Smart Voice-Based Meeting Analyzer - Code Documentation

Step 1: Install Dependencies

Install all the required Python libraries including Hugging Face Transformers, torchaudio, librosa, pyannote for diarization, and Gradio for UI.

```
!pip install -q transformers torchaudio librosa pyannote.audio gradio
```

Step 2: Login to Hugging Face

Authenticate using your Hugging Face account to access models such as Whisper and pyannote.

```
from huggingface_hub import notebook_login
notebook_login()
```

Step 3: Upload Audio File

Allows the user to upload an audio file for analysis. Typically in .wav or .mp3 format.

```
from google.colab import files
uploaded = files.upload()
```

Step 4: Load Whisper ASR Model

Load Hugging Face's Whisper model for speech-to-text transcription.

```
from transformers import pipeline
asr = pipeline("automatic-speech-recognition", model="openai/whisper-base")
```

Step 5: Transcribe Audio

Read the uploaded audio file using torchaudio and transcribe it using Whisper.

```
import torchaudio
file_path = list(uploaded.keys())[0]
waveform, sample_rate = torchaudio.load(file_path)
transcription = asr(file_path, chunk_length_s=30, stride_length_s=(5,2))
print(transcription["text"])
```

Step 6: Keyword Search

Search for specific keywords in the transcription.

```
def search_keywords(transcript, keywords):
    results = []
```

```
    for keyword in keywords:
        if keyword.lower() in transcript.lower():
            results.append(f"[OK] Found '{keyword}'")
        else:
            results.append(f"[X] Not Found '{keyword}'")
    return results

keywords = ["deadline", "project", "budget"]
print(search_keywords(transcription["text"], keywords))
```

Step 7: Speaker Diarization (Optional)

Use pyannote's diarization model to identify who is speaking at different times in the audio.

```
from pyannote.audio import Pipeline
pipeline = Pipeline.from_pretrained("pyannote/speaker-diarization")
diarization = pipeline(file_path)
for turn, _, speaker in diarization.itertracks(yield_label=True):
    print(f"{speaker} speaks from {turn.start:.1f}s to {turn.end:.1f}s")
```

Step 8: Gradio Interface for Keyword Search (Optional UI)

Launch a simple web interface to let users input keywords and search the transcript interactively.

```
import gradio as gr

def keyword_search_ui(keywords):
    result = search_keywords(transcription["text"], keywords.split(","))
    return "\n".join(result)

gr.Interface(fn=keyword_search_ui,
             inputs="text",
             outputs="text",
             title="Keyword Search in Audio Transcript").launch()
```