MP3
Audio player

Created by:

- Salma Sherif
- Menna Azab
- Nada Samir

Presented to:

DR/ Eslam Shaalan

Tuneytones

• Abstract

This project is a Python-based music player with additional functionality such as noise reduction, volume control, pitch shifting, and speed adjustment.

The program uses various libraries and modules.

The program allows users to play individual songs or create playlists, and includes features such as shuffling and navigation between songs.

This project demonstrates the power and versatility of Python for audio processing, playback and serves as a great example of what can be achieved with Python in this field.

• Table of contents

• List of figures

1. Introduction

TuneyTones is a Python-based music player with additional functionality such as noise reduction, volume control, pitch shifting, and speed adjustment. The program uses various libraries and modules, including Pygame.mixer, Tkinter, Librosa, Pydub and more libraries, to create a user-friendly interface with easy-to-use controls. The program allows users to play individual songs or create playlists, and includes features such as shuffling and navigation between songs. The user can also add white noise to the currently playing song, reduce noise from the audio file, and adjust the amplitude, pitch, and speed of the audio. Additionally, the program includes a text-to-speech feature that converts user input into speech and adds it to the playlist. Overall, this project demonstrates the power and versatility of Python for audio processing, playback, and serves as a great example of what can be achieved with Python in this field.

2. Applications

TuneyTones have various practical applications and use cases. For example:

- ✓ Personal music library: Users can use the audio player to manage their personal music collection, create playlists, and listen to their favorite songs with added functionalities for adjusting
- ✓ volume, skipping tracks, shuffling the playlist, and applying audio effects such as white noise, noise reduction, and amplification.
- ✓ Podcasts and audiobooks: Users can use the audio player to listen to their favorite podcasts and audiobooks with added functionalities for adjusting playback speed and introducing a delay.
- ✓ Audio editing and manipulation: Users can use the audio player as a tool for editing and manipulating audio files with added functionalities for changing pitch, amplifying/attenuating audio, and applying various audio effects.
- ✓ Text-to-speech: The audio player's text-to-speech feature can be used by users who want to convert text to speech and listen to the generated audio files.

3. Audio Player GUI

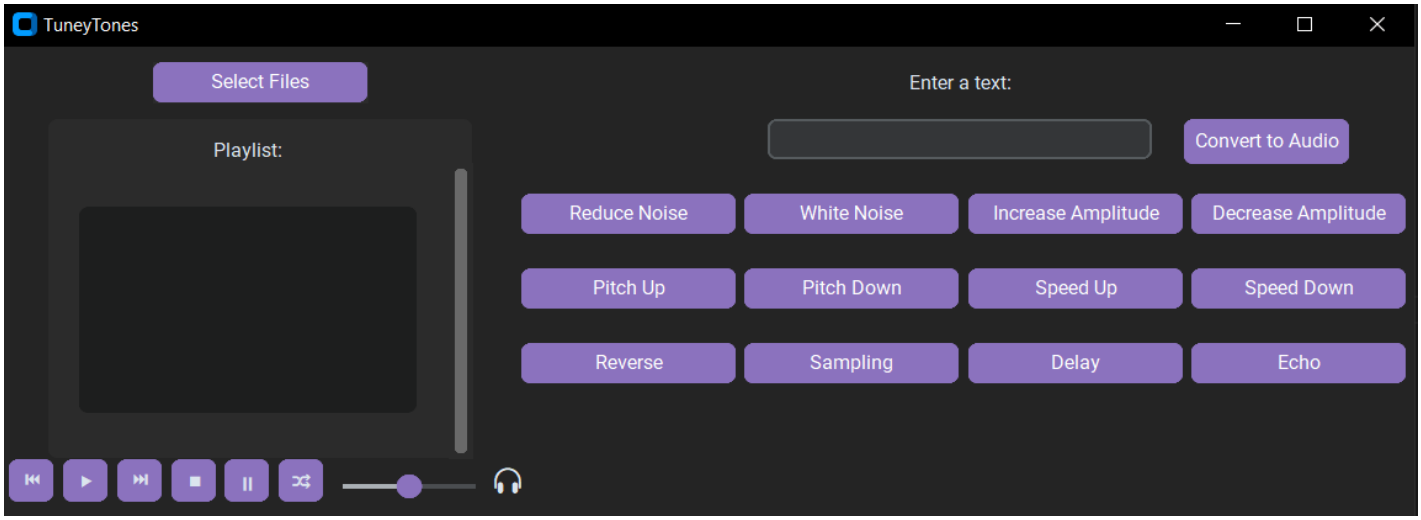

Figure1.TuneyTones GUI

4. Original Waveform of a selected soundtrack

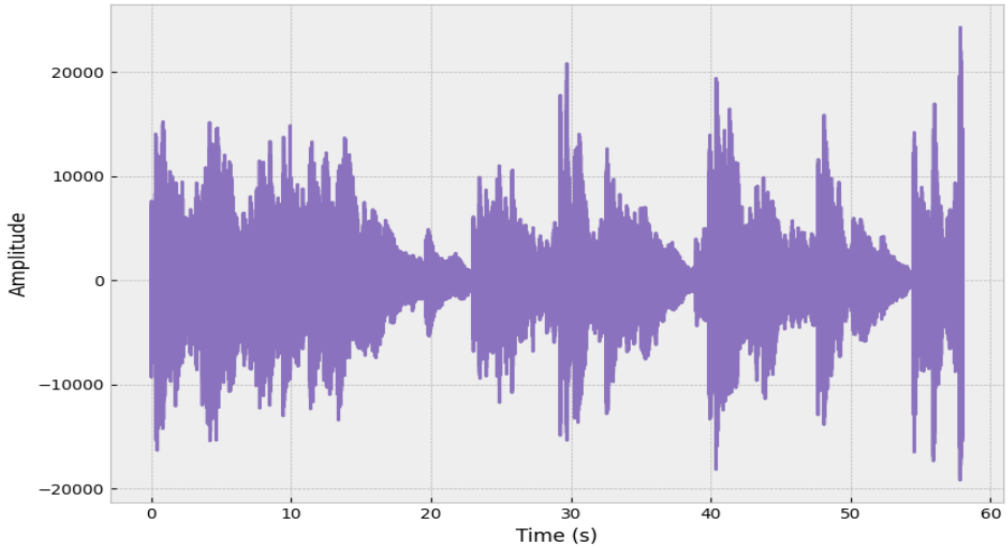

Figure2. Original Waveform

5. TuneyTones Functions

5.1. def speed_up():

The speed_up() function uses the librosa library to speed up the playback of the currently playing song in the playlist. The function loads the current song into a NumPy array, sets a desired stretch factor and computes the phase vocoder transformation of the audio signal using librosa.phase_vocoder(). The resulting transformed audio signal is then inverted using the inverse short-time Fourier transform with librosa.istft() to obtain the stretched audio time series. The amplitude of the audio signal is increased by a gain of 6dB using the formula y_stretch = y_stretch * 10**(6/20). The stretched audio signal is saved to a temporary file and added to the playlist for playback. Finally, the original song is restored in the playlist.

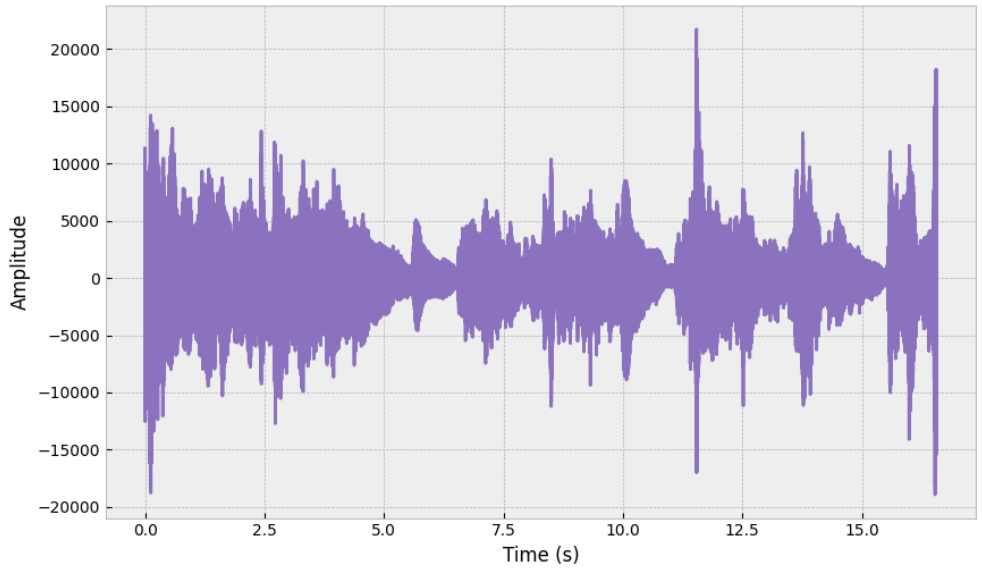

Figure3.Speed Up Waveform

When we apply speed up or time compression to an audio waveform, we reduce its duration without changing its pitch or frequency content. This means that the waveform is compressed in time to adjust for the change in duration. More specifically, time compression involves reducing the duration of the audio signal by a certain factor. As a result of time compression, the waveform's fundamental frequency and harmonics remain the same, but the waveform is compressed in time. This means that the waveform's temporal features, such as its attack and decay times, are also compressed and occur more rapidly. This can result in a higher perceived tempo or speed of the audio.In summary, time compression changes the duration of an audio waveform without changing its pitch or frequency content, while compressing the waveform's temporal features and making it sound faster.

5.2. def speed_down():

The speed_down() function uses the librosa library to slow down the playback of the currently playing song in the playlist. The function loads the current song into a NumPy array, sets a desired stretch factor and computes the phase vocoder transformation of the audio signal using librosa.phase_vocoder(). The resulting transformed audio signal is then inverted using the inverse short-time Fourier transform with librosa.istft() to obtain the stretched audio time series. The amplitude of the audio signal is increased by a gain of 6dB using the formula y_stretch = y_stretch * 10**(6/20). The stretched audio signal is saved to a temporary file and added to the playlist for playback. Finally, the original song is restored in the playlist.

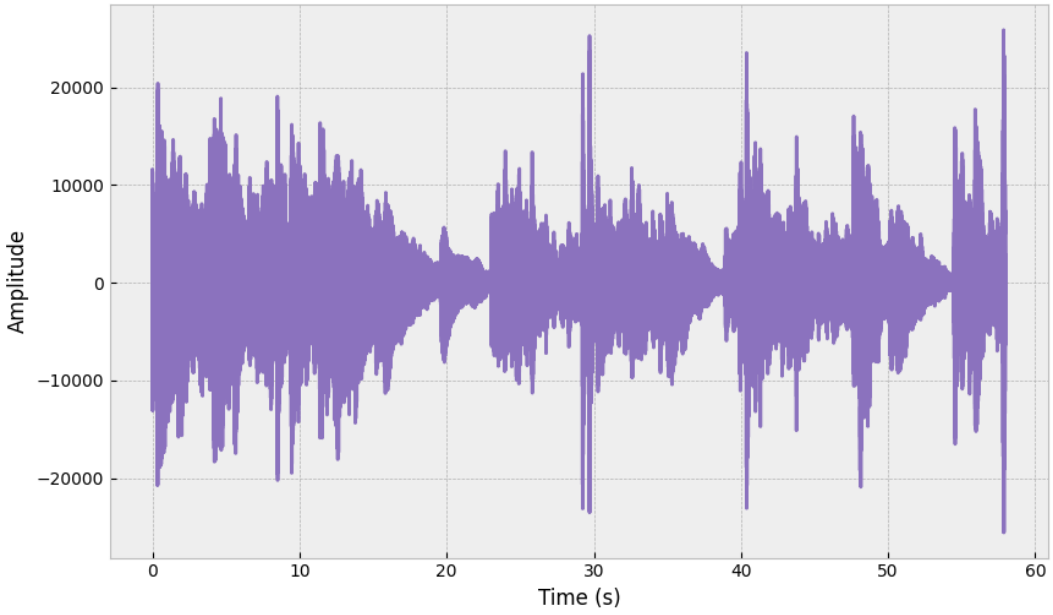

Figure4.Speed Down Waveform

When we apply speed down or time expansion to an audio waveform, we increase its duration without changing its pitch or frequency content. This means that the waveform is stretched in time to adjust for the change in duration. More specifically, time expansion involves increasing the duration of the audio signal by a certain factor. As a result of time expansion, the waveform's fundamental frequency and harmonics remain the same, but the waveform is stretched in time. This means that the waveform's temporal features, such as its attack and decay times, are also stretched and occur more slowly. This can result in a lower perceived tempo or speed of the audio. In summary, time expansion changes the duration of an audio waveform without changing its pitch or frequency content, while stretching the waveform's temporal features and making it sound slower.

5.3. def pitch_up():

In the pitch_up() function modifies the frame rate of the audio signal, causing a change in the frequency content of the audio signal in the frequency domain. Specifically, the frequency content of the signal is shifted upwards by the desired number of semitones. This means that the fundamental frequency and all of the harmonics are shifted upwards by the same amount in Hertz.

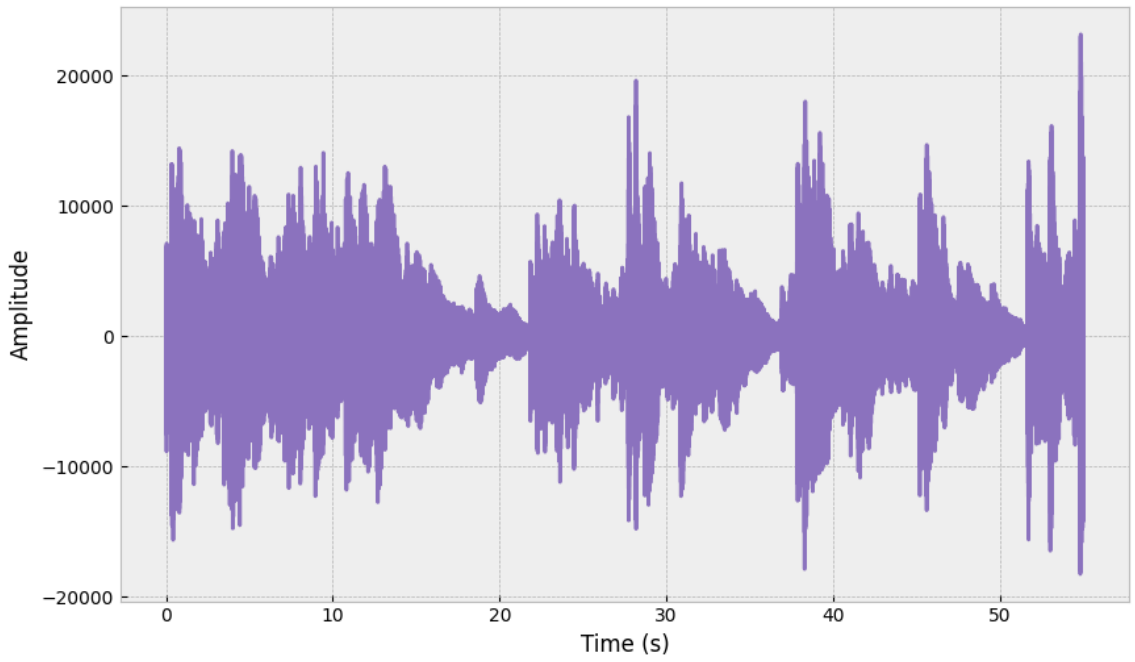

Figure5.Pitch Up Waveform

When we apply pitch up to an audio waveform, we raise its pitch or frequency without changing its duration. This means that the waveform is compressed in time to adjust for the change in pitch. More specifically, pitch up involves increasing the frequency of the audio signal by a certain factor.

As a result of pitch up, the waveform's fundamental frequency is increased, and all of its harmonics are shifted proportionally upward in frequency. This creates a higher-pitched sound. However, since the duration of the waveform remains the same, the waveform is compressed in time to accommodate the higher frequency.

5.4. def pitch_down():

In the pitch_down() function, the pitch is shifted by modifying the frame rate of the audio signal using the _spawn() method, causing a change in the frequency content of the audio signal in the frequency domain. The frequency content of the signal is shifted downwards by the desired number of semitones, resulting in a lower-pitched audio signal.

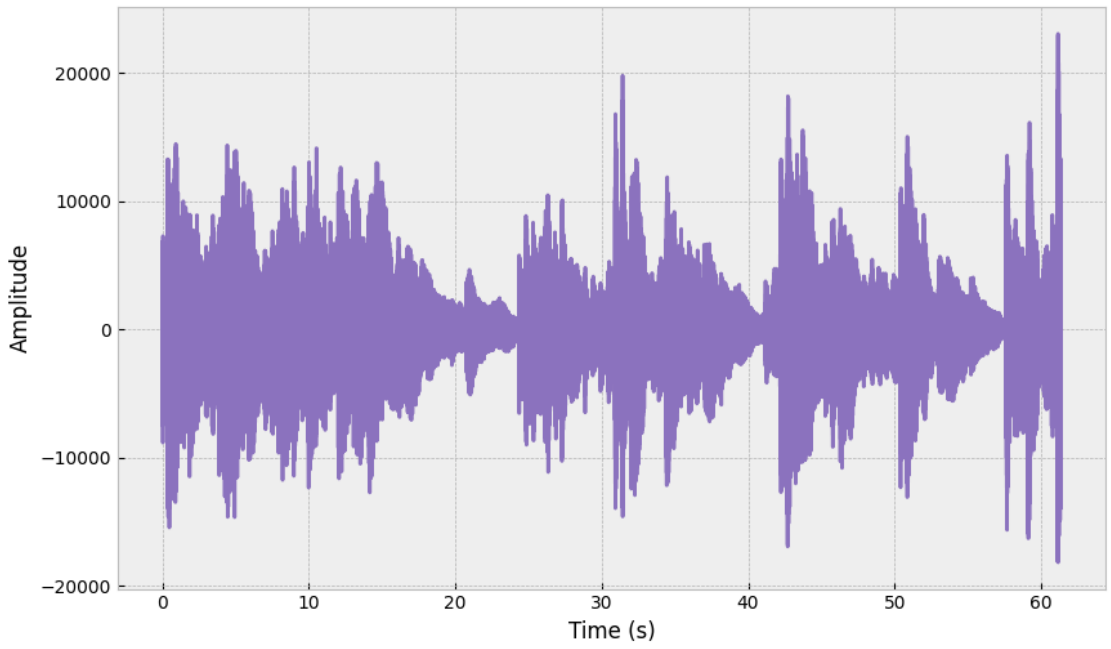

Figure6.Pitch Down Waveform

When we apply pitch down to an audio waveform, we lower its pitch or frequency without changing its duration. This means that the waveform is stretched in time to adjust for the change in pitch. More specifically, pitch down involves reducing the frequency of the audio signal by a certain factor. As a result of pitch down, the waveform's fundamental frequency is lowered, and all of its harmonics are shifted proportionally downward in frequency. This creates a deeper, lower-pitched sound. However, since the duration of the waveform remains the same, the waveform is stretched in time to accommodate the lower frequency.

5.5. def increase_amplitude():

The increase_amplitude() function increases the volume or loudness of the currently playing song in MP3 audio player. It does this by applying a gain value in decibels to the audio segment of the song. The currently playing song is increased by the specified gain in decibels. The waveform of the song is stretched out in the time domain, causing the signal to become louder. However, increasing the amplitude of an audio signal can also lead to clipping, which occurs when the amplitude of the waveform exceeds the maximum amplitude that can be represented. This can result in distortion and loss of information in the signal.

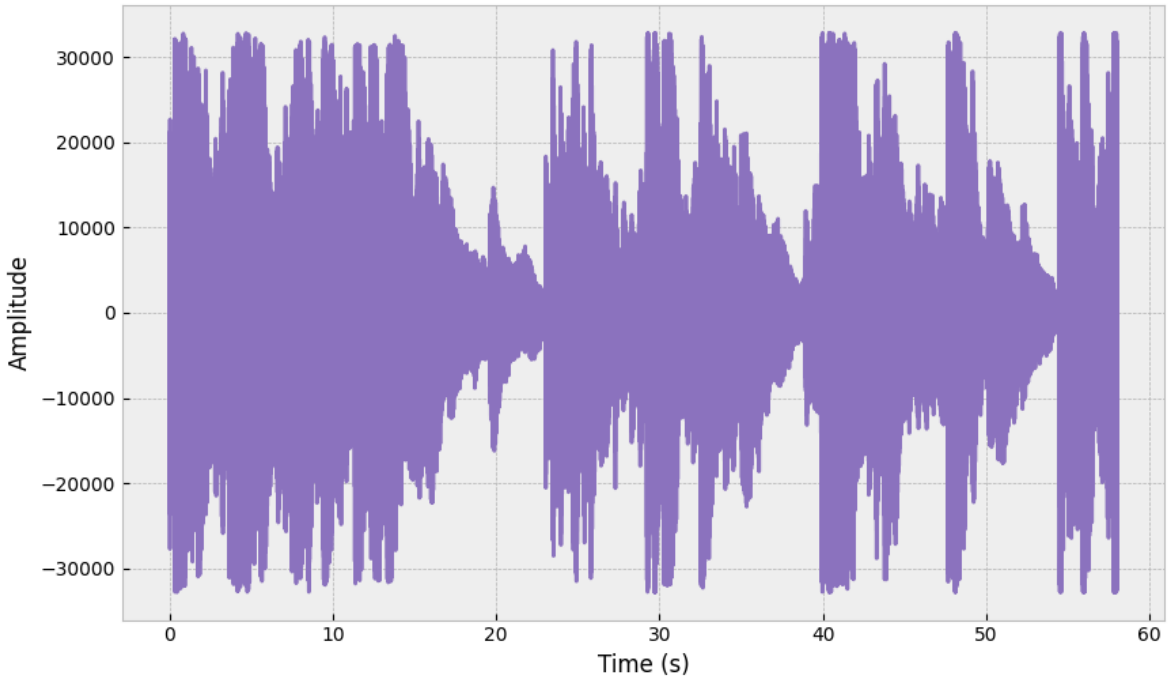

Figure7.Increase Amplitude Waveform

Increasing the amplitude of an audio waveform increases the volume or loudness of the sound by scaling up the waveform's amplitude, resulting in a more intense or louder sound. One consequence of increasing the amplitude is that the waveform may become distorted if the amplitude is increased too much. This is because the waveform's peaks may exceed the maximum amplitude that can be represented by the digital audio system, resulting in clipping or distortion.

5.6. def decrease_amplitude():

The decrease_amplitude() function decreases the volume or loudness of the currently playing song in MP3 audio player. It does this by applying a negative gain value to the audio segment of the song.The currently playing song is decreased by the specified gain in decibels.

The waveform of the song is compressed in the time domain, causing the signal to become quieter. Decreasing the amplitude of an audio signal does not typically lead to clipping, since reducing the amplitude of the waveform will not cause it to exceed the maximum amplitude that can be represented.

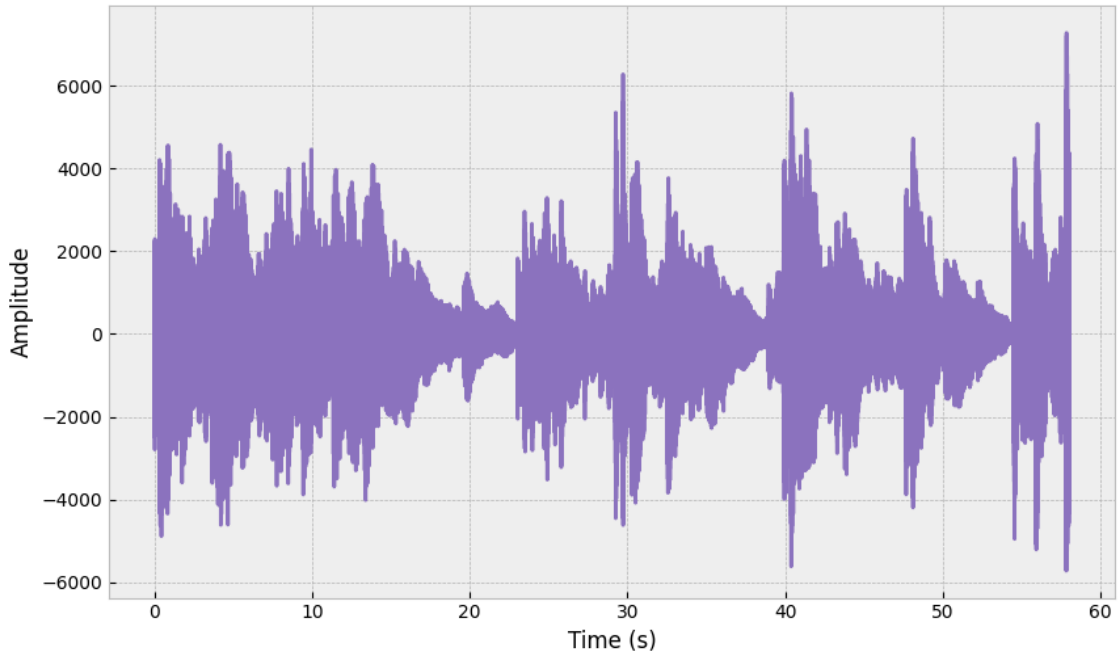

Figure8.Decrease Amplitude Waveform

Decreasing the amplitude of an audio waveform decreases the volume or loudness of the sound by scaling down the waveform's amplitude, resulting in a less intense or quieter sound. One consequence of decreasing the amplitude is that the waveform's signal-to-noise ratio may be reduced. This is because the noise floor of the audio signal remains the same, but the signal level is decreased, making the noise more noticeable.

5.7. def white_noise():

The white_noise() function adds white noise to a given audio signal by generating a random noise signal with the same standard deviation as the original signal, and then adding it to the original signal multiplied by a noise percentage factor. The add_white_noise() function uses the white_noise() function to add white noise to the current audio file in a playlist. It loads the audio signal and sample rate, applies the white_noise() function, writes the resulting noisy signal to a temporary file, updates the file name in the playlist, plays the noisy audio, and then restores the original file name in the playlist.

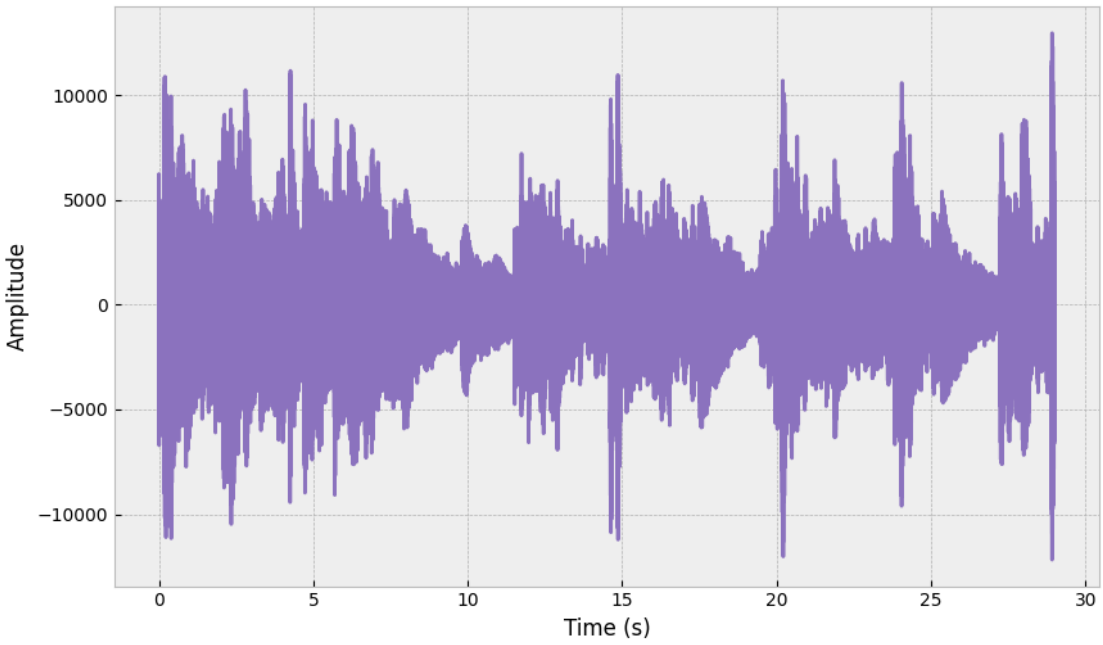

Figure9.White Noise Waveform

Adding white noise to an audio waveform results in a sound that contains equal energy at all frequencies, resulting in a hissing or shushing sound with no discernible pitch.As a result of adding white noise to an audio waveform, the waveform's amplitude is altered at each sample point by a random amount. This results in a sound that has a constant level of energy across all frequencies and that is perceived as a hissing or shushing sound.One consequence of adding white noise to an audio waveform is that it can mask other sounds or distort their frequency content. This is because the added noise can cover up or interfere with other sounds in the audio signal.

5.8. def noise_reduction():

The noise_reduction() function applies a noise reduction algorithm to the currently playing song in the playlist. It first applies a high-pass filter to remove low-frequency noise and a low-pass filter to remove high-frequency noise. It then normalizes the audio to improve the overall volume level. The resulting audio is saved as an MP3 file with the filename format "{current_song_filename}_reduced.mp3" and added to the playlist at the current song index. The function then plays the filtered song using play_music(). Finally, the original song is restored in the playlist.

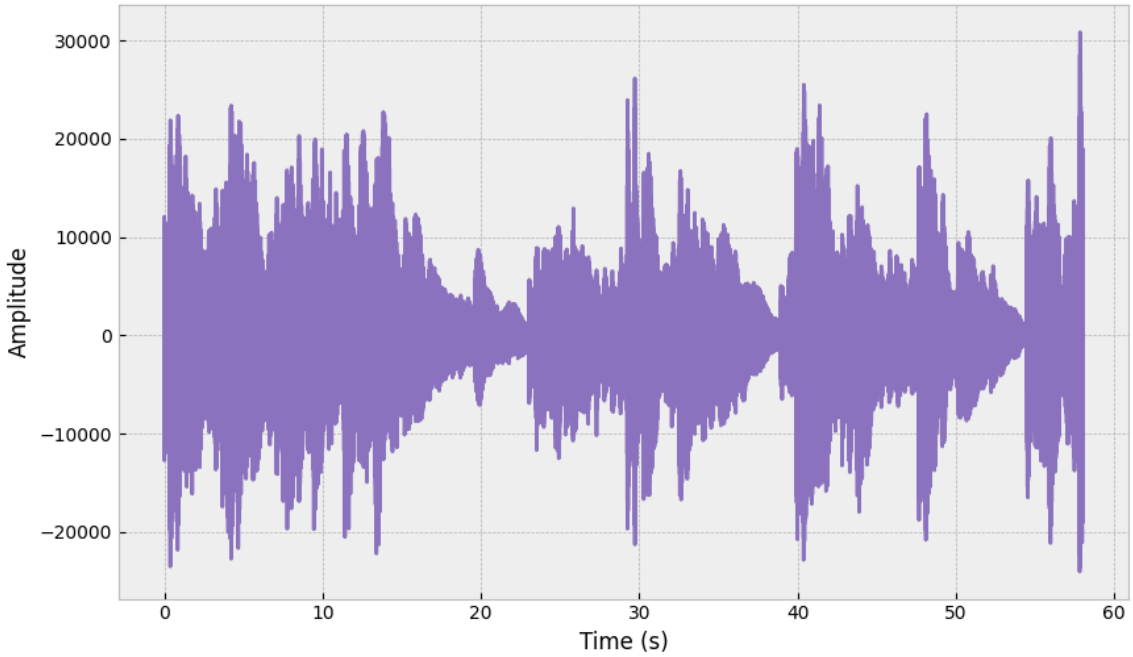

Figure10.Noise Reduce Waveform

When we apply noise reduction to an audio waveform, we remove or reduce unwanted noise from the signal, resulting in a cleaner and clearer sound. This means that the waveform's frequency content is altered by selectively attenuating certain frequencies or frequency ranges. One consequence of noise reduction is that it can also remove or attenuate some of the desired signal in the audio waveform, especially if the noise and signal are in the same frequency range. This can result in a loss of some of the desired audio content, such as high-frequency detail or harmonics.

5.9. def echo():

The echo() function creates a delayed copy of the audio signal using a fractional delay of 0.3 seconds (specified in samples as fractional_delay = int(0.3 * sr)) and then adds the delayed copy to the original signal with a gain of 0.5 (echoed_signal = y + 0.5 * delayed_signal),When an audio signal is processed with an echo effect, a delayed version of the signal is added to the original signal, resulting in a repeated sound that gradually fades away over time. The delay time determines how long the repeated sound will take to occur after the original sound, and the decay rate determines how quickly the repeated sound will fade away. An echo effect in audio processing refers to the repetition of a sound after a short delay, which creates a perception of space and depth in the sound. The delay time, decay rate, and amplitude of the echo can be adjusted to create the desired effect.

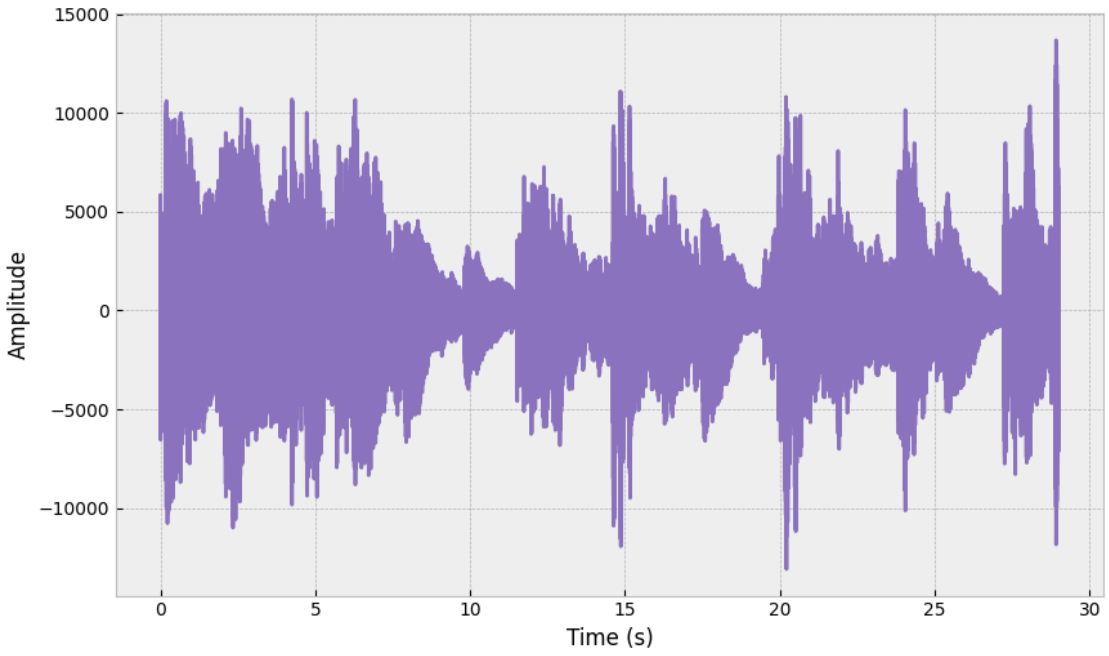

Figure11.Echo Waveform

This results in a new waveform that contains the original audio signal, followed by a repetition of the signal that is shifted in time by the delay amount and has a lower amplitude due to the gain factor. The delay time and gain factor can be adjusted to create different echo effects, such as short or long echoes, multiple echoes, or reverberation.

5.10. def reverse_audio():

The reverse_audio() function reverses the playback of an audio file by extracting the audio data as a numpy array, reversing the order of the samples in the array, creating a new AudioSegment object with the reversed audio data, exporting the reversed audio data to a temporary file, and playing the temporary file. The function then updates the playlist to restore the original audio file to its original position in the playlist.

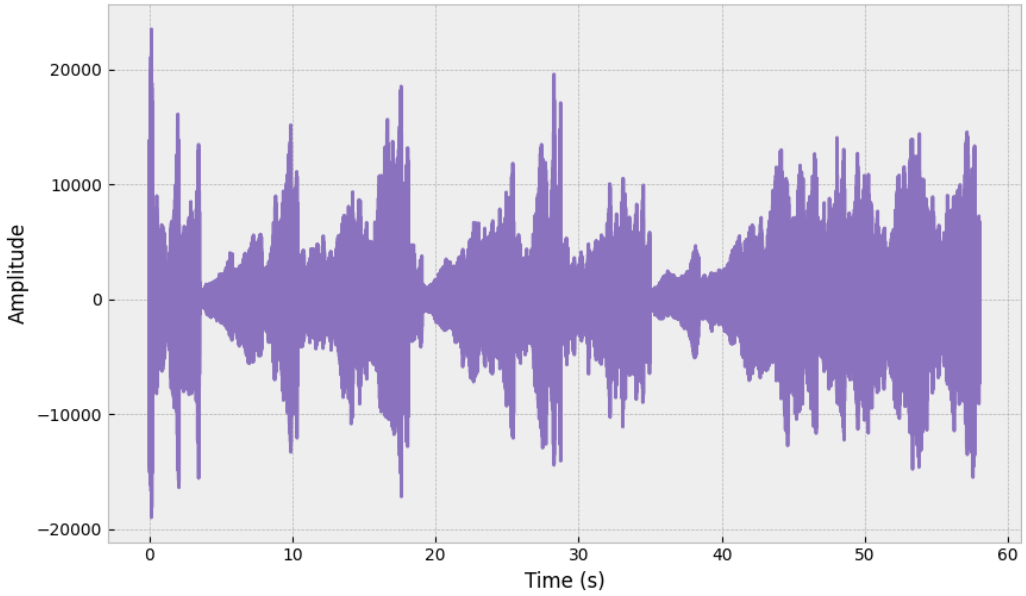

Figure12.Reverse Waveform

When we apply the reverse operation to a waveform, we simply flip the waveform horizontally, so that the samples that were originally at the end of the waveform are now at the beginning, and vice versa. This means that the waveform is effectively inverted, producing a mirror image of the original waveform

5.11. def add_sample():

the add_sampling() function resamples the audio to a lower sample rate of 8 kHz. This means that the number of samples per second is reduced from the original sample rate of the audio file to 8000 samples per second. Resampling the audio file to a lower sample rate can reduce the file size, which can be useful for storage or transmission purposes. However, the lower sample rate can also result in a loss of high-frequency content and a reduction in overall audio quality. Additionally, resampling can introduce artifacts such as aliasing, which is a distortion caused by the inability of the lower sample rate to accurately represent high-frequency components of the audio signal.

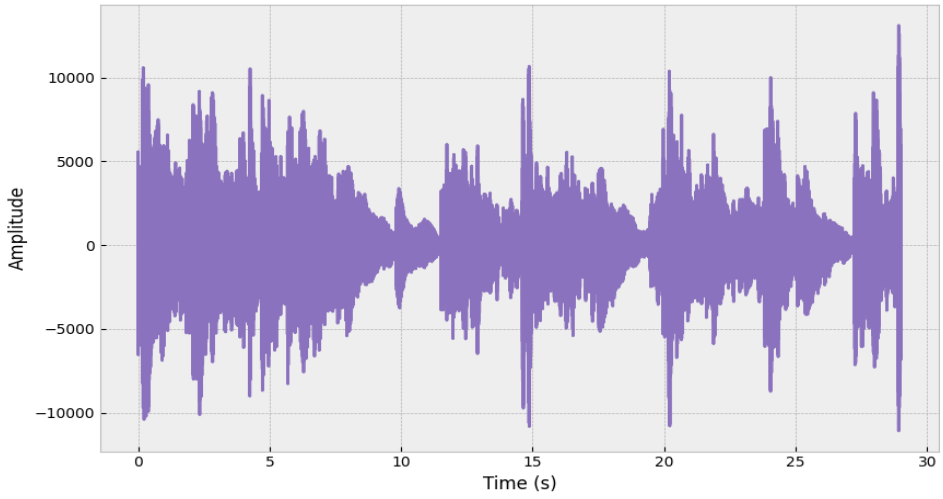

Figure13.Sampling Waveform

When we reduce the sampling rate of an audio signal, it can result in the following changes: Reduction in the number of samples, resulting in a loss of detail and resolution, especially in complex and high-frequency parts of the waveform. Changes in the frequency content, potentially resulting in the loss or aliasing of high-frequency content and the introduction of distortion and artifacts in the waveform. Changes in the amplitude and dynamic range of the waveform due to quantization noise introduced by reducing the bit depth. Changes in the timing and phase relationships between different parts of the waveform, potentially affecting the perceived rhythm and timing of the audio.

5.12. def delay():

The delay() function adds a delay of 10 seconds to the currently playing song in the playlist and then resumes playing the song from the beginning.

5.13. def text_to_speech():

Defines a function called text_to_speech(), which takes user input from a text entry widget and converts it into speech using Google's Text-to-Speech (gTTS) API. The converted speech is then saved as an MP3 file with the first 5 characters of the input text as the file name. The file path is then added to a playlist, which is updated in listbox widget. If the playlist contains only one file, the play_music() function is called to play the speech.

6. Conclusion

TuneyTones is an MP3 audio player project that is helpful to learn about digital signal processing, audio file formats, and graphical user interfaces. To create TuneyTones in Python, we used a combination of libraries and frameworks, such as Pygame for audio playback, Pydub, Librosa and numpy for audio file manipulation, Matplotlib for plotting waveform, CustomTkinter and Tkinter for building the graphical user interface.

Some of the key features of TuneyTones:

- Loading and playing MP3 audio files
- Providing controls for play, pause, stop, resume, next song and previous song.
- Implementing features such as volume control, noise reduction, and audio effects such as pitch shift, speed adjustment, increasing and decreasing amplitude, applying white noise or delaying a sound to start after 10 seconds.
- Allowing users to create playlists and manage their music library.
- Building a user-friendly graphical interface with buttons, sliders, and visual feedback, such as a waveform display
- Text-to-speech converter

Overall, TuneyTones audio player is a way to explore the world of digital audio processing and user interface design in Python.

References

- https://pypi.org/
- https://www.tutorialspoint.com/how-to-plot-a-wav-file-using-matplotlib
- https://customtkinter.tomschimansky.com/documentation/
- https://towardsdatascience.com/get-to-know-audio-feature-extraction-in-python-a499fdaefe42
- https://daehnhardt.com/blog/2023/03/05/python-audio-signal-processing-with-librosa/
- https://www.i2symbol.com/
- https://youtu.be/4rzpMA6CUPg
- https://youtu.be/umAXGVzVvwQ
- https://www.youtube.com/watch?v=kNgaO1etjXM&feature=youtu.be