🎧 What Is "Simulated Hearing Loss"?

You're taking a **normal audio clip** (e.g., music, voice) and modifying it to **mimic how someone with hearing loss would perceive it** — particularly:

- **High-frequency hearing loss** (very common with age)
- **Mild or moderate general hearing loss**

You apply **digital audio filters** to reduce or eliminate certain frequencies — creating an audio illusion of real hearing impairment.

💡 Why Add This?

- **Educates users**: most people have no idea what hearing loss sounds like
- **Empathy-building**: perfect for demo video or live presentations
- **Unique and creative** — few apps actually simulate hearing loss this way
- Makes your app *not just diagnostic*, but also *experiential*

🛠️ How It Works (Technically)

1. Load a normal `.wav` audio file (speech/music)
2. Apply **frequency filters** using `scipy.signal` or `librosa`
    - Drop **high frequencies** to simulate presbycusis (age-related loss)
    - Drop **low AND high frequencies** to simulate “mild to moderate” loss
3. Export the modified audio
4. Let user toggle:
    - “Original”

    - “Mild hearing loss”
    - “High-frequency loss”

✅ Libraries Needed

```
pip install scipy soundfile librosa numpy
```

🔄 Implementation Steps in Streamlit

🔹 Step 1:Load a Clean Audio Clip

```
import streamlit as st
import soundfile as sf
import librosa
import numpy as np

st.title("Hearing Loss Simulator")

audio_file = “audi_file.wav”

if audio_file:
    y, sr = librosa.load(audio_file, sr=None)
    st.audio(audio_file, format="audio/wav", start_time=0)
```

🔹 Step 2: Apply Hearing Loss Filters

You can apply a **lowpass filter** or **bandstop filter** using `scipy.signal`.

```
from scipy.signal import butter, lfilter

def butter_bandstop(lowcut, highcut, fs, order=5):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='bandstop')
    return b, a

def apply_filter(data, lowcut, highcut, fs, order=6):
    b, a = butter_bandstop(lowcut, highcut, fs, order=order)
    y = lfilter(b, a, data)
    return y
```

Example configurations:

- **Mild hearing loss**: drop 300–3000 Hz slightly
- **High-frequency loss**: drop 4000–8000 Hz more aggressively

```
def simulate_mild_loss(audio, sr):
    return apply_filter(audio, 400, 3000, sr)

def simulate_high_freq_loss(audio, sr):
    return apply_filter(audio, 4000, 8000, sr)
```

🔹 Step 3: Play Modified Audio in Streamlit

```
import io

def convert_to_wav_bytes(y, sr):
    wav_bytes = io.BytesIO()
    sf.write(wav_bytes, y, sr, format='WAV')
    wav_bytes.seek(0)
    return wav_bytes

if audio_file:
    st.markdown("### Simulated Audio Versions")

    if st.button("Simulate Mild Hearing Loss"):
        y_mild = simulate_mild_loss(y, sr)
        st.audio(convert_to_wav_bytes(y_mild, sr), format='audio/wav')

    if st.button("Simulate High-Frequency Hearing Loss"):
        y_high = simulate_high_freq_loss(y, sr)
        st.audio(convert_to_wav_bytes(y_high, sr), format='audio/wav')
```

🔍 Optional Enhancements

- Show a **spectrogram** of original vs filtered signal (`librosa.display.specshow`)
- Add volume normalization

- Let users **download** simulated files via `st.download_button`

🧠 Bonus Ideas

- Auto-play your own clean `.wav` file (e.g., spoken instruction or music clip)
- Add checkboxes to toggle multiple types of loss
- Let users adjust **severity sliders** (filter strength)

✅ Summary

What	How
Simulate hearing loss	Apply digital filters to real audio
Mild loss	Drop 300–3000 Hz mildly
High-freq loss	Drop 4000–8000 Hz sharply
Libraries	`scipy`, `librosa`, `soundfile`, `numpy`
Integration	`st.audio()` for playback