```
#pip install whisper openai diffusers pydub
```

```
#from google.colab import drive
#drive.mount('/content/drive')
```

```
#pip install moviepy speechrecognition transformers pillow
```

```
import moviepy.editor as mp

def extract_audio(video_path, audio_path):
    video = mp.VideoFileClip(video_path)
    audio = video.audio
    audio.write_audiofile(audio_path)
    print(f"Audio extracted and saved to {audio_path}")

# Example usage
video_path = '/content/drive/MyDrive/Datasciencedemo.mp4'
audio_path = 'extracted_audio.wav'
extract_audio(video_path, audio_path)
```

```
MoviePy - Writing audio in extracted_audio.wav
MoviePy - Done.
Audio extracted and saved to extracted_audio.wav
```

```
import whisper

def audio_to_text(audio_path, text_path):
    # Load the Whisper model
    model = whisper.load_model("base")

    # Transcribe the audio
    result = model.transcribe(audio_path)
    text = result["text"]

    # Save the transcribed text to a file
    with open(text_path, 'w') as file:
        file.write(text)

    print(f"Text transcribed and saved to {text_path}")
    return text

# Example usage
audio_path = 'extracted_audio.wav'
text_path = 'transcribed_text.txt'
text = audio_to_text(audio_path, text_path)
```

```
100%|███████████████████████████████████████| 139M/139M [00:02<00:00, 66.1MiB/s]
Text transcribed and saved to transcribed_text.txt
```

```python
# Open and read the contents of the transcribed text file
with open(text_path, 'r') as file:
    transcribed_text = file.read()

# Print the transcribed text
print("Transcribed Text:")
print(transcribed_text)
```

```
 get the latest updates on more such interesting videos. Thank you and keep learning.
```

```python
#pip install --upgrade whisper
```

```python
#pip install openai-whisper
```

```python
#pip install openai==0.28
```

```python
#pip install transformers
```

```
from transformers import BartForConditionalGeneration, BartTokenizer

def summarize_text(text, summary_path):
    # Load pre-trained BART model and tokenizer
    model_name = "facebook/bart-large-cnn"
    tokenizer = BartTokenizer.from_pretrained(model_name)
    model = BartForConditionalGeneration.from_pretrained(model_name)

    # Encode the input text
    inputs = tokenizer.encode("summarize: " + text, return_tensors="pt", max_length=1024,

    # Generate summary
    summary_ids = model.generate(inputs, max_length=150, min_length=40, length_penalty=2.
    summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)

    # Save the summary to a file
    with open(summary_path, 'w') as file:
        file.write(summary)

    print(f"Summarized text saved to {summary_path}")
    return summary

# Example usage
text_path = 'transcribed_text.txt'
summary_path = 'summarized_text.txt'

# Read the transcribed text from the file
with open(text_path, 'r') as file:
    transcribed_text = file.read()

# Summarize the transcribed text
summarized_text = summarize_text(transcribed_text, summary_path)

# Print the summarized text
print("Summarized Text:")
print(summarized_text)
```

```
WARNING:py.warnings:/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_tc
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
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 mc
  warnings.warn(
```

```
vocab.json: 100%                          899k/899k [00:00<00:00, 3.45MB/s]
merges.txt: 100%                          456k/456k [00:00<00:00, 20.0MB/s]
tokenizer.json: 100%                      1.36M/1.36M [00:00<00:00, 19.4MB/s]
config.json: 100%                         1.58k/1.58k [00:00<00:00, 109kB/s]
model.safetensors: 100%                   1.63G/1.63G [00:14<00:00, 80.3MB/s]
generation_config.json: 100%              363/363 [00:00<00:00, 26.5kB/s]
```

```
Summarized text saved to summarized_text.txt
Summarized Text:
The median base salaries of a data scientist can range from $95,000 to $165,000. Witl
```

```
#pip install transformers diffusers torch torchvision torchaudio
```

```
from transformers import BartForConditionalGeneration, BartTokenizer
from diffusers import StableDiffusionPipeline
import torch
from PIL import Image

# Function to summarize text
def summarize_text(text, summary_path):
    model_name = "facebook/bart-large-cnn"
    tokenizer = BartTokenizer.from_pretrained(model_name)
    model = BartForConditionalGeneration.from_pretrained(model_name)

    inputs = tokenizer.encode("summarize: " + text, return_tensors="pt", max_length=1024,
    summary_ids = model.generate(inputs, max_length=150, min_length=40, length_penalty=2.
    summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)

    with open(summary_path, 'w') as file:
        file.write(summary)

    print(f"Summarized text saved to {summary_path}")
    return summary

# Function to create an image banner based on summarized text
def create_image_banner(summary_text, image_path):
    model_id = "CompVis/stable-diffusion-v1-4"  # or use another model if available
    device = "cuda" if torch.cuda.is_available() else "cpu"

    pipe = StableDiffusionPipeline.from_pretrained(model_id)
    pipe = pipe.to(device)

    with torch.no_grad():
        image = pipe(summary_text).images[0]

    image.save(image_path)
    print(f"Banner image created and saved to {image_path}")

# Example usage
text_path = 'transcribed_text.txt'
summary_path = 'summarized_text.txt'
image_path = 'banner_image.png'

# Read the transcribed text from the file
with open(text_path, 'r') as file:
    transcribed_text = file.read()

# Summarize the transcribed text
summarized_text = summarize_text(transcribed_text, summary_path)

# Print the summarized text
print("Summarized Text:")
print(summarized_text)

# Create an image banner based on the summarized text
create_image_banner(summarized_text, image_path)
```

```
Summarized text saved to summarized_text.txt
Summarized Text:
The median base salaries of a data scientist can range from $95,000 to $165,000. Witl
model_index.json: 100%                                          541/541 [00:00<00:00, 39.4kB/s]
Fetching 16 files: 100%                                          16/16 [00:52<00:00,  3.63s/it]
safety_checker/config.json: 100%                              4.56k/4.56k [00:00<00:00, 198kB/s]
(…)ature_extractor/preprocessor_config.json: 100%               342/342 [00:00<00:00, 11.8kB/s]
tokenizer/merges.txt: 100%                                    525k/525k [00:00<00:00, 3.02MB/s]
(…)kpoints/scheduler_config-                                    209/209 [00:00<00:00, 1.91kB/s]
checkpoint.json: 100%
scheduler/scheduler_config.json: 100%                           313/313 [00:00<00:00, 3.39kB/s]
text_encoder/config.json: 100%                                  592/592 [00:00<00:00, 5.78kB/s]
tokenizer/special_tokens_map.json: 100%                         472/472 [00:00<00:00, 7.67kB/s]
tokenizer/tokenizer_config.json: 100%                           806/806 [00:00<00:00, 8.98kB/s]
unet/config.json: 100%                                          743/743 [00:00<00:00, 12.3kB/s]
tokenizer/vocab.json: 100%                                  1.06M/1.06M [00:00<00:00, 7.37MB/s]
vae/config.json: 100%                                           551/551 [00:00<00:00, 9.44kB/s]
model.safetensors: 100%                                       492M/492M [00:11<00:00, 25.7MB/s]
model.safetensors: 100%                                     1.22G/1.22G [00:20<00:00, 93.2MB/s]
diffusion_pytorch_model.safetensors: 100%                     335M/335M [00:07<00:00, 54.8MB/s]
diffusion_pytorch_model.safetensors: 100%                   3.44G/3.44G [00:50<00:00, 158MB/s]
Loading pipeline components...: 100%                                 7/7 [00:02<00:00,  2.71it/s]
Token indices sequence length is longer than the specified maximum sequence length fc
The following part of your input was truncated because CLIP can only handle sequences
100%                                            50/50 [00:23<00:00,  2.23it/s]
Banner image created and saved to banner_image.png
```

```
from IPython.display import Image as IPImage, display

# Display the generated image
display(IPImage(filename="banner_image.png"))
```

```
from transformers import BartForConditionalGeneration, BartTokenizer
from diffusers import StableDiffusionPipeline
import torch
from PIL import Image

# Function to summarize text
def summarize_text(text, summary_path):
    model_name = "facebook/bart-large-cnn"
    tokenizer = BartTokenizer.from_pretrained(model_name)
    model = BartForConditionalGeneration.from_pretrained(model_name)

    inputs = tokenizer.encode("summarize: " + text, return_tensors="pt", max_length
    summary_ids = model.generate(inputs, max_length=150, min_length=40, length_pena
    summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)

    with open(summary_path, 'w') as file:
        file.write(summary)

    print(f"Summarized text saved to {summary_path}")
    return summary

# Function to create an image banner based on summarized text
def create_image_banner(summary_text, image_path):
    model_id = "CompVis/stable-diffusion-v1-4"
    device = "cuda" if torch.cuda.is_available() else "cpu"
    pipe = StableDiffusionPipeline.from_pretrained(model_id)
    pipe = pipe.to(device)

    # Explicitly mention language and context
    prompt = f"A detailed and clear banner image for an educational video titled: '

    with torch.no_grad():
        image = pipe(prompt).images[0]

    image.save(image_path)
    print(f"Banner image created and saved to {image_path}")

# Example usage
text_path = 'transcribed_text.txt'
summary_path = 'summarized_text.txt'
image_path = 'banner_image.png'

# Read the transcribed text from the file
with open(text_path, 'r') as file:
    transcribed_text = file.read()

# Summarize the transcribed text
summarized_text = summarize_text(transcribed_text, summary_path)

# Print the summarized text
print("Summarized Text:")
print(summarized_text)

# Create an image banner based on the summarized text
create_image_banner(summarized_text, image_path)
```

```
Summarized text saved to summarized_text.txt
Summarized Text:
The median base salaries of a data scientist can range from $95,000 to $165,000. With
Loading pipeline components...: 100%                              7/7 [00:02<00:00,  2.70it/s]
Token indices sequence length is longer than the specified maximum sequence length fc
The following part of your input was truncated because CLIP can only handle sequences
100%                                          50/50 [00:22<00:00,  2.21it/s]
Banner image created and saved to banner_image.png
```

```
from IPython.display import Image as IPImage, display

# Display the generated image
display(IPImage(filename="banner_image.png"))
```

```
from transformers import BartForConditionalGeneration, BartTokenizer
from diffusers import StableDiffusionPipeline
import torch
from PIL import Image

# Function to summarize text
def summarize_text(text, summary_path):
    model_name = "facebook/bart-large-cnn"
    tokenizer = BartTokenizer.from_pretrained(model_name)
    model = BartForConditionalGeneration.from_pretrained(model_name)

    inputs = tokenizer.encode("summarize: " + text, return_tensors="pt", max_length
    summary_ids = model.generate(inputs, max_length=150, min_length=40, length_pena
    summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)

    with open(summary_path, 'w') as file:
        file.write(summary)

    print(f"Summarized text saved to {summary_path}")
    return summary

# Function to create an image banner based on summarized text
def create_image_banner(summary_text, image_path):
    model_id = "CompVis/stable-diffusion-v1-4"
    device = "cuda" if torch.cuda.is_available() else "cpu"
    pipe = StableDiffusionPipeline.from_pretrained(model_id)
    pipe = pipe.to(device)

    # Explicit and detailed prompt
    prompt = (f"Create a professional and visually appealing banner image for an ec
              f"The banner should clearly depict the topic of the video, which is:
              "The image should include elements that represent key concepts or the
              "Use a clean and modern design, with clear text in English, and ensur
              "Include appropriate symbols, illustrations, or icons that reflect th

    with torch.no_grad():
        image = pipe(prompt).images[0]

    image.save(image_path)
    print(f"Banner image created and saved to {image_path}")

# Example usage
text_path = 'transcribed_text.txt'
summary_path = 'summarized_text.txt'
image_path = 'banner_image.png'

# Read the transcribed text from the file
with open(text_path, 'r') as file:
    transcribed_text = file.read()

# Summarize the transcribed text
summarized_text = summarize_text(transcribed_text, summary_path)

# Print the summarized text
print("Summarized Text:")
print(summarized_text)

# Create an image banner based on the summarized text
create_image_banner(summarized_text, image_path)

from IPython.display import Image as IPImage, display

# Display the generated image
display(IPImage(filename="banner_image.png"))
```

```
Summarized text saved to summarized_text.txt
Summarized Text:
The median base salaries of a data scientist can range from $95,000 to $165,000. With
```

```
Loading pipeline components...: 100%                                7/7 [00:01<00:00,  3.45it/s]
```

```
Token indices sequence length is longer than the specified maximum sequence length fc
The following part of your input was truncated because CLIP can only handle sequences
```

```
100%                                        50/50 [00:22<00:00,  2.24it/s]
```

```
Banner image created and saved to banner_image.png
```

```
from transformers import BartForConditionalGeneration, BartTokenizer
from diffusers import StableDiffusionPipeline
import torch
from PIL import Image

# Function to create an image banner based on summarized text
def create_image_banner(summary_text, image_path):
    model_id = "CompVis/stable-diffusion-v1-4"
    device = "cuda" if torch.cuda.is_available() else "cpu"

    pipe = StableDiffusionPipeline.from_pretrained(model_id)
    pipe = pipe.to(device)

    # Explicit and detailed prompt
    prompt = (f"Create a professional and visually appealing banner image for an ec
              f"The banner should clearly depict the topic of the video, which is:
              "The image should include elements that represent key concepts or the
              "Use a clean and modern design, with clear text in English, and ensur
              "Include appropriate symbols, illustrations, or icons that reflect th

    with torch.no_grad():
        image = pipe(prompt).images[0]

    image.save(image_path)
```