, many years ago- that was in 1986, which
9: so so many changes have happened in the real world.
10: so many changes have happened in the requirement of the industry, requirement of Manpower for the industry.
11: however, there were no corresponding changes which happened?
12: education system.
13: so, therefore, I feel that this was the right time when we brought in the education policy 2020, honorable prime minister, de
14: 2020 is a policy, the NP 2020 is a policy, and for implementing a policy, you need a framework.
15: now, why we call it a framework?
16: we call it a framework because this is very flexible.
17: this allows you all the Innovation, the way you educate your kids, you educate your students.
18: still, it provides you the basic guidelines, the, the framework, the outer layer it provides, and that layer is mostly the en
19: that is such an enabling layer that it has broken the shackles which were there in the education sector, right?
20: so, yes, it is a big Revolution and this is going to change the way we have been educating our kids and this will be Game Cha
21: so, sir, I think, um, uh, let's go with this example of, let's say, I started off living in a small 2bhk apartment M and I sl
22: right, and one fine day, you came and you changed my kitchen completely.
23: I was using a bicycle, I moved to a scooter and a car and now you're asking me to fly and you're giving me an aircraft.
24: NEP sounds more like that for me.
25: how do I do the transition?
26: I fear that I will.
27: I will crash if I use a aeroplane without training.
28: I'm talking about all the teachers in the in the country, all the schools in the country.
29: we have been driving buses at Max.
30: now we should fly.
31: how do we do this?
32: okay, look at the requirement of the industry.
33: requirement of the industry has been moving very fast.
34: the technology is emerging every day and the industry is moving with that speed.
35: so when a student is coming out of your Institute and is going out in the market, he finds that whatever he has been taught h
36: when industry is moving that fast, when the requirement is moving that fast, don't you think it is important for us to change
37: yes, how long can we wait?
38: yes, it will take a lot of effort for every one of us to adapt to this change.
39: but this change is going to be not only beneficial, mhm, but also very facilitative for all of us, very liberating for all of
40: this is going to be highly liberative and choice-based system.
41: there are number of choices which are available to you, which are available to every student.
42: yes, when we introduce a new system, we have to really create, create new things, create new ways of doing things, learn some
43: but once we learn it, there is no limit to Innovation and creativity which this will bring.
44: there's going to be a learning curve here.
45: you mean, it's going to be difficult to fly a fighter, not really difficult to apply, because all of us we have that kind of
46: only thing is that we were not translating our vision and Innovative Minds into the education sector, whereas we are applying
47: know we are, we are moving very fast.
48: it's only education and I think the education sector is the smartest sector, which is very good at learning, and I'm I'm very
49: I can tell you that already a number of Institutions have adopted the NEP and ncrf to varying degrees and whereever they have
50: so then, um, um, I have spoken to many academics.
51: most of us do not understand NEP.
52: let on ncrf, which is the next step on how to implement NEP.
53: if I can request you to give me an elevator pitch for NEP and then an elevator pitch for ncrf, for the exclusive reason that
```

```
import nltk
from nltk.tokenize import sent_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

# Download NLTK data
nltk.download('punkt')

def read_transcript(file_path):
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            transcript = file.read()
        return transcript
    except Exception as e:
        print(f"Error reading file: {e}")
        return None

def split_into_sentences(transcript):
    return sent_tokenize(transcript)

def calculate_cosine_similarity(sentences):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(sentences)
    cosine_sim = cosine_similarity(tfidf_matrix)
    return cosine_sim

def save_cosine_similarity_to_csv(cosine_sim, sentences, output_file):
    # Create sentence names (S1, S2, S3, ...)
    sentence_names = [f"S{i+1}" for i in range(len(sentences))]

    # Create a DataFrame for the cosine similarity matrix
    df = pd.DataFrame(cosine_sim, index=sentence_names, columns=sentence_names)

    # Save the DataFrame to a CSV file
    df.to_csv(output_file)
    print(f"Cosine similarity matrix saved to {output_file}")

def process_transcript(file_path, output_file):
    # Read the transcript
    transcript = read_transcript(file_path)
    if transcript is None:
        return

    # Split the transcript into sentences
    sentences = split_into_sentences(transcript)
    print("Sentences extracted:")
    for i, sentence in enumerate(sentences):
        print(f"S{i + 1}: {sentence}")

    # Calculate cosine similarity between sentences
    cosine_sim = calculate_cosine_similarity(sentences)

    # Save the cosine similarity matrix to a CSV file
    save_cosine_similarity_to_csv(cosine_sim, sentences, output_file)

# File paths (adjust as needed)
file_path = '/content/transcript (5).txt'  # Path to your uploaded transcript file
output_file = '/content/cosine_similarity_matrix1.csv'  # Output CSV file name

# Process the transcript and generate the CSV file
process_transcript(file_path, output_file)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Sentences extracted:
S1: so, sir, we know that India has seen a huge Revolution with digital payments.
S2: we all thought that India is a place- at least the West thought- that India is a place where, uh, many people do not get a s
S3: that was the narrative some 30 years ago.
S4: and not many are literate- people cannot read, um.
S5: but then we have now shown that digital payments number one is India, while people thought that it wouldn't even come to top
S6: I think, immediately after UPA, the next big revolution, personally, I think, is in education, and the complete homework for
S7: sir, my question is: do you think ncrf plus NEP put together will be the next big revolution after UPA in India?
S8: absolutely, and why I think so is because in education, the last policy came up many, many years ago- that was in 1986, which
S9: so so many changes have happened in the real world.
S10: so many changes have happened in the requirement of the industry, requirement of Manpower for the industry.
S11: however, there were no corresponding changes which happened?
S12: education system.
S13: so, therefore, I feel that this was the right time when we brought in the education policy 2020, honorable prime minister,
S14: 2020 is a policy, the NP 2020 is a policy, and for implementing a policy, you need a framework.
S15: now, why we call it a framework?
```

```
S16: we call it a framework because this is very flexible.
S17: this allows you all the Innovation, the way you educate your kids, you educate your students.
S18: still, it provides you the basic guidelines, the, the framework, the outer layer it provides, and that layer is mostly the e
S19: that is such an enabling layer that it has broken the shackles which were there in the education sector, right?
S20: so, yes, it is a big Revolution and this is going to change the way we have been educating our kids and this will be Game Ch
S21: so, sir, I think, um, uh, let's go with this example of, let's say, I started off living in a small 2bhk apartment M and I s
S22: right, and one fine day, you came and you changed my kitchen completely.
S23: I was using a bicycle, I moved to a scooter and a car and now you're asking me to fly and you're giving me an aircraft.
S24: NEP sounds more like that for me.
S25: how do I do the transition?
S26: I fear that I will.
S27: I will crash if I use a aeroplane without training.
S28: I'm talking about all the teachers in the in the country, all the schools in the country.
S29: we have been driving buses at Max.
S30: now we should fly.
S31: how do we do this?
S32: okay, look at the requirement of the industry.
S33: requirement of the industry has been moving very fast.
S34: the technology is emerging every day and the industry is moving with that speed.
S35: so when a student is coming out of your Institute and is going out in the market, he finds that whatever he has been taught
S36: when industry is moving that fast, when the requirement is moving that fast, don't you think it is important for us to chang
S37: yes, how long can we wait?
S38: yes, it will take a lot of effort for every one of us to adapt to this change.
S39: but this change is going to be not only beneficial, mhm, but also very facilitative for all of us, very liberating for all o
S40: this is going to be highly liberative and choice-based system.
S41: there are number of choices which are available to you, which are available to every student.
S42: yes, when we introduce a new system, we have to really create, create new things, create new ways of doing things, learn som
S43: but once we learn it, there is no limit to Innovation and creativity which this will bring.
S44: there's going to be a learning curve here.
S45: you mean, it's going to be difficult to fly a fighter, not really difficult to apply, because all of us we have that kind of
S46: only thing is that we were not translating our vision and Innovative Minds into the education sector, whereas we are applyin
S47: know we are, we are moving very fast.
S48: it's only education and I think the education sector is the smartest sector, which is very good at learning, and I'm I'm ver
S49: I can tell you that already a number of Institutions have adopted the NEP and ncrf to varying degrees and whereever they hav
S50: so then, um, um, I have spoken to many academics.
S51: most of us do not understand NEP.
S52: let on ncrf, which is the next step on how to implement NEP.
S53: if I can request you to give me an elevator pitch for NEP and then an elevator pitch for ncrf, for the exclusive reason that
```

```python
df = pd.read_csv('/content/cosine_similarity_matrix1.csv')
df
```

	Unnamed: 0	S1	S2	S3	S4	S5	S6	S7	S8	S9	...	S458	S459	S460
0	S1	1.000000	0.144704	0.018483	0.000000	0.246278	0.091548	0.230970	0.024449	0.060579	...	0.000000	0.066609	0.000000
1	S2	0.144704	1.000000	0.034696	0.150481	0.230813	0.084158	0.141569	0.094473	0.073300	...	0.000000	0.104808	0.000000
2	S3	0.018483	0.034696	1.000000	0.000000	0.025220	0.038285	0.011759	0.295992	0.018366	...	0.000000	0.031778	0.000000
3	S4	0.000000	0.150481	0.000000	1.000000	0.056117	0.116833	0.000000	0.145069	0.120146	...	0.000000	0.000000	0.127874
4	S5	0.246278	0.230813	0.025220	0.056117	1.000000	0.055102	0.053087	0.019148	0.028013	...	0.088463	0.096468	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
462	S463	0.019424	0.026095	0.024720	0.000000	0.056461	0.026382	0.027060	0.065532	0.054914	...	0.093098	0.033396	0.000000
463	S464	0.061773	0.042473	0.034434	0.000000	0.109107	0.113300	0.036147	0.028488	0.033839	...	0.051788	0.062190	0.000000
464	S465	0.058840	0.064283	0.012052	0.071909	0.063745	0.085637	0.083713	0.093791	0.049924	...	0.089716	0.091426	0.000000
465	S466	0.087732	0.005907	0.011192	0.000000	0.011617	0.120295	0.104724	0.026829	0.010998	...	0.046377	0.029276	0.000000
466	S467	0.000000	0.000000	0.000000	0.027857	0.060456	0.018291	0.043638	0.027710	0.000000	...	0.000000	0.000000	0.000000

467 rows × 468 columns

```python
# Import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the uploaded CSV file
file_path = '/content/cosine_similarity_matrix1.csv'
data = pd.read_csv(file_path)

# Set the first column as the index and remove the 'Unnamed: 0' column
data.set_index('Unnamed: 0', inplace=True)

# Generate the correlation matrix
correlation_matrix = data.corr()

# Print the correlation matrix summary
print("Correlation Matrix Summary:")
print(correlation_matrix.describe())

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()
```

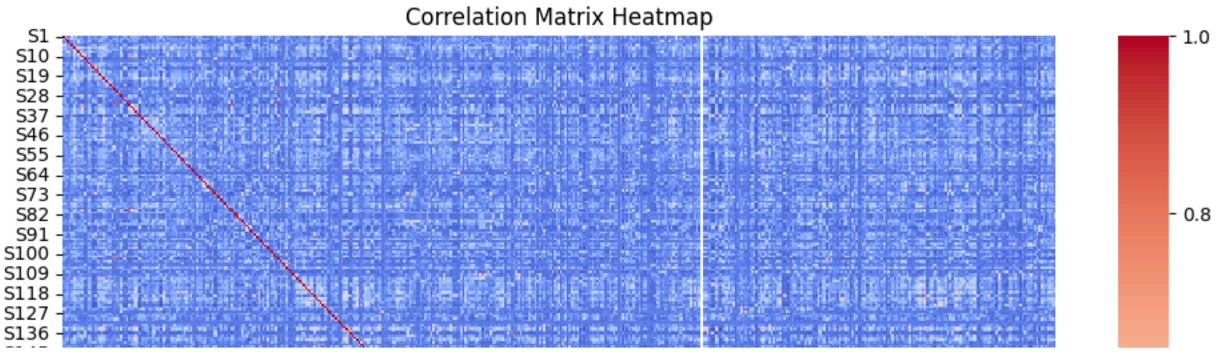
```
Correlation Matrix Summary:
               S1          S2          S3          S4          S5          S6  \
count  466.000000  466.000000  466.000000  466.000000  466.000000  466.000000   
mean     0.032835    0.075637    0.045360    0.033332    0.078733    0.150554   
std      0.084350    0.111592    0.089276    0.093679    0.101235    0.143893   
min     -0.063055   -0.099254   -0.074572   -0.080767   -0.080913   -0.126219   
25%     -0.024712   -0.006972   -0.014654   -0.031607    0.000692    0.039624   
50%      0.014187    0.062486    0.030391    0.004558    0.070433    0.141213   
75%      0.071348    0.135054    0.079533    0.075962    0.139049    0.249313   
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000   

               S7          S8          S9         S10  ...        S458  \
count  466.000000  466.000000  466.000000  466.000000  ...  466.000000   
mean     0.069673    0.077276    0.061499    0.083850  ...    0.062465   
std      0.103796    0.104836    0.106447    0.127883  ...    0.125686   
min     -0.097341   -0.090755   -0.090614   -0.106429  ...   -0.111943   
25%     -0.003994    0.003120   -0.018873   -0.015841  ...   -0.035533   
50%      0.056906    0.061219    0.046579    0.064468  ...    0.035272   
75%      0.128066    0.135750    0.115556    0.147639  ...    0.135771   
max      1.000000    1.000000    1.000000    1.000000  ...    1.000000   

             S459        S460        S461        S462        S463        S464  \
count  466.000000  466.000000  466.000000  466.000000  466.000000  466.000000   
mean     0.079398   -0.015808    0.099405    0.091793    0.031906    0.099529   
std      0.143378    0.103117    0.129924    0.116551    0.111166    0.129182   
min     -0.123665   -0.103533   -0.095591   -0.089121   -0.084055   -0.117284   
25%     -0.037101   -0.059438   -0.003425    0.004502   -0.036968   -0.008316   
50%      0.043437   -0.038165    0.083142    0.079135   -0.001110    0.087770   
75%      0.177067   -0.016561    0.164672    0.156014    0.063807    0.182374   
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000   

             S465        S466        S467  
count  466.000000  466.000000  466.000000  
mean     0.108244    0.083739    0.034295  
std      0.124239    0.110004    0.104721  
min     -0.119478   -0.085893   -0.084523  
25%      0.012929    0.003796   -0.029172  
50%      0.101166    0.064838    0.000970  
75%      0.179246    0.148362    0.063946  
max      1.000000    1.000000    1.000000  

[8 rows x 467 columns]
```



```
import nltk
from nltk.tokenize import sent_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

# Download NLTK data
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt')
def read_transcript(file_path):
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            transcript = file.read()
        return transcript
    except Exception as e:
        print(f"Error reading file: {e}")
        return None
```

```python

def split_into_sentences(transcript):
    return sent_tokenize(transcript)

def calculate_cosine_similarity(sentences):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(sentences)
    cosine_sim = cosine_similarity(tfidf_matrix)
    return cosine_sim

def segment_sentences(sentences, cosine_sim, threshold=0.5):
    visited = [False] * len(sentences)
    segments = []

    for i in range(len(sentences)):
        if not visited[i]:
            segment = [sentences[i]]
            visited[i] = True

            # Check for similar sentences
            for j in range(i + 1, len(sentences)):
                if not visited[j] and cosine_sim[i][j] >= threshold:
                    segment.append(sentences[j])
                    visited[j] = True

            segments.append(segment)

    return segments

def print_segments(segments):
    for idx, segment in enumerate(segments, start=1):
        print(f"\nSegment {idx}:")
        for sentence in segment:
            print(f" - {sentence}")

def process_transcript(file_path, threshold=0.5):
    # Read the transcript
    transcript = read_transcript(file_path)
    if transcript is None:
        return

    # Split the transcript into sentences
    sentences = split_into_sentences(transcript)
    print("Sentences extracted:")
    for i, sentence in enumerate(sentences):
        print(f"S{i + 1}: {sentence}")

    # Calculate cosine similarity between sentences
    cosine_sim = calculate_cosine_similarity(sentences)

    # Segment sentences based on similarity
    segments = segment_sentences(sentences, cosine_sim, threshold)

    # Print segmented sentences
    print("\nSegmented Sentences:")
    print_segments(segments)

# File path to the transcript
file_path = '/content/transcript (8).txt'  # Replace with your file path

# Process the transcript and print segments with a similarity threshold of 0.5
process_transcript(file_path, threshold=0.1)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
Sentences extracted:
S1: 0 - 30: so sorry we know that India has seen a huge Revolution with digital payments we all thought that India is a place at
30 - 60: I think immediately after you play the next big revolution personally I think is an education and the complete homework
60 - 90: do you think ncrf plus any people together will be the next big revolution after up in India absolutely and why I think
90 - 120: how many changes have happened in the real world so many changes have happened in the requirement of the industry requi
120 - 150: July 2020 and we recently celebrated the 4th anniversary of NP 2020 ncrf has been brought to implement the intent of 2
150 - 180: this allows you all the Innovation the way you educate your kids you educate your students still it provides you the b
180 - 210: broken the shackles of which were there in the education sector right so yes it is a big Revolution and this is going
210 - 240: pull up my kitchen to add to my taste buds in a way that it's convenient for me and one fine day you came and you char
240 - 270: we have been driving buses at Max now we should fly how do we do this look at the requirement of the industry requirem
270 - 300: link that whatever he has been taught has no relevance to the real life world when industry is moving that fast when t
300 - 330: will you be not only beneficial but also very facilitated for all of us very liberating for all of us this is going to
330 - 360: new things create new ways of doing things learn something new but once we learn it there is no limit to Innovation an
360 - 390: creating our vision and Innovative Minds into the education sector where is we are applying it elsewhere everywhere el
390 - 420: to be fun for everyone and I can I can tell you that already a number of Institutions have adopted the any pain and CF
420 - 450: next step on how to implement an AP if I can request you to give me an elevator pitch for an EP and then an elevator p
450 - 480: it allows for creditor of all learnings weather in academics orange killing or an experiential learning and all these
```

```
480 - 510: and people who are already skilled or already in the professional area not there the experiential learning would play
510 - 540: increase of technology which has been created by single Department good question so therefore all of us we have to wor
540 - 570: all kinds of learning are being contractors including learning of soft skills employability skills life skills your ha
570 - 600: go out it's all very flexible so that there's no Dropout there's no Dropout so these three things coupled with use of
600 - 630: I want my son to be an engineer don't you think if we create a give me five approach to Credit Systems everybody will
630 - 660: turn off infrastructure India has today we created in last 1775 years we are going to double that infrastructure in ne
660 - 690: more number of other branches even liberal arts social sciences if I want to be an award Society I would need a proper
690 - 720: have you already seen the Fallout of this know you can see the photo you can see how many Engineers are there for Desi
720 - 750: [Unintelligible]
750 - 780: any of the new technology machines current any laser-based machines any automated operating machines robotic operation
780 - 810: robotic process and that is killing is equally important and this is important in multiple areas and therefore multidi
810 - 840: me and you want to know design a VTech in CSC syllabus or be taking AI syllabus that is Nip complaint under the ncrf
840 - 870: are you teaching Teddy teaching Terry is not sufficient if you want the student to really understand and reply that co
870 - 900: those horses are skill bass courses you divide every subject into 30 and its application how do you apply the theory
900 - 930: who is learning which is happening which which you are going through so then this looks like let's say my student stay
930 - 960: extension of BSC physics or BSC chemistry and you give him the actual knowledge of computer science right in the first
960 - 990: that's unbelievable so that 50% of the time which means two full years and a btech program a person can stay outside
990 - 1020: are you learning outcomes and their alignment with the overall curricular structure and then once it comes back we ha
1020 - 1050: he gets the credits okay so here is where I have talked with some inhibitions about the entire setup where you are
1050 - 1080: write although we can keep that a check it is not easy for us to keep some zones green some zones red operational co
1080 - 1110: what is the guarantee that the student is learning in the campus is there a is there an accident on some kind of a
1110 - 1140: either online or with some time stamps it is it is being documented know what time is done and CVT has videographed
1140 - 1170: she claims that all right he is a good technician and he can repair any kind of car so she prepares you open the BMW
1170 - 1200: Julie appointed by the awarding body which Awards the certificate and then that video is kept forever so you imagine
1200 - 1230: who is giving us I think that's a nail on the head where I think we all should pass for a moment and then think is
1230 - 1260: experimenting something like this I don't think we'll be damaging I think all the Institute should come out of their
1260 - 1290: [Unintelligible]
1290 - 1320: play nursery and she says it lightly and the answer lies in this new framework and know how many times as you rightl
1320 - 1350: it can be adopted but colleges that are approved by a city let's say a state Technical University may not know these
1350 - 1380: of course a price them go and talk to them and then educate them but over and Beyond this there are some subtle prob
1380 - 1410: play by 50% or I'll engage my faculty for the betterment of my students at a level which need not necessarily be tea
1410 - 1440: picture Affiliated to these 1200 University leaving all the ionis that is Institute of national importance but they
S2: 3 4 4
1440 - 1470: do you think this ratio is sufficient to teach a technical subject know once we are sending our students out again
1470 - 1500: it's not easy to create a project which is outcome based creating that project itself is going to take a lot of time
```

```python
import nltk
from nltk.tokenize import sent_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

# Download NLTK data
nltk.download('punkt')

def read_transcript(file_path):
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            transcript = file.read()
        return transcript
    except Exception as e:
        print(f"Error reading file: {e}")
        return None

def split_into_sentences(transcript):
    return sent_tokenize(transcript)

def calculate_cosine_similarity(sentences):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(sentences)
    cosine_sim = cosine_similarity(tfidf_matrix)
    return cosine_sim

def segment_sentences(sentences, cosine_sim, threshold=0.5):
    visited = [False] * len(sentences)
    segments = []

    for i in range(len(sentences)):
        if not visited[i]:
            segment = [sentences[i]]
            visited[i] = True

            # Check for similar sentences
            for j in range(i + 1, len(sentences)):
                if not visited[j] and cosine_sim[i][j] >= threshold:
                    segment.append(sentences[j])
                    visited[j] = True

            segments.append(segment)

    return segments

def print_segments(segments):
    for idx, segment in enumerate(segments, start=1):
        print(f"\nSegment {idx}:")
        for sentence in segment:
            print(f" - {sentence}")

# 🆕 New Function to Save Segments to a Text File
def save_segments_to_file(segments, output_file):
    try:
        with open(output_file, 'w', encoding='utf-8') as file:
            for idx, segment in enumerate(segments, start=1):
                file.write(f"Segment {idx}:\n")
                for sentence in segment:
                    file.write(f" - {sentence}\n")
                file.write("\n")
        print(f"Segmented output saved to {output_file}")
    except Exception as e:
        print(f"Error saving file: {e}")

def process_transcript(file_path, threshold=0.5, output_file='segmented_output.txt'):
    # Read the transcript
    transcript = read_transcript(file_path)
    if transcript is None:
        return

    # Split the transcript into sentences
    sentences = split_into_sentences(transcript)
    print("Sentences extracted:")
    for i, sentence in enumerate(sentences):
        print(f"S{i + 1}: {sentence}")

    # Calculate cosine similarity between sentences
    cosine_sim = calculate_cosine_similarity(sentences)

    # Segment sentences based on similarity
    segments = segment_sentences(sentences, cosine_sim, threshold)

    # Print segmented sentences
    print("\nSegmented Sentences:")
    print_segments(segments)

    # Save segments to a text file
    save_segments_to_file(segments, output_file)

# File paths
file_path = '/content/transcript (5).txt'  # Replace with your file path
output_file = '/content/segmented_output.txt'  # Output text file path

# Process the transcript, segment it, and save to a text file
process_transcript(file_path, threshold=0.15, output_file=output_file)
```

```
Sentences extracted:
S1: so, sir, we know that India has seen a huge Revolution with digital payments.
S2: we all thought that India is a place- at least the West thought- that India is a place where, uh, many people do not get a sc
S3: that was the narrative some 30 years ago.
S4: and not many are literate- people cannot read, um.
S5: but then we have now shown that digital payments number one is India, while people thought that it wouldn't even come to top
S6: I think, immediately after UPA, the next big revolution, personally, I think, is in education, and the complete homework for
S7: sir, my question is: do you think ncrf plus NEP put together will be the next big revolution after UPA in India?
S8: absolutely, and why I think so is because in education, the last policy came up many, many years ago- that was in 1986, which
S9: so so many changes have happened in the real world.
S10: so many changes have happened in the requirement of the industry, requirement of Manpower for the industry.
S11: however, there were no corresponding changes which happened?
S12: education system.
S13: so, therefore, I feel that this was the right time when we brought in the education policy 2020, honorable prime minister, c
S14: 2020 is a policy, the NP 2020 is a policy, and for implementing a policy, you need a framework.
S15: now, why we call it a framework?
S16: we call it a framework because this is very flexible.
S17: this allows you all the Innovation, the way you educate your kids, you educate your students.
S18: still, it provides you the basic guidelines, the, the framework, the outer layer it provides, and that layer is mostly the e
S19: that is such an enabling layer that it has broken the shackles which were there in the education sector, right?
S20: so, yes, it is a big Revolution and this is going to change the way we have been educating our kids and this will be Game Ch
S21: so, sir, I think, um, uh, let's go with this example of, let's say, I started off living in a small 2bhk apartment M and I s
S22: right, and one fine day, you came and you changed my kitchen completely.
S23: I was using a bicycle, I moved to a scooter and a car and now you're asking me to fly and you're giving me an aircraft.
S24: NEP sounds more like that for me.
S25: how do I do the transition?
S26: I fear that I will.
S27: I will crash if I use a aeroplane without training.
S28: I'm talking about all the teachers in the in the country, all the schools in the country.
S29: we have been driving buses at Max.
S30: now we should fly.
S31: how do we do this?
S32: okay, look at the requirement of the industry.
S33: requirement of the industry has been moving very fast.
S34: the technology is emerging every day and the industry is moving with that speed.
S35: so when a student is coming out of your Institute and is going out in the market, he finds that whatever he has been taught
```

```
S36: when industry is moving that fast, when the requirement is moving that fast, don't you think it is important for us to chang
S37: yes, how long can we wait?
S38: yes, it will take a lot of effort for every one of us to adapt to this change.
S39: but this change is going to be not only beneficial, mhm, but also very facilitative for all of us, very liberating for all o
S40: this is going to be highly liberative and choice-based system.
S41: there are number of choices which are available to you, which are available to every student.
S42: yes, when we introduce a new system, we have to really create, create new things, create new ways of doing things, learn so
S43: but once we learn it, there is no limit to Innovation and creativity which this will bring.
S44: there's going to be a learning curve here.
S45: you mean, it's going to be difficult to fly a fighter, not really difficult to apply, because all of us we have that kind o
S46: only thing is that we were not translating our vision and Innovative Minds into the education sector, whereas we are applyi
S47: know we are, we are moving very fast.
S48: it's only education and I think the education sector is the smartest sector, which is very good at learning, and I'm I'm ver
S49: I can tell you that already a number of Institutions have adopted the NEP and ncrf to varying degrees and whereever they hav
S50: so then, um, um, I have spoken to many academics.
S51: most of us do not understand NEP.
S52: let on ncrf, which is the next step on how to implement NEP.
S53: if I can request you to give me an elevator pitch for NEP and then an elevator pitch for ncrf, for the exclusive reason that
S54: okay, see, if I talk about one simple line on NP and ncf, that is, it is crediti of all learning.
S55: it allows for crediti of all learnings, whether in academics or in Skilling or in experiential learning, and all these three
```

```python
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

# Download NLTK data
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords

def read_transcript(file_path):
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            transcript = file.read()
        return transcript
    except Exception as e:
        print(f"Error reading file: {e}")
        return None

def split_into_sentences(transcript):
    return sent_tokenize(transcript)

def calculate_cosine_similarity(sentences):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(sentences)
    cosine_sim = cosine_similarity(tfidf_matrix)
    return cosine_sim

def segment_sentences(sentences, cosine_sim, threshold=0.5):
    visited = [False] * len(sentences)
    segments = []
    for i in range(len(sentences)):
        if not visited[i]:
            segment = [sentences[i]]
            visited[i] = True
            for j in range(i + 1, len(sentences)):
                if not visited[j] and cosine_sim[i][j] >= threshold:
                    segment.append(sentences[j])
                    visited[j] = True
            segments.append(segment)
    return segments

def remove_stopwords(segment):
    stop_words = set(stopwords.words('english'))
    filtered_words = []
    for sentence in segment:
        words = word_tokenize(sentence)
        filtered_words.extend([word.lower() for word in words if word.isalnum() and word.lower() not in stop_words])
    return filtered_words

def find_keywords(filtered_words):
    if len(filtered_words) < 2:
        return None, None, 0  # Not enough words for comparison

    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(filtered_words)
    cosine_sim = cosine_similarity(tfidf_matrix)

    max_sim = 0
    keyword_pair = (None, None)
    for i in range(len(filtered_words)):
        for j in range(i + 1, len(filtered_words)):
            if cosine_sim[i][j] > max_sim:
                max_sim = cosine_sim[i][j]
                keyword_pair = (filtered_words[i], filtered_words[j])
    return keyword_pair[0], keyword_pair[1], max_sim

def save_segments_to_csv(segments, output_file):
    with pd.ExcelWriter(output_file) as writer:
        for idx, segment in enumerate(segments, start=1):
            filtered_words = remove_stopwords(segment)
            word1, word2, max_sim = find_keywords(filtered_words)

            # Create a DataFrame for the segment
            df = pd.DataFrame({
                'Word 1': [word1] if word1 else [],
                'Word 2': [word2] if word2 else [],
                'Cosine Similarity': [max_sim] if word1 and word2 else [],
                'Keyword': [f"{word1}, {word2}"] if word1 and word2 else []
            })

            # Save each segment as a separate sheet
            df.to_excel(writer, sheet_name=f'Segment {idx}', index=False)

    print(f"Segmented keywords saved to {output_file}")

def process_transcript(file_path, threshold=0.5, output_file='segmented_keywords.xlsx'):
    transcript = read_transcript(file_path)
    if transcript is None:
        return

    sentences = split_into_sentences(transcript)
    cosine_sim = calculate_cosine_similarity(sentences)
    segments = segment_sentences(sentences, cosine_sim, threshold)

    save_segments_to_csv(segments, output_file)

    print("\nChosen Keywords for Each Segment:")
    for idx, segment in enumerate(segments, start=1):
        filtered_words = remove_stopwords(segment)
        word1, word2, max_sim = find_keywords(filtered_words)
        if word1 and word2:
            print(f"Segment {idx}: Keywords = {word1}, {word2} (Similarity: {max_sim:.2f})")
        else:
            print(f"Segment {idx}: Not enough data for keywords")

# File paths
file_path = '/content/transcript (5).txt'  # Replace with your file path
output_file = '/content/segmented_keywords.xlsx'  # Output Excel file path

# Run the process
process_transcript(file_path, threshold=0.1, output_file=output_file)

```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Segmented keywords saved to /content/segmented_keywords.xlsx

Chosen Keywords for Each Segment:
Segment 1: Keywords = sir, sir (Similarity: 1.00)
Segment 2: Keywords = 30, 30 (Similarity: 1.00)
Segment 3: Keywords = many, many (Similarity: 1.00)
Segment 4: Keywords = many, many (Similarity: 1.00)
Segment 5: Keywords = education, education (Similarity: 1.00)
Segment 6: Keywords = brought, brought (Similarity: 1.00)
Segment 7: Keywords = call, call (Similarity: 1.00)
Segment 8: Keywords = allows, allows (Similarity: 1.00)
Segment 9: Keywords = um, um (Similarity: 1.00)
Segment 10: Keywords = car, car (Similarity: 1.00)
Segment 11: Keywords = yes, yes (Similarity: 1.00)
Segment 12: Keywords = know, know (Similarity: 1.00)
Segment 13: Keywords = use, use (Similarity: 1.00)
Segment 14: Keywords = level, level (Similarity: 1.00)
Segment 15: Keywords = yes, yes (Similarity: 1.00)
Segment 16: Keywords = going, going (Similarity: 1.00)
Segment 17: Keywords = thing, thing (Similarity: 1.00)
Segment 18: Keywords = number, number (Similarity: 1.00)
Segment 19: Keywords = okay, okay (Similarity: 1.00)
Segment 20: Keywords = students, students (Similarity: 1.00)
Segment 21: Not enough data for keywords
Segment 22: Keywords = single, single (Similarity: 1.00)
Segment 23: Keywords = good, good (Similarity: 1.00)
Segment 24: Keywords = curricular, curricular (Similarity: 1.00)
Segment 25: Keywords = catch, catch (Similarity: 1.00)
Segment 26: Keywords = short, short (Similarity: 1.00)
Segment 27: Keywords = need, need (Similarity: 1.00)
Segment 28: Keywords = important, important (Similarity: 1.00)
Segment 29: Keywords = unable, unable (Similarity: 1.00)
Segment 30: Keywords = syllabus, syllabus (Similarity: 1.00)
Segment 31: Keywords = already, already (Similarity: 1.00)
Segment 32: Keywords = theory, theory (Similarity: 1.00)
Segment 33: Keywords = student, student (Similarity: 1.00)
Segment 34: Keywords = three, three (Similarity: 1.00)
Segment 35: Keywords = talking, talking (Similarity: 1.00)
Segment 36: Not enough data for keywords
Segment 37: Not enough data for keywords
Segment 38: Not enough data for keywords
Segment 39: Not enough data for keywords
Segment 40: Not enough data for keywords
Segment 41: Keywords = project, project (Similarity: 1.00)
Segment 42: Keywords = inside, inside (Similarity: 1.00)
Segment 43: Keywords = think, think (Similarity: 1.00)
Segment 44: Keywords = contain, contain (Similarity: 1.00)
Segment 45: Keywords = really, really (Similarity: 1.00)
Segment 46: Not enough data for keywords
Segment 47: Not enough data for keywords
Segment 48: Keywords = mhm, mhm (Similarity: 1.00)
Segment 49: Not enough data for keywords
Segment 50: Not enough data for keywords
Segment 51: Not enough data for keywords
```

```python
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

# Download NLTK data
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords

def read_segmented_file(file_path):
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            content = file.read()
            # Split the file content into segments based on a delimiter (e.g., "\n\n")
            segments = [segment.strip() for segment in content.split("\n\n") if segment.strip()]
        return segments
    except Exception as e:
        print(f"Error reading file: {e}")
        return None

def remove_stopwords(segment):
    stop_words = set(stopwords.words('english'))
    words = word_tokenize(segment)
    return [word.lower() for word in words if word.isalnum() and word.lower() not in stop_words]

def calculate_word_cosine_similarity(filtered_words):
    if len(filtered_words) < 2:
        return None, None, 0, None  # Not enough words for comparison

    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(filtered_words)
    cosine_sim = cosine_similarity(tfidf_matrix)

    max_sim = 0
    keyword_pair = (None, None)
    for i in range(len(filtered_words)):
        for j in range(i + 1, len(filtered_words)):
            if cosine_sim[i][j] > max_sim:
                max_sim = cosine_sim[i][j]
                keyword_pair = (filtered_words[i], filtered_words[j])

    return keyword_pair[0], keyword_pair[1], max_sim, cosine_sim

def save_word_similarity_to_excel(cosine_sim, filtered_words, sheet_name, writer):
    if cosine_sim is not None:
        df = pd.DataFrame(cosine_sim, index=filtered_words, columns=filtered_words)
        df.to_excel(writer, sheet_name=sheet_name)
    else:
        df = pd.DataFrame(columns=['Info'])
        df.loc[0] = ["Not enough words for comparison"]
        df.to_excel(writer, sheet_name=sheet_name, index=False)

def save_keywords_summary_to_csv(keywords, output_file):
    summary_df = pd.DataFrame(keywords, columns=['Segment', 'Word 1', 'Word 2', 'Cosine Similarity', 'Keyword'])
    summary_df.to_csv(output_file, index=False)
    print(f"Keywords summary saved to {output_file}")
```

```python

def process_segments(file_path, output_excel='segment_word_similarity.xlsx', summary_csv='segment_keywords_summary.csv'):
    segments = read_segmented_file(file_path)
    if segments is None:
        return

    keywords_summary = []

    with pd.ExcelWriter(output_excel) as writer:
        for idx, segment in enumerate(segments, start=1):
            # Remove stopwords
            filtered_words = remove_stopwords(segment)

            # Calculate cosine similarity between words
            word1, word2, max_sim, cosine_sim = calculate_word_cosine_similarity(filtered_words)

            # Save word similarity matrix for each segment to Excel
            save_word_similarity_to_excel(cosine_sim, filtered_words, f'Segment {idx}', writer)

            # Prepare summary data
            if word1 and word2:
                keyword = f"{word1}, {word2}"
                keywords_summary.append([f"Segment {idx}", word1, word2, max_sim, keyword])
            else:
                keywords_summary.append([f"Segment {idx}", "N/A", "N/A", 0, "N/A"])

    # Save summary of keywords to CSV
    save_keywords_summary_to_csv(keywords_summary, summary_csv)

    # Print chosen keywords for each segment
    print("\nChosen Keywords for Each Segment:")
    for row in keywords_summary:
        print(f"{row[0]}: Keywords = {row[4]} (Similarity: {row[3]:.2f})")

# File paths
segmented_file_path = '/content/segmented_output.txt'  # Replace with your segmented text file path
output_excel = '/content/segment_word_similarity.xlsx'  # Excel file for word similarity matrices
summary_csv = '/content/segment_keywords_summary.csv'  # CSV file for keywords summary

# Run the process
process_segments(segmented_file_path, output_excel=output_excel, summary_csv=summary_csv)

```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Keywords summary saved to /content/segment_keywords_summary.csv

Chosen Keywords for Each Segment:
Segment 1: Keywords = sir, sir (Similarity: 1.00)
Segment 2: Keywords = thought, thought (Similarity: 1.00)
Segment 3: Keywords = 30, 30 (Similarity: 1.00)
Segment 4: Keywords = think, think (Similarity: 1.00)
Segment 5: Keywords = however, however (Similarity: 1.00)
Segment 6: Keywords = education, education (Similarity: 1.00)
Segment 7: Keywords = 2020, 2020 (Similarity: 1.00)
Segment 8: Keywords = call, call (Similarity: 1.00)
Segment 9: Keywords = way, way (Similarity: 1.00)
Segment 10: Keywords = provides, provides (Similarity: 1.00)
Segment 11: Keywords = uh, uh (Similarity: 1.00)
Segment 12: Keywords = N/A (Similarity: 0.00)
Segment 13: Keywords = saying, saying (Similarity: 1.00)
Segment 14: Keywords = nep, nep (Similarity: 1.00)
Segment 15: Keywords = say, say (Similarity: 1.00)
Segment 16: Keywords = fact, fact (Similarity: 1.00)
Segment 17: Keywords = without, without (Similarity: 1.00)
Segment 18: Keywords = talking, talking (Similarity: 1.00)
Segment 19: Keywords = N/A (Similarity: 0.00)
Segment 20: Keywords = okay, okay (Similarity: 1.00)
Segment 21: Keywords = student, student (Similarity: 1.00)
Segment 22: Keywords = yes, yes (Similarity: 1.00)
Segment 23: Keywords = yes, yes (Similarity: 1.00)
Segment 24: Keywords = new, new (Similarity: 1.00)
Segment 25: Keywords = going, going (Similarity: 1.00)
Segment 26: Keywords = minds, minds (Similarity: 1.00)
Segment 27: Keywords = already, already (Similarity: 1.00)
Segment 28: Keywords = um, um (Similarity: 1.00)
Segment 29: Keywords = okay, okay (Similarity: 1.00)
Segment 30: Keywords = already, already (Similarity: 1.00)
Segment 31: Keywords = interdisciplinarity, interdisciplinarity (Similarity: 1.00)
Segment 32: Keywords = single, single (Similarity: 1.00)
Segment 33: Keywords = good, good (Similarity: 1.00)
Segment 34: Keywords = curricular, curricular (Similarity: 1.00)
Segment 35: Keywords = learning, learning (Similarity: 1.00)
Segment 36: Keywords = multiple, multiple (Similarity: 1.00)
Segment 37: Keywords = three, three (Similarity: 1.00)
Segment 38: Keywords = short, short (Similarity: 1.00)
Segment 39: Keywords = think, think (Similarity: 1.00)
Segment 40: Keywords = infrastructure, infrastructure (Similarity: 1.00)
Segment 41: Keywords = N/A (Similarity: 0.00)
Segment 42: Keywords = feel, feel (Similarity: 1.00)
Segment 43: Keywords = important, important (Similarity: 1.00)
Segment 44: Keywords = imagine, imagine (Similarity: 1.00)
Segment 45: Keywords = unable, unable (Similarity: 1.00)
Segment 46: Keywords = design, design (Similarity: 1.00)
Segment 47: Keywords = computer, computer (Similarity: 1.00)
Segment 48: Keywords = courses, courses (Similarity: 1.00)
Segment 49: Keywords = 50, 50 (Similarity: 1.00)
Segment 50: Keywords = keep, keep (Similarity: 1.00)
Segment 51: Keywords = N/A (Similarity: 0.00)
```

```python
import nltk
from nltk.tokenize import word_tokenize
import re
import pandas as pd

# Download NLTK data if not already available
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords

# Define filler words to remove
FILLER_WORDS = {'umm', 'uh', 'oh', 'okay', 'like', 'you know', 'actually', 'basically', 'literally', 'well', 'so', 'just', 'i mean',

def read_segmented_file(file_path):
    """Read segmented text file and split into segments."""
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            content = file.read()
            # Split the file content into segments based on double newlines
            segments = [segment.strip() for segment in content.split("\n\n") if segment.strip()]
        return segments
    except Exception as e:
        print(f"Error reading file: {e}")
        return None

def clean_text(segment):
    """Remove stopwords, fillers, and dates from a segment."""
    stop_words = set(stopwords.words('english'))

    # Remove dates (e.g., 12/03/2023, March 12, 2023)
    segment = re.sub(r'\b(\d{1,2}[/-]\d{1,2}[/-]\d{2,4}|\b(?:January|February|March|April|May|June|July|August|September|October|Nov

    # Tokenize and clean words
    words = word_tokenize(segment)
    filtered_words = [
        word.lower() for word in words
        if word.isalnum() and word.lower() not in stop_words and word.lower() not in FILLER_WORDS
    ]
    return ' '.join(filtered_words)

def save_to_csv(segments, cleaned_segments, output_file):
    """Save original and cleaned segments to a CSV file."""
    df = pd.DataFrame({
        'Original Segment': segments,
        'Cleaned Segment (No Stopwords or Fillers)': cleaned_segments
    })
    df.to_csv(output_file, index=False)
    print(f"Cleaned segments saved to {output_file}")

def process_segments(file_path, output_csv='cleaned_segments.csv'):
    """Process the segmented text file."""
    segments = read_segmented_file(file_path)
    if segments is None:
        return

    cleaned_segments = [clean_text(segment) for segment in segments]
    save_to_csv(segments, cleaned_segments, output_csv)

    # Print a preview of cleaned segments
    print("\nPreview of Cleaned Segments:")
    for idx, (orig, clean) in enumerate(zip(segments, cleaned_segments), start=1):
        print(f"\nSegment {idx} (Original): {orig}")
        print(f"Segment {idx} (Cleaned): {clean}")

# File paths
segmented_file_path = '/content/segmented_output.txt'  # Replace with your segmented text file path
output_csv = '/content/cleaned_segments.csv'  # CSV file for cleaned segments

```

```
# Run the process
process_segments(segmented_file_path, output_csv=output_csv)

```

```
Cleaned segments saved to /content/cleaned_segments.csv

Preview of Cleaned Segments:

Segment 1 (Original): Segment 1:
 - so, sir, we know that India has seen a huge Revolution with digital payments.
 - but then we have now shown that digital payments number one is India, while people thought that it wouldn't even come to top 5
 - sir, my question is: do you think ncrf plus NEP put together will be the next big revolution after UPA in India?
 - know we are, we are moving very fast.
 - wonderful, sir.
 - we all know a lot of scandals.
 - so that has been provided here.
 - not only that India is grappling with this problem.
Segment 1 (Cleaned): segment 1 sir know india seen huge revolution digital payments shown digital payments number one india peopl

Segment 2 (Original): Segment 2:
 - we all thought that India is a place- at least the West thought- that India is a place where, uh, many people do not get a squ
 - and not many are literate- people cannot read, um.
 - how do we do this?
 - you have place for everything.
 - okay, you look at the kind of the, the requirement of infrastructure India has today.
 - so here is where I have um, um, um thought, with some inhibitions, about the entire setup, where you are simply assuming a lot
 - so that I know the student is learning, the student knows that here is a genuine place and the person there also knows that we
 - the university is not giving me what I thought it would.
 - we have, uh, which is not appearing at the surface level, which is the student teacher ratio.
 - it is uh, it is really uh, against the students, right, right, already, it's already so, but then, uh, the colleges should not
 - so here's where I think there's a foot for Thought, where I will, uh, try to slightly disagree.
 - I I see that as as this is a wonderful thought, but I think the students wants to see those short videos not in these alignmen
 - that is where the problem lies.
 - that that is where the problem lies.
 - that is, the hecaton is is all about.
 - that's a very good thought.
 - why is it that we have not thought about all these things worldwide?
 - so for why see why this was not not even thought about?
 - we did that, mhm, and that has been done not only once but at least twice, multiple times.
 - that is the framework.
 - it gives me some non-trivial money, at least enough for me to eat and stay in a place.
Segment 2 (Cleaned): segment 2 thought india least west india place many people get square meal right many people read um place e

Segment 3 (Original): Segment 3:
 - that was the narrative some 30 years ago.
 - absolutely, and why I think so is because in education, the last policy came up many, many years ago- that was in 1986, which
 - I was trying to answer.
 - that was the second part which I wanted to answer.
 - it was.
 - Manan is going to be 30 seconds only.
 - and even in that 7 and a half minutes, that concept is broken into three parts of 2 minutes each, followed by 30 Seconds of sc
 - earlier this was not there.
Segment 3 (Cleaned): segment 3 narrative 30 years ago absolutely think education last policy came many many years 1986 slightly t

Segment 4 (Original): Segment 4:
 - I think, immediately after UPA, the next big revolution, personally, I think, is in education, and the complete homework for t
 - so so many changes have happened in the real world.
 - so many changes have happened in the requirement of the industry, requirement of Manpower for the industry.
 - so, therefore, I feel that this was the right time when we brought in the education policy 2020, honorable prime minister, dec
 - we call it a framework because this is very flexible.
 - that is such an enabling layer that it has broken the shackles which were there in the education sector, right?
```

```
import json
import re

def load_transcript_from_file(file_path):
    """
    Loads the transcript from a file.
    Args:
        file_path (str): Path to the transcript file.
    Returns:
        list: List of dictionaries with 'start', 'end', and 'text' keys.
    """
    try:
        with open(file_path, "r", encoding="utf-8") as file:
            if file_path.endswith(".json"):
                transcript = json.load(file)
            else:
                # For plain text files, assume each line is in the format: [start] - [end]: [text]
                transcript = []
                for line in file:
                    match = re.match(r"(\d+\.\d+) - (\d+\.\d+): (.+)", line.strip())
                    if match:
                        start, end, text = match.groups()
                        transcript.append({
                            "start": float(start),
                            "end": float(end),
                            "text": text
                        })
            return transcript
    except Exception as e:
        print(f"Error loading transcript: {e}")
        return None

def save_transcript_with_timestamps(transcript, output_path="transcript_with_timestamps.txt"):
    """
    Saves the transcript with timestamps to a text file.
    Args:
        transcript (list): List of dictionaries with 'start', 'end', and 'text'.
        output_path (str): The path to save the output file.
    """
    with open(output_path, "w", encoding="utf-8") as file:
        for segment in transcript:
            file.write(f"{segment['start']} - {segment['end']}: {segment['text']}\n")
    print(f"Transcript with timestamps saved to {output_path}")

# Example usage
if __name__ == "__main__":
    # Ask the user for the transcript file path
    transcript_path = input("Enter the path to the transcript file: ")
    output_path = input("Enter the path to save the transcript with timestamps (default: transcript_with_timestamps.txt): ") or "tra

    # Load the transcript from the file
    transcript = load_transcript_from_file(transcript_path)
    if not transcript:
        print("Failed to load transcript. Exiting.")
        exit()

    # Print the transcript with timestamps
    print("\nTranscript with Timestamps:")
    for segment in transcript:
        print(f"{segment['start']} - {segment['end']}: {segment['text']}")

    # Save the transcript with timestamps to a file
    save_transcript_with_timestamps(transcript, output_path)
```

```
Enter the path to the transcript file: https://youtu.be/sK8SILOM37I
Enter the path to save the transcript with timestamps (default: transcript_with_timestamps.txt):
Error loading transcript: [Errno 2] No such file or directory: 'https://youtu.be/sK8SILOM37I'
Failed to load transcript. Exiting.

Transcript with Timestamps:
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-31-41113dfd0e0a> in <cell line: 0>()
     57     # Print the transcript with timestamps
     58     print("\nTranscript with Timestamps:")
---> 59     for segment in transcript:
     60         print(f"{segment['start']} - {segment['end']}: {segment['text']}")
     61 

TypeError: 'NoneType' object is not iterable
```

```
import re
import urllib.parse
import requests
from youtube_transcript_api import YouTubeTranscriptApi
from pytube import YouTube
import speech_recognition as sr
from pydub import AudioSegment
import os
import yt_dlp
def extract_video_id(video_url):
    """
    Extracts the YouTube video ID from various URL formats.
    """
    parsed_url = urllib.parse.urlparse(video_url)
    query_params = urllib.parse.parse_qs(parsed_url.query)

    if "v" in query_params:
        return query_params["v"][0]

    match = re.search(r"(youtu\.be/|youtube\.com/embed/|youtube\.com/shorts/)([\w-]+)", video_url)
    if match:
        return match.group(2)

    return None

```

```
def download_audio(video_url):
    """
    Downloads the audio using yt-dlp with cookies and returns the file path.
    """
    try:
        ydl_opts = {
            'format': 'bestaudio/best',
            'outtmpl': 'audio.%(ext)s',
            'cookiefile': '/content/cookies (2).txt',  # Use the exported cookies
            'postprocessors': [{
                'key': 'FFmpegExtractAudio',
                'preferredcodec': 'mp3',
                'preferredquality': '192',
            }],
        }
        with yt_dlp.YoutubeDL(ydl_opts) as ydl:
            info = ydl.extract_info(video_url, download=True)
            return "audio.mp3"
    except Exception as e:
        return f"Error downloading audio: {str(e)}"

def convert_audio_to_wav(audio_file):
    """
    Converts the downloaded MP3 audio to WAV format using pydub.
    """
    wav_file = "audio.wav"
    try:
        AudioSegment.from_mp3(audio_file).export(wav_file, format="wav")
        return wav_file
    except Exception as e:
        return f"Error converting to WAV: {str(e)}"

def transcribe_audio(audio_path, chunk_length=30):
    """
    Splits audio into smaller chunks and transcribes each chunk separately.
    Args:
        audio_path (str): Path to the audio file.
        chunk_length (int): Length of each chunk in seconds (default: 30).
    Returns:
        list: List of dictionaries containing transcribed text and timestamps.
    """
    recognizer = sr.Recognizer()
    audio = AudioSegment.from_wav(audio_path)
    total_duration = len(audio) / 1000  # Convert to seconds
    transcribed_segments = []

    print("Transcribing audio in chunks...")

    # Split and transcribe audio in chunks
    for start in range(0, int(total_duration), chunk_length):
        end = min(start + chunk_length, int(total_duration))
        chunk = audio[start * 1000:end * 1000]  # Extract chunk in milliseconds
        chunk.export("chunk.wav", format="wav")  # Save chunk temporarily

        with sr.AudioFile("chunk.wav") as source:
            try:
                audio_data = recognizer.record(source)
                text = recognizer.recognize_google(audio_data)
                transcribed_segments.append({
                    "start": start,
                    "end": end,
                    "text": text
                })
            except sr.UnknownValueError:
                transcribed_segments.append({
                    "start": start,
                    "end": end,
                    "text": "[Unintelligible]"
                })
            except sr.RequestError as e:
                return f"Error with the speech recognition service: {str(e)}"

    os.remove("chunk.wav")  # Clean up temporary chunk file
    return transcribed_segments

def get_transcript_unlisted(video_url):
    """
    Tries to fetch the transcript using youtube_transcript_api first,
    then falls back to downloading and transcribing audio if necessary.
    """
    video_id = extract_video_id(video_url)
    if not video_id:
```

```python
        return "Invalid YouTube URL."

    # Try to fetch transcript using youtube_transcript_api
    try:
        transcript = YouTubeTranscriptApi.get_transcript(video_id)
        # Add 'end' time to each segment
        for segment in transcript:
            segment["end"] = segment["start"] + segment["duration"]
        return transcript  # Return transcript with timestamps
    except:
        print("Transcript not available via API, attempting audio transcription...")

    # Download and transcribe audio if no transcript is available
    audio_file = download_audio(video_url)
    if "Error" in audio_file:
        return audio_file

    wav_file = convert_audio_to_wav(audio_file)
    if "Error" in wav_file:
        return wav_file

    transcription = transcribe_audio(wav_file)

    # Cleanup temporary files
    os.remove(audio_file)
    os.remove(wav_file)

    return transcription

def save_transcript_to_file(transcript, filename="transcript.txt"):
    """
    Saves the transcript to a text file.
    Args:
        transcript (list or str): The transcript to save.
        filename (str): The name of the output file.
    """
    with open(filename, "w", encoding="utf-8") as file:
        if isinstance(transcript, list):
            for segment in transcript:
                file.write(f"{segment['start']} - {segment['end']}: {segment['text']}\n")
        else:
            file.write(transcript)
    print(f"Transcript saved to {filename}")

# Example usage
if __name__ == "__main__":
    video_url = input("Enter the YouTube video URL: ")
    transcript = get_transcript_unlisted(video_url)

    if isinstance(transcript, list):
        print("\nTranscript with Timestamps:")
        for segment in transcript:
            print(f"{segment['start']} - {segment['end']}: {segment['text']}")
    else:
        print("\nTranscript:\n", transcript)

    # Save transcript to a text file
    save_transcript_to_file(transcript, "transcript.txt")
```

```
Enter the YouTube video URL: https://youtu.be/sK8SILOM37I
Transcript not available via API, attempting audio transcription...
[youtube] Extracting URL: https://youtu.be/sK8SILOM37I
[youtube] sK8SILOM37I: Downloading webpage
[youtube] sK8SILOM37I: Downloading tv client config
[youtube] sK8SILOM37I: Downloading player f6e09c70
[youtube] sK8SILOM37I: Downloading tv player API JSON
[info] sK8SILOM37I: Downloading 1 format(s): 251
[download] Destination: audio.webm
[download] 100% of   39.37MiB in 00:00:01 at 31.86MiB/s
[ExtractAudio] Destination: audio.mp3
Deleting original file audio.webm (pass -k to keep)
Transcribing audio in chunks...

Transcript with Timestamps:
0 - 30: so sorry we know that India has seen a huge Revolution with digital payments we all thought that India is a place at leas
30 - 60: I think immediately after you play the next big revolution personally I think is an education and the complete homework
60 - 90: do you think ncrf plus any people together will be the next big revolution after up in India absolutely and why I think
90 - 120: how many changes have happened in the real world so many changes have happened in the requirement of the industry requi
120 - 150: July 2020 and we recently celebrated the 4th anniversary of NP 2020 ncrf has been brought to implement the intent of 2
150 - 180: this allows you all the Innovation the way you educate your kids you educate your students still it provides you the b
180 - 210: broken the shackles of which were there in the education sector right so yes it is a big Revolution and this is going
210 - 240: pull up my kitchen to add to my taste buds in a way that it's convenient for me and one fine day you came and you char
240 - 270: we have been driving buses at Max now we should fly how do we do this look at the requirement of the industry requirem
270 - 300: link that whatever he has been taught has no relevance to the real life world when industry is moving that fast when t
```

```
300 - 330: will you be not only beneficial but also very facilitated for all of us very liberating for all of us this is going t
330 - 360: new things create new ways of doing things learn something new but once we learn it there is no limit to Innovation a
360 - 390: creating our vision and Innovative Minds into the education sector where is we are applying it elsewhere everywhere e
390 - 420: to be fun for everyone and I can I can tell you that already a number of Institutions have adopted the any pain and C
420 - 450: next step on how to implement an AP if I can request you to give me an elevator pitch for an EP and then an elevator
450 - 480: it allows for creditor of all learnings weather in academics orange killing or an experiential learning and all these
480 - 510: and people who are already skilled or already in the professional area not there the experiential learning would play
510 - 540: increase of technology which has been created by single Department good question so therefore all of us we have to wo
540 - 570: all kinds of learning are being contractors including learning of soft skills employability skills life skills your h
570 - 600: go out it's all very flexible so that there's no Dropout there's no Dropout so these three things coupled with use of
600 - 630: I want my son to be an engineer don't you think if we create a give me five approach to Credit Systems everybody will
630 - 660: turn off infrastructure India has today we created in last 1775 years we are going to double that infrastructure in n
660 - 690: more number of other branches even liberal arts social sciences if I want to be an award Society I would need a prope
690 - 720: have you already seen the Fallout of this know you can see the photo you can see how many Engineers are there for Des
720 - 750: [Unintelligible]
750 - 780: any of the new technology machines current any laser-based machines any automated operating machines robotic operatio
780 - 810: robotic process and that is killing is equally important and this is important in multiple areas and therefore multid
810 - 840: me and you want to know design a VTech in CSC syllabus or be taking AI syllabus that is Nip complaint under the ncrf
840 - 870: are you teaching Teddy teaching Terry is not sufficient if you want the student to really understand and reply that c
870 - 900: those horses are skill bass courses you divide every subject into 30 and its application how do you apply the theory
900 - 930: who is learning which is happening which which you are going through so then this looks like let's say my student sta
930 - 960: extension of BSC physics or BSC chemistry and you give him the actual knowledge of computer science right in the firs
960 - 990: that's unbelievable so that 50% of the time which means two full years and a btech program a person can stay outside
990 - 1020: are you learning outcomes and their alignment with the overall curricular structure and then once it comes back we h
1020 - 1050: he gets the credits okay so here is where I have talked with some inhibitions about the entire setup where you are
1050 - 1080: write although we can keep that a check it is not easy for us to keep some zones green some zones red operational
1080 - 1110: what is the guarantee that the student is learning in the campus is there a is there an accident on some kind of a
1110 - 1140: either online or with some time stamps it is it is being documented know what time is done and CVT has videographed
1140 - 1170: she claims that all right he is a good technician and he can repair any kind of car so she prepares you open the BM
1170 - 1200: Julie appointed by the awarding body which Awards the certificate and then that video is kept forever so you imagin
1200 - 1230: who is giving us I think that's a nail on the head where I think we all should pass for a moment and then think is
```

```
pip install yt-dlp
```

```
Collecting yt-dlp
  Downloading yt_dlp-2025.2.19-py3-none-any.whl.metadata (171 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 171.9/171.9 kB 3.1 MB/s eta 0:00:00
Downloading yt_dlp-2025.2.19-py3-none-any.whl (3.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.2/3.2 MB 36.3 MB/s eta 0:00:00
Installing collected packages: yt-dlp
Successfully installed yt-dlp-2025.2.19
```

```
pip install youtube-transcript-api
```

```
Collecting youtube-transcript-api
  Downloading youtube_transcript_api-0.6.3-py3-none-any.whl.metadata (17 kB)
Requirement already satisfied: defusedxml<0.8.0,>=0.7.1 in /usr/local/lib/python3.11/dist-packages (from youtube-transcript-api) (0
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from youtube-transcript-api) (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->youtube-transcri
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->youtube-transcript-api) (3.1
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->youtube-transcript-api
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->youtube-transcript-api
Downloading youtube_transcript_api-0.6.3-py3-none-any.whl (622 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 622.3/622.3 kB 9.6 MB/s eta 0:00:00
Installing collected packages: youtube-transcript-api
Successfully installed youtube-transcript-api-0.6.3
```

```
pip install SpeechRecognition
```

```
Collecting SpeechRecognition
  Downloading SpeechRecognition-3.14.1-py3-none-any.whl.metadata (31 kB)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from SpeechRecognition) (4.12.2)
Downloading SpeechRecognition-3.14.1-py3-none-any.whl (32.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 32.9/32.9 MB 34.0 MB/s eta 0:00:00
Installing collected packages: SpeechRecognition
Successfully installed SpeechRecognition-3.14.1
```

```
pip install pydub
```

```
Collecting pydub
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Installing collected packages: pydub
Successfully installed pydub-0.25.1
```

```
pip install pytube
```

```
Collecting pytube
  Downloading pytube-15.0.0-py3-none-any.whl.metadata (5.0 kB)
Downloading pytube-15.0.0-py3-none-any.whl (57 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 57.6/57.6 kB 3.2 MB/s eta 0:00:00
Installing collected packages: pytube
Successfully installed pytube-15.0.0
```

```python
import nltk
from nltk.tokenize import sent_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

# Download NLTK data
nltk.download('punkt')

def read_transcript(file_path):
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            transcript = file.read()
        return transcript
    except Exception as e:
        print(f"Error reading file: {e}")
        return None

def split_into_sentences(transcript):
    return sent_tokenize(transcript)

def calculate_cosine_similarity(sentences):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(sentences)
    cosine_sim = cosine_similarity(tfidf_matrix)
    return cosine_sim

def segment_sentences(sentences, cosine_sim, threshold=0.5, min_sentences=5):
    visited = [False] * len(sentences)
    segments = []

    for i in range(len(sentences)):
        if not visited[i]:
            segment = [sentences[i]]
            visited[i] = True

            for j in range(i + 1, len(sentences)):
                if not visited[j] and cosine_sim[i][j] >= threshold:
                    segment.append(sentences[j])
                    visited[j] = True

            segments.append(segment)

    # Merge smaller segments
    merged_segments = []
    temp_segment = []

    for segment in segments:
        temp_segment.extend(segment)
        if len(temp_segment) >= min_sentences:
            merged_segments.append(temp_segment)
            temp_segment = []

    if temp_segment:
        if merged_segments:
            merged_segments[-1].extend(temp_segment)
        else:
            merged_segments.append(temp_segment)

    return merged_segments

def print_segments(segments):
    for idx, segment in enumerate(segments, start=1):
        print(f"\nSegment {idx}:")
        for sentence in segment:
            print(f" - {sentence}")

def save_segments_to_file(segments, output_file):
    try:
        with open(output_file, 'w', encoding='utf-8') as file:
            for idx, segment in enumerate(segments, start=1):
                file.write(f"Segment {idx}:\n")
                for sentence in segment:
                    file.write(f" - {sentence}\n")
                file.write("\n")
        print(f"Segmented output saved to {output_file}")
    except Exception as e:
        print(f"Error saving file: {e}")

def process_transcript(file_path, threshold=0.5, min_sentences=5, output_file='segmented_output.txt'):
    transcript = read_transcript(file_path)
    if transcript is None:
        return

    sentences = split_into_sentences(transcript)
    print("Sentences extracted:")
    for i, sentence in enumerate(sentences):
        print(f"S{i + 1}: {sentence}")

    cosine_sim = calculate_cosine_similarity(sentences)

    segments = segment_sentences(sentences, cosine_sim, threshold, min_sentences)

    print("\nSegmented Sentences:")
    print_segments(segments)

    save_segments_to_file(segments, output_file)

file_path = '/content/transcript.txt'  # Replace with your file path
output_file = '/content/segmented_output.txt'
process_transcript(file_path, threshold=0.15, min_sentences=5, output_file=output_file)
```