In [49]:
```python
# Data Manipulation and Analysis
import pandas as pd
import numpy as np

# Visualization Libraries
import matplotlib.pyplot as plt
import seaborn as sns

# File Handling
from glob import glob

# Audio Processing
import librosa
import librosa.display

# Interactive Audio Playback
import IPython.display as ipd

# Utility Functions
from itertools import cycle

# Set Plotting Theme
sns.set_theme(style="white", palette=None)

# Define Color Palette and Cycle for Visualizations
color_pal = plt.rcParams["axes.prop_cycle"].by_key()["color"]
color_cycle = cycle(color_pal)

# SciKIt libraries(this is what your machine learning thing is derived from
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
```

In [50]:
```python
audio_file1 = glob("/Users/Parag/Documents/TMRT_data/noteA.mp3")
audio_file2 = glob("/Users/Parag/Documents/TMRT_data/noteB.mp3")
```

In [53]:
```python
#audio_file1
print("Play the A note on the tuning fork at 440Hz")
ipd.Audio(audio_file1[0])
```

```
Play the A note on the tuning fork at 440Hz
```

Out[53]: 0:00 / 0:20

In [55]:
```python
#audio_file2

print("Play the B note on the tuning fork at 440Hz")
ipd.Audio(audio_file2[0])
```

```
Play the B note on the tuning fork at 440Hz
```

Out[55]: 0:00 / 0:20

In [57]:
```python
#next Loading the audio files data
#first for the first audio file
Atime , Asr = librosa.load(audio_file1[0])
#displaying the first ten samples
print(f"The first 10 samples of note A {Atime[:10]}")
print(f"The shape of Atime {Atime.shape}")
print(f"The sample rate for the A note file {Asr}")
```

```
The first 10 samples of note A [-1.1166938e-23  9.0989867e-24 -4.4460958e-24  1.9438744e-23
 -4.6528909e-25 -1.6130022e-23 -6.6174449e-24 -6.4106497e-24
  8.4786013e-24  2.8537731e-23]
The shape of Atime (443904,)
The sample rate for the A note file 22050
```

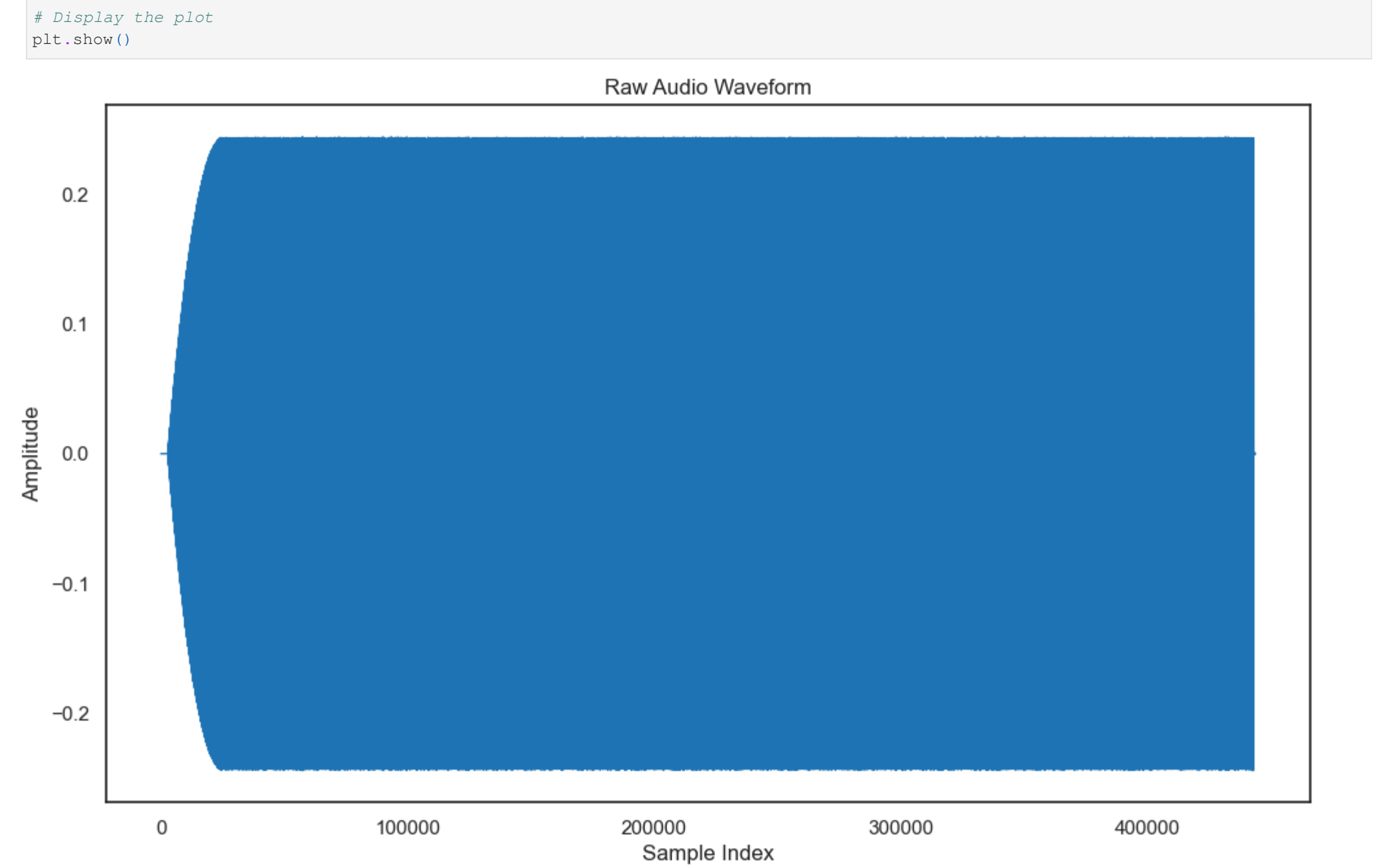
In [59]:
```python
#for the second audio file
Btime,Bsr = librosa.load(audio_file2[0])
#displaying the first ten samples,shape and samplerate
print(f"The first 10 samples of note B{Btime[:10]}")
print(f"The shape of Btime {Btime.shape}")
print(f"The sample rate for the B note file {Bsr} \n")
print("I looked up why im gettin a bunch of zeros in the first then samples -> Theres this thing called a DC offset which gets added
```

```
The first 10 samples of note B[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
The shape of Btime (443904,)
The sample rate for the B note file 22050

I looked up why im gettin a bunch of zeros in the first then samples -> Theres this thing called a DC offset which gets added to a fi
les beginning , librosa counteracts by substracting the audio file with the mean of the samples and so we get a value that is very cl
ose to zero which gets approximated to zero
```

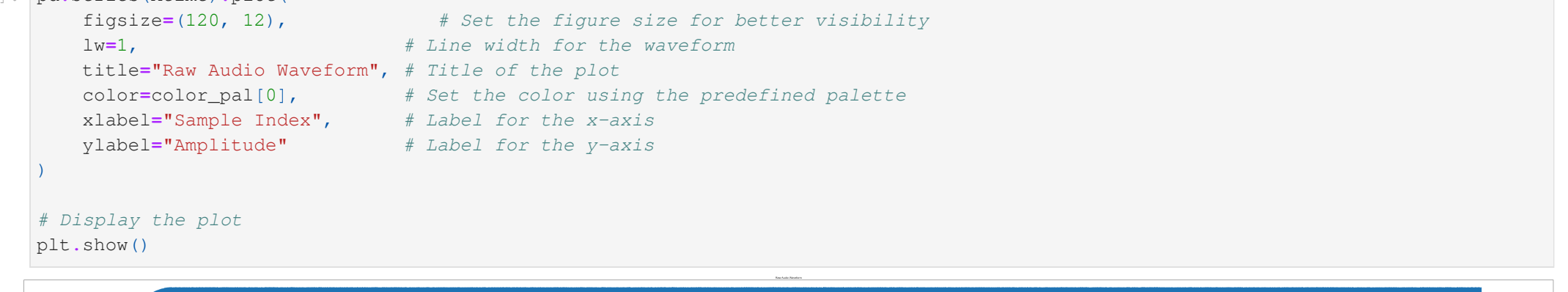
In [61]:
```python
#Plotting the raw audiowaveform for the A note
# Plot the raw audio waveform
pd.Series(Atime).plot(
    figsize=(12, 7),              # Set the figure size for better visibility
    lw=1,                        # Line width for the waveform
    title="Raw Audio Waveform", # Title of the plot
    color=color_pal[0],          # Set the color using the predefined palette
    xlabel="Sample Index",       # Label for the x-axis
    ylabel="Amplitude"           # Label for the y-axis
)

# Display the plot
plt.show()
```



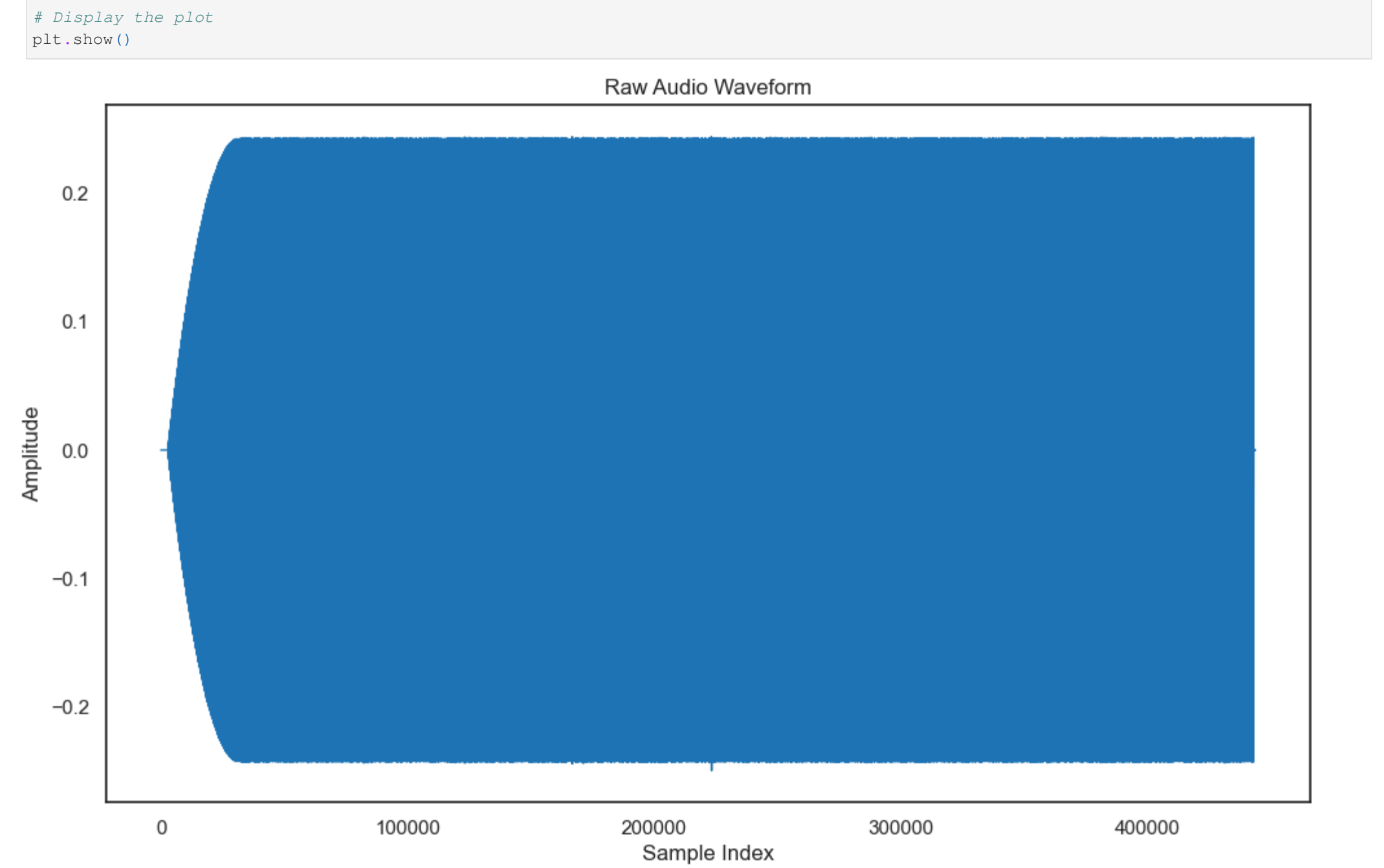
In [62]:
```python
pd.Series(Atime).plot(
    figsize=(120, 12),              # Set the figure size for better visibility
    lw=1,                        # Line width for the waveform
    title="Raw Audio Waveform", # Title of the plot
    color=color_pal[0],          # Set the color using the predefined palette
    xlabel="Sample Index",       # Label for the x-axis
    ylabel="Amplitude"           # Label for the y-axis
)

# Display the plot
plt.show()
```



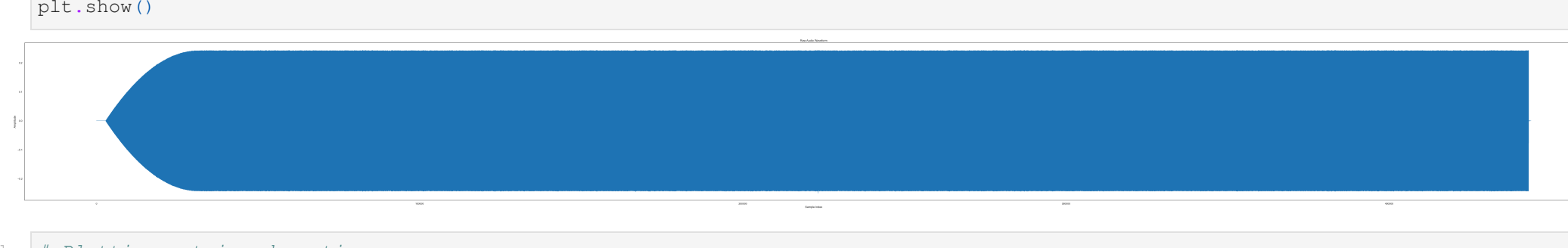
In [63]:
```python
#for the B note
pd.Series(Btime).plot(
    figsize=(12, 7),              # Set the figure size for better visibility
    lw=1,                        # Line width for the waveform
    title="Raw Audio Waveform", # Title of the plot
    color=color_pal[0],          # Set the color using the predefined palette
    xlabel="Sample Index",       # Label for the x-axis
    ylabel="Amplitude"           # Label for the y-axis
)

# Display the plot
plt.show()
```



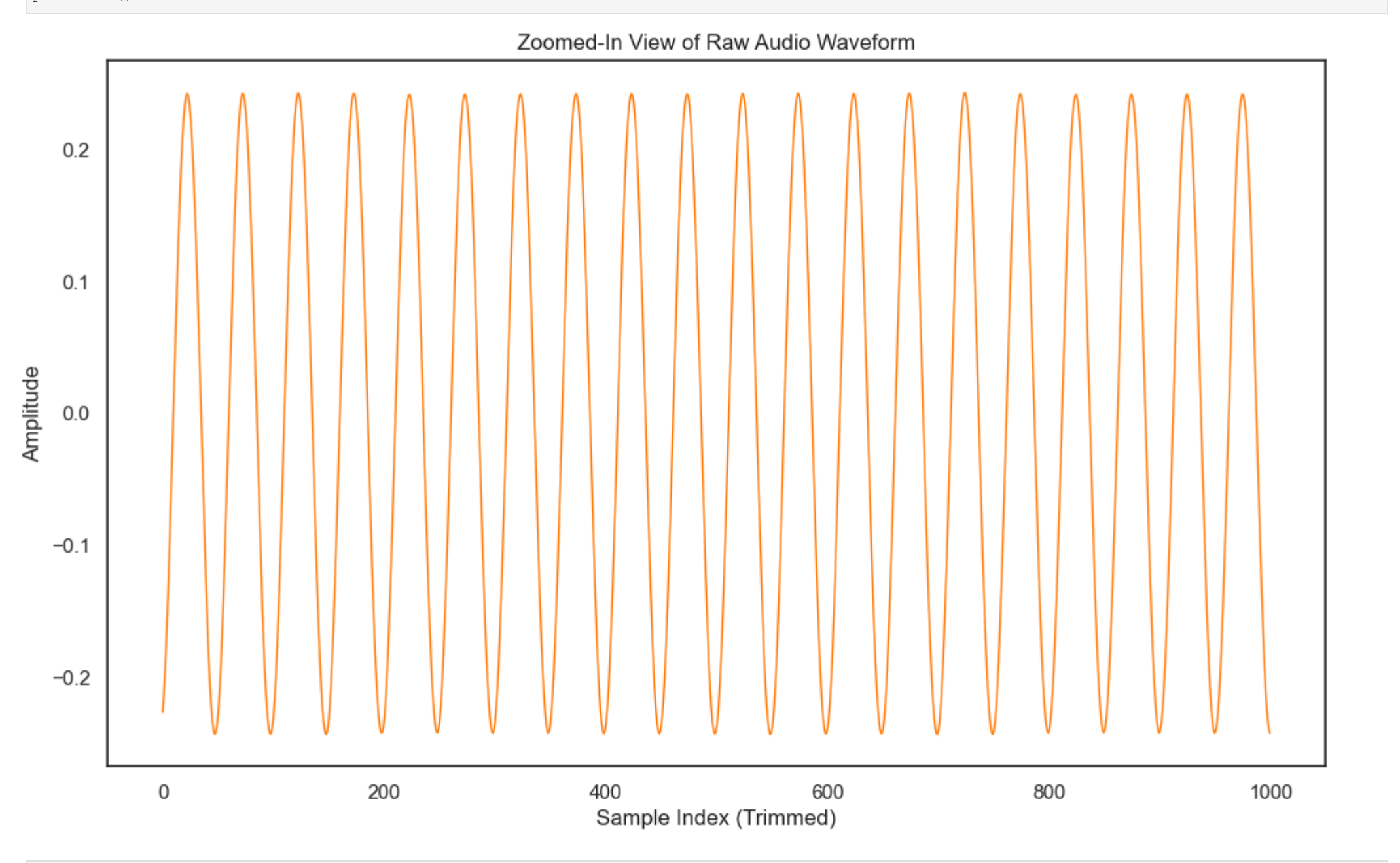
In [65]:
```python
pd.Series(Btime).plot(
    figsize=(120, 12),              # Set the figure size for better visibility
    lw=1,                        # Line width for the waveform
    title="Raw Audio Waveform", # Title of the plot
    color=color_pal[0],          # Set the color using the predefined palette
    xlabel="Sample Index",       # Label for the x-axis
    ylabel="Amplitude"           # Label for the y-axis
)

# Display the plot
plt.show()
```



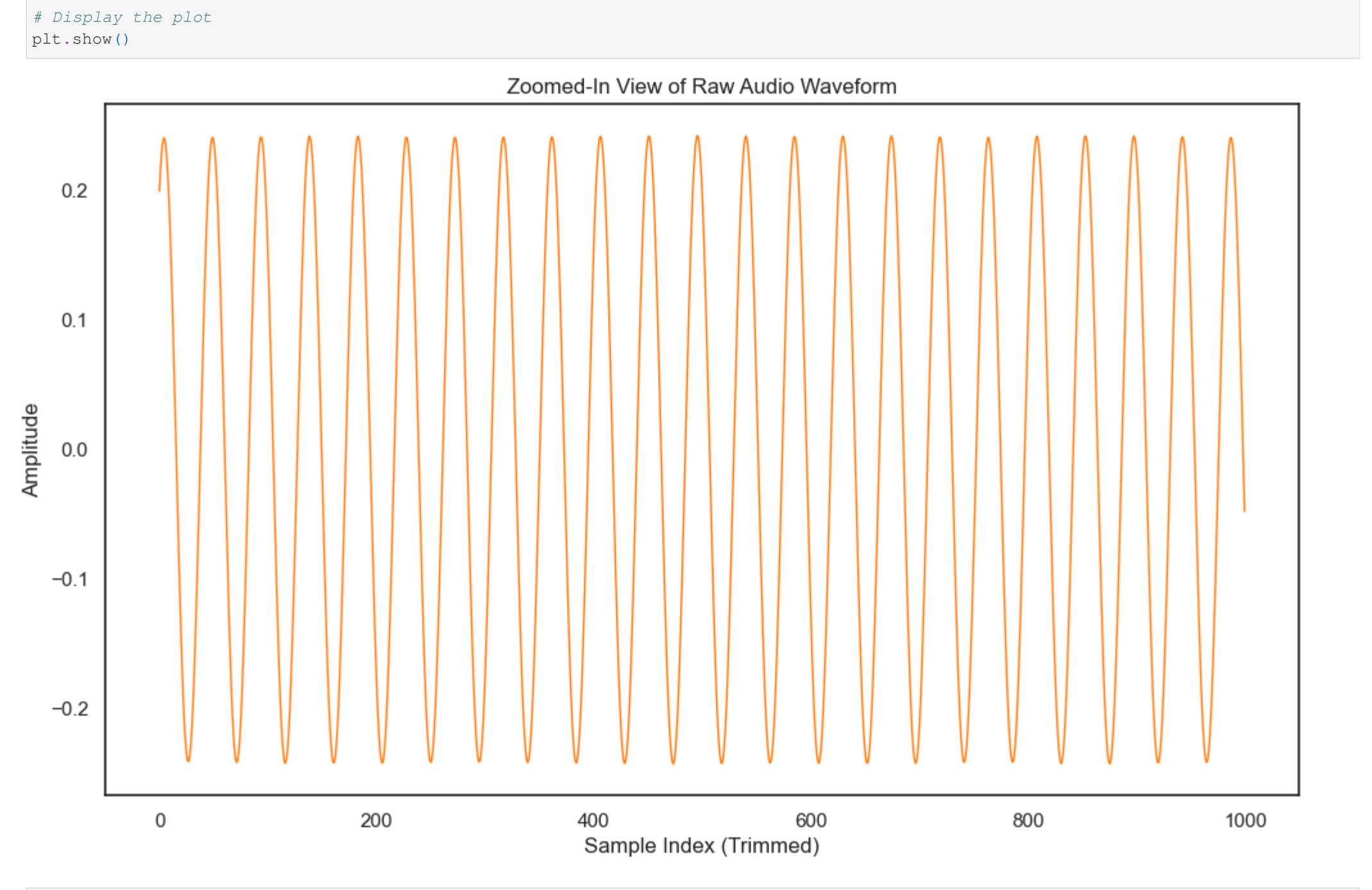
In [66]:
```python
# Plotting a trimmed section
pd.Series(Atime[30000:31000]).plot(
    figsize=(12, 7),               # Set figure size for better visibility
    lw=1,                          # Line width for the waveform
    title="Zoomed-In View of Raw Audio Waveform",  # Descriptive title for clarity
    color=color_pal[1],            # Use a consistent color palette for aesthetics
    xlabel="Sample Index (Trimmed)", # X-axis label indicating the zoomed-in range
    ylabel="Amplitude"             # Y-axis label for amplitude values
)

# Display the plot
plt.show()
```



In [67]:
```python
# Plotting a trimmed section
pd.Series(Btime[30000:31000]).plot(
    figsize=(12, 7),               # Set figure size for better visibility
    lw=1,                          # Line width for the waveform
    title="Zoomed-In View of Raw Audio Waveform",  # Descriptive title for clarity
    color=color_pal[1],            # Use a consistent color palette for aesthetics
    xlabel="Sample Index (Trimmed)", # X-axis label indicating the zoomed-in range
    ylabel="Amplitude"             # Y-axis label for amplitude values
)

# Display the plot
plt.show()
```



In [68]:
```python
#now taking the short time fourier transforms of both the notes
#this is to convert the graph that we have from a time domain to a frequency domain to then further use in a spectrograph
Astft = librosa.stft(Atime)
Bstft = librosa.stft(Btime)
# conveting both the fourier transforms into decibels (logarithmic scael)->this is more closer to what humans hear and understand
Astft_db = librosa.amplitude_to_db(np.abs(Astft),ref = np.max)
Bstft_db = librosa.amplitude_to_db(np.abs(Bstft),ref = np.max)
print(f"printing the shape of the A notes spectrogram {Astft_db.shape}")
print(f"printing the shape of the B notes spectrogram {Bstft_db.shape}")
# (n_freq_bins, n_time_frames) ->Shape
```

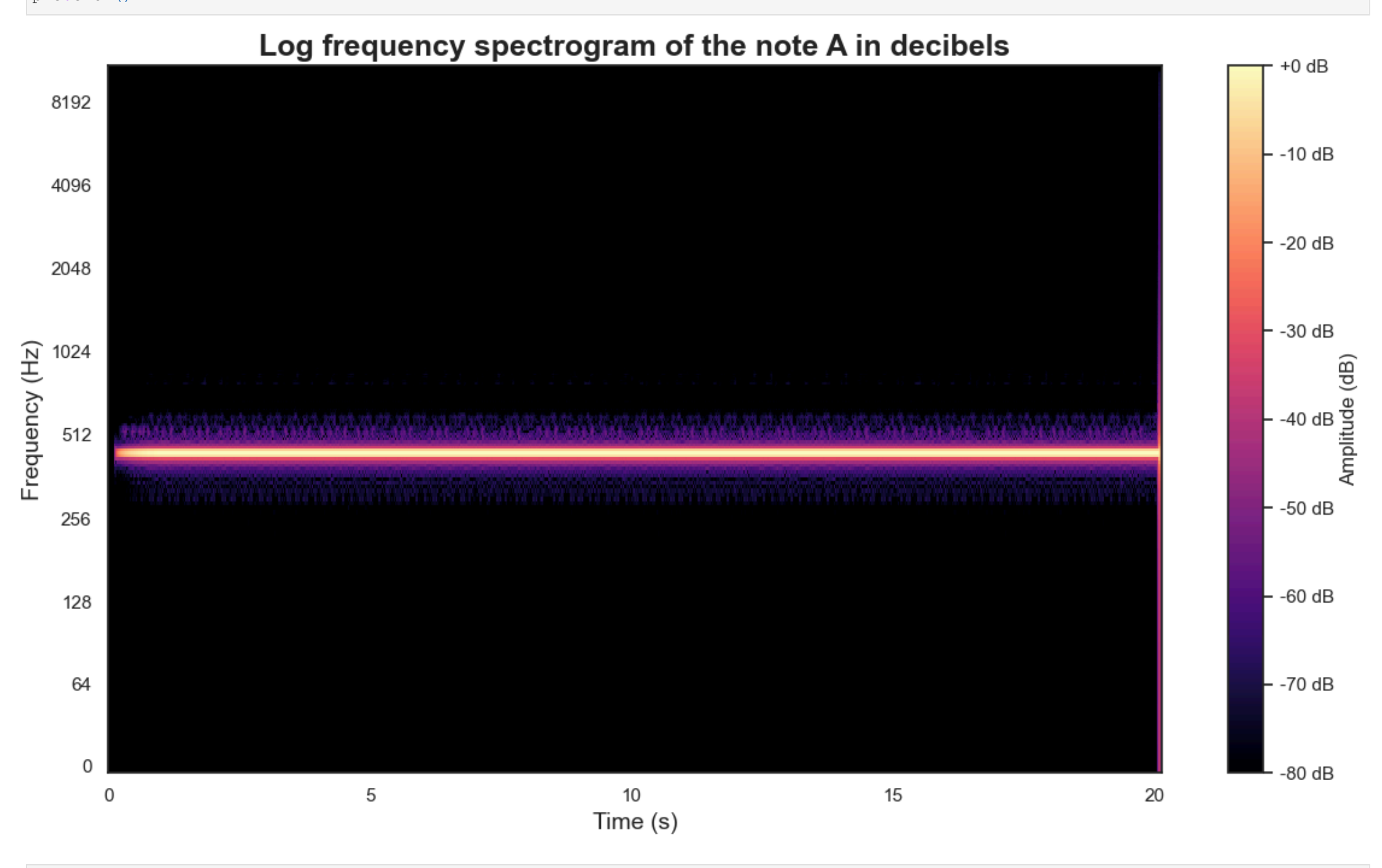
```
printing the shape of the A notes spectrogram (1025, 868)
printing the shape of the B notes spectrogram (1025, 868)
```

In [69]:
```python
# plotting the log frequency spectrogram for the note A

fig ,ax = plt.subplots(figsize=(12,7))
img = librosa.display.specshow(Astft_db, x_axis = "time",y_axis = "log" , sr = Asr,cmap = "magma" , ax=ax)
ax.set_title("Log frequency spectrogram of the note A in decibels", fontsize=18, fontweight='bold')
ax.set_xlabel("Time (s)", fontsize=14)
ax.set_ylabel("Frequency (Hz)", fontsize=14)

# Add a color bar to indicate decibel levels
cbar = fig.colorbar(img, ax=ax, format="%+2.0f dB")
cbar.set_label("Amplitude (dB)", fontsize=12)

# Show the plot
plt.tight_layout()  # Adjust layout to prevent clipping
plt.show()
```

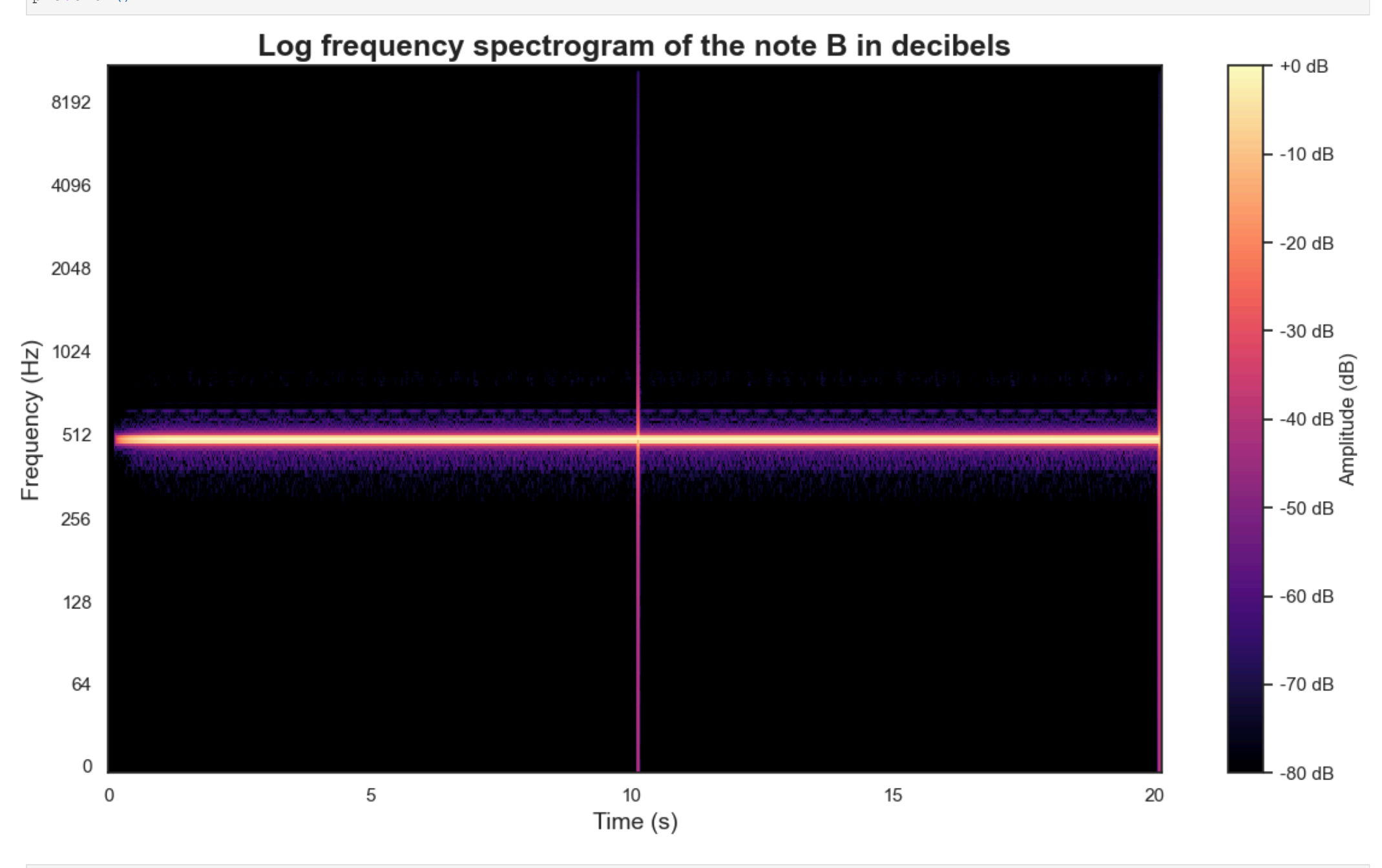


In [70]:
```python
# plotting the log frequency spectrogram for the note B

fig ,ax = plt.subplots(figsize=(12,7))
img = librosa.display.specshow(Bstft_db, x_axis = "time",y_axis = "log" , sr = Asr,cmap = "magma" , ax=ax)
ax.set_title("Log frequency spectrogram of the note B in decibels", fontsize=18, fontweight='bold')
ax.set_xlabel("Time (s)", fontsize=14)
ax.set_ylabel("Frequency (Hz)", fontsize=14)

# Add a color bar to indicate decibel levels
cbar = fig.colorbar(img, ax=ax, format="%+2.0f dB")
cbar.set_label("Amplitude (dB)", fontsize=12)

# Show the plot
plt.tight_layout()  # Adjust layout to prevent clipping
plt.show()
```



In [71]:
```python
ipd.Image(url='https://wikimedia.org/api/rest_v1/media/math/render/svg/102948b0039fc05f0516b9f1e8e3f452062d3be6')
```

Out[71]:

$$m = 2595\log_{10}\left(1+\frac{f}{700}\right).$$

In [72]:
```python
print("A Mel Spectrogram is a type of spectrogram where the frequency axis is scaled according to the Mel scale, which aligns closely
```

```
A Mel Spectrogram is a type of spectrogram where the frequency axis is scaled according to the Mel scale, which aligns closely with t
he way humans perceive sound and pitch. It is widely used in audio processing and machine learning, particularly in tasks like speech
recognition, music analysis, and audio classification.
```

In [73]:
```python
A_mel_spectrogram_shape =librosa.feature.melspectrogram(y=Atime ,sr =Asr,n_mels = 128)
print(f"mel spectrograms shape for note A : {A_mel_spectrogram_shape}")
```

```
mel spectrograms shape for note A : [[7.0717218e-31 6.8558216e-21 1.5559253e-14 ... 1.8056446e-01
  2.1694279e-01 9.6334480e-03]
 [4.3253702e-31 7.2070235e-21 2.7842371e-14 ... 1.8755974e-01
  2.2593592e-01 1.0065172e-02]
 [1.8615114e-31 6.5560902e-21 1.9178788e-14 ... 1.9394916e-01
  2.3142473e-01 1.0027794e-02]
 ...
 [1.4668376e-30 8.8734677e-21 2.3761473e-14 ... 6.6056702e-05
  7.7382079e-05 3.2629134e-06]
 [1.6819144e-30 1.5885492e-20 1.7985261e-14 ... 3.8485075e-05
  4.5087931e-05 1.9151814e-06]
 [3.5028750e-30 1.8873335e-21 7.5541283e-16 ... 3.4517634e-06
  4.0610471e-06 1.7709561e-07]]
```

In [75]:
```python
B_mel_spectrogram_shape =librosa.feature.melspectrogram(y=Btime ,sr =Bsr,n_mels = 128)
print(f"mel spectrograms shape for note B : {B_mel_spectrogram_shape}")
```

```
mel spectrograms shape for note B : [[4.8042363e-38 1.2066133e-34 3.8391848e-22 ... 1.5539832e-01
  1.8000789e-01 7.4191825e-03]
 [3.7043558e-38 1.1818378e-34 1.4688693e-21 ... 1.6528858e-01
  1.9109669e-01 7.8213010e-03]
 [2.4573080e-38 1.5024826e-34 2.0297249e-21 ... 1.6981709e-01
  1.9681512e-01 8.1464257e-03]
 ...
 [8.6387915e-38 6.2689509e-34 3.2417341e-21 ... 7.4305572e-05
  8.7448920e-05 3.7238660e-06]
 [2.1167743e-37 1.5469581e-33 2.9109984e-21 ... 3.8816263e-05
  4.5599532e-05 1.9473023e-06]
 [3.9284979e-37 4.1980760e-34 1.9522087e-22 ... 3.4946554e-06
  4.0923733e-06 1.7744749e-07]]
```

In [76]:
```python
# Convert the Mel Spectrogram to Decibels (Logarithmic Scale)
A_db_mel = librosa.amplitude_to_db(A_mel_spectrogram_shape, ref=np.max)

# Display Information
print(f"Mel Spectrogram (in dB) Shape for A: {A_db_mel.shape}")
print(f"Decibel Range: {A_db_mel.min():.2f} dB to {A_db_mel.max():.2f} dB")
```

```
Mel Spectrogram (in dB) Shape for A: (128, 868)
Decibel Range: -80.00 dB to 0.00 dB
```

In [77]:
```python
# Convert the Mel Spectrogram to Decibels (Logarithmic Scale)
B_db_mel = librosa.amplitude_to_db(B_mel_spectrogram_shape, ref=np.max)

# Display Information
print(f"Mel Spectrogram (in dB) Shape: {B_db_mel.shape}")
print(f"Decibel Range: {B_db_mel.min():.2f} dB to {B_db_mel.max():.2f} dB")
```

```
Mel Spectrogram (in dB) Shape: (128, 868)
Decibel Range: -80.00 dB to 0.00 dB
```

In [79]:
```python
#plotting the log scale mel spectrogram for the A note

fig,ax = plt.subplots(figsize = (12,7))
img = librosa.display.specshow(A_db_mel , sr =Asr
                               ,x_axis = 'time',
                               y_axis = 'mel',
                               cmap = 'magma',
                               ax=ax)
ax.set_title("Mel Spectrogram for A (Log Scale in Decibels)", fontsize=18, fontweight="bold")
ax.set_xlabel("Time",fontsize = 14)
ax.set_ylabel("Mel Frequency (Hz)", fontsize=14)

# Add a Color Bar to Show the Decibel Range
cbar = fig.colorbar(img, ax=ax, format="%+2.0f dB")
cbar.set_label("Amplitude (dB)", fontsize=12)

# Adjust Layout for Better Visualization
plt.tight_layout()
plt.show()
```

In [80]:
```python
#plotting the log scale mel spectrogram for the B note

fig,ax = plt.subplots(figsize = (12,7))
img = librosa.display.specshow(B_db_mel , sr =Asr
                               ,x_axis = 'time',
                               y_axis = 'mel',
                               cmap = 'magma',
                               ax=ax)
ax.set_title("Mel Spectrogram for B (Log Scale in Decibels)", fontsize=18, fontweight="bold")
ax.set_xlabel("Time",fontsize = 14)
ax.set_ylabel("Mel Frequency (Hz)", fontsize=14)

# Add a Color Bar to Show the Decibel Range
cbar = fig.colorbar(img, ax=ax, format="%+2.0f dB")
cbar.set_label("Amplitude (dB)", fontsize=12)

# Adjust Layout for Better Visualization
plt.tight_layout()
plt.show()
```

In [81]:
```python
print("A note on decibels: The scale is a relative scale, 0dbs is the minimum the human ear can hear,-ve decibels indicate that the s
```

```
A note on decibels: The scale is a relative scale, 0dbs is the minimum the human ear can hear,-ve decibels indicate that the sound is
very little in amplitude and not a negetive sound(because ofc it isnt)
 also in digital systems 0db is the loudest sound you can get before any distortion is induced into the audio
```

In [82]:
```python
print("the orignal plan was to get multiple of these files that carried note sounds and base my classification on that but i dont hav
```

```
the orignal plan was to get multiple of these files that carried note sounds and base my classification on that but i dont have that
much of storage on my system
 so im tryna create snippets of the two audio files and feed that to the dataset
```

In [104…
```python
snippet_duration = 3  # seconds
samples_per_snippet_A = int(Asr * snippet_duration)
samples_per_snippet_B = int(Bsr * snippet_duration)


snippets_A = [Atime[i:i + samples_per_snippet_A] for i in range(0, len(Atime), samples_per_snippet_A) if len(Atime[i:i + samples_per_
snippets_B = [Btime[i:i + samples_per_snippet_B] for i in range(0, len(Btime), samples_per_snippet_B) if len(Btime[i:i + samples_per_

print(f"Created {len(snippets_A)} snippets for Note A.")
print(f"Created {len(snippets_B)} snippets for Note B.")


# Extract MFCC features from each snippet.
features_A = [librosa.feature.mfcc(y=snippet, sr=Asr, n_mfcc=13).mean(axis=1) for snippet in snippets_A]
features_B = [librosa.feature.mfcc(y=snippet, sr=Bsr, n_mfcc=13).mean(axis=1) for snippet in snippets_B]


# Combine the features and create corresponding labels.
X = np.array(features_A + features_B)
y = np.array([0] * len(features_A) + [1] * len(features_B))


# Split the data so we can test the model's performance on unseen data.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


# It's important to scale the features so the model can train properly.
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


# We'll use a simple K-Nearest Neighbors (KNN) classifier.
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train_scaled, y_train)


# Make predictions and check the model's accuracy.
y_pred = model.predict(X_test_scaled)
print("\nModel Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=['Note A', 'Note B']))


# Let's try to classify a snippet from the original files.
new_snippet_path = audio_file1[0]  # Or audio_file2[0]
new_snippet, _ = librosa.load(new_snippet_path, duration=3) # Load just a 3-second snippet

# Process the new snippet in the same way as the training data.
new_snippet_features = librosa.feature.mfcc(y=new_snippet, sr=Asr, n_mfcc=13).mean(axis=1)
new_snippet_features = new_snippet_features.reshape(1, -1) # Reshape for the model

# Scalin the new snippet's features using the same scaler
new_snippet_scaled = scaler.transform(new_snippet_features)

# Make a prediction
prediction = model.predict(new_snippet_scaled)
predicted_class = "Note A" if prediction[0] == 0 else "Note B"

print(f"\nPrediction for the new snippet: {predicted_class}")
ipd.Audio(new_snippet, rate=Asr)
```

```
Created 6 snippets for Note A.
Created 6 snippets for Note B.

Model Evaluation:
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

      Note A       1.00      1.00      1.00         1
      Note B       1.00      1.00      1.00         3

    accuracy                           1.00         4
   macro avg       1.00      1.00      1.00         4
weighted avg       1.00      1.00      1.00         4


Prediction for the new snippet: Note A
```

Out[104…: 0:00 / 0:03

In [106…
```python
#checking with the second file
new_snippet_path = audio_file2[0]  # Or audio_file2[0]
new_snippet, _ = librosa.load(new_snippet_path, duration=3) # Load just a 3-second snippet

# Process the new snippet in the same way as the training data.
new_snippet_features = librosa.feature.mfcc(y=new_snippet, sr=Asr, n_mfcc=13).mean(axis=1)
new_snippet_features = new_snippet_features.reshape(1, -1) # Reshape for the model

# Scalin the new snippet's features using the same scaler
new_snippet_scaled = scaler.transform(new_snippet_features)

# Make a prediction
prediction = model.predict(new_snippet_scaled)
predicted_class = "Note A" if prediction[0] == 0 else "Note B"

print(f"\nPrediction for the new snippet: {predicted_class}")
ipd.Audio(new_snippet, rate=Asr)
```

```
Prediction for the new snippet: Note B
```

Out[106…: 0:00 / 0:03

In [ ]: