To process sound signals, transfer them to the frequency domain, apply different filters, and hear the filtered sounds, follow these steps:

1. **Read the Sound Signal:**
   - Use a sound processing library to read the audio file and extract the signal.

2. **Transform to Frequency Domain:**
   - Apply the Fast Fourier Transform (FFT) to convert the time-domain signal to the frequency domain.

3. **Apply Filters:**
   - Implement low-pass, high-pass, and band-pass filters in the frequency domain by modifying the frequency components.

4. **Transform Back to Time Domain:**
   - Apply the Inverse Fast Fourier Transform (IFFT) to convert the filtered signal back to the time domain.

5. **Play the Filtered Sound:**
   - Use an audio library to play the modified sound.

```python
import numpy as np
from scipy.io import wavfile
from scipy.fft import fft, ifft
import matplotlib.pyplot as plt
import IPython.display as ipd

# Load the file from your device
from google.colab import files

uploaded = files.upload()

# Specify the uploaded file name here
filename = list(uploaded.keys())[0]

# Step 1: Read the Sound Signal
sample_rate, signal = wavfile.read(filename)

# Normalize the signal
signal = signal / np.max(np.abs(signal))

# Convert signal to 16-bit signed integers
signal = signal.astype(np.int16)

# Step 2: Transform to Frequency Domain
frequency_domain = fft(signal)

# Frequency axis
frequencies = np.fft.fftfreq(len(frequency_domain), 1/sample_rate)

# Function to apply a filter
def apply_filter(frequency_domain, filter_type, cutoff_low=None, cutoff_high=None):
    if filter_type == 'low_pass':
        frequency_domain[np.abs(frequencies) > cutoff_low] = 0
    elif filter_type == 'high_pass':
        frequency_domain[np.abs(frequencies) < cutoff_high] = 0
    elif filter_type == 'band_pass':
        frequency_domain[(np.abs(frequencies) < cutoff_low) | (np.abs(frequencies) > cutoff_high)] = 0
    return frequency_domain

# Step 3: Apply Filters
# Low-pass filter (cutoff at 1000 Hz)
low_pass_filtered = apply_filter(frequency_domain.copy(), 'low_pass', cutoff_low=1000)

# High-pass filter (cutoff at 1000 Hz)
high_pass_filtered = apply_filter(frequency_domain.copy(), 'high_pass', cutoff_high=1000)

# Band-pass filter (1000 Hz to 2000 Hz)
band_pass_filtered = apply_filter(frequency_domain.copy(), 'band_pass', cutoff_low=1000, cutoff_high=2000)
```

```
# Step 4: Transform Back to Time Domain
low_pass_time_domain = ifft(low_pass_filtered)
high_pass_time_domain = ifft(high_pass_filtered)
band_pass_time_domain = ifft(band_pass_filtered)

# Step 5: Play the Filtered Sound
print("Original Sound:")
ipd.display(ipd.Audio(signal, rate=sample_rate))

print("Low-pass Filtered Sound:")
ipd.display(ipd.Audio(np.real(low_pass_time_domain), rate=sample_rate))

print("High-pass Filtered Sound:")
ipd.display(ipd.Audio(np.real(high_pass_time_domain), rate=sample_rate))

print("Band-pass Filtered Sound:")
ipd.display(ipd.Audio(np.real(band_pass_time_domain), rate=sample_rate))

# Optional: Plot the original and filtered signals in frequency domain
plt.figure(figsize=(14, 7))
plt.subplot(2, 2, 1)
plt.plot(frequencies, np.abs(frequency_domain))
plt.title('Original Signal')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')

plt.subplot(2, 2, 2)
plt.plot(frequencies, np.abs(low_pass_filtered))
plt.title('Low-pass Filtered Signal')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')

plt.subplot(2, 2, 3)
plt.plot(frequencies, np.abs(high_pass_filtered))
plt.title('High-pass Filtered Signal')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')

plt.subplot(2, 2, 4)
plt.plot(frequencies, np.abs(band_pass_filtered))
plt.title('Band-pass Filtered Signal')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')

plt.tight_layout()
plt.show()
```