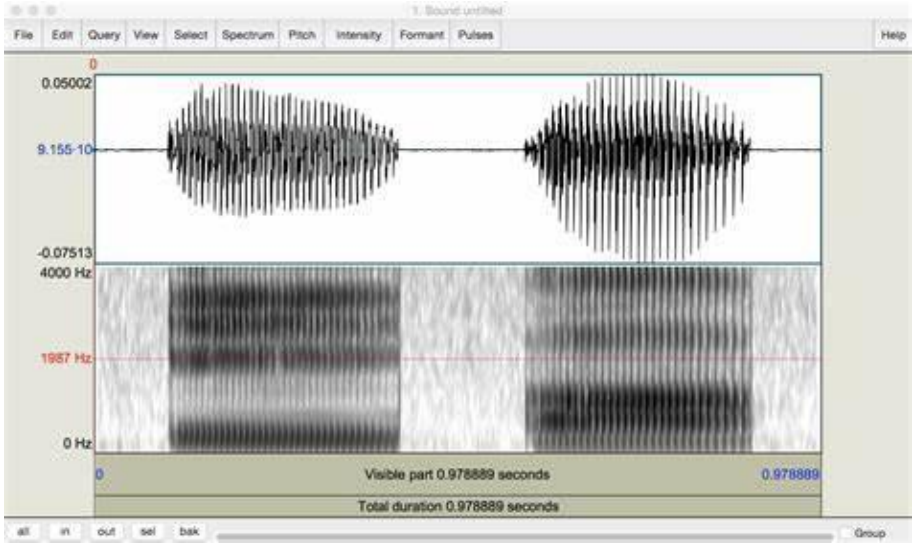
INSTITUTO TECNOLÓGICO BELTRÁN
Centro de Tecnología e Innovación

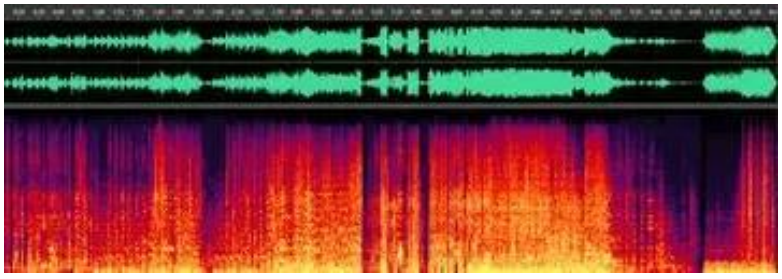
Análisis del Audio

"Técnicas de procesamiento del habla"

***Alumno** = Nicolas mesquiatti*

***Materia** = Técnicas de procesamiento del habla*





Ejercicio

Dado el audio: **AnalisisTextos.mp3**

1-Analizar el audio AnalisisTextos.mp3 con MediaInfo
Informar:

- → formato
- → tasa de bits
- → canales
- → frecuencia de muestreo

2-Realizar el sampleo con ffmpeg
3-Analizar el audio nuevamente con MediaInfo
Informar:

- → formato
- → tasa de bits
- → canales
- → formato de muestreo

4-Con Python:

- ♦ -Mostrar el Vector de la señal segmentada
- ♦ -Mostrar la cantidad de elementos de la muestra
- ♦ -Mostrar la Frecuencia de Muestreo
- ♦ -Mostrar la duración en segundos del audio

5-Imprimir la señal sonora
6-Reproducir la señal original
7-Modificar la frecuencia de muestreo para que dure más y menos tiempo. Explicar que sucede con el sonido
8-Bajar la calidad del audio y reproducir la señal.

- → Explicar cuál es el proceso

1)Análisis con MediaInfo :

→ formato = MPEG audio (sonido de alta calidad en tamaño resumido)

→ tasa de bits = 256 kb/s (ocupa 256 kb por segundo)

→ canales = 1 canal (un solo canal de sonido no es tan eficiente la percepción espacial del sonido que cuando Tenes 2 canales)

→ frecuencia de muestreo = 48.0 kHz (48 mil muestras por segundo)

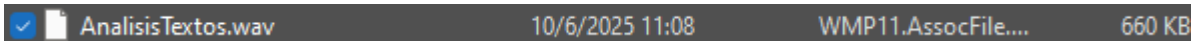
```
MediaInfo report of "AnalisisTextos.mp3" :

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

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
Sampling rate                            : 48.0 kHz
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                          : LAME
```

2)

Se realizo el sampleo con ffmpeg:

```
[mp3 @ 000001e36b4839c0] Estimating duration from bitrate, this may be inaccurate
Input #0, mp3, from 'C:\ffmpeg-master-latest-win64-gpl-shared\ffmpeg-master-latest-win64-gpl-shared\AnalisisTextos.mp3':
  Duration: 00:00:07.01, start: 0.000000, bitrate: 256 kb/s
  Stream #0:0: Audio: mp3 (mp3float), 48000 Hz, mono, fltp, 256 kb/s
Stream mapping:
  Stream #0:0 -> #0:0 (mp3 (mp3float) -> pcm_s16le (native))
Press [q] to stop, [?] for help
Output #0, wav, to 'C:\ffmpeg-master-latest-win64-gpl-shared\ffmpeg-master-latest-win64-gpl-shared\AnalisisTextos.wav':
  Metadata:
    ISFT            : Lavf62.1.100
  Stream #0:0: Audio: pcm_s16le ([1][0][0][0] / 0x0001), 48000 Hz, mono, s16, 768 kb/s
    Metadata:
      encoder         : Lavc62.3.101 pcm_s16le
[mp3float @ 000001e36d211e40] invalid new backstep -1
[out#0/wav @ 000001e36b4db1c0] video:0KiB audio:659KiB subtitle:0KiB other streams:0KiB global headers:0KiB muxing overhead:
0.011554%
size=     659KiB time=00:00:07.03 bitrate= 768.1kbits/s speed= 590x elapsed=0:00:00.01
```

3)



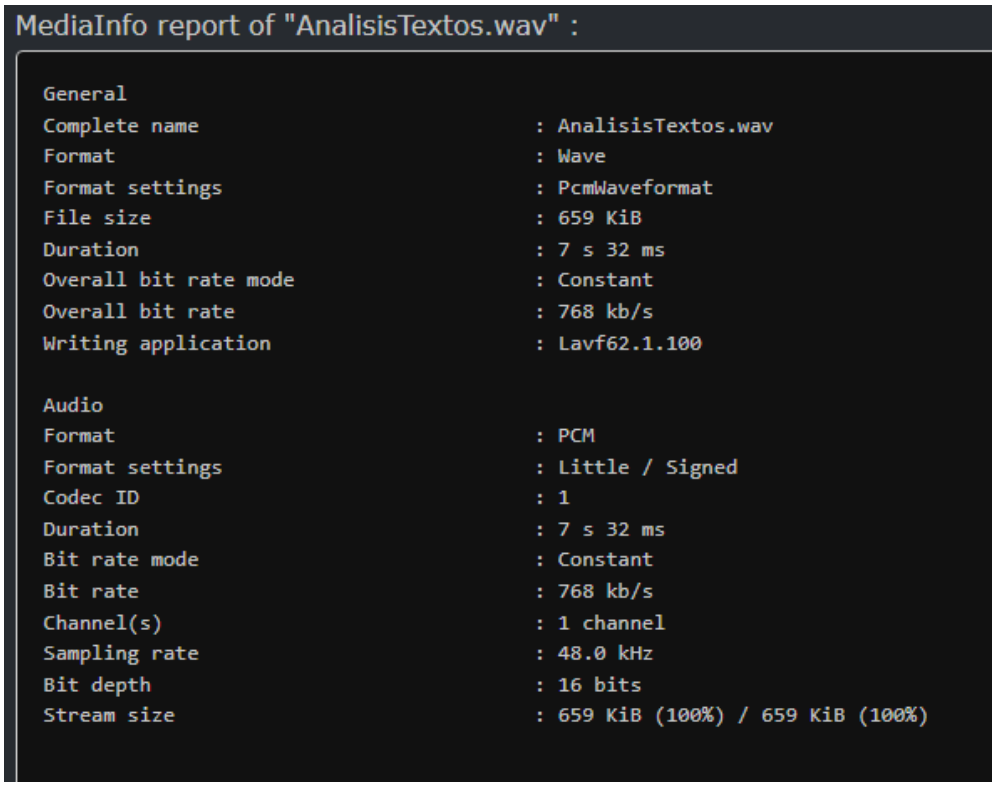
Luego de analizarlo con MediaInfo estos fueron los resultados:

→ formato = Wave (formato **sin compresión**, lo que mantiene el audio en su calidad original, pero con un tamaño de archivo mucho mayor.)

→ tasa de bits = 768 kb/s (es más pesado que el mp3 ya que este ocupa 768 kb por segundo, mientras que el mp3 256kb)

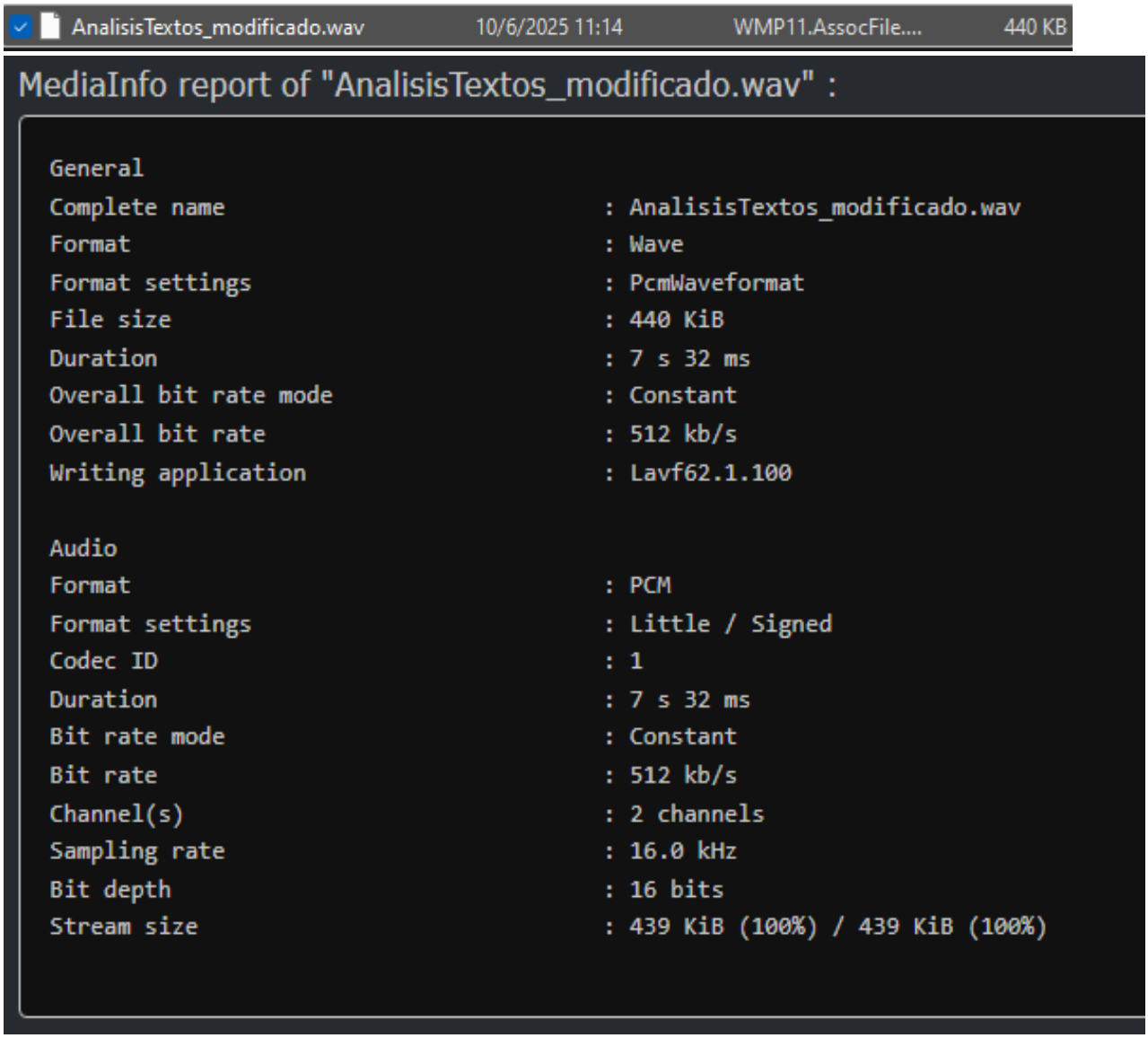
→ canales = 1 canal (un solo canal de sonido no es tan eficiente la percepción espacial del sonido que cuando Tenes 2 canales)

→ frecuencia de muestreo = 48.0 kHz (48 mil muestras por segundo)



Conclusión: Aunque ambos tienen la misma frecuencia de muestreo, **el WAV debería sonar más claro y definido**, especialmente en sonidos sutiles o en frecuencias altas, debido a que conserva toda la información del audio sin pérdida... Sinceramente casi no noto la diferencia de mejora entre ambos a primeras escuchas, Seguramente se encuentre la diferencia en un análisis más profundo en Python

Además de este archivo WAV, converti otro a .WAV cambiando el canal de 1 a 2 y cambiando la frecuencia de muestreo a menos de 48.0 kHz a 16 kHz



Análisis

→ formato = Wave (formato **sin compresión**, lo que mantiene el audio en su calidad original, pero con un tamaño de archivo mucho mayor.)

→ tasa de bits = 512 kb/s (es más pesado que el mp3(256), pero menos que el WAV (768) anterior, un punto intermedio entre los dos)

→ canales = 2 *canales (Si el archivo original era **mono**, convertirlo a **estéreo no añade información nueva**, solo copia el canal original en dos. Esto no mejora la calidad sino solo la especialización.)*

→ frecuencia de muestreo = 16.0 kHz (Reduce la calidad, utilizado en telefonía y voz en algunos sistemas. La pérdida de agudos puede hacer que el audio suene más opaco o menos definido.)

De los tres audios, este último es el que peor se escucha por más que tenga 2 canales, el cambio de frecuencia a un número menor afecta en la calidad del sonido

*4) **Análisis con Python***

Antes de hacer el análisis con Python debemos tener instaladas las siguientes librerías

"pip install librosa numpy matplotlib pygame soundfile pydub"

→ **librosa**: Carga, procesa y analiza señales de audio. Lo use para cargar el audio, para obtener la duración, hacer el grafico con forma de onda y para el espectrograma.

→ **numpy**: Manejo de matrices y cálculos numéricos, esencial para el procesamiento de señales. La use para la transformada de Fourier que se utiliza para el espectro de frecuencias.

→ **matplotlib.pyplot**: Generación de gráficos, incluyendo espectros y espectrogramas. Con matplot fue posible poder graficar la señal del audio, mostrar el espectrograma y personalizar esos gráficos con distintos títulos.

→ **pygame**: Reproducción de audio, útil cuando `librosa` no lo soporta. Gracias a pygame pude cargar el audio y ejecutarlo.

→ **soundfile**: Carga y guarda archivos de audio en formatos como WAV sin pérdidas.

→ **pydub**: Manipulación de audio: La use para Convertir formatos, Modificar frecuencia de muestreo y Exportar archivos con distintas calidades.

Realice el análisis de ambos audios 1) AnalisisTextos.mp3 y 2) AnalisisTextos.wav más que nada para ver en que cambiaban los gráficos

1) *AnalisisTextos.mp3*

Cargamos el audio:

```
audio_path = "AnalisisTextos.mp3"
y, sr = librosa.load(audio_path, sr=None)
```

- **4)-Mostrar el Vector de la señal segmentada**

(El primer vector se conforma con los primeros 50 elementos, como en el audio se encuentra un silencio muy breve, devuelve un vector de todos 0, pero si adelantamos un poco más el audio ya devuelve valores porque encuentra sonidos)

```
# 4) a)Vector de la señal segmentada
segmento = y[:50]  #Si traigo los primeros 50 elementos se escucha en silen
print("Vector en inicio de audio , que se encuentra silencios :", segmento)
segmento = y[70000:70200]  # Devuelve valor de 200 elementos ya que no hay
print("Nuevo segmento de la señal:", segmento)
```

- **4)-Mostrar la cantidad de elementos de la muestra**

```
# 4) b)Cantidad total de muestras
num_elementos = len(y)
print("Cantidad total de muestras:", num_elementos)
```

- **4)-Mostrar la Frecuencia de Muestreo**

```
# 4) c)Frecuencia de muestreo
print("Frecuencia de muestreo:", sr, "Hz")
```

- **4)-Mostrar la duración en segundos del audio**

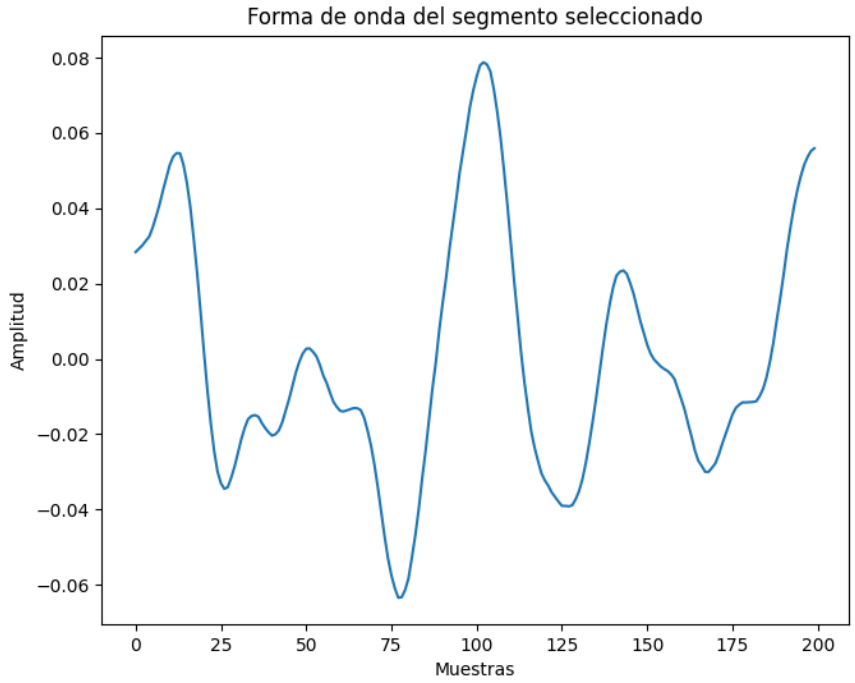
```
# 4) d) Duración del audio en segundos
duracion = librosa.get_duration(y=y, sr=sr)
print("Duración del audio:", duracion, "segundos")
```

5-Imprimir la señal sonora:

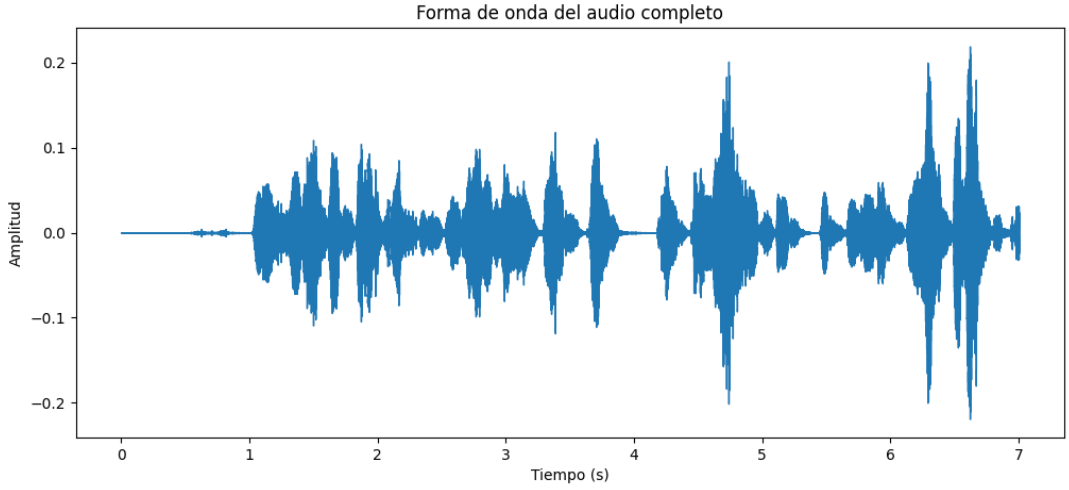
Realice un gráfico de onda a la parte segmentada del vector y otro grafico con forma de onda del audio completo

```
# 5) Imprimir la señal segmentada
plt.plot(segmento)
plt.title("Forma de onda del segmento seleccionado")
plt.xlabel("Muestras")
plt.ylabel("Amplitud")
plt.show()
# Graficar la forma de onda del audio completo
plt.figure(figsize=(12, 5))
librosa.display.waveshow(y, sr=sr)
plt.title("Forma de onda del audio completo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Amplitud")
plt.show()
```

Imprimí la señal segmentada



Imprimir la forma de onda del audio completa



6 - Reproducir la señal original

(Utilice la biblioteca de pygame : esta es una biblioteca útil a la hora de integrar audio dentro de un juego o app interactiva, en ese caso lo

uso para reproducir un audio sin dependencias, maneja la carga y reproducción del archivo MP3)

```
# 6) Reproducir el audio utilizando pygame
# Inicializar pygame
pygame.init()
print("Reproduciendo archivo de audio original...")

# Cargar y reproducir el audio
pygame.mixer.init()
pygame.mixer.music.load("AnalisisTextos.mp3")
pygame.mixer.music.play()

# Esperar hasta que termine la reproducción
while pygame.mixer.music.get_busy():
    continue
```

7-Modificar la frecuencia de muestreo para que
dure más y menos tiempo. Explicar que sucede con el sonido

En este paso use la librería de librosa para realizar el resampleo del audio, cambiando la frecuencia del muestreo y utilice la librería soundfile para guardar archivos de audios

```
# === 7. Modificar frecuencia de muestreo y reproducir ===

# Más lento
y_lento = librosa.resample(y, orig_sr=sr, target_sr=int(sr / 2))
sf.write("audio_mas_lento.wav", y_lento, int(sr / 2))

# Más rápido
y_rapido = librosa.resample(y, orig_sr=sr, target_sr=int(sr * 1.5))
sf.write("audio_mas_rapido.wav", y_rapido, int(sr * 1.5))

print("Archivos exportados con nueva frecuencia de muestreo.")
```

8-Bajar la calidad del audio y reproducir la señal.

```python
# === 8. Reducir calidad y reproducir ===
audio = AudioSegment.from_wav("AnalisisTextos.wav")
audio_baja_calidad = audio.set_frame_rate(8000).set_channels(1)
audio_baja_calidad.export("audio_baja_calidad.mp3", format="mp3", bitrate="32k")

print("Audio exportado en baja calidad (8000 Hz, 32 kbps).")

# === Reproducir los archivos con pygame ===

def reproducir_audio(archivo):
    pygame.init()
    pygame.mixer.init()
    pygame.mixer.music.load(archivo)
    pygame.mixer.music.play()
    while pygame.mixer.music.get_busy():
        continue

print("Reproduciendo archivo de menor frecuencia de muestreo...")
reproducir_audio("audio_mas_lento.wav")

print("Reproduciendo archivo de mayor frecuencia de muestreo...")
reproducir_audio("audio_mas_rapido.wav")

print("Reproduciendo archivo de baja calidad...")
reproducir_audio("audio_baja_calidad.mp3")
```

Explicar cuál es el proceso

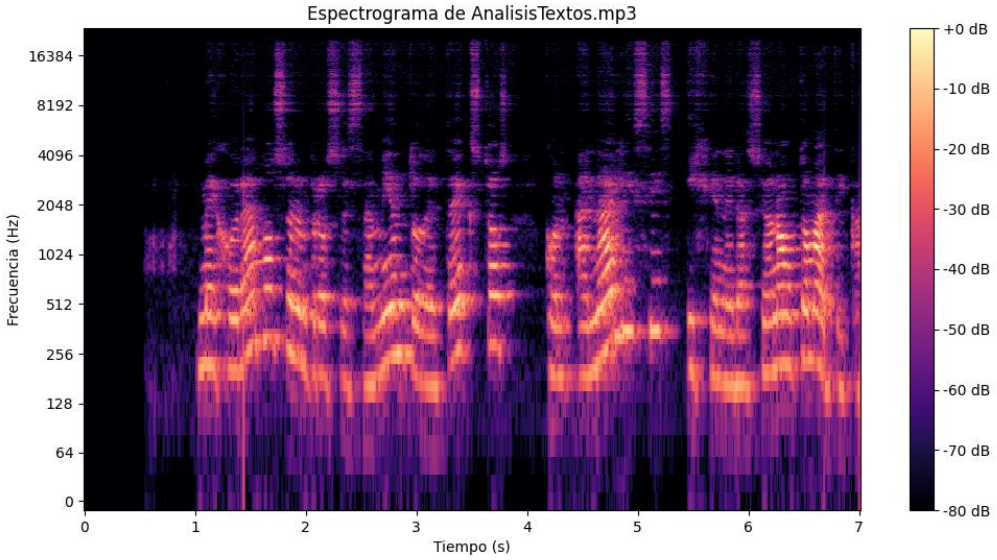
(En este paso utilicé la librería pydub para modificar las características de un archivo wav que habia mapleado con ffmpeg, y luego lo converti a mp3, después modifiqué la calidad reduciendo la frecuencia de muestreo a 8000 kHz, y lo cambié a un solo canal luego hice una función utilizando pygame para poder reproducir los audios de este ejercicio y el anterior y luego los reproduci.)

Gráfico de espectrograma y espectro de frecuencias

El espectrograma visualiza la evolución de **frecuencia** en el tiempo. Se genera usando la **Transformada Rápida de Fourier de Corto Tiempo (STFT)**, que descompone la señal en pequeñas ventanas temporales:

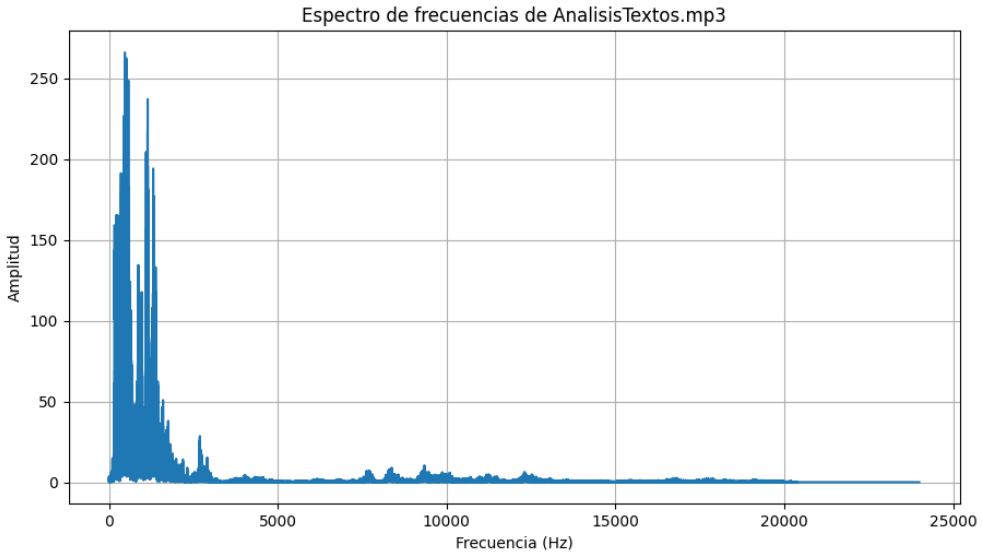
*Aplica la **STFT** a la señal y para obtener los componentes de frecuencia en pequeños segmentos de tiempo. Toma los valores absolutos de la matriz resultante para*

representar la magnitud de cada frecuencia. Convierte los valores de **amplitud** a **decibeles** (escala logarítmica) para una mejor interpretación visual. *Muestra el espectrograma con el tiempo en el eje X y la frecuencia en el eje Y. La escala es logarítmica, lo que ayuda a visualizar mejor las* ***frecuencias bajas***. Agrega una barra de colores para indicar los niveles de amplitud en decibeles.



El **espectro de frecuencias** muestra **qué frecuencias están presentes** en el sonido y con qué amplitud.

La transformada de Fourier Es una herramienta matemática que descompone una señal en sus frecuencias constituyentes y nos permite ver qué frecuencias están presentes en el audio y con qué intensidad.



Análisis con el audio sampleado .wav , a diferencia del mp3 el formato wav tiene más calidad sin perdidas, debería tener mayor resolución en espectrogramas, frecuencias

más altas se mantienen generando gráficos más detallados, igualmente mucha diferencia no se verá debido a que ambos audios tienen la misma frecuencia de muestreo

Debemos tener las mismas librerías instaladas que para analizas el archivo mp3

pip install librosa numpy matplotlib pygame soundfile

2) *AnalisisTextos.wav*

Cargamos el audio:

```
# Cargar el archivo de audio
audio_path = "AnalisisTextos.wav"
y, sr = librosa.load(audio_path, sr=None)
```

4)-a) Mostrar el Vector de la señal segmentada -b) Mostrar la cantidad de elementos de la muestra -c) Mostrar la Frecuencia de Muestreo -d) Mostrar la duración en segundos del audio

```
# 4) a)Vector de la señal segmentada
segmento = y[:50]  #Si traigo los primeros 50 elementos se escucha en silenc
print("Vector en inicio de audio , que se encuentra silencios :", segmento)

segmento = y[80000:80500]  # Devuelve valor de 500 elementos ya que no hay s
print("Nuevo segmento de la señal:", segmento)

# 4) b)Cantidad total de muestras
num_elementos = len(y)
print("Cantidad total de muestras:", num_elementos)

# 4) c)Frecuencia de muestreo
print("Frecuencia de muestreo:", sr, "Hz")

# 4) d) Duración del audio en segundos
duracion = librosa.get_duration(y=y, sr=sr)
print("Duración del audio:", duracion, "segundos")
```

5-Imprimir la señal sonora

```python
# 5) Imprimir la señal segmentada
plt.plot(segmento)
plt.title("Forma de onda del segmento seleccionado")
plt.xlabel("Muestras")
plt.ylabel("Amplitud")
plt.show()
# Graficar la forma de onda del audio completo
plt.figure(figsize=(12, 5))
librosa.display.waveshow(y, sr=sr)
plt.title("Forma de onda del audio completo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Amplitud")
plt.show()
```

Gráfico de onda del segmento (segmento = y [80000:80500])

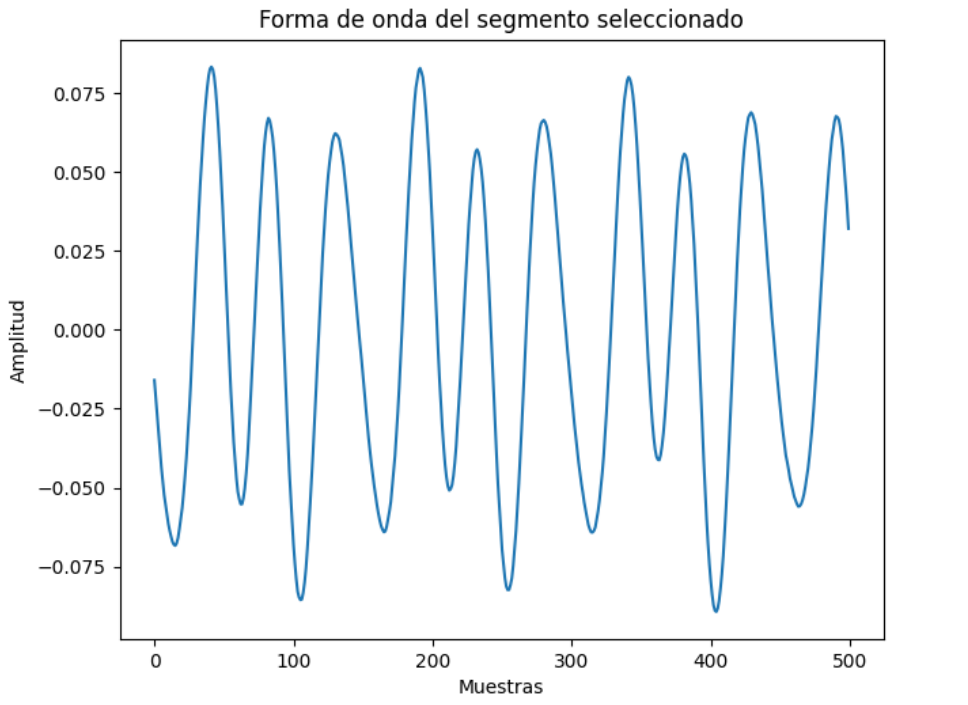
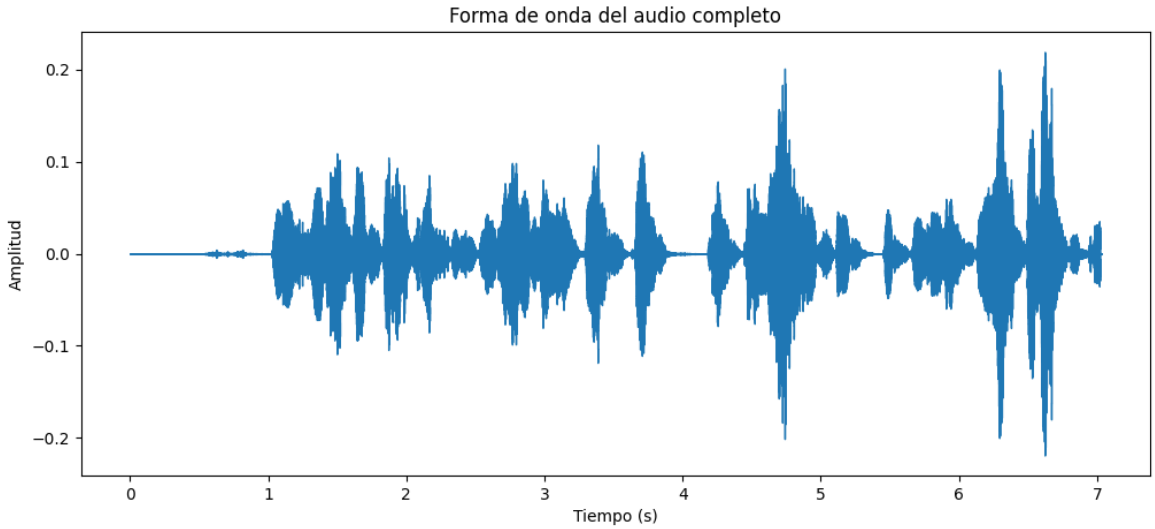


Gráfico de onda del audio completo



6-Reproducir la señal original

```
# 6) Reproducir el audio utilizando pygame
# Inicializar pygame
pygame.init()
print("Reproduciendo archivo de audio original...")

# Cargar y reproducir el audio
pygame.mixer.init()
pygame.mixer.music.load("AnalisisTextos.wav")
pygame.mixer.music.play()

# Esperar hasta que termine la reproducción
while pygame.mixer.music.get_busy():
    continue
```

7-Modificar la frecuencia de muestreo para que dure más y menos tiempo. Explicar que sucede con el sonido

A diferencia del anterior análisis acá reproduci dentro del mismo ejercicio sin hacer una función que se llame reproducir audio

```python
#7) Cambiar la frecuencia de muestreo del audio
# Inicializar pygame
#Printeo los khz de cada audio reproducidos
data_menos, sr_menos = sf.read("output_menos.wav")
data_mas, sr_mas = sf.read("output_mas.wav")
pygame.init()
pygame.mixer.init()

# Reproducir el archivo con frecuencia reducida
pygame.mixer.music.load("output_menos.wav")
pygame.mixer.music.play()
print("Reproduciendo: Audio con frecuencia reducida")
print("Frecuencia de muestreo reducida:", sr_menos, "Hz")
# Esperar a que termine
while pygame.mixer.music.get_busy():
    continue

# Reproducir el archivo con frecuencia aumentada
pygame.mixer.music.load("output_mas.wav")
pygame.mixer.music.play()
print("Reproduciendo: Audio con frecuencia aumentada")
print("Frecuencia de muestreo aumentada:", sr_mas, "Hz")

# Esperar a que termine
while pygame.mixer.music.get_busy():
    continue

print("Reproducción completada")
```

8-Bajar la calidad del audio y reproducir la señal.

```python
#8)Bajar la calidad del audio
# Reproducir el archivo de audio con calidad reducida En e

pygame.init()
pygame.mixer.init()
pygame.mixer.music.load("AnalisisTextos_modificado.wav")
pygame.mixer.music.play()

print("Reproduciendo archivo de calidad reducida...")

while pygame.mixer.music.get_busy():
    continue

print("Reproducción completada")
```

Explicar cuál es el proceso

Reproduci el archivo AnalisisTextos_modificado.wav, es un archivo que habia sampleado con ffmpeg donde lo transforme en 2 canales, y le baje la frecuencia de muestreo a 16 kHz

Gráfico de espectrograma y espectro de frecuencias

Con AnalisisTextos.wav y AnalisisTextos_modificado.wav PARA COMPROBAR LAS DIFERENCIAS EN TODOS LOS GRAFICOS

AnalisisTextos.wav

Espectrograma

```
#Espectograma
# Convertir el audio a un espectrograma
D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)

# Visualizar el espectrograma
plt.figure(figsize=(10, 4))
librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='log')
plt.colorbar(format="%+2.0f dB")
plt.title("Espectrograma")
plt.show()
# Guardar el espectrograma como imagen
#plt.savefig("espectrograma.png")
```

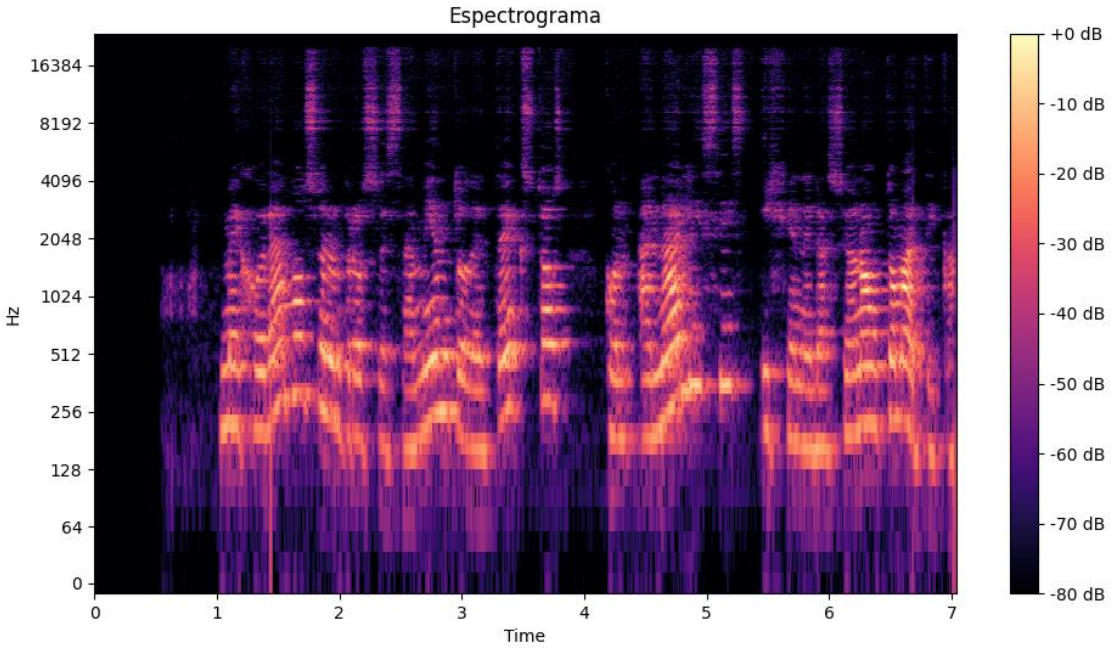
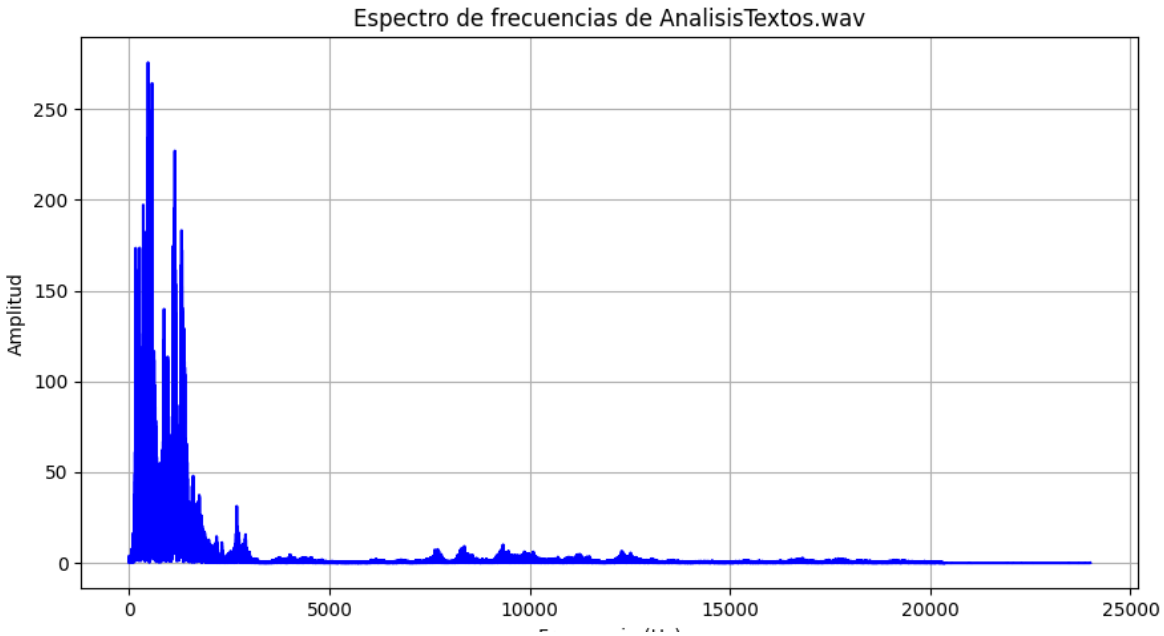


Gráfico del espectro

```python
# Espectro de frecuencias del audio original .wav

# Aplicar la Transformada de Fourier (FFT)
fft = np.fft.fft(y)
frecuencias = np.fft.fftfreq(len(y), 1/sr)

# Tomar solo la mitad del espectro (frecuencias positivas)
fft_magnitud = np.abs(fft[:len(fft)//2])
frecuencias = frecuencias[:len(frecuencias)//2]

# Graficar el espectro
plt.figure(figsize=(10, 5))
plt.plot(frecuencias, fft_magnitud, color="blue")
plt.xlabel("Frecuencia (Hz)")
plt.ylabel("Amplitud")
plt.title("Espectro de frecuencias de AnalisisTextos.wav")
plt.grid()
plt.show()
```



AnalisisTextos_modificado.wav

Espectrograma

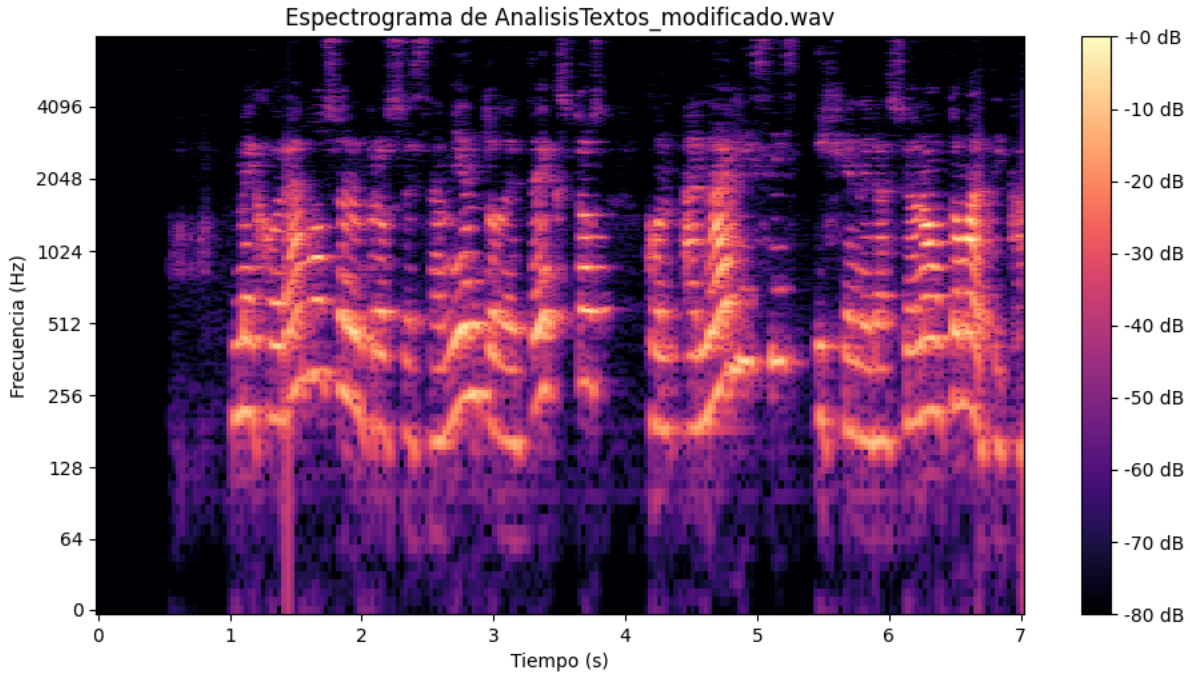
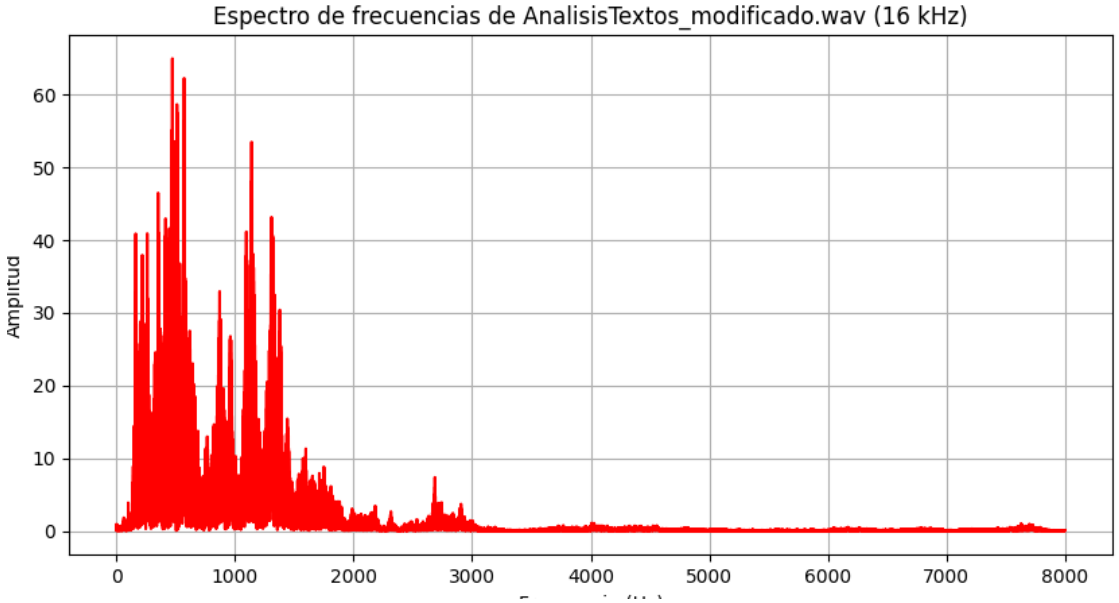


Gráfico del espectro



Conclusión de los gráficos

→ El espectrograma de 16384 se ve más “suave” o detallado en frecuencia, pero más borroso en el eje del tiempo.
→ El de 4096 es más “afilado” en tiempo, pero más grueso en frecuencia.

En un espectrograma con *n_fft=4096*, como hay mejor resolución temporal, los eventos cortos (incluidos los silencios breves) aparecen más definidos y visibles. En cambio, en uno con *n_fft=16384*, al tener menor resolución temporal, los silencios pueden quedar “diluidos” o fundidos con otros sonidos, haciéndolos menos evidentes. Es como mirar una película en cámara lenta (4096) versus verla en cámara súper lenta pero borrosa (16384): en la primera vez mejor los momentos exactos donde pasa algo, como un silencio.