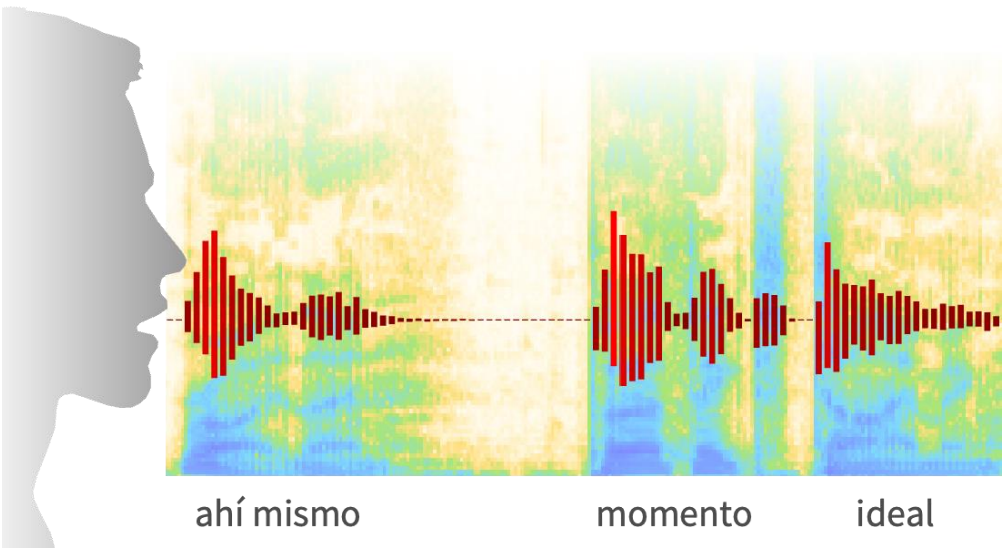
INSTITUTO TECNOLÓGICO BELTRÁN
Centro de Tecnología e Innovación

Análisis de Audio

Alumno: Marco Medina

Fechas: 21/06

Institución: Instituto Tecnologico Beltran



En esta práctica analizamos con scripts de Python señales de audio. El código realiza operaciones de carga, visualización, manipulación y reproducción de archivos WAV, implementando diversas bibliotecas para modificar características sonoras.

Para llevar a cabo esto, dividimos el código en etapas para visualizar facilmente los puntos a analizar:

1. Configuración inicial e importación de bibliotecas
2. Carga y análisis básico del archivo de audio
3. Visualización de la forma de onda
4. Manipulación de la señal mediante cuatro técnicas diferentes
5. Reproducción y almacenamiento de resultados

1- En primera instancia importamos las bibliotecas que usaremos para manipular el audio.

```
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np
import soundfile as sf
from IPython.display import Audio, display
import seaborn as sns
import sounddevice as sd
sns.set_context("poster")
```

- **De estas las más importantes son**:
  - **librosa**: procesamiento avanzado del audio
  - **sounddevice**: reproducción en tiempo real
  - **soundfile**: manejo de archivos de audio
  - **matplotlib/seaborn**: visualización profesional

- **Cabe destacar el estilo visual**: “poster” optimiza los gráficos para presentaciones

2- Luego cargamos el archivo de audio en .WAV y analizamos las partes importantes del mismo.

```
# Cargar archivo WAV convertido desde MP3
audio, sr = librosa.load("ruta_del_archivo.WAV", sr=None)  # sr=None mantiene la frecuencia original
```

- **Se aclara las variables**:
  - **Audio**: Array Numpy con amplitudes normalizadas.
  - **sr**: Frecuencia de muestreo en Hz

```
# Mostrar vector de la señal
indice_inicio = int(0.7 * sr)
print("Vector a partir del segundo 0.7:")
print(audio[indice_inicio:])

# Largo del array
print("Cantidad de elementos de la muestra: ", len(audio))

# Frecuencia de muestreo
print("Frecuencia de Muestreo: ", sr)

# Duración en segundos
duracion = len(audio) / sr
print("Duración (segundos): ", duracion)
```

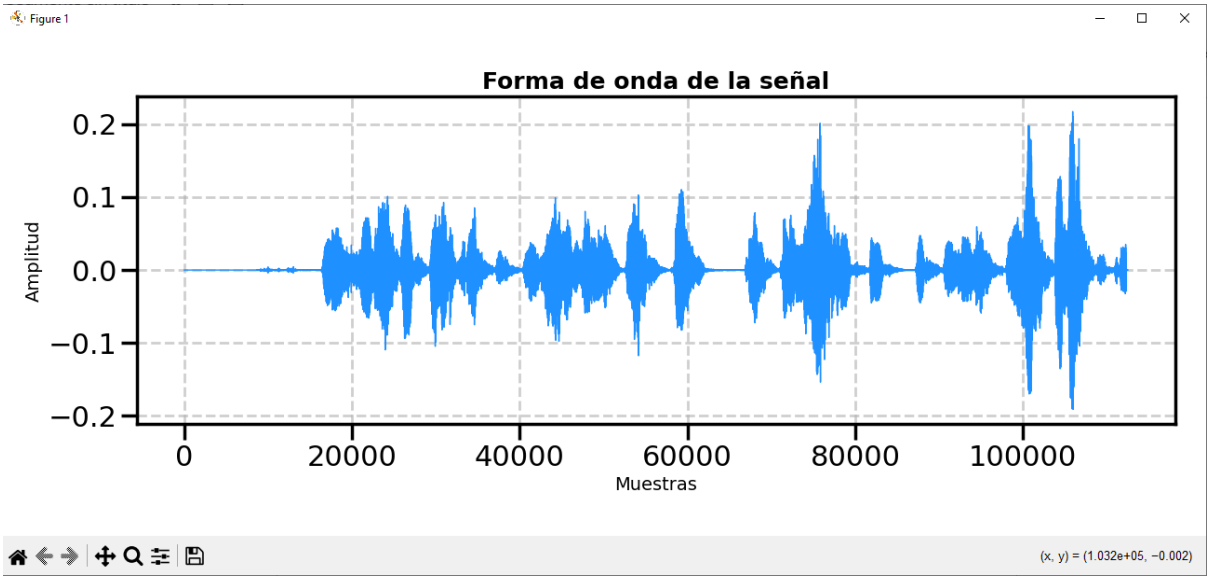
- **Con el audio ya cargado, visualizamos los datos que necesitamos. Por ejemplo**:
  - indice_inicio: que arranca en 0.7 , ya que, al haber huecos vacíos de audio se necesita vectorizar a partir que se empieza a captar el sonido.

- **Luego generamos**:
  - largo del array utilizando la función “len()”.
  - frecuencia llamando a “sr”.
  - Duración del audio en segundos, dividiendo el largo del array por la frecuencia.

3- Visualizamos el gráfico de las ondas de audio (Personalizando la las muestras y la amplitud con colores visibles sin ser invasivos):

```
# ------------------------------------------------
# PASO 5: IMPRIMIR LA SEÑAL SONORA (FORMA DE ONDA)
# ------------------------------------------------

plt.figure(figsize=(14, 5))  # Un poco más alto
plt.plot(audio, color='dodgerblue', linewidth=1)  # Color más vistoso y línea más fina
plt.title("Forma de onda de la señal", fontsize=18, fontweight='bold')
plt.xlabel("Muestras", fontsize=14)
plt.ylabel("Amplitud", fontsize=14)
plt.grid(True, linestyle='--', alpha=0.6)  # Agrega grilla suave
plt.tight_layout()  # Ajusta automáticamente los márgenes
plt.show()
```



4- Ahora reproducimos el audio original y luego lo manipulamos el audio para que se escuche de otras formas:

- **Audio original:**

```
print("Reproduciendo audio original con sounddevice...")
sd.play(audio, sr)
sd.wait()  # Espera hasta que termine de reproducirse
```

- **Audio mas largo (Sonido mas grave y lento):**

```
# Para hacer que el audio dure más (sonido más grave y lento)
sr_mas_lento = sr // 2  # Mitad de la frecuencia original
audio_lento = librosa.resample(audio, orig_sr=sr, target_sr=sr_mas_lento)
```

- **Audio mas corto (Sonido más agudo y rápido):**

```
# Para hacer que el audio dure menos (sonido más agudo y rápido)
sr_mas_rapido = sr * 2  # Doble de la frecuencia original
audio_rapido = librosa.resample(audio, orig_sr=sr, target_sr=sr_mas_rapido)
```

- **Audio con menos calidad (se reduce a la mitad de la frecuencia original)**

```
# Reducir la frecuencia de muestreo a la mitad (calidad más baja)
nueva_sr = sr // 2  # Mitad de la frecuencia original
audio_baja_calidad = librosa.resample(audio, orig_sr=sr, target_sr=nueva_sr)
```

- **Audio con menor profundidad de bits (de 32/64 bits a 8 bits):**

```
# Reducir la profundidad de bits (de 32/64 bits a 8 bits por ejemplo)
def reducir_profundidad_bits(audio, bits=8):
    """Reduce la profundidad de bits de la señal"""
    max_val = 2**(bits-1)
    return np.round(audio * max_val) / max_val

audio_8bits = reducir_profundidad_bits(audio, bits=8)
```

- **Audio que simula la compresión que tiene los audios.MP3:**

```python
# Simular compresión MP3 eliminando componentes frecuenciales
def comprimir_audio(audio, sr, factor=0.5):
    """Elimina parte del contenido frecuencial"""
    D = librosa.stft(audio)  # Transformada de Fourier
    magnitudes = np.abs(D)

    # Eliminar una parte de las frecuencias altas
    cutoff = int(magnitudes.shape[0] * factor)
    magnitudes[cutoff:, :] = 0

    # Reconstruir la señal
    audio_comprimido = librosa.istft(magnitudes * np.exp(1j * np.angle(D)))
    return audio_comprimido

audio_comprimido = comprimir_audio(audio, sr, factor=0.7)
```

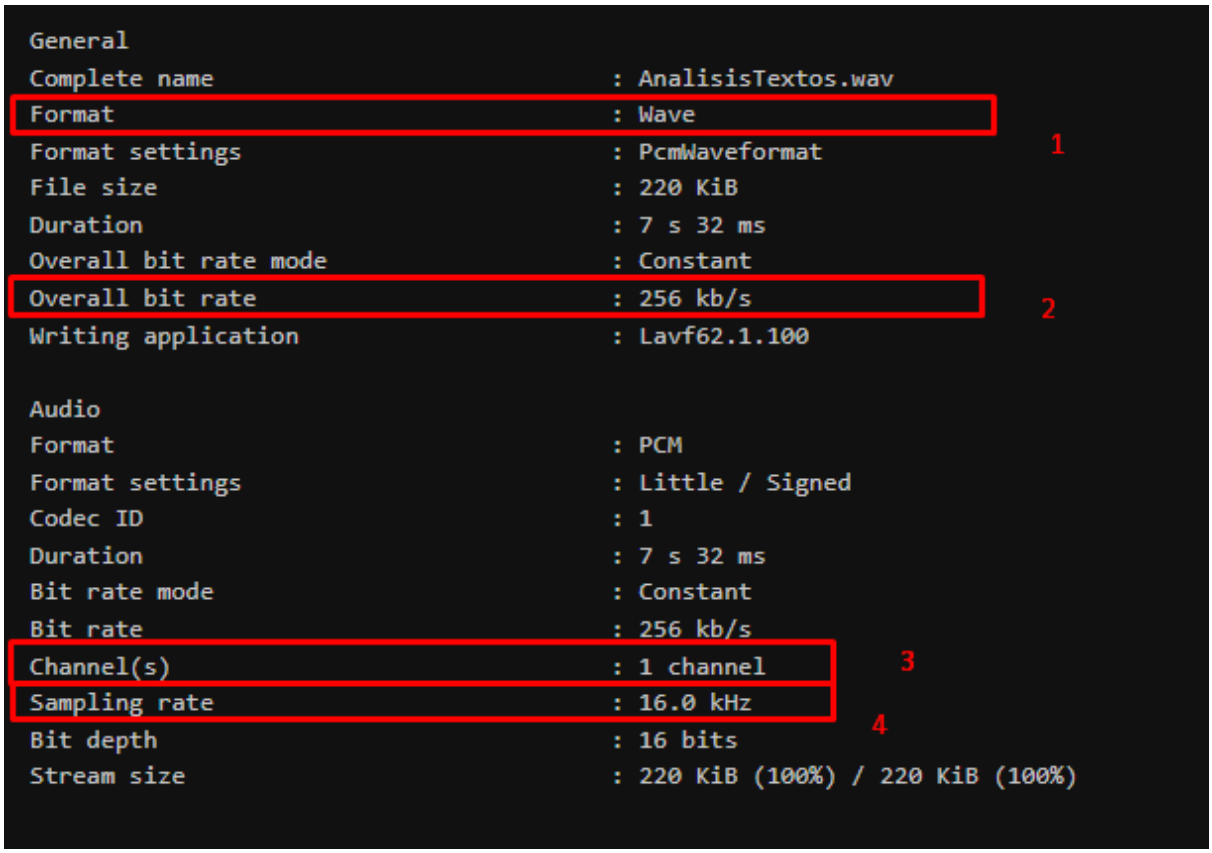
Análisis del audio.MP3 Y .WAV utilizando MediaInfo

- **Audio.MP3:**

```
General
Complete name                            : AnalisisTextos.mp3
Format                                   : MPEG Audio                    1
File size                                : 219 KiB
Duration                                 : 7 s 12 ms
Overall bit rate mode                    : Constant
Overall bit rate                         : 256 kb/s                      2
Writing library                          : LAME3.100

Audio
Format                                   : MPEG Audio
Format version                           : Version 1
Format profile                           : Layer 3
Duration                                 : 7 s 12 ms
Bit rate mode                            : Constant
Bit rate                                 : 256 kb/s
Channel(s)                               : 1 channel                     3
Sampling rate                            : 48.0 kHz                      4
Frame rate                               : 41.667 FPS (1152 SPF)
Compression mode                         : Lossy
Stream size                              : 219 KiB (100%) / 219 KiB (100%) / 219 KiB (100%) / 219 KiB (100%)
Writing library                          : LAME3.100
```

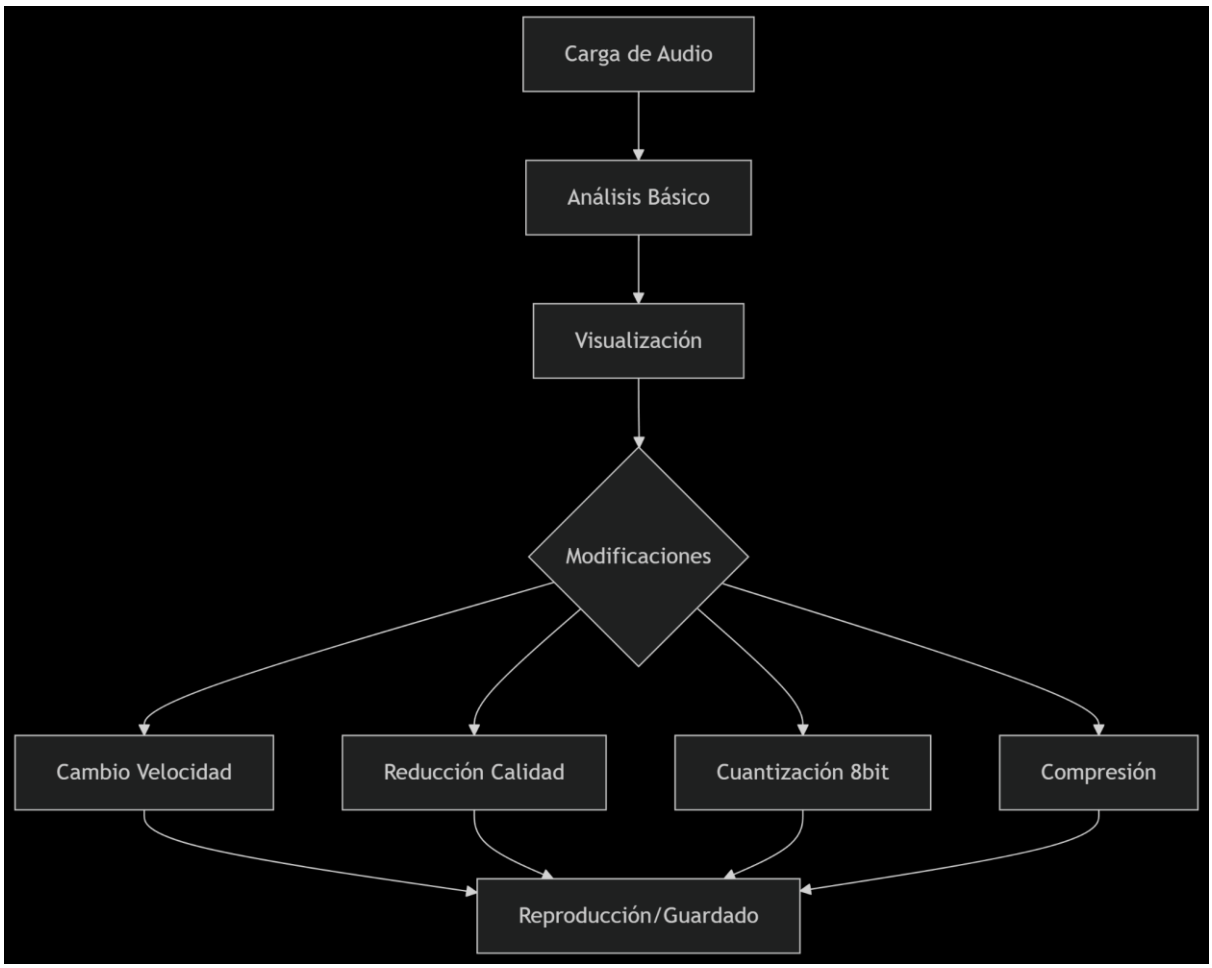
- **Audio.WAV:**



- En las anteriores imágenes se visualizamos los datos que nos brinda MediaInfo, de los cuales remarcamos: el formato(1), la tasa de bits(2), canales(3) y formato de muestreo(4).

Conclusión

- El código implementa un pipeline completo de procesamiento de audio digital con:
    - 4 técnicas distintas de transformación
    - Visualización profesional
    - Manejo adecuado de formatos WAV
    - Modularidad para extensiones futuras



- Las operaciones realizadas son fundamentales en aplicaciones de:

- Producción musical digital
- Procesamiento de voz (ASR, TTS)
- Análisis de señales acústicas
- Desarrollo de efectos de audio