FINAL CODE FOR SEGMENTATION:

```
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 11.6 MB/s eta 0:00:00
Downloading yt_dlp-2025.2.19-py3-none-any.whl (3.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.2/3.2 MB 56.9 MB/s eta 0:00:00
Installing collected packages: yt_dlp
Successfully installed yt_dlp-2025.2.19
```

```
import yt_dlp

def download_audio(video_url):
    ydl_opts = {
    'format': 'bestaudio/best',
    'outtmpl': '/content/audio.mp3',
    'cookiefile': '/content/cookies (3).txt',  # Update this path
    'postprocessors': [{
        'key': 'FFmpegExtractAudio',
        'preferredcodec': 'mp3',
        'preferredquality': '192',
    }],
  }

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

    return "/content/audio.mp3"

video_url = input("Enter YouTube video URL: ")
file_path = download_audio(video_url)
print(f"Downloaded audio: {file_path}")
```

```
Enter YouTube video URL: https://youtu.be/sK8SILOM37I
[youtube] Extracting URL: https://youtu.be/sK8SILOM37I
[youtube] sK8SILOM37I: Downloading webpage
[youtube] sK8SILOM37I: Downloading tv client config
[youtube] sK8SILOM37I: Downloading player 7d1d50a6
[youtube] sK8SILOM37I: Downloading tv player API JSON
[info] sK8SILOM37I: Downloading 1 format(s): 251
[download] Destination: /content/audio.mp3
[download] 100% of   39.37MiB in 00:00:03 at 13.06MiB/s
[ExtractAudio] Destination: /content/audio.mp3.mp3
Deleting original file /content/audio.mp3 (pass -k to keep)
Downloaded audio: /content/audio.mp3
```

```
import os

# Check the files in /content directory
print("Files in content:", os.listdir("/content"))

# Fix the file name if needed
old_path = "/content/audio.mp3.mp3"
new_path = "/content/audio.mp3"

if os.path.exists(old_path):
    os.rename(old_path, new_path)
    print(f"Renamed: {old_path} → {new_path}")
else:
    print("Audio file not found!")
```

```
Files in content: ['.config', 'cookies (3).txt', 'audio.mp3.mp3', 'sample_data']
Renamed: /content/audio.mp3.mp3 → /content/audio.mp3
```

```
import IPython.display as ipd

# Set the correct file path
audio_path = "/content/audio.mp3"

# Play the audio with the correct sample rate (e.g., 44100 Hz)
ipd.Audio(audio_path, rate=44100)
```

```python
import os
import whisper
from pydub import AudioSegment

def convert_audio_to_wav(audio_file):
    """Converts an MP3 audio file to WAV format for better transcription."""
    wav_file = "audio.wav"
    if audio_file.endswith(".mp3"):
        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)}"
    return audio_file  # If already WAV, return as is

def transcribe_audio_with_whisper(audio_path):
    """Transcribes audio using OpenAI's Whisper model."""
    model = whisper.load_model("base")  # Options: "tiny", "base", "small", "medium", "large"

    print("🔹 Transcribing with Whisper... (this may take some time)")

    result = model.transcribe(audio_path)
    return result["text"]

def save_transcript_to_file(transcript, filename="transcript.txt"):
    """Saves the transcript to a text file."""
    with open(filename, "w", encoding="utf-8") as file:
        file.write(transcript)
    print(f"✅ Transcript saved to {filename}")

# Example usage
if __name__ == "__main__":
    audio_file = input("Enter the path to your audio file (MP3/WAV): ")

    if not os.path.exists(audio_file):
        print("❌ Error: File not found.")
    else:
        wav_file = convert_audio_to_wav(audio_file)

        if "Error" in wav_file:
            print(wav_file)
        else:
            transcript = transcribe_audio_with_whisper(wav_file)
            print("\n🔹 Transcript:\n", transcript)

            # Save transcript
            save_transcript_to_file(transcript, "transcript.txt")

            # Cleanup WAV file if it was converted
            if wav_file != audio_file:
                os.remove(wav_file)
```

```
Enter the path to your audio file (MP3/WAV): /content/audio.mp3
100%|████████████████████████████████████████| 139M/139M [00:01<00:00, 115MiB/s]
/usr/local/lib/python3.11/dist-packages/whisper/__init__.py:150: FutureWarning: You are using `torch.load` with `weights_only=False`
  checkpoint = torch.load(fp, map_location=device)
🔹 Transcribing with Whisper... (this may take some time)
/usr/local/lib/python3.11/dist-packages/whisper/transcribe.py:126: UserWarning: FP16 is not supported on CPU; using FP32 instead
  warnings.warn("FP16 is not supported on CPU; using FP32 instead")

🔹 Transcript:
  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
✅ Transcript saved to transcript.txt
```

```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')
nltk.download('punkt_tab')
nltk.download('stopword')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('averaged_perceptron_tagger')

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)
```

```
[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.
[nltk_data] Error loading stopword: Package 'stopword' not found in
[nltk_data]     index
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
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 many people do not get a square m
S3: That was an narrative some 30 years ago.
S4: And not many are literate, people cannot read.
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 UPI the next big revolution, personally I think is in education.
S7: And the complete homework for this has happened in the form of NEP, the documentation of which many of us have read and reali
S8: So, followed by which we got NCRF framework done, which is a national creative framework, surrounding which we will be discus
S9: So, my question is, do you think NCRF plus NEP put together will be the next big revolution after UPI in India?
S10: Absolutely.
S11: And why I think so is because in education the last policy came up many many years ago that was in 1986, which was slightly
S12: And thereafter so many changes have happened in the real world, so many changes have happened in the requirement of the indu
S13: However there were no corresponding changes which happened in education system.
S14: So, therefore I feel that this was the right time when we brought in the education policy 2020, honorable prime minister dec
S15: And we recently celebrated the fourth anniversary of NEP 2020.
S16: NCRF has been brought to implement the intent of 2020, 2020 is a policy, NEP 2020 is a policy.
S17: And for implementing a policy you need a framework.
S18: Now, why we call it a framework?
S19: We call it a framework because this is very flexible.
S20: This allows you all the innovation, the way you educate your students.
S21: Still it provides you the basic guidelines, the framework, the outer layer it provides.
S22: And that layer is mostly the enabling layer.
S23: That is such a enabling layer that it has broken the shackles which were there in the education sector.
S24: 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
S25: So, sir I think let us go with this example of let us say I started off living in a small 2BHK apartment and I slowly develo
S26: And one fine day you came and you changed my kitchen completely.
S27: I was using a bicycle, I moved to a scooter and a car and now we are asking me to fly and giving me an aircraft.
S28: NEP sounds more like that for me.
S29: How do I do the transition?
S30: I fear that I will crash if I use a aeroplane without training.
S31: I am talking about all the teachers in the country, all the schools in the country.
S32: We have been driving buses at max.
S33: Now we should fly how do we do this?
S34: Okay.
S35: Look at the requirement of the industry.
S36: The requirement of the industry has been moving very fast.
S37: The technology is emerging every day and the industry is moving with that speed.
S38: 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
S39: When industry is moving that fast, when the requirement is moving that fast, don't you think it is important for us to chang
S40: Yes, how long can we wait?
S41: Yes, it will take a lot of effort for every one of us to adapt to this change.
S42: But this change is going to be not only beneficial but also very facilitated for all of us, very liberating for all of us.
S43: This is going to be highly liberative and choice based system.
S44: There are number of choices which are available to you, which are available to every student.
S45: Yes, when we introduce a new system, we have to really create new things, create new ways of doing things, learn something n
```

Keyword Generation

```
!pip install gensim spacy nltk
```

```
Collecting gensim
  Downloading gensim-4.3.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (8.1 kB)
Requirement already satisfied: spacy in /usr/local/lib/python3.11/dist-packages (3.7.5)
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages (3.9.1)
Requirement already satisfied: numpy<2.0,>=1.18.5 in /usr/local/lib/python3.11/dist-packages (from gensim) (1.26.4)
Collecting scipy<1.14.0,>=1.7.0 (from gensim)
  Downloading scipy-1.13.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (60 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.6/60.6 kB 3.6 MB/s eta 0:00:00
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.11/dist-packages (from gensim) (7.1.0)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.0.12)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.11)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.0.9)
Requirement already satisfied: thinc<8.3.0,>=8.2.2 in /usr/local/lib/python3.11/dist-packages (from spacy) (8.2.5)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.1.3)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.5.1)
```

```
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.10)
Requirement already satisfied: weasel<0.5.0,>=0.1.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (0.4.1)
Requirement already satisfied: typer<1.0.0,>=0.3.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (0.15.2)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (4.67.1)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.32.3)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.16
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.1.6)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from spacy) (75.1.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (24.2)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.5.0)
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: language-data>=1.2 in /usr/local/lib/python3.11/dist-packages (from langcodes<4.0.0,>=3.2.0->spacy
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.6
Requirement already satisfied: pydantic-core==2.27.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (3.1
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open>=1.8.1->gensim) (1.17.2)
Requirement already satisfied: blis<0.8.0,>=0.7.8 in /usr/local/lib/python3.11/dist-packages (from thinc<8.3.0,>=8.2.2->spacy) (6
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.11/dist-packages (from thinc<8.3.0,>=8.2.2->spa
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->spacy) (1
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->spacy) (13.9.4
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from weasel<0.5.0,>=0.1.0->
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->spacy) (3.0.2)
Requirement already satisfied: marisa-trie>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from language-data>=1.2->langcodes<
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0.0,
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->
Downloading gensim-4.3.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (26.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 26.7/26.7 MB 38.2 MB/s eta 0:00:00
Downloading scipy-1.13.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (38.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 38.6/38.6 MB 20.5 MB/s eta 0:00:00
Installing collected packages: scipy, gensim
  Attempting uninstall: scipy
    Found existing installation: scipy 1.14.1
    Uninstalling scipy-1.14.1:
```

Using LDA

```
import re
import gensim
import spacy
import nltk
from gensim import corpora
from gensim.models import LdaModel
from nltk.corpus import stopwords

# Download stopwords if not available
nltk.download("stopwords")
stop_words = set(stopwords.words("english"))

# Load spaCy NLP model
nlp = spacy.load("en_core_web_sm")

# Function to preprocess text
def preprocess_text(text):
    text = re.sub(r'\s+', ' ', text)  # Remove extra spaces
    doc = nlp(text.lower())  # Convert to lowercase and tokenize
    tokens = [token.lemma_ for token in doc if token.is_alpha and token.lemma_ not in stop_words]
    return tokens

# Read the transcript from file
input_file = "/content/segmented_output.txt"  # Change this to your actual file name
with open(input_file, "r", encoding="utf-8") as file:
    transcript = file.read()

# Preprocess each segment in the transcript
processed_texts = [preprocess_text(segment) for segment in transcript.split("\n") if segment.strip()]

# Create Dictionary and Corpus for LDA
dictionary = corpora.Dictionary(processed_texts)
corpus = [dictionary.doc2bow(text) for text in processed_texts]

# Train LDA Model
num_topics = 3  # Adjust based on requirement
lda_model = LdaModel(corpus, num_topics=num_topics, id2word=dictionary, passes=10)

# Display Extracted Topics
print("\nExtracted Topics:")
```

```
for idx, topic in lda_model.print_topics():
    print(f"Topic {idx + 1}: {topic}")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

Extracted Topics:
Topic 1: 0.066*"I" + 0.039*"segment" + 0.021*"say" + 0.017*"student" + 0.015*"think" + 0.012*"number" + 0.011*"see" + 0.011*"go" + 0
Topic 2: 0.030*"I" + 0.018*"ncrf" + 0.012*"framework" + 0.012*"learn" + 0.012*"absolutely" + 0.011*"way" + 0.009*"outcome" + 0.008*
Topic 3: 0.021*"student" + 0.019*"skill" + 0.016*"go" + 0.014*"year" + 0.014*"time" + 0.012*"teach" + 0.011*"course" + 0.010*"teache
```

```
import re
import gensim
import spacy
import nltk
from gensim import corpora
from gensim.models import LdaModel
from nltk.corpus import stopwords

# Download stopwords if not available
nltk.download("stopwords")
stop_words = set(stopwords.words("english"))

# Load spaCy NLP model
nlp = spacy.load("en_core_web_sm")

# Function to preprocess text
def preprocess_text(text):
    text = re.sub(r'\s+', ' ', text)  # Remove extra spaces
    doc = nlp(text.lower())  # Convert to lowercase and tokenize
    tokens = [token.lemma_ for token in doc if token.is_alpha and token.lemma_ not in stop_words]
    return tokens

# Read the transcript from file
input_file = "/content/segmented_output.txt"  # Change this to your actual file name
with open(input_file, "r", encoding="utf-8") as file:
    transcript = file.readlines()

# Preprocess each segment
processed_texts = [preprocess_text(segment) for segment in transcript if segment.strip()]

# Create Dictionary and Corpus for LDA
dictionary = corpora.Dictionary(processed_texts)
corpus = [dictionary.doc2bow(text) for text in processed_texts]

# Train LDA Model
num_topics = 5  # Adjust based on requirement
lda_model = LdaModel(corpus, num_topics=num_topics, id2word=dictionary, passes=10)

# Extract the dominant topic for each segment and save to file
output_file = "output_topics.txt"
with open(output_file, "w", encoding="utf-8") as out_file:
    for i, bow in enumerate(corpus):
        topic_distribution = lda_model.get_document_topics(bow)
        dominant_topic = max(topic_distribution, key=lambda x: x[1])[0]  # Get the most probable topic
        topic_keywords = lda_model.print_topic(dominant_topic)

        out_file.write(f"Segment {i + 1}:\n")
        out_file.write(f"Topic: {topic_keywords}\n\n")

print(f"Extracted topics saved to {output_file}")

```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Extracted topics saved to output_topics.txt
```

```
import re
import gensim
import spacy
import nltk
from gensim import corpora
from gensim.models import LdaModel
from nltk.corpus import stopwords

# Download stopwords if not available
nltk.download("stopwords")
stop_words = set(stopwords.words("english"))

# Load spaCy NLP model
nlp = spacy.load("en_core_web_sm")
```

```python

# Function to preprocess text
def preprocess_text(text):
    """Cleans and tokenizes text for topic modeling"""
    text = re.sub(r'\s+', ' ', text)  # Remove extra spaces
    doc = nlp(text.lower())  # Convert to lowercase and tokenize
    tokens = [token.lemma_ for token in doc if token.is_alpha and token.lemma_ not in stop_words]
    return tokens

# Read the transcript file
input_file = "/content/segmented_output (2).txt"  # Change this to your actual file name
with open(input_file, "r", encoding="utf-8") as file:
    transcript = file.read()

# Split transcript into segments using a pattern like 'Segment X:' or other logical markers
segments = re.split(r'Segment \d+:', transcript)[1:]  # Removes empty first split

# Preprocess each full segment
processed_texts = [preprocess_text(segment.strip()) for segment in segments if segment.strip()]

# Create Dictionary and Corpus for LDA
dictionary = corpora.Dictionary(processed_texts)
corpus = [dictionary.doc2bow(text) for text in processed_texts]

# Train LDA Model
num_topics = min(len(processed_texts), 5)  # Set topic count based on available data
lda_model = LdaModel(corpus, num_topics=num_topics, id2word=dictionary, passes=10)

# Extract the top keyword for each full segment
output_file = "top_keywords.txt"
with open(output_file, "w", encoding="utf-8") as out_file:
    for i, bow in enumerate(corpus):
        topic_distribution = lda_model.get_document_topics(bow)
        dominant_topic = max(topic_distribution, key=lambda x: x[1])[0]  # Get the most probable topic
        topic_keywords = lda_model.print_topic(dominant_topic)

        # Extract the highest probability keyword
        keywords = topic_keywords.split(" + ")
        top_keyword = max(keywords, key=lambda x: float(x.split("*")[0]))  # Get highest weight word
        top_word = top_keyword.split("*")[1].replace("'", "").strip()  # Extract the actual word

        out_file.write(f"Segment {i + 1}: {top_word}\n")

print(f"Top keywords saved to {output_file}")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Top keywords saved to top_keywords.txt
```

```python
import re
import gensim
import spacy
import nltk
from gensim import corpora
from gensim.models import LdaModel
from nltk.corpus import stopwords

# Download stopwords if not available
nltk.download("stopwords")
stop_words = set(stopwords.words("english"))

# Load spaCy NLP model
nlp = spacy.load("en_core_web_sm")

# Function to preprocess text
def preprocess_text(text):
    """Cleans and tokenizes text for topic modeling"""
    text = re.sub(r'\s+', ' ', text)  # Remove extra spaces
    doc = nlp(text.lower())  # Convert to lowercase and tokenize
    tokens = [token.lemma_ for token in doc if token.is_alpha and token.lemma_ not in stop_words]
    return tokens

# Read the transcript file
input_file = "/content/segmented_output (2).txt"  # Change this to your actual file name
with open(input_file, "r", encoding="utf-8") as file:
    transcript = file.read()

# Split transcript into segments using 'Segment X:' markers
segments = re.split(r'Segment \d+:', transcript)[1:]  # Removes empty first split

# Preprocess each full segment
processed_texts = [preprocess_text(segment.strip()) for segment in segments if segment.strip()]
```

```

# Create Dictionary and Corpus for LDA
dictionary = corpora.Dictionary(processed_texts)
corpus = [dictionary.doc2bow(text) for text in processed_texts]

# Train LDA Model
num_topics = min(len(processed_texts), 5)  # Set topic count based on available data
lda_model = LdaModel(corpus, num_topics=num_topics, id2word=dictionary, passes=10)

# Extract the top 3 keywords for each full segment
output_file = "top_keywords2.txt"
with open(output_file, "w", encoding="utf-8") as out_file:
    for i, bow in enumerate(corpus):
        topic_distribution = lda_model.get_document_topics(bow)
        dominant_topic = max(topic_distribution, key=lambda x: x[1])[0]  # Get the most probable topic
        topic_keywords = lda_model.print_topic(dominant_topic)

        # Extract top 3 highest probability keywords
        keywords = topic_keywords.split(" + ")
        top_keywords = sorted(keywords, key=lambda x: float(x.split("*")[0]), reverse=True)[:3]
        top_words = [word.split("*")[1].replace("'", "").strip() for word in top_keywords]

        out_file.write(f"Segment {i + 1}: {', '.join(top_words)}\n")

print(f"Top keywords saved to {output_file}")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Top keywords saved to top_keywords2.txt
```

Using KeyBERT

```
pip install keybert
```

```
Collecting keybert
  Downloading keybert-0.9.0-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.11/dist-packages (from keybert) (1.26.4)
Requirement already satisfied: rich>=10.4.0 in /usr/local/lib/python3.11/dist-packages (from keybert) (13.9.4)
Requirement already satisfied: scikit-learn>=0.22.2 in /usr/local/lib/python3.11/dist-packages (from keybert) (1.6.1)
Requirement already satisfied: sentence-transformers>=0.3.8 in /usr/local/lib/python3.11/dist-packages (from keybert) (3.4.1)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.4.0->keybert) (3.0.0
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.4.0->keybert) (2.18
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.22.2->keybert) (1.13.1
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.22.2->keybert) (1.4.2
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.22.2->keybert)
Requirement already satisfied: transformers<5.0.0,>=4.41.0 in /usr/local/lib/python3.11/dist-packages (from sentence-transformers>=0
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from sentence-transformers>=0.3.8->keybert) (4.67.1
Requirement already satisfied: torch>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from sentence-transformers>=0.3.8->keybert
Requirement already satisfied: huggingface-hub>=0.20.0 in /usr/local/lib/python3.11/dist-packages (from sentence-transformers>=0.3.8
Requirement already satisfied: Pillow in /usr/local/lib/python3.11/dist-packages (from sentence-transformers>=0.3.8->keybert) (11.1
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sentence-transform
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sentence-t
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sentence-tr
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sentence-transf
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sentence-transform
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.4.0->keyb
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers>=0.3.8
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers>=0.3.8->
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sent
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->se
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sent
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->senter
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->senter
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sent
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-tra
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-t
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sente
Requirement already satisfied: triton==3.1.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers>=
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers>=
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch>=1.11.0->ser
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->sente
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->sent
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch>=1.11.0->sentence-tran
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.20.0->sent
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.20.0
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.20.0
Downloading keybert-0.9.0-py3-none-any.whl (41 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 41.4/41.4 kB 1.9 MB/s eta 0:00:00
Installing collected packages: keybert
Successfully installed keybert-0.9.0
```

```python
import re
import spacy
import nltk
from keybert import KeyBERT
from nltk.corpus import stopwords

# Download stopwords if not available
nltk.download("stopwords")
stop_words = set(stopwords.words("english"))

# Load spaCy NLP model
nlp = spacy.load("en_core_web_sm")

# Initialize KeyBERT model
kw_model = KeyBERT()

# Function to preprocess text
def preprocess_text(text):
    """Cleans text and removes unnecessary characters."""
    text = re.sub(r'\s+', ' ', text)  # Remove extra spaces
    doc = nlp(text.lower())  # Convert to lowercase and tokenize
    tokens = [token.lemma_ for token in doc if token.is_alpha and token.lemma_ not in stop_words]
    return " ".join(tokens)  # Return as cleaned text

# Read the transcript file
input_file = "/content/segmented_output (2).txt"  # Change this to your actual file name
with open(input_file, "r", encoding="utf-8") as file:
    transcript = file.read()

# Split transcript into segments using 'Segment X:' markers
segments = re.split(r'Segment \d+:', transcript)[1:]  # Removes empty first split

# Preprocess each segment
cleaned_segments = [preprocess_text(segment.strip()) for segment in segments if segment.strip()]

# Extract top 3 keywords using KeyBERT
output_file = "top_keywords.txt"
with open(output_file, "w", encoding="utf-8") as out_file:
    for i, segment in enumerate(cleaned_segments):
        keywords = kw_model.extract_keywords(segment, keyphrase_ngram_range=(1, 1), top_n=3)

        # Get top 3 keywords
        top_words = [word[0] for word in keywords]

        out_file.write(f"Segment {i + 1}: {', '.join(top_words)}\n")

print(f"Top keywords saved to {output_file}")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
/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 s
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(
modules.json: 100%  349/349 [00:00<00:00, 29.3kB/s]
config_sentence_transformers.json: 100%  116/116 [00:00<00:00, 7.47kB/s]
README.md: 100%  10.5k/10.5k [00:00<00:00, 726kB/s]
sentence_bert_config.json: 100%  53.0/53.0 [00:00<00:00, 3.12kB/s]
config.json: 100%  612/612 [00:00<00:00, 37.6kB/s]
model.safetensors: 100%  90.9M/90.9M [00:01<00:00, 111MB/s]
tokenizer_config.json: 100%  350/350 [00:00<00:00, 20.0kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 4.20MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 19.9MB/s]
special_tokens_map.json: 100%  112/112 [00:00<00:00, 10.3kB/s]
config.json: 100%  190/190 [00:00<00:00, 8.24kB/s]
Top keywords saved to top_keywords.txt
```

Linkedin Transcript

```
import yt_dlp

def download_audio(video_url):
    ydl_opts = {
        'format': 'bestaudio/best',
        'outtmpl': 'linkedin_audio.mp3',  # Save file with a meaningful name
        'cookiefile': '/content/cookies (4).txt',  # Ensure you have LinkedIn authentication
        'postprocessors': [{
            'key': 'FFmpegExtractAudio',
            'preferredcodec': 'mp3',
            'preferredquality': '192',
        }],
    }

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

    return "linkedin_audio.mp3"

video_url = input("Enter LinkedIn video URL: ")
file_path = download_audio(video_url)
print(f"Downloaded audio: {file_path}")

```

```
Enter LinkedIn video URL: https://www.linkedin.com/posts/guillermoflor_elon-musk-wants-the-workplace-to-be-vibrant-activity-7208090
[LinkedIn] Extracting URL: https://www.linkedin.com/posts/guillermoflor_elon-musk-wants-the-workplace-to-be-vibrant-activity...nuG8
[LinkedIn] 7208090955906707457: Downloading webpage
ERROR: [LinkedIn] 7208090955906707457: Unable to extract video; please report this issue on  https://github.com/yt-dlp/yt-dlp/issue
---------------------------------------------------------------------------
RegexNotFoundError                        Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/yt_dlp/YoutubeDL.py in wrapper(self, *args, **kwargs)
   1636                 try:
-> 1637                     return func(self, *args, **kwargs)
   1638                 except (CookieLoadError, DownloadCancelled, LazyList.IndexError, PagedList.IndexError):

                              11 frames
RegexNotFoundError: [LinkedIn] 7208090955906707457: Unable to extract video; please report this issue on  https://github.com/yt-
dlp/yt-dlp/issues?q= , filling out the appropriate issue template. Confirm you are on the latest version using  yt-dlp -U

During handling of the above exception, another exception occurred:

DownloadError                             Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/yt_dlp/YoutubeDL.py in trouble(self, message, tb, is_error)
   1032             else:
   1033                 exc_info = sys.exc_info()
-> 1034             raise DownloadError(message, exc_info)
   1035         self._download_retcode = 1
   1036

DownloadError: ERROR: [LinkedIn] 7208090955906707457: Unable to extract video; please report this issue on  https://github.com/yt-
dlp/yt-dlp/issues?q= , filling out the appropriate issue template. Confirm you are on the latest version using  yt-dlp -U
```

```
!pip install -U yt-dlp

```

```
Requirement already satisfied: yt-dlp in /usr/local/lib/python3.11/dist-packages (2025.2.19)
```

```
!yt-dlp --cookies-from-browser chrome "https://www.linkedin.com/posts/guillermoflor_elon-musk-wants-the-workplace-to-be-vibrant-acti
```

```
[LinkedIn] Extracting URL: https://www.linkedin.com/posts/guillermoflor_elon-musk-wants-the-workplace-to-be-vibrant-activity...nuG8
[LinkedIn] 7208090955906707457: Downloading webpage
Extracting cookies from chrome
ERROR: could not find chrome cookies database in "/root/.config/google-chrome"
```

```
yt-dlp --cookies-from-browser chrome "https://www.linkedin.com/posts/guillermoflor_elon-musk-wants-the-workplace-to-be-vibrant-acti
```

```
  File "<ipython-input-26-af5310eec6a0>", line 1
    yt-dlp --cookies-from-browser chrome "https://www.linkedin.com/posts/guillermoflor_elon-musk-wants-the-workplace-to-be-vibrant-
activity-7208090955906707457-YQ__?utm_source=share&utm_medium=member_desktop&rcm=ACoAAC472oABr5i8229nuG8UuSMZV1GiCnsrmuE"
                    ^
SyntaxError: invalid syntax
```

TRANSCRIPT SEGMENTATION

```python
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Ensure necessary resources are downloaded
nltk.download('punkt')
nltk.download('stopwords')

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

def calculate_similarity(segment_words, new_sentence_words):
    vectorizer = TfidfVectorizer()
    combined_text = [" ".join(segment_words), " ".join(new_sentence_words)]
    tfidf_matrix = vectorizer.fit_transform(combined_text)
    return cosine_similarity(tfidf_matrix[0], tfidf_matrix[1])[0][0]

def segment_transcript(file_path, output_path, threshold=0.5):
    with open(file_path, 'r', encoding='utf-8') as file:
        lines = file.readlines()

    lines = [line.strip() for line in lines if line.strip()]
    segments = []
    i = 0

    while i < len(lines):
        segment = lines[i:i+5]
        segment_words = preprocess_text(" ".join(segment))

        new_segment = segment[:]
        i += 5

        while i < len(lines):
            new_sentence = lines[i]
            new_sentence_words = preprocess_text(new_sentence)

            similarity = calculate_similarity(segment_words, new_sentence_words)

            if similarity >= threshold:
                new_segment.append(new_sentence)
                segment_words.extend(new_sentence_words)
                i += 1
            else:
                break

        segments.append(" ".join(new_segment))

        # Start the next segment from the current sentence (not skipping lines)

    with open(output_path, 'w', encoding='utf-8') as output_file:
        for idx, segment in enumerate(segments):
            output_file.write(f"Segment {idx + 1}:\n{segment}\n\n")

    return segments

# Example usage
file_path = "/content/transcript (15).txt"  # Path to the input .txt file
output_path = "segmented_transcript.txt"  # Path to the output .txt file
segments = segment_transcript(file_path, output_path)
for idx, segment in enumerate(segments):
    print(f"Segment {idx + 1} saved to file.")
```

```
Segment 1 saved to 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!
```

```python
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Ensure necessary resources are downloaded
nltk.download('punkt')
nltk.download('stopwords')

```

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

def calculate_similarity(reference_text, new_sentence):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform([" ".join(reference_text), new_sentence])
    return cosine_similarity(tfidf_matrix[0], tfidf_matrix[1])[0][0]

def segment_transcript(file_path, output_path, threshold=1.0):
    with open(file_path, 'r', encoding='utf-8') as file:
        lines = [line.strip() for line in file.readlines() if line.strip()]

    segments = []
    i = 0

    while i < len(lines):
        segment = lines[i:i+5]  # First 5 lines as an initial segment
        segment_words = preprocess_text(" ".join(segment))

        new_segment = segment[:]
        i += 5  # Move forward

        while i < len(lines):
            new_sentence = lines[i]
            new_sentence_words = preprocess_text(new_sentence)

            similarity = calculate_similarity(segment_words, " ".join(new_sentence_words))

            if similarity >= threshold:
                new_segment.append(new_sentence)
                i += 1
            else:
                break  # Start a new segment

        segments.append(" ".join(new_segment))

    with open(output_path, 'w', encoding='utf-8') as output_file:
        for idx, segment in enumerate(segments):
            output_file.write(f"Segment {idx + 1}:\n{segment}\n\n")

    return segments

# Example usage
file_path = "/content/transcript (15).txt"
output_path = "segmented_transcript.txt"
segments = segment_transcript(file_path, output_path, threshold=0.6)

for idx, segment in enumerate(segments):
    print(f"Segment {idx + 1} saved to file.")
```

```
Segment 1 saved to 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!
```

```
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Download stopwords if not already downloaded
nltk.download('stopwords')
nltk.download('punkt')

# Function to clean text by removing stopwords
def clean_text(text):
    stop_words = set(stopwords.words('english'))
    words = nltk.word_tokenize(text.lower())  # Tokenize and convert to lowercase
    return ' '.join([word for word in words if word.isalnum() and word not in stop_words])

# Read transcript
def read_transcript(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        return file.readlines()

# Function to group sentences into segments and save them
def segment_transcript(sentences, output_file):
    segments = []
    current_segment = sentences[:5]  # First 5 lines as the initial segment
```

```
    cleaned_segment_words = clean_text(' '.join(current_segment))  # Remove stopwords

    for i in range(5, len(sentences)):
        sentence = sentences[i]
        cleaned_sentence = clean_text(sentence)

        if not cleaned_sentence:  # Skip empty sentences after stopword removal
            continue

        # Calculate similarity
        vectorizer = TfidfVectorizer().fit_transform([cleaned_segment_words, cleaned_sentence])
        similarity = cosine_similarity(vectorizer)[0, 1]

        if similarity > 0.5:  # Threshold to decide if it belongs in the same segment
            current_segment.append(sentence)
        else:
            segments.append(current_segment)  # Save the current segment
            current_segment = [sentence] + sentences[i+1:i+5]  # New segment with next 4 lines
            cleaned_segment_words = clean_text(' '.join(current_segment))  # Recalculate cleaned words

    segments.append(current_segment)  # Append the last segment

    # Save segmented transcript to a new file
    with open(output_file, 'w', encoding='utf-8') as file:
        for idx, segment in enumerate(segments):
            file.write(f"Segment {idx + 1}:\n")
            file.write("".join(segment) + "\n")
            file.write("-" * 50 + "\n")

    return segments

# Example usage
if __name__ == "__main__":
    transcript_lines = read_transcript("/content/transcript (15).txt")
    segmented_text = segment_transcript(transcript_lines, "segmented_transcript.txt")

    for idx, segment in enumerate(segmented_text):
        print(f"Segment {idx + 1}:")
        print("".join(segment))
        print("-" * 50)
```

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

```
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Download stopwords if not already downloaded
nltk.download('stopwords')
nltk.download('punkt')

# Function to clean text by removing stopwords
def clean_text(text):
    stop_words = set(stopwords.words('english'))
    words = nltk.word_tokenize(text.lower())  # Tokenize and convert to lowercase
    return ' '.join([word for word in words if word.isalnum() and word not in stop_words])

# Read transcript
def read_transcript(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        return file.readlines()

# Function to group sentences into segments and save them
def segment_transcript(sentences, output_file):
    segments = []
    i = 0  # Pointer to track position in sentences

    while i < len(sentences):
        current_segment = sentences[i:i+5]  # First 5 lines as the initial segment
        cleaned_segment_words = clean_text(' '.join(current_segment))  # Remove stopwords
        i += 5  # Move the pointer forward

        while i < len(sentences):
            cleaned_sentence = clean_text(sentences[i])
```

```

            if not cleaned_sentence:  # Skip empty sentences after stopword removal
                i += 1
                continue

            # Calculate similarity
            vectorizer = TfidfVectorizer().fit_transform([cleaned_segment_words, cleaned_sentence])
            similarity = cosine_similarity(vectorizer)[0, 1]

            if similarity > 0.5:  # Threshold to decide if it belongs in the same segment
                current_segment.append(sentences[i])
                i += 1  # Move to the next sentence
            else:
                break  # Stop adding to this segment, start a new one

        segments.append(current_segment)

    # Save segmented transcript to a new file
    with open(output_file, 'w', encoding='utf-8') as file:
        for idx, segment in enumerate(segments):
            file.write(f"Segment {idx + 1}:\n")
            file.write("".join(segment) + "\n")
            file.write("-" * 50 + "\n")

    return segments

# Example usage
if __name__ == "__main__":
    transcript_lines = read_transcript("/content/transcript (15).txt")
    segmented_text = segment_transcript(transcript_lines, "segmented_transcript.txt")

    for idx, segment in enumerate(segmented_text):
        print(f"Segment {idx + 1}:")
        print("".join(segment))
        print("-" * 50)
```

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

```
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Download stopwords if not already downloaded
nltk.download('stopwords')
nltk.download('punkt')

# Function to clean text by removing stopwords
def clean_text(text):
    stop_words = set(stopwords.words('english'))
    words = nltk.word_tokenize(text.lower())  # Tokenize and convert to lowercase
    return ' '.join([word for word in words if word.isalnum() and word not in stop_words])

# Read transcript
def read_transcript(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        return file.readlines()

# Function to group sentences into segments and save them
def segment_transcript(sentences, output_file):
    segments = []
    i = 0  # Pointer to track position in sentences

    while i < len(sentences):
        current_segment = sentences[i:i+5]  # First 5 lines as the initial segment
        cleaned_segment_words = clean_text(' '.join(current_segment))  # Remove stopwords
        i += 5  # Move the pointer forward

        while i < len(sentences):
            cleaned_sentence = clean_text(sentences[i])

            if not cleaned_sentence:  # Skip empty sentences after stopword removal
                i += 1
                continue

```

```
            # Calculate similarity
            vectorizer = TfidfVectorizer().fit_transform([cleaned_segment_words, cleaned_sentence])
            similarity = cosine_similarity(vectorizer)[0, 1]

            if similarity > 0.2:  # Threshold to decide if it belongs in the same segment
                current_segment.append(sentences[i])
                i += 1  # Move to the next sentence
            else:
                break  # Stop adding to this segment, start a new one

        segments.append(current_segment)
        i += 5  # Move the pointer forward to ensure the next segment starts correctly

    # Save segmented transcript to a new file
    with open(output_file, 'w', encoding='utf-8') as file:
        for idx, segment in enumerate(segments):
            file.write(f"Segment {idx + 1}:\n")
            file.write("".join(segment) + "\n")
            file.write("-" * 50 + "\n")

    return segments

# Example usage
if __name__ == "__main__":
    transcript_lines = read_transcript("/content/transcript (15).txt")
    segmented_text = segment_transcript(transcript_lines, "segmented_transcript.txt")

    for idx, segment in enumerate(segmented_text):
        print(f"Segment {idx + 1}:")
        print("".join(segment))
        print("-" * 50)

```

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

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

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

# Function to clean text by removing stopwords
def clean_text(text):
    stop_words = set(stopwords.words('english'))
    words = nltk.word_tokenize(text.lower())  # Tokenize and convert to lowercase
    return ' '.join([word for word in words if word.isalnum() and word not in stop_words])

# Read transcript file
def read_transcript(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        return sent_tokenize(file.read())  # Use sentence tokenizer instead of readlines

# Function to segment transcript
def segment_transcript(sentences, output_file):
    segments = []
    i = 0  # Pointer to track position

    while i < len(sentences):
        current_segment = [sentences[i]]  # Start with first sentence
        cleaned_segment = clean_text(sentences[i])
        i += 1  # Move forward

        while i < len(sentences):
            cleaned_sentence = clean_text(sentences[i])

            if not cleaned_sentence:  # Skip empty cleaned sentences
                i += 1
                continue

            # Vectorize segment and new sentence
            vectorizer = TfidfVectorizer()
```

```
            vectors = vectorizer.fit_transform([cleaned_segment, cleaned_sentence])
            similarity = cosine_similarity(vectors)[0, 1]

            if similarity > 0.9:  # Threshold for grouping
                current_segment.append(sentences[i])
                cleaned_segment += " " + cleaned_sentence  # Update segment content
                i += 1  # Move to next sentence
            else:
                break  # Stop adding, create a new segment

        segments.append(current_segment)

    # Save segmented transcript
    with open(output_file, 'w', encoding='utf-8') as file:
        for idx, segment in enumerate(segments):
            file.write(f"Segment {idx + 1}:\n")
            file.write(" ".join(segment) + "\n")
            file.write("-" * 50 + "\n")

    return segments

# Example usage
if __name__ == "__main__":
    transcript_sentences = read_transcript("/content/transcript (15).txt")
    segmented_text = segment_transcript(transcript_sentences, "segmented_transcript.txt")

    for idx, segment in enumerate(segmented_text):
        print(f"Segment {idx + 1}:")
        print(" ".join(segment))
        print("-" * 50)

```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Segment 1:
 So, sir, we know that India has seen a huge revolution with digital payments.
--------------------------------------------------
Segment 2:
We all thought that India is a place, at least the West thought that India is a place where many people do not get a square mill,
--------------------------------------------------
Segment 3:
That was an narrative some 30 years ago.
--------------------------------------------------
Segment 4:
And not many are literate, people cannot read.
--------------------------------------------------
Segment 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 50,
--------------------------------------------------
Segment 6:
I think immediately after UPI the next big revolution, personally I think is in education.
--------------------------------------------------
Segment 7:
And the complete homework for this has happened in the form of NEP, the documentation of which many of us have read and realized
--------------------------------------------------
Segment 8:
So, followed by which we got NCRF framework done, which is a national creative framework, surrounding which we will be discussing
--------------------------------------------------
Segment 9:
So, my question is, do you think NCRF plus NEP put together will be the next big revolution after UPI in India?
--------------------------------------------------
Segment 10:
Absolutely.
--------------------------------------------------
Segment 11:
And why I think so is because in education the last policy came up many many years ago that was in 1986, which was slightly tweak
--------------------------------------------------
Segment 12:
And thereafter so many changes have happened in the real world, so many changes have happened in the requirement of the industry,
--------------------------------------------------
Segment 13:
However there were no corresponding changes which happened in education system.
--------------------------------------------------
Segment 14:
So, therefore I feel that this was the right time when we brought in the education policy 2020, honorable prime minister declared
--------------------------------------------------
Segment 15:
And we recently celebrated the fourth anniversary of NEP 2020.
--------------------------------------------------
Segment 16:
NCRF has been brought to implement the intent of 2020, 2020 is a policy, NEP 2020 is a policy.
--------------------------------------------------
Segment 17:
And for implementing a policy you need a framework.
--------------------------------------------------
```

```
Segment 18:
```

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

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

# Read transcript file
def read_transcript(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        return sent_tokenize(file.read())  # Tokenize entire text into sentences

# Function to segment transcript using similarity
def segment_transcript(sentences, output_file, threshold=0.9):
    segments = []
    i = 0  # Pointer to track position

    # Fit a single TF-IDF vectorizer on all sentences for consistency
    vectorizer = TfidfVectorizer(stop_words="english")
    sentence_vectors = vectorizer.fit_transform(sentences)  # Vectorize entire text

    while i < len(sentences):
        current_segment = [sentences[i]]  # Start with the first sentence
        segment_indices = [i]
        i += 1  # Move pointer forward

        while i < len(sentences):
            # Compute cosine similarity between last sentence of segment and next sentence
            similarity = cosine_similarity(sentence_vectors[segment_indices[-1]], sentence_vectors[i])[0, 0]

            if similarity >= threshold:  # If similar, add to the segment
                current_segment.append(sentences[i])
                segment_indices.append(i)  # Track index of included sentence
                i += 1
            else:
                break  # Stop and start a new segment

        segments.append(current_segment)  # Store the formed segment

    # Save segmented transcript
    with open(output_file, 'w', encoding='utf-8') as file:
        for idx, segment in enumerate(segments):
            file.write(f"Segment {idx + 1}:\n")
            file.write(" ".join(segment) + "\n")
            file.write("-" * 50 + "\n")

    return segments

# Example usage
if __name__ == "__main__":
    transcript_sentences = read_transcript("/content/transcript (15).txt")
    segmented_text = segment_transcript(transcript_sentences, "segmented_transcript.txt")

    for idx, segment in enumerate(segmented_text):
        print(f"Segment {idx + 1}:")
        print(" ".join(segment))
        print("-" * 50)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Segment 1:
 So, sir, we know that India has seen a huge revolution with digital payments.
--------------------------------------------------
Segment 2:
We all thought that India is a place, at least the West thought that India is a place where many people do not get a square mill,
--------------------------------------------------
Segment 3:
That was an narrative some 30 years ago.
--------------------------------------------------
Segment 4:
And not many are literate, people cannot read.
--------------------------------------------------
Segment 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 50,
--------------------------------------------------
Segment 6:
I think immediately after UPI the next big revolution, personally I think is in education.
--------------------------------------------------
Segment 7:
And the complete homework for this has happened in the form of NEP, the documentation of which many of us have read and realized
```

```
--------------------------------------------------
Segment 8:
So, followed by which we got NCRF framework done, which is a national creative framework, surrounding which we will be discussing
--------------------------------------------------
Segment 9:
So, my question is, do you think NCRF plus NEP put together will be the next big revolution after UPI in India?
--------------------------------------------------
Segment 10:
Absolutely.
--------------------------------------------------
Segment 11:
And why I think so is because in education the last policy came up many many years ago that was in 1986, which was slightly tweak
--------------------------------------------------
Segment 12:
And thereafter so many changes have happened in the real world, so many changes have happened in the requirement of the industry,
--------------------------------------------------
Segment 13:
However there were no corresponding changes which happened in education system.
--------------------------------------------------
Segment 14:
So, therefore I feel that this was the right time when we brought in the education policy 2020, honorable prime minister declared
--------------------------------------------------
Segment 15:
And we recently celebrated the fourth anniversary of NEP 2020.
--------------------------------------------------
Segment 16:
NCRF has been brought to implement the intent of 2020, 2020 is a policy, NEP 2020 is a policy.
--------------------------------------------------
Segment 17:
And for implementing a policy you need a framework.
--------------------------------------------------
Segment 18:
Now, why we call it a framework?
--------------------------------------------------
Segment 19:
```

```
# prompt: 1. Read the transcript
# 2. Separate each line
# 3. By default we have the first 5 lines in segment 1
# 4. Then we will remove the stop words from that segment and put the remaining words in a new list
# 5. Now we will start with the comparison with nest lines one by one
# 6. Remove the stop words from the sentences below them and compare with those of the list from segment 1
# 7. If the stop words from the sentence 6 are similar to those of segment 1 (Finding the cosine similarity) Put them in segment 1 e
# 8. The next segment will consists of this sentence 6 and the next 4 chronological sentences should also be kept in segment 2.
# 9. Now we will repeat the steps 4-8 for the rest of the transcript.

import nltk
from nltk.tokenize import sent_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from nltk.corpus import stopwords

# Download required data
nltk.download('punkt')
nltk.download('stopwords')

# Function to clean text by removing stopwords
def clean_text(text):
    stop_words = set(stopwords.words('english'))
    words = nltk.word_tokenize(text.lower())
    return ' '.join([word for word in words if word.isalnum() and word not in stop_words])

# Read transcript file
def read_transcript(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        return sent_tokenize(file.read())

# Function to segment transcript using similarity
def segment_transcript(sentences, output_file, threshold=1):
    segments = []
    i = 0

    # Fit a single TF-IDF vectorizer on all sentences for consistency
    vectorizer = TfidfVectorizer(stop_words="english")
    sentence_vectors = vectorizer.fit_transform(sentences)

    while i < len(sentences):
        current_segment = [sentences[i]]
        segment_indices = [i]
        i += 1

        while i < len(sentences):
            similarity = cosine_similarity(sentence_vectors[segment_indices[-1]], sentence_vectors[i])[0, 0]

```

```
            if similarity >= threshold:
                current_segment.append(sentences[i])
                segment_indices.append(i)
                i += 1
            else:
                break

        segments.append(current_segment)

    # Save segmented transcript
    with open(output_file, 'w', encoding='utf-8') as file:
        for idx, segment in enumerate(segments):
            file.write(f"Segment {idx + 1}:\n")
            file.write(" ".join(segment) + "\n")
            file.write("-" * 50 + "\n")

    return segments

# Example usage
if __name__ == "__main__":
    transcript_sentences = read_transcript("/content/transcript (15).txt")
    segmented_text = segment_transcript(transcript_sentences, "segmented_transcript.txt")

    for idx, segment in enumerate(segmented_text):
        print(f"Segment {idx + 1}:")
        print(" ".join(segment))
        print("-" * 50)

```

```
[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!
Segment 1:
 So, sir, we know that India has seen a huge revolution with digital payments.
--------------------------------------------------
Segment 2:
We all thought that India is a place, at least the West thought that India is a place where many people do not get a square mill,
--------------------------------------------------
Segment 3:
That was an narrative some 30 years ago.
--------------------------------------------------
Segment 4:
And not many are literate, people cannot read.
--------------------------------------------------
Segment 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 50,
--------------------------------------------------
Segment 6:
I think immediately after UPI the next big revolution, personally I think is in education.
--------------------------------------------------
Segment 7:
And the complete homework for this has happened in the form of NEP, the documentation of which many of us have read and realized
--------------------------------------------------
Segment 8:
So, followed by which we got NCRF framework done, which is a national creative framework, surrounding which we will be discussing
--------------------------------------------------
Segment 9:
So, my question is, do you think NCRF plus NEP put together will be the next big revolution after UPI in India?
--------------------------------------------------
Segment 10:
Absolutely.
--------------------------------------------------
Segment 11:
And why I think so is because in education the last policy came up many many years ago that was in 1986, which was slightly tweak
--------------------------------------------------
Segment 12:
And thereafter so many changes have happened in the real world, so many changes have happened in the requirement of the industry,
--------------------------------------------------
Segment 13:
However there were no corresponding changes which happened in education system.
--------------------------------------------------
Segment 14:
So, therefore I feel that this was the right time when we brought in the education policy 2020, honorable prime minister declared
--------------------------------------------------
Segment 15:
And we recently celebrated the fourth anniversary of NEP 2020.
--------------------------------------------------
Segment 16:
NCRF has been brought to implement the intent of 2020, 2020 is a policy, NEP 2020 is a policy.
--------------------------------------------------
Segment 17:
And for implementing a policy you need a framework.
--------------------------------------------------
Segment 18:
Now, why we call it a framework?
```

```
#Using LLM
```

```
pip uninstall bitsandbytes -y
```

```
WARNING: Skipping bitsandbytes as it is not installed.
```

```
pip install bitsandbytes --no-cache-dir
```

```
Requirement already satisfied: bitsandbytes in /usr/local/lib/python3.11/dist-packages (0.45.3)
Requirement already satisfied: torch<3,>=2.0 in /usr/local/lib/python3.11/dist-packages (from bitsandbytes) (2.6.0+cu124)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from bitsandbytes) (2.0.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsandbytes) (3.17.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsandbyte
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsandbytes) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsandbytes) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsandbytes) (2024.10.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bits
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bi
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bits
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsandby
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsandb
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsandby
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsar
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsar
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bits
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsanc
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsandbytes
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsandbyt
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsa
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsandbytes) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsandbytes) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch<3,>=2.0->bit
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch<3,>=2.0->bitsandbytes
```

```
from transformers import pipeline
```

```
from transformers import AutoModelForCausalLM, AutoTokenizer, pipeline
```

```
import torch
```

```
# Ensure GPU is available
assert torch.cuda.is_available(), "GPU is not available! Check your CUDA installation."
```

```
pipe = pipeline("text-generation", model="deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B")
```

```
config.json: 100%                                   679/679 [00:00<00:00, 16.8kB/s]
model.safetensors: 100%                             3.55G/3.55G [00:23<00:00, 163MB/s]
generation_config.json: 100%                        181/181 [00:00<00:00, 6.96kB/s]
tokenizer_config.json: 100%                         3.07k/3.07k [00:00<00:00, 193kB/s]
tokenizer.json: 100%                                7.03M/7.03M [00:00<00:00, 24.3MB/s]
```

```
with open("/content/transcript (15).txt", "r", encoding="utf-8") as file:
    transcript_text = file.read()
```

```
inputs = [
    {
        "role": "system",
        "content": "You are a helpful assistant tasked with segmenting a long video lecture transcript into clear, distinct topic se
    },
    {
        "role": "user",
        "content": f"You have been provided with a long video lecture transcript. Your task is Read the transcript, split it into li
    },
    {
        "role": "assistant",
        "content": "Please segment the transcript provided below into clear sections without altering the original content. Ensure t
    },
    {
        "role": "user",
        "content": transcript_text
    }
]
```

```
a = pipe(inputs, max_length=500000)
```

Start coding or generate with AI.

```
#The Method we discussed
```

```
import nltk
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from nltk.corpus import stopwords

nltk.download('stopwords')
nltk.download('punkt')
nltk.download('punkt_tab')
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    words = nltk.word_tokenize(text.lower())
    return ' '.join([word for word in words if word.isalnum() and word not in stop_words])

def segment_transcript(transcript):
    lines = transcript.strip().split('\n')
    segments = []
    current_segment = lines[:5]  # First 5 lines
    processed_segment = [preprocess_text(line) for line in current_segment]
    vectorizer = TfidfVectorizer()
    segment_vectors = vectorizer.fit_transform(processed_segment)

    i = 5
    while i < len(lines):
        sentence = lines[i]
        processed_sentence = preprocess_text(sentence)
        sentence_vector = vectorizer.transform([processed_sentence])

        similarity_scores = cosine_similarity(sentence_vector, segment_vectors).flatten()
        max_similarity = np.max(similarity_scores) if len(similarity_scores) > 0 else 0

        if max_similarity > 0.5:  # Threshold for similarity
            current_segment.append(sentence)
            processed_segment.append(processed_sentence)
            segment_vectors = vectorizer.fit_transform(processed_segment)
        else:
            segments.append(current_segment)  # Store completed segment
            current_segment = [sentence] + lines[i+1:i+5]  # Start new segment
            processed_segment = [preprocess_text(line) for line in current_segment]
            segment_vectors = vectorizer.fit_transform(processed_segment)
            i += 4  # Skip next 4 lines as they are in the new segment

        i += 1

    if current_segment:
        segments.append(current_segment)  # Append last segment

    return segments

# Example usage
transcript = """ So, sir, we know that India has seen a huge revolution with digital payments. We all thought that India is a place,
segmented_output = segment_transcript(transcript)

for idx, segment in enumerate(segmented_output, 1):
    print(f"Segment {idx}:")
    print("\n".join(segment))
    print("-" * 50)
```

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

Start coding or generate with AI.