



Work2 librosa

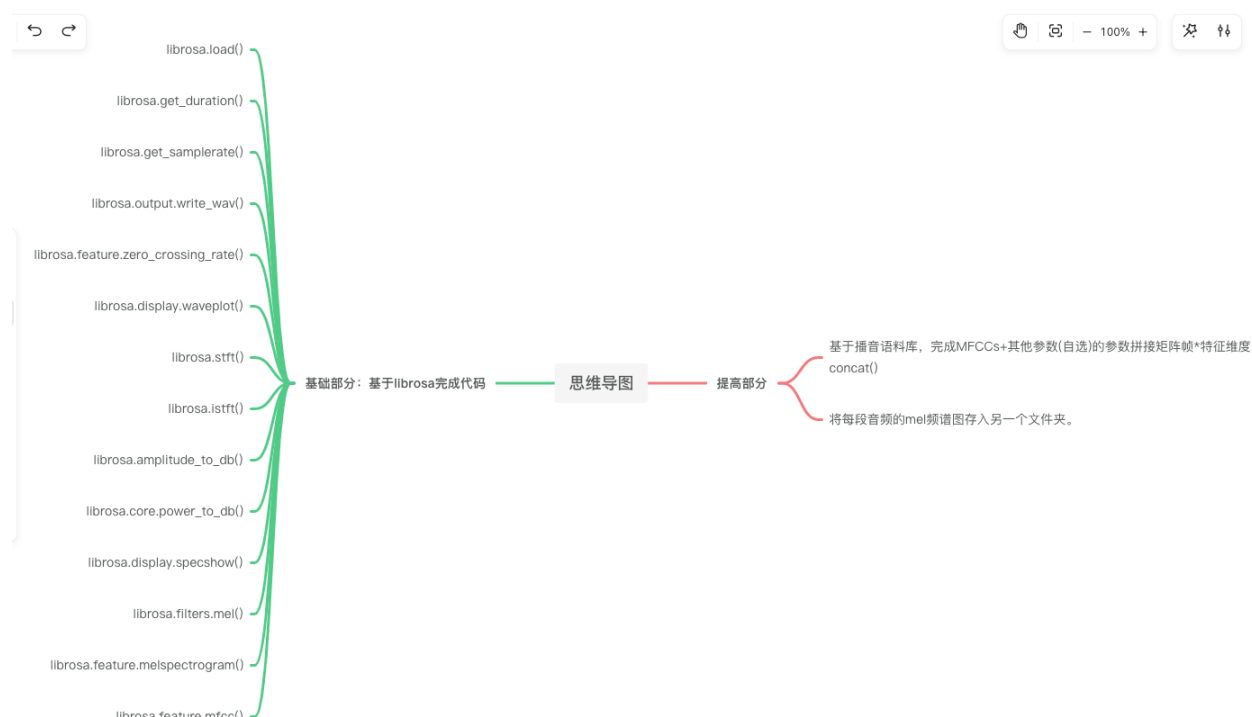
☒ Reviewed ☒

实验报告

实验进展：

已完成基础部分和提高部分所有实验内容

实验思维导图：



实验内容：

1) 提高部分：

- 基于播音语料库，完成MFCCs+其他参数(自选)的参数拼接矩阵帧*特征维度concat()
- 将每段音频的mel频谱图存入另一个文件夹。提示：plt.savefig('路径/文件名')

a、基于播音语料库，完成MFCCs+其他参数(自选)的参数拼接矩阵帧*特征维度concat()

此处自定义参数：

MFCC特征维度：13，其他自选参数的维度：5

```
import os
import librosa
import numpy as np

# 播音语料库文件夹路径
haixia_folder = 'haixia'
kanghui_folder = 'kanghui'

# 定义MFCC和其他自选参数的参数
n_mfcc = 13 # MFCC特征维度
n_other_params = 5 # 其他自选参数的维度

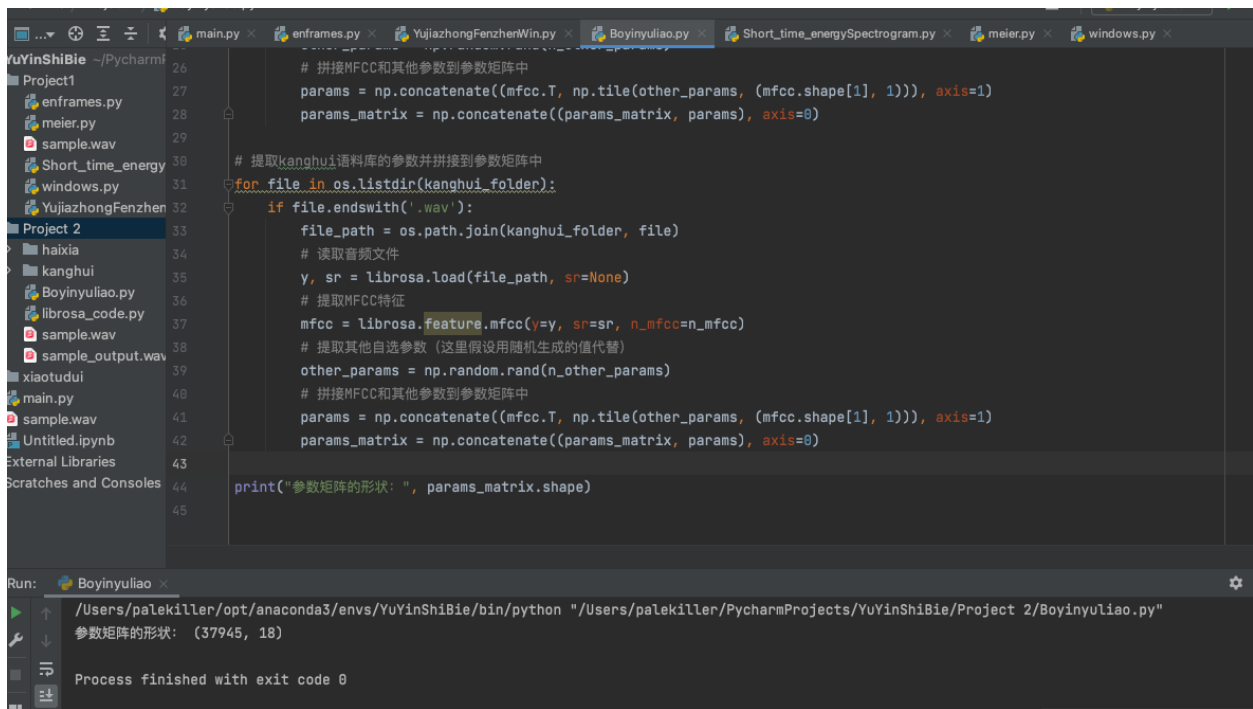
# 创建一个空的参数矩阵
params_matrix = np.empty((0, n_mfcc + n_other_params))

# 提取haixia语料库的参数并拼接到参数矩阵中
for file in os.listdir(haixia_folder):
    if file.endswith('.wav'):
        file_path = os.path.join(haixia_folder, file)
        # 读取音频文件
        y, sr = librosa.load(file_path, sr=None)
        # 提取MFCC特征
        mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc)
        # 提取其他自选参数（这里假设用随机生成的值代替）
        other_params = np.random.rand(n_other_params)
        # 拼接MFCC和其他参数到参数矩阵中
        params = np.concatenate((mfcc.T, np.tile(other_params, (mfcc.shape[1], 1))), axis=1)
        params_matrix = np.concatenate((params_matrix, params), axis=0)

# 提取kanghui语料库的参数并拼接到参数矩阵中
for file in os.listdir(kanghui_folder):
    if file.endswith('.wav'):
        file_path = os.path.join(kanghui_folder, file)
        # 读取音频文件
        y, sr = librosa.load(file_path, sr=None)
        # 提取MFCC特征
        mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc)
        # 提取其他自选参数（这里假设用随机生成的值代替）
        other_params = np.random.rand(n_other_params)
        # 拼接MFCC和其他参数到参数矩阵中
        params = np.concatenate((mfcc.T, np.tile(other_params, (mfcc.shape[1], 1))), axis=1)
        params_matrix = np.concatenate((params_matrix, params), axis=0)

print("参数矩阵的形状：", params_matrix.shape)
```

运行结果：



b、将每段音频的mel频谱图存入另一个文件夹'Boyin_mel_save'。提示：plt.savefig('路径/文件名')

```

import os
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np

# 播音语料库文件夹路径
haixia_folder = 'haixia'
kanghui_folder = 'kanghui'

# 保存Mel频谱图的文件路径
save_folder = 'Boyin_mel_save'

# 提取haixia语料库的Mel频谱图并保存
for file in os.listdir(haixia_folder):
    if file.endswith('.wav'):
        file_path = os.path.join(haixia_folder, file)
        # 读取音频文件
        y, sr = librosa.load(file_path, sr=None)
        # 提取Mel频谱图
        mel_spec = librosa.feature.melspectrogram(y=y, sr=sr)
        # 将频谱图转换为dB刻度
        mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)
        # 绘制频谱图
        plt.figure(figsize=(10, 6))
        librosa.display.specshow(mel_spec_db, sr=sr, x_axis='time', y_axis='mel')
        plt.colorbar(format='%+2.0f dB')
        plt.title('Mel Spectrogram')
        plt.xlabel('Time (s)')
        plt.ylabel('Mel Frequency')
        # 保存频谱图
        save_path = os.path.join(save_folder, file.replace('.wav', '.png'))
        plt.savefig(save_path)
        plt.close()

# 提取kanghui语料库的Mel频谱图并保存
for file in os.listdir(kanghui_folder):
    if file.endswith('.wav'):
        file_path = os.path.join(kanghui_folder, file)
        # 读取音频文件

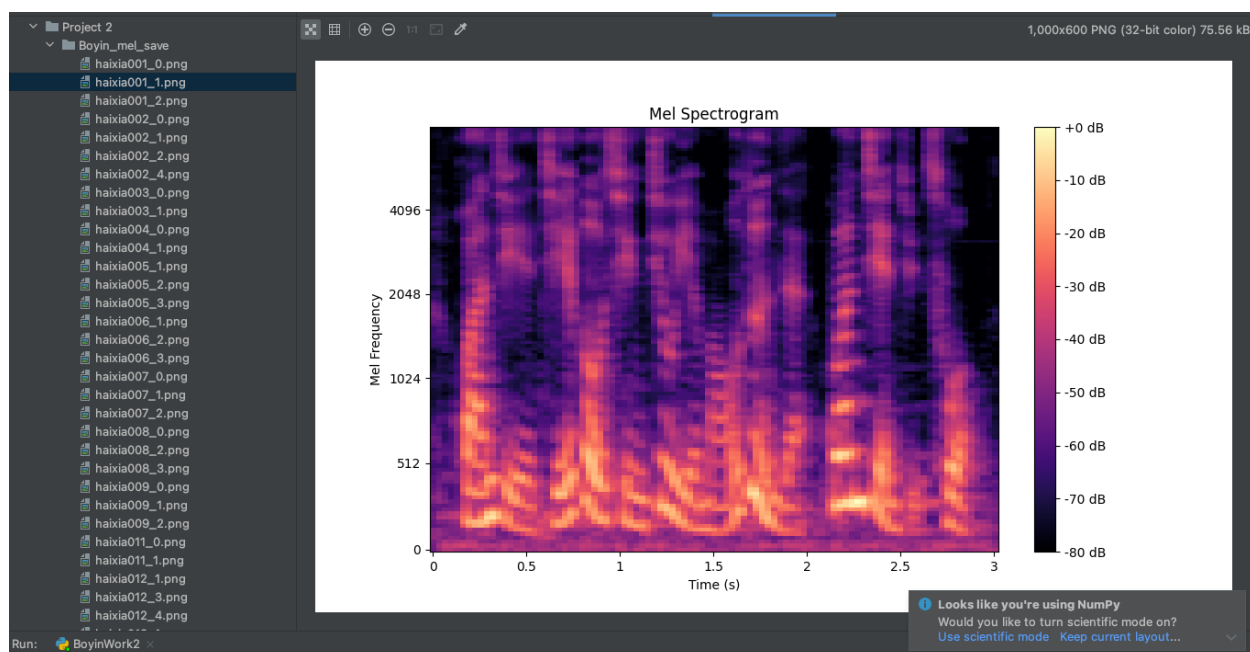
```

```

y, sr = librosa.load(file_path, sr=None)
# 提取Mel频谱图
mel_spec = librosa.feature.melspectrogram(y=y, sr=sr)
# 将频谱图转换为dB刻度
mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)
# 绘制频谱图
plt.figure(figsize=(10, 6))
librosa.display.specshow(mel_spec_db, sr=sr, x_axis='time', y_axis='mel')
plt.colorbar(format='%+2.0f dB')
plt.title('Mel Spectrogram')
plt.xlabel('Time (s)')
plt.ylabel('Mel Frequency')
# 保存频谱图
save_path = os.path.join(save_folder, file.replace('.wav', '.png'))
plt.savefig(save_path)
plt.close()

```

运行结果：



2) 基础部分：

基于librosa完成代码<https://www.cnblogs.com/LXP-Never/p/11561355.html#blogTitle6>

Librosa

Librosa（音乐信号分析Python库）是一个用于音频分析和处理的Python库。它专门设计用于音频和音乐处理应用，并提供了丰富的工具和函数，用于从音频中提取特征、处理音频信号和进行音乐分析。

librosa.load() 读取音频文件

```

'''
librosa.load()
librosa.load(path, sr=22050, mono=True, offset=0.0, duration=None)

```

读取音频文件。默认采样率是22050，如果要保留音频的原始采样率，使用sr = None。

参数：

path：音频文件的路径。

sr : 采样率, 如果为“None”使用音频自身的采样率
mono : bool, 是否将信号转换为单声道
offset : float, 在此时间之后开始阅读 (以秒为单位)
持续时间: float, 仅加载这么多的音频 (以秒为单位)

返回:

y : 音频时间序列
sr : 音频的采样率

```
'''  
y, sr = librosa.load('sample.wav')  
print(' y : 音频时间序列:', y, 'sr : 音频的采样率', sr)
```

运行截图:



librosa.get_duration() 计算时间序列、特征矩阵、文件的持续时间 (以秒为单位)

```
'''  
librosa.get_duration()  
librosa.get_duration(y=None, sr=22050, S=None, n_fft=2048, hop_length=512, center=True, filename=None)
```

计算时间序列的持续时间 (以秒为单位)

参数:

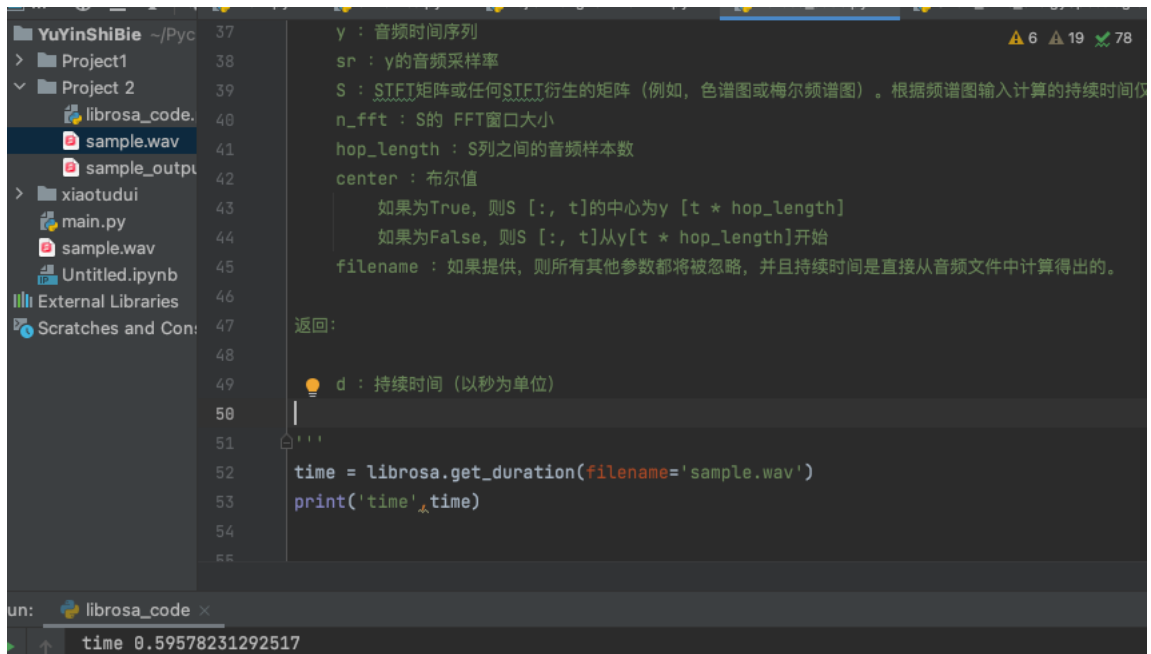
y : 音频时间序列
sr : y的音频采样率
S : STFT矩阵或任何STFT衍生的矩阵 (例如, 色谱图或梅尔频谱图)。根据频谱图输入计算的持续时间仅在达到帧分辨率之前才是准确的。如果需要高精度, 则最好直接使用音频时间序列。
n_fft : S的 FFT窗口大小
hop_length : S列之间的音频样本数
center : 布尔值
 如果为True, 则S[:, t]的中心为y[t * hop_length]
 如果为False, 则S[:, t]从y[t * hop_length]开始
filename : 如果提供, 则所有其他参数都将被忽略, 并且持续时间是直接从音频文件中计算得出的。

返回:

d : 持续时间 (以秒为单位)

```
'''  
time = librosa.get_duration(filename='sample.wav')  
print('time', time)
```

运行截图：



The screenshot shows a Jupyter Notebook interface. On the left is a file explorer showing a project structure with files like 'librosa_code.py', 'sample.wav', and 'sample_output'. The main area displays the documentation for the `librosa.get_duration` function. The documentation includes parameters: `y` (audio time series), `sr` (sampling rate), `S` (STFT matrix or derived), `n_fft` (FFT window size), `hop_length` (audio samples between S rows), `center` (boolean for centering), and `filename` (file path). It also shows the return value `d` (duration in seconds). Below the documentation, a code cell contains the following Python code:

```
time = librosa.get_duration(filename='sample.wav')
print('time', time)
```

The output of the code cell is visible at the bottom: `time 0.59578231292517`.

`librosa.get_samplerate()` 读取采样率

```
...
librosa.get_samplerate(path) 读取采样率

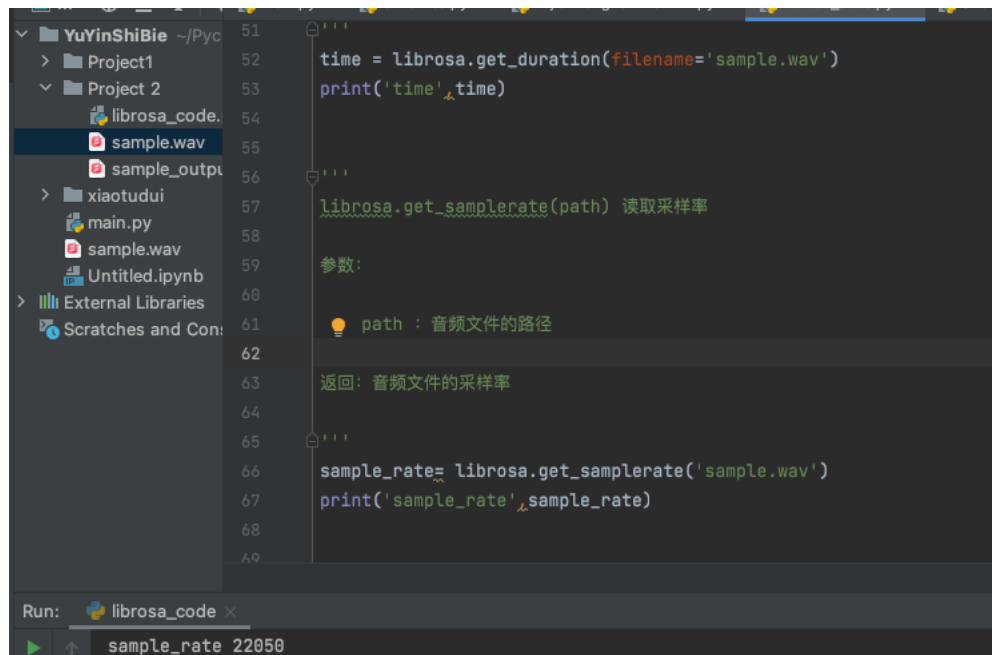
参数：

    path : 音频文件的路径

返回：音频文件的采样率

...
sample_rate= librosa.get_samplerate('sample.wav')
print('sample_rate', sample_rate)
```

运行截图：



librosa.output.write_wav() 将时间序列输出为.wav文件

```
'''
librosa.output.write_wav()
librosa.output.write_wav(path, y, sr, norm=False)

将时间序列输出为.wav文件

参数:

    path: 保存输出wav文件的路径
    y : 音频时间序列。
    sr : y的采样率
    norm: bool, 是否启用幅度归一化。将数据缩放到[-1, +1]范围。

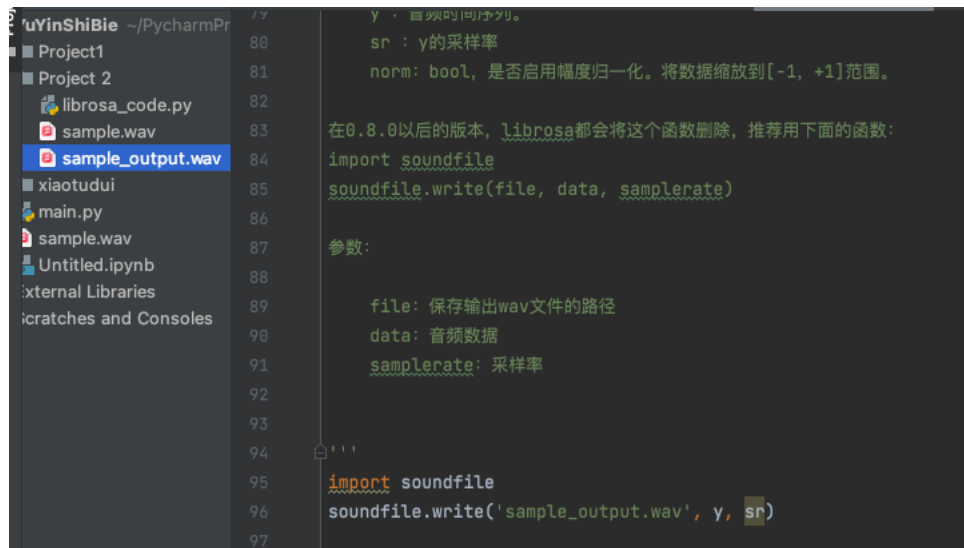
在0.8.0以后的版本, librosa都会将这个函数删除, 推荐用下面的函数:
import soundfile
soundfile.write(file, data, samplerate)

参数:

    file: 保存输出wav文件的路径
    data: 音频数据
    samplerate: 采样率

'''
import soundfile
soundfile.write('sample_output.wav', y, sr)
```

运行截图:



`librosa.feature.zero_crossing_rate()` 计算一段音频序列的过零率

```
'''
librosa.feature.zero_crossing_rate()
librosa.feature.zero_crossing_rate(y, frame_length = 2048, hop_length = 512, center = True)

计算音频时间序列的过零率。

参数：

    y : 音频时间序列
    frame_length : 帧长
    hop_length : 帧移
    center : bool, 如果为True, 则通过填充y的边缘来使帧居中。

返回：

    zcr : zcr[0, i]是第i帧中的过零率

'''
zcr = librosa.feature.zero_crossing_rate(y, frame_length= 2048, hop_length= 512, center = True)
print('过零率', zcr)
```

运行截图：


```
102 计算音频时间序列的过零率。
103
104 参数:
105
106     y : 音频时间序列
107     frame_length : 帧长
108     hop_length : 帧移
109     center: bool, 如果为True, 则通过填充y的边缘来使帧居中。
110
111 返回:
112
113     zcr: zcr[0, i]是第i帧中的过零率
114
115 '''
116 zcr = librosa.feature.zero_crossing_rate(y, frame_length=2048, hop_length=512, center=True)
117 print('过零率', zcr)
118
119
```

Run: librosa_code x

```
过零率 [[0.02392578 0.05371094 0.07519531 0.08740234 0.08544922 0.0703125
0.06445312 0.05126953 0.04101562 0.02832031 0.01708984 0.03564453
0.05371094 0.06542969 0.0703125 0.05517578 0.05175781 0.06201172
0.07373047 0.06933594 0.05810547 0.04003906 0.02539062 0.02441406
0.02392578 0.01757812]]
```

librosa.display.waveplot() 绘制波形的幅度包络线

```
'''
librosa.display.waveplot()
librosa.display.waveplot(y, sr=22050, x_axis='time', offset=0.0, ax=None)

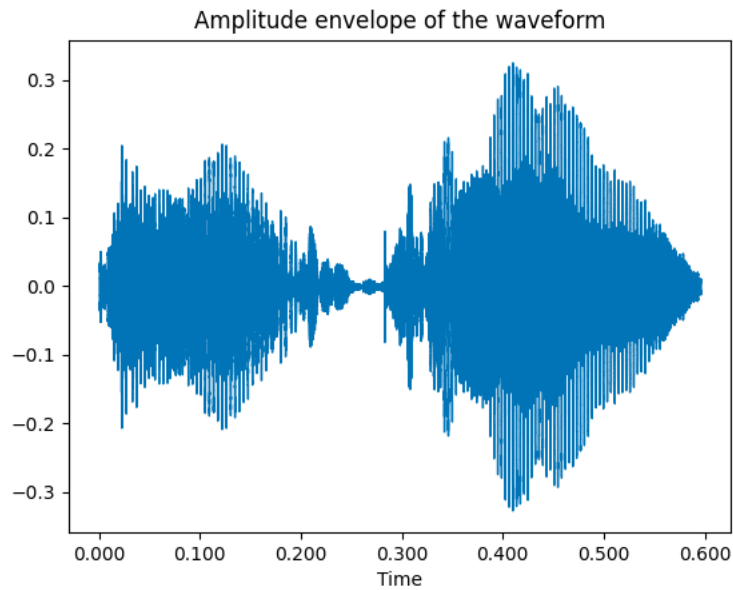
绘制波形的幅度包络线

参数:

    y : 音频时间序列
    sr : y的采样率
    x_axis : str {'time', 'off', 'none'}或None, 如果为“时间”, 则在x轴上给定时间刻度线。
    offset : 水平偏移 (以秒为单位) 开始波形图

'''
import librosa.display
librosa.display.waveshow(y, sr=sr)
plt.title('Amplitude envelope of the waveform')
plt.show()
```

运行结果：



librosa.stft() 短时傅立叶变换（STFT），返回一个复数矩阵D(F, T)

```
'''
librosa.stft()
librosa.stft(y, n_fft=2048, hop_length=None, win_length=None, window='hann', center=True, pad_mode='reflect')

短时傅立叶变换（STFT），返回一个复数矩阵D(F, T)

参数：

y：音频时间序列
n_fft：FFT窗口大小，n_fft=hop_length+overlapping
hop_length：帧移，如果未指定，则默认win_length / 4。
win_length：每一帧音频都由window () 加窗。窗长win_length，然后用零填充以匹配N_FFT。默认win_length=n_fft。
window：字符串，元组，数字，函数 shape = (n_fft, )
    窗口（字符串，元组或数字）；
    窗函数，例如scipy.signal.hanning
    长度为n_fft的向量或数组
center：bool
    如果为True，则填充信号y，以使帧 D[:, t]以y[t * hop_length]为中心。
    如果为False，则D[:, t]从y[t * hop_length]开始
dtype：D的复数值类型。默认值为64-bit complex复数
pad_mode：如果center = True，则在信号的边缘使用填充模式。默认情况下，STFT使用reflection padding。

返回：

    STFT矩阵，shape = (1 + n_fft/2, t)

'''
stft = librosa.stft(y)
print('stft矩阵', stft)
```

运行截图：

```
main.py x enframes.py x YuJiazhongFenzhenWin.py x librosa_code.py x Sh...
'YuYinShiBie ~/PycharmPr 161
162 返回:
163
164 STFT矩阵, shape = (1 + n_fft/2, t)
165
166
167 stft = librosa.stft(y)
168 print('stft矩阵',stft)
169
170
Scratches and Consoles
Run: librosa_code x
stft矩阵 [[-1.2049651e+00+0.0000000e+00j -2.2843478e+00+0.0000000e+00j
2.3120918e+00+0.0000000e+00j ... 1.1385580e+00+0.0000000e+00j
2.5151622e+00+0.0000000e+00j 3.2161636e+00+0.0000000e+00j]
[-3.9646356e+00-4.8219271e+00j 8.2311153e+00+3.8646221e+00j
-7.1988249e+00+6.1588013e-01j ... -2.1783781e-01+4.1062522e-01j
-1.3809110e+00+9.1298324e-01j -2.1840229e+00-2.1140006e-01j]
[ 1.0554677e+01-2.4934361e+00j -2.0589975e+01-4.7575369e+00j
1.2949191e+01+1.1484971e+01j ... -3.6486751e-01-3.5773374e-02j
3.0329210e-01-5.0304419e-01j 3.6845496e-01-3.3544299e-01j]
...
[ 1.2369737e-02+9.2161947e-04j -3.7951078e-03+6.0298800e-04j
-1.7561831e-02-3.6523312e-03j ... 1.2769577e-03-4.8353332e-03j
2.9613206e-03+4.4135610e-03j -2.7822526e-03-1.7650318e-03j]
[-1.2240043e-02+6.7730347e-04j 7.4838137e-04+1.3139565e-02j
1.4028014e-02+1.9103336e-03j ... -2.1512380e-03+1.0997737e-03j
-3.1251737e-03-1.0558778e-03j -1.2183007e-03+2.3466668e-03j]
[ 1.0946919e-02+0.0000000e+00j -3.5174384e-03+0.0000000e+00j
-1.9990452e-02+0.0000000e+00j ... 5.0860606e-03+0.0000000e+00j
4.8479149e-03+0.0000000e+00j 3.4057524e-03+0.0000000e+00j]]
```

librosa.istft() 短时傅立叶逆变换 (ISTFT)，将复数值 $D(f,t)$ 频谱矩阵转换为时间序列 y ，窗函数、帧移等参数应与stft相同

```
...
librosa.istft()
librosa.istft(stft_matrix, hop_length=None, win_length=None, window='hann', center=True, length=None)
```

短时傅立叶逆变换 (ISTFT)，将复数值 $D(f,t)$ 频谱矩阵转换为时间序列 y ，窗函数、帧移等参数应与stft相同

参数：

stft_matrix : 经过STFT之后的矩阵
hop_length : 帧移, 默认为

win_length : 窗长, 默认为n_fft
window : 字符串, 元组, 数字, 函数或shape = (n_fft,)
窗口 (字符串, 元组或数字)
窗函数, 例如scipy.signal.hanning
长度为n_fft的向量或数组

center : bool
如果为True, 则假定D具有居中的帧
如果False, 则假定D具有左对齐的帧

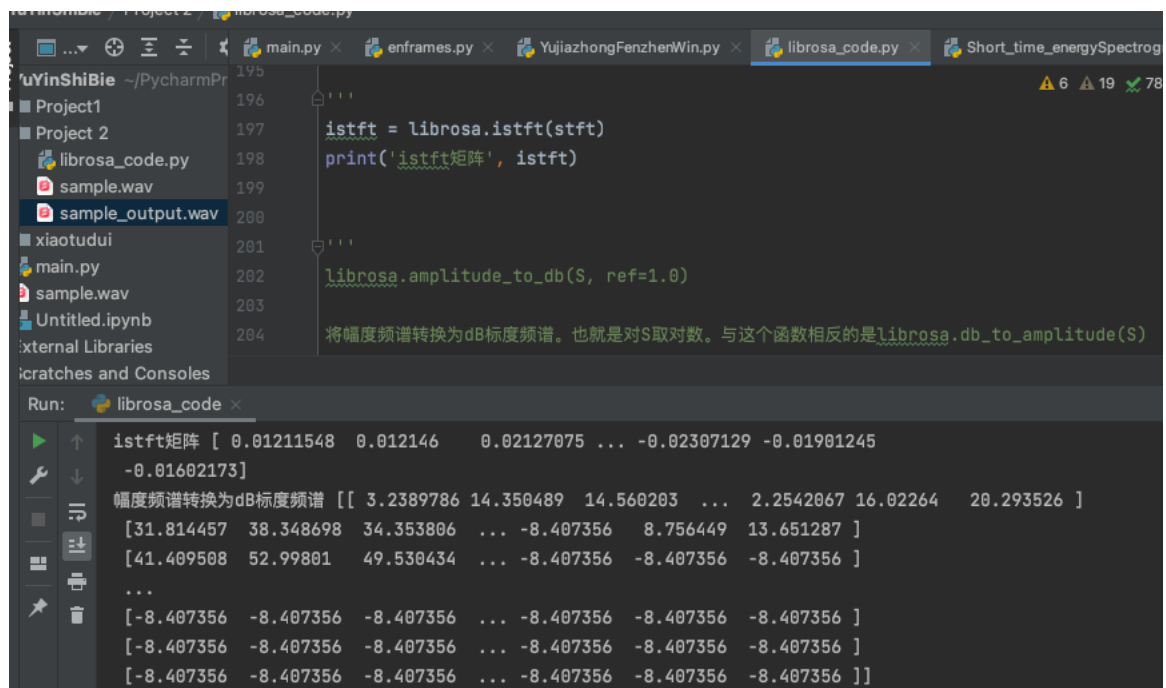
length : 如果提供, 则输出y为零填充或剪裁为精确长度音频

返回：

y : 时域信号

```
...
istft = librosa.istft(stft)
print('istft矩阵', istft)
```

运行截图：



```
195
196
197     istft = librosa.istft(stft)
198     print('istft矩阵', istft)
199
200
201
202     librosa.amplitude_to_db(S, ref=1.0)
203
204     将幅度频谱转换为dB标度频谱。也就是对S取对数。与这个函数相反的是librosa.db_to_amplitude(S)
```

Run: librosa_code ×

```
istft矩阵 [ 0.01211548  0.012146   0.02127075 ... -0.02307129 -0.01901245
-0.01602173]
幅度频谱转换为dB标度频谱 [[ 3.2389786 14.350489 14.560203 ... 2.2542067 16.02264 20.293526 ]
[31.814457 38.348698 34.353806 ... -8.407356 8.756449 13.651287 ]
[41.409508 52.99801 49.530434 ... -8.407356 -8.407356 -8.407356 ]
...
[-8.407356 -8.407356 -8.407356 ... -8.407356 -8.407356 -8.407356 ]
[-8.407356 -8.407356 -8.407356 ... -8.407356 -8.407356 -8.407356 ]
[-8.407356 -8.407356 -8.407356 ... -8.407356 -8.407356 -8.407356 ]]
```

`librosa.amplitude_to_db()` 将幅度频谱转换为dB标度频谱。也就是对S取对数。与这个函数相反的是`librosa.db_to_amplitude(S)`

```
...
librosa.amplitude_to_db(S, ref=1.0)

将幅度频谱转换为dB标度频谱。也就是对S取对数。与这个函数相反的是librosa.db_to_amplitude(S)

参数：

S : 输入幅度
ref : 参考值, 幅值 abs(S) 相对于ref进行缩放,

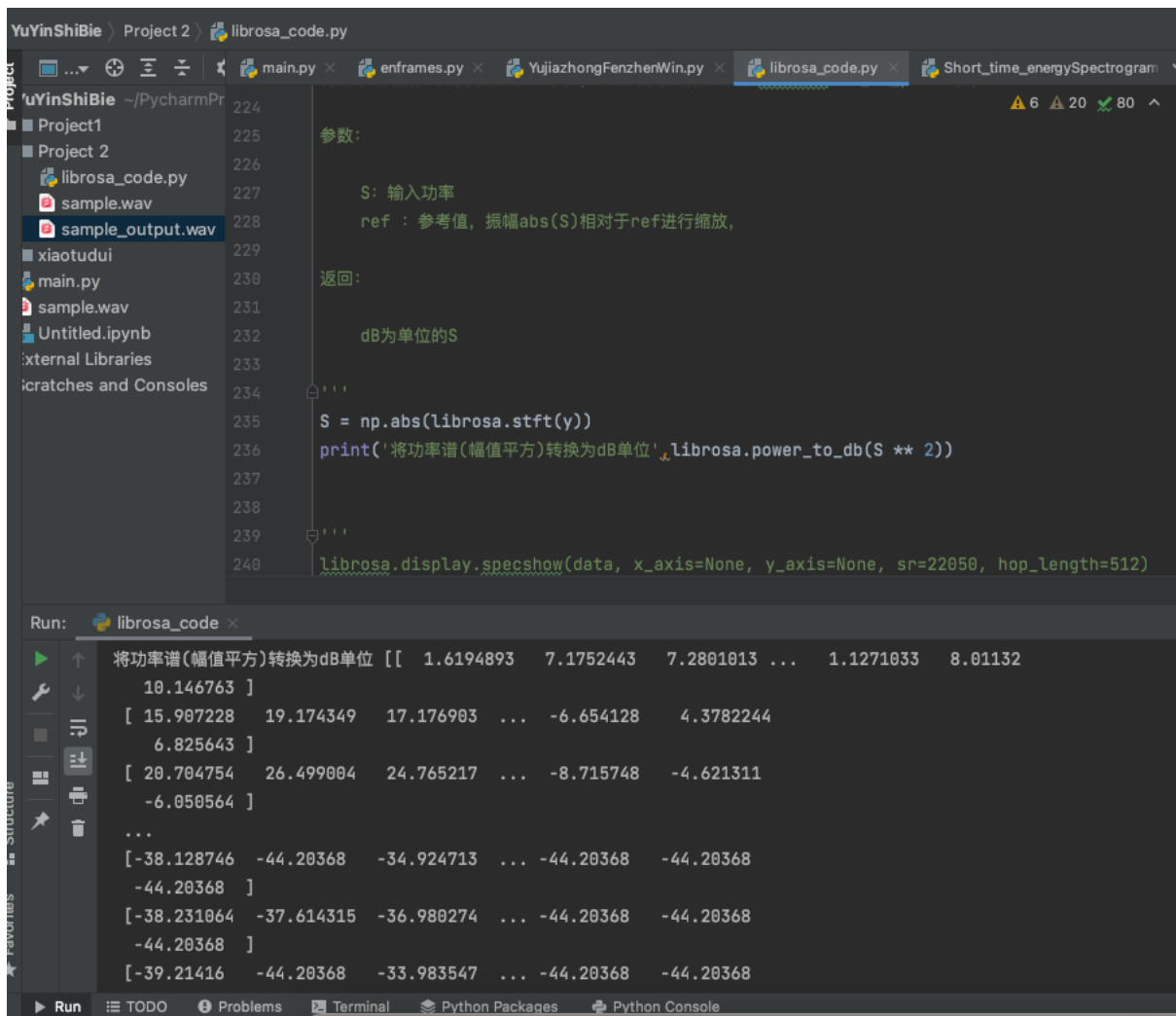
返回：

dB为单位的S

...

S = np.abs(librosa.stft(y))
print('幅度频谱转换为dB标度频谱', librosa.amplitude_to_db(S ** 2))
```

运行截图：



librosa.display.specshow() 绘制频谱图

绘制功率谱和转换为dB单位的功率谱

```

...
librosa.display.specshow(data, x_axis=None, y_axis=None, sr=22050, hop_length=512)

```

绘制频谱图

参数：

data：要显示的矩阵
sr：采样率
hop_length：帧移
x_axis y_axis：x和y轴的范围
频率类型
'linear'：频率范围由FFT窗口和采样率确定
'log'：频谱以对数刻度显示
'mel'：频率由mel标度决定
时间类型
time：标记以毫秒，秒，分钟或小时显示。值以秒为单位绘制。
s：标记显示为秒。
ms：标记以毫秒为单位显示。
所有频率类型均以Hz为单位绘制

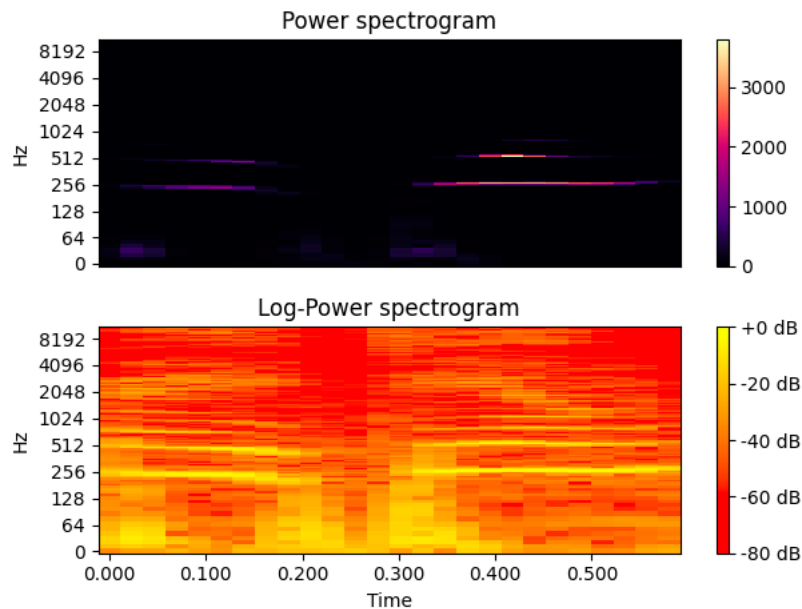
```

...
plt.figure()

```

```
plt.subplot(2, 1, 1)
librosa.display.specshow(S ** 2, sr=sr, y_axis='log') # 绘制功率谱
plt.colorbar()
plt.title('Power spectrogram')
plt.subplot(2, 1, 2)
# 相对于峰值功率计算dB, 那么其他的dB都是负的, 注意看右边cmap值
librosa.display.specshow(librosa.power_to_db(S ** 2, ref=np.max), sr=sr, y_axis='log', x_axis='time') # 绘制对数功率谱
plt.colorbar(format='%+2.0f dB')
plt.title('Log-Power spectrogram')
plt.set_cmap("autumn")
plt.tight_layout()
plt.show()
```

运行结果：



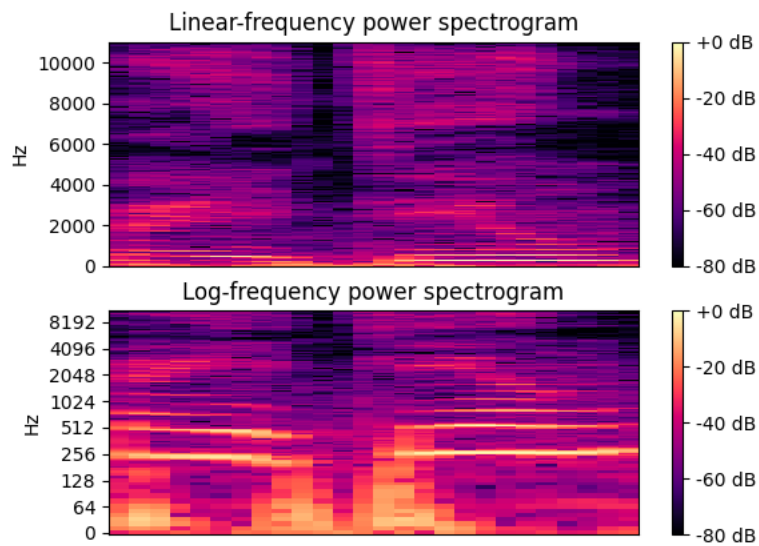
绘制幅值转dB的线性频率功率谱和对数频率功率谱

```
plt.figure()

D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)
plt.subplot(2, 1, 1)
librosa.display.specshow(D, y_axis='linear')
plt.colorbar(format='%+2.0f dB')
plt.title('Linear-frequency power spectrogram') # 线性频率功率谱

plt.subplot(2, 1, 2)
librosa.display.specshow(D, y_axis='log')
plt.colorbar(format='%+2.0f dB')
plt.title('Log-frequency power spectrogram') # 对数频率功率谱
plt.show()
```

运行结果：



`librosa.filters.mel()` 创建一个滤波器组矩阵以将FFT合并成Mel频率

```
'''
librosa.filters.mel(sr, n_fft, n_mels=128, fmin=0.0, fmax=None, htk=False, norm=1)

创建一个滤波器组矩阵以将FFT合并成Mel频率

参数：

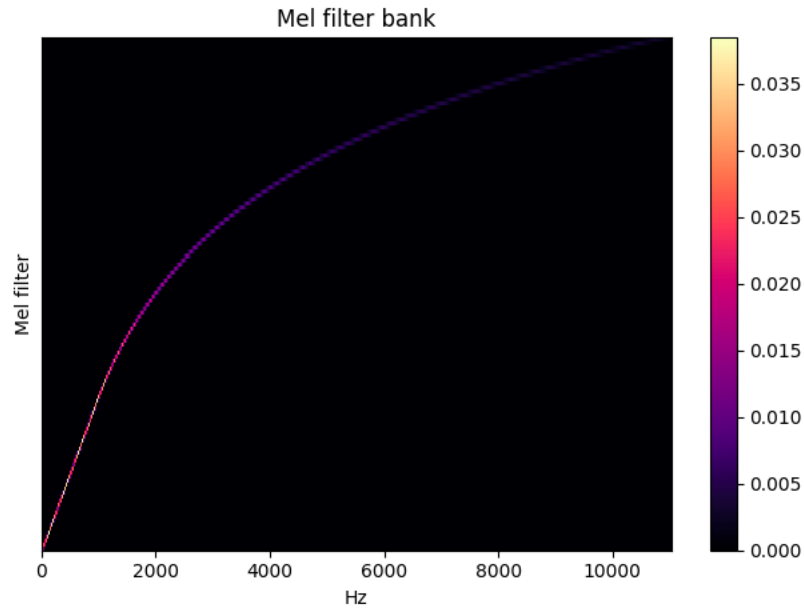
    sr : 输入信号的采样率
    n_fft : FFT组件数
    n_mels : 产生的梅尔带数
    fmin : 最低频率 (Hz)
    fmax : 最高频率 (以Hz为单位)。如果为None, 则使用fmax = sr / 2.0
    norm : {None, 1, np.inf} [标量]
        如果为1, 则将三角mel权重除以mel带的宽度 (区域归一化)。否则, 保留所有三角形的峰值为1.0

返回：Mel变换矩阵

'''

melfb = librosa.filters.mel(sr=22050, n_fft=2048)
plt.figure()
librosa.display.specshow(melfb, x_axis='linear')
plt.ylabel('Mel filter')
plt.title('Mel filter bank')
plt.colorbar()
plt.tight_layout()
plt.show()
```

运行结果：



librosa.feature.melspectrogram() 求取Mel频谱

```
'''
librosa.feature.melspectrogram(y=None, sr=22050, S=None, n_fft=2048, hop_length=512, win_length=None, window='hann', center=True, pad_mode=

计算Mel频谱

如果提供了频谱图输入S, 则通过mel_f.dot (S) 将其直接映射到mel_f上。

如果提供了时间序列输入y, sr, 则首先计算其幅值频谱S, 然后通过mel_f.dot (S ** power) 将其映射到mel scale上。默认情况下, power= 2在功率谱上运行。

参数:

    y : 音频时间序列
    sr : 采样率
    S : 频谱
    n_fft : FFT窗口的长度
    hop_length : 帧移
    win_length : 窗口的长度为win_length, 默认win_length = n_fft
    window : 字符串, 元组, 数字, 函数或shape = (n_fft, )
        窗口规范 (字符串, 元组或数字); 看到scipy.signal.get_window
        窗口函数, 例如 scipy.signal.hanning
        长度为n_fft的向量或数组
    center : bool
        如果为True, 则填充信号y, 以使帧 t以y [t * hop_length]为中心。
        如果为False, 则帧t从y [t * hop_length]开始
    power : 幅度谱的指数。例如1代表能量, 2代表功率, 等等
    n_mels : 滤波器组的个数 1288
    fmax : 最高频率

返回: Mel频谱shape=(n_mels, t)

'''

# 方法一: 使用时间序列求Mel频谱
print('Mel频谱', librosa.feature.melspectrogram(y=y, sr=sr))

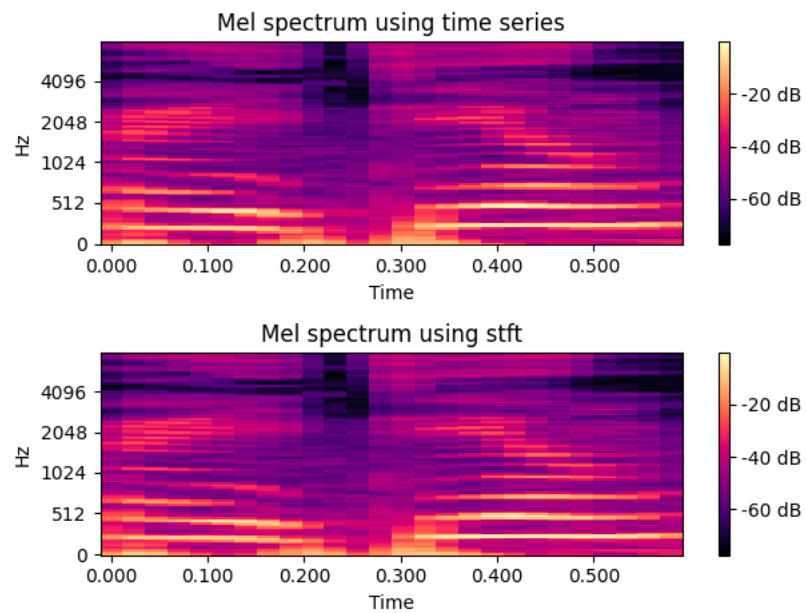
# 方法二: 使用stft频谱求Mel频谱
D = np.abs(librosa.stft(y)) ** 2 # stft频谱
S = librosa.feature.melspectrogram(S=D) # 使用stft频谱求Mel频谱

plt.figure(figsize=(10, 4))
librosa.display.specshow(librosa.power_to_db(S, ref=np.max), y_axis='mel', fmax=8000, x_axis='time')

plt.colorbar(format='%+2.0f dB')
```

```
plt.title('Mel spectrogram')
plt.tight_layout()
plt.show()
```

运行结果：



```
YinShiBie > Project 2 > librosa_code.py
... 7 21 80 ^ v
Project1
Project 2
librosa_code.py
sample.wav
sample_output.wav
xiaotudui
main.py
sample.wav
Untitled.ipynb
External Libraries
Cratches and Consoles
352
353 # 方法一：使用时间序列求Mel频谱
354 print('Mel频谱', librosa.feature.melspectrogram(y=y, sr=sr))
355
356 # 方法二：使用stft频谱求Mel频谱
357 D = np.abs(librosa.stft(y)) ** 2 # stft频谱
358 S = librosa.feature.melspectrogram(S=D) # 使用stft频谱求Mel频谱
359
360 # 可视化使用时间序列求Mel频谱
361 plt.subplot(2, 1, 1)
362 librosa.display.specshow(librosa.power_to_db(librosa.feature.melspectrogram(y=y, sr=sr))
363 plt.colorbar(format='%+2.0f dB')
364 plt.title('Mel spectrum using time series')
365 plt.tight_layout()
Run: librosa_code x
Mel频谱 [[8.49792004e+00 3.88988228e+01 2.35739956e+01 ... 1.36033446e-02
6.99590296e-02 1.19607508e-01]
[3.17283082e+00 2.33829212e+01 1.35104685e+01 ... 5.55578768e-02
8.65228176e-02 1.00905247e-01]
[5.38794219e-01 2.41469407e+00 1.75359464e+00 ... 1.27279714e-01
1.33379593e-01 1.01538472e-01]
...
[2.40159617e-03 1.18778413e-02 1.16650397e-02 ... 2.53711274e-04
1.05742954e-04 4.39646610e-05]
[6.61151716e-04 3.07304226e-03 4.24639042e-03 ... 1.32512403e-04
5.32691774e-05 2.45274587e-05]
[1.01015547e-04 5.38025051e-04 2.14530504e-03 ... 2.52284590e-05
1.32360728e-05 6.19555931e-06]]
```

提取Log-Mel Spectrogram 特征

```
# 提取Log-Mel Spectrogram 特征
# 提取 mel spectrogram feature
melspec = librosa.feature.melspectrogram(y=y, sr=sr, n_fft=1024, hop_length=512, n_mels=128)
logmelspec = librosa.power_to_db(melspec) # 转换到对数刻度

print('Log-Mel Spectrogram 特征', logmelspec.shape) # (128, 65)
```

运行结果：

```
375 # 提取Log-Mel Spectrogram 特征
376 # 提取 mel spectrogram feature
377 melspec = librosa.feature.melspectrogram(y=y, sr=sr, n_fft=1024, hop_length=512, n_mels=128)
378 logmelspec = librosa.power_to_db(melspec) # 转换到对数刻度
379
380 print('Log-Mel Spectrogram 特征', logmelspec.shape) # (128, 65)
381
382
383
Run: librosa_code x
Log-Mel Spectrogram 特征 (128, 26)
```

librosa.feature.mfcc() 提取MFCC系数

```
'''
librosa.feature.mfcc(y=None, sr=22050, S=None, n_mfcc=20, dct_type=2, norm='ortho', **kwargs)

提取MFCC系数

参数：

    y：音频数据
    sr：采样率
    S：np.ndarray，对数功能梅尔谱图
    n_mfcc：int>0，要返回的MFCC数量
    dct_type：None, or {1, 2, 3} 离散余弦变换（DCT）类型。默认情况下，使用DCT类型2。
    norm：None or 'ortho' 规范。如果dct_type为2或3，则设置norm='ortho'使用正交DCT基础。 标准化不支持dct_type = 1。

返回：

    M：MFCC序列

'''

# 提取 MFCC feature
mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40)
# 可视化 MFCC feature
print('MFCC feature', mfccs.shape)      # (40, 65)
librosa.display.specshow(mfccs, x_axis='time')
plt.colorbar()
plt.title('MFCC')
plt.tight_layout()
plt.show()
```

运行结果：

