

```
In [31]: import numpy as np
import feature_extraction.simple_extractor_py3 as extractor
import pretreatment.pretreatment as pretreatment
import tools.drawer_utils as drawer_utils
import tools.sound_utils as sound_utils
from scipy.io import wavfile
from imp import reload
%load_ext autoreload
%autoreload 2
%matplotlib inline
```

The autoreload extension is already loaded. To reload it, use:  
%reload\_ext autoreload

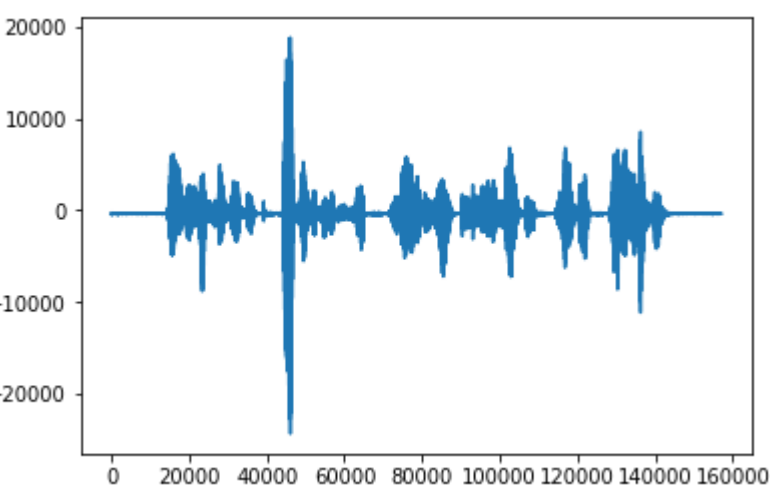
```
In [32]: wave_file = r"/datas/A2_0.wav"
```

```
In [33]: sampling_freq, audio = wavfile.read(wave_file) # 读出采样率和时域数据矩阵
sound_utils.play_sound(audio,sampling_freq)
```

```
In [34]: print(sampling_freq,audio)
```

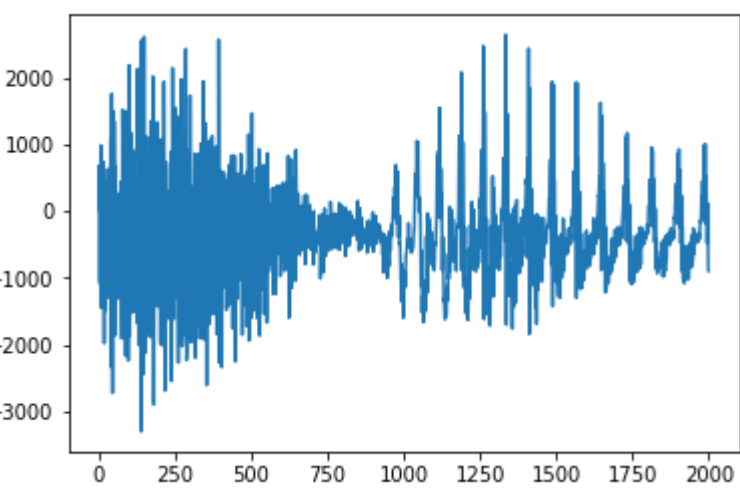
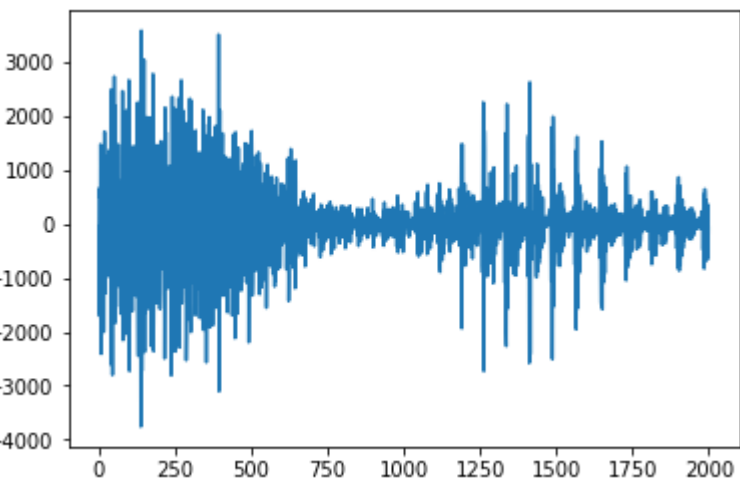
16000 [-296 -424 -392 ... -394 -379 -390]

```
In [30]: drawer_utils.plot_array(audio)
```



预加重 pre-emphasis

```
In [36]: pre_emphasised = pretreatment.pre_emphasis(audio,0.98)
drawer_utils.plot_array(pre_emphasised[20000:22000])
drawer_utils.plot_array(audio[20000:22000])
```



```
In [37]: sound_utils.play_sound(pre_emphasised,sampling_freq)
# from IPython.display import Audio
# Audio(data=pre_emphasised, rate=sampling_freq)
```

作用

- 增强高频能量，提升高频分辨率，衰弱200HZ以下的频率成分

问题

- 预加重后，反而噪音变强了

发现

- 预加重后，时域波形图明显变“密”了（高音部分变多了，所以直观地听起来就是更嘈杂了）
- 频率越高声音越尖，频率越低声音越低沉
- 明显听后声音更加低沉

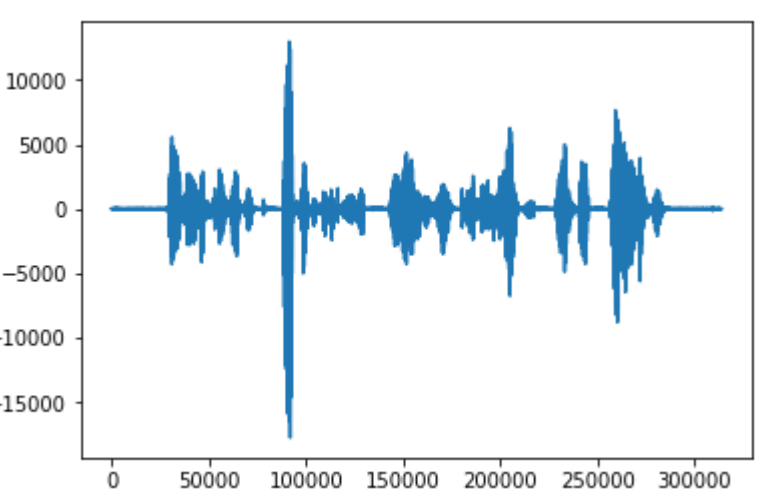
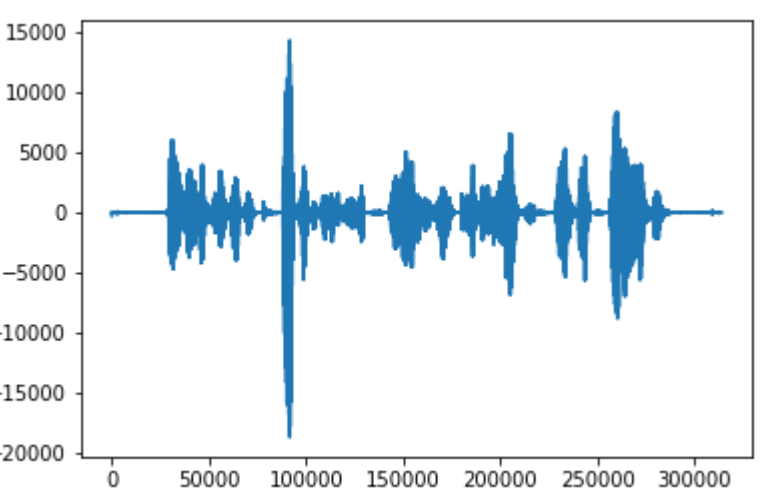
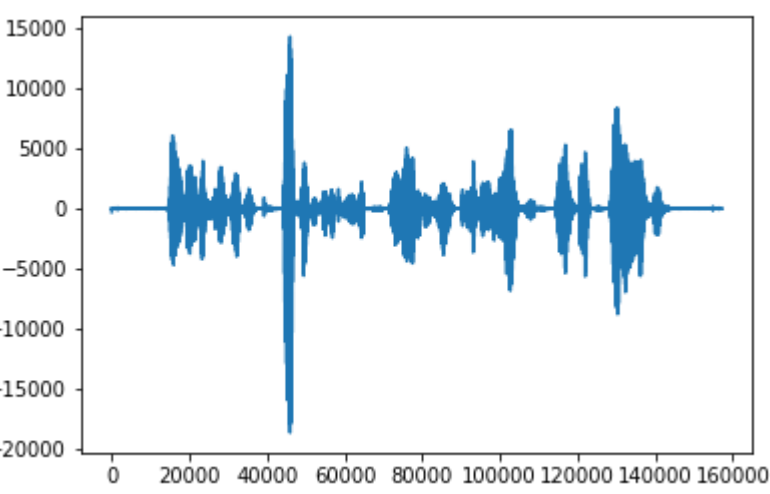
个人理解

- 虽然噪声变大了，但是发现预处理后，整个音频都充斥着这个噪声，在计算机看来，这个噪声其实可以理解成“没有噪声”（只是相对的）

分帧、加窗

```
In [38]: # 无加窗
frames = pretreatment.enframe(pre_emphasised,512,256)
# 加hanning窗
import scipy.signal as signal
hanning_frames = pretreatment.enframe(pre_emphasised,512,256,signal.hanning(512))
```

```
In [39]: drawer_utils.plot_array(pre_emphasised)
drawer_utils.plot_array(frames.flatten())
drawer_utils.plot_array(hanning_frames.flatten())
```



作用

- 语音信号本是一种典型的非平稳信号，但是相对于声波振动的速度，发音器官的运动就是非常缓慢的了，所以工程上通常认为10-30ms这样长度的时间段中，语音信号是平稳的

问题

- 无

发现

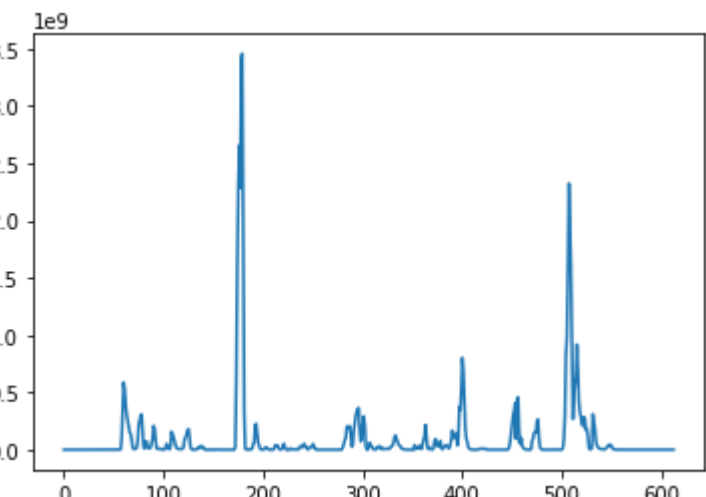
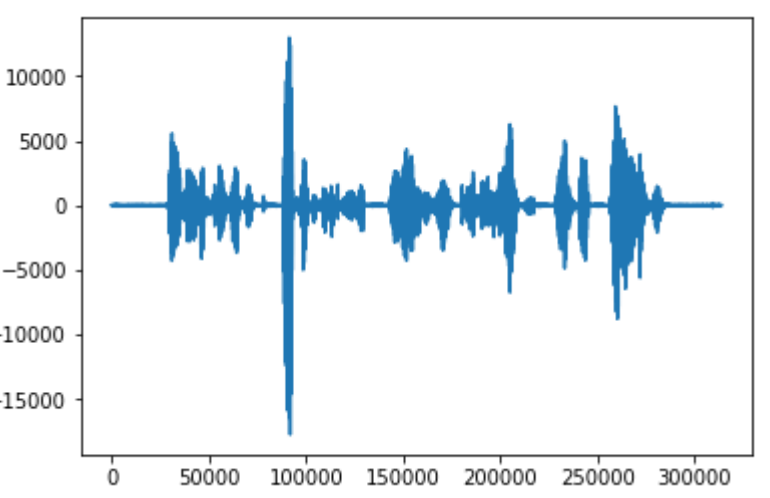
- 整体音频变长了，说话变慢了，毕竟有叠音。
- 未加窗的有明显的拖长和重声的现象
- 加了hanning窗后，重声现象有效减缓

个人理解

- 整体作用如上所述，这样变换后，让音频更像“图像”了。但是比图像的像素更具连续性的特点
- hanning窗，保留中间部分，削弱边缘部分，能够有效抑制重音现象

短时能量

```
In [40]: drawer_utils.plot_array(hanning_frames.flatten())
drawer_utils.plot_array(extractor.cal_energy(hanning_frames))
```



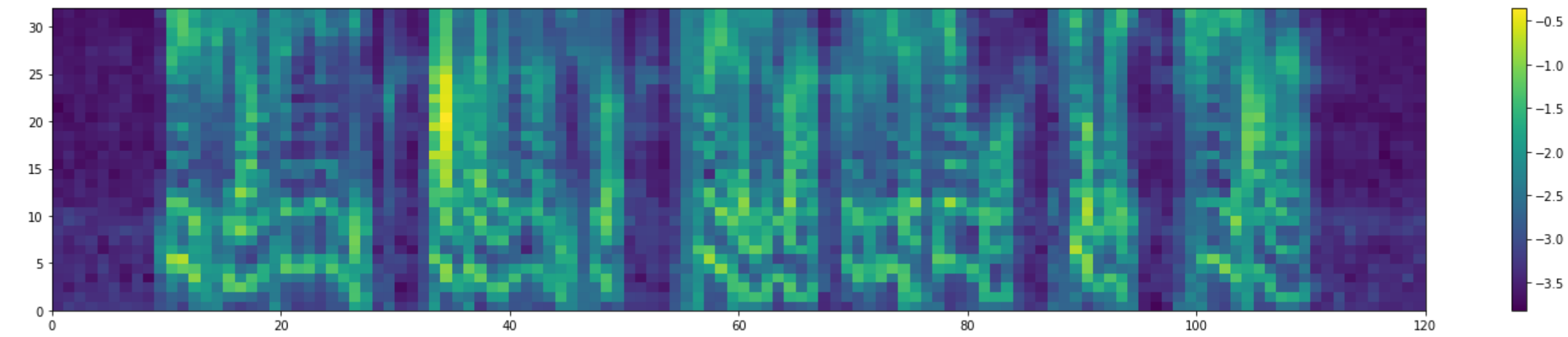




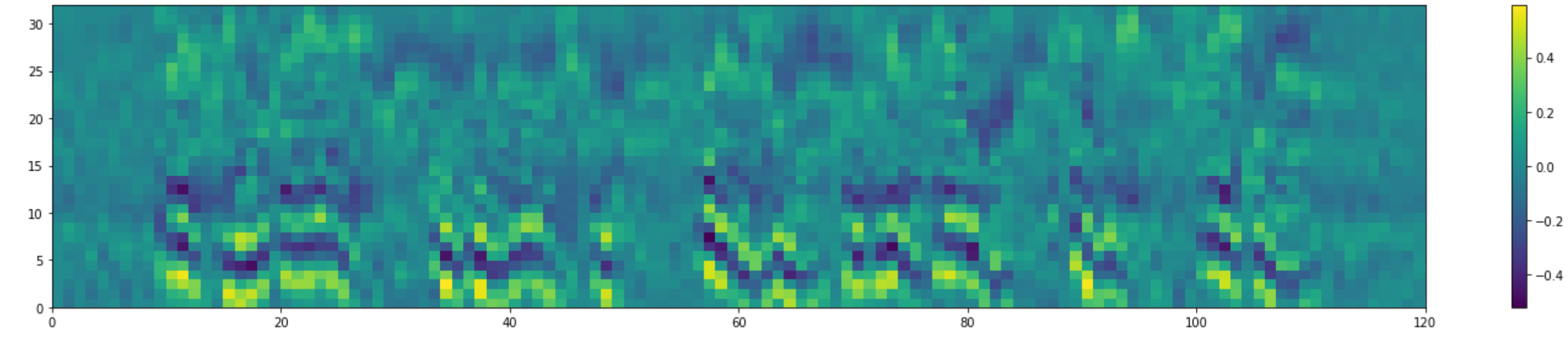


```
In [52]: mel_spec = mfcc.make_mel(
        spectrogram_wave,
        mel_filter,
        shorten_factor = shorten_factor
    )
    drawer_utils.plot_colorbar(mel_spec, astype("float"))

e:\anaconda3\envs\mypython\lib\site-packages\scipy\ndimage\interpolation.py:616: UserWarning: From scipy 0.13.0, the output shape of zoom() is calculated with round() instead of int() - for these inputs the size of the returned array has changed.
  "the returned array has changed.", UserWarning)
```



```
In [53]: # mel变化差值
        drawer_utils.plot_colorbar(mfcc.get_delta_mfcc(mel_spec, 2))
```



作用

- 模拟人耳对不同频率的声音不同的感知能力
- 1000HZ以下感知能力和频率呈线性关系
- 1000HZ以上呈对数关系

问题

- 无

发现

- 通过修改mel滤波器的个数可以控制声谱图的纵向“分辨率”
- 对低频具有一定的放大作用，高频有一定的压缩作用（类似于“归一化”）

个人理解

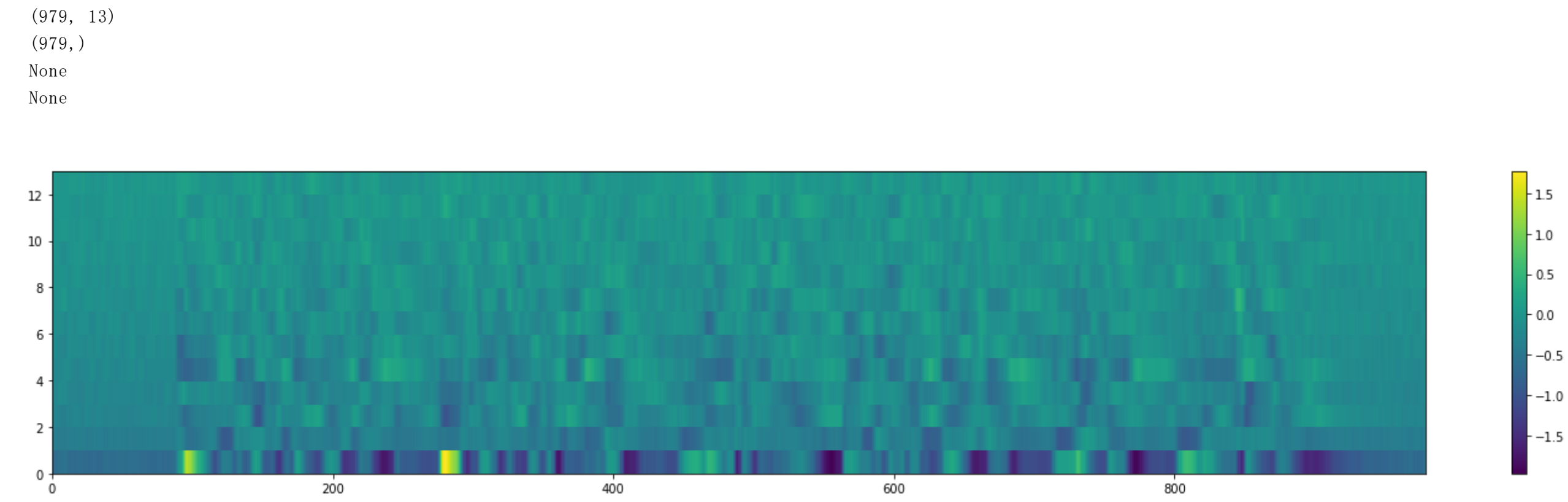
- 与声谱图相比，降维、滤波、映射成更符合听觉感知的频谱图

PLP

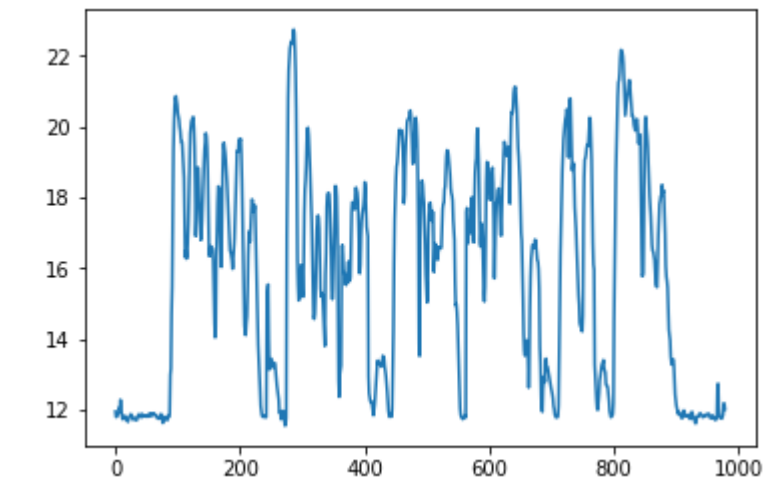
```
In [54]: import feature_extraction.plp as plp

e:\anaconda3\envs\mypython\lib\site-packages\h5py\_init_.py:36: FutureWarning: Conversion of the second argument of issubdtype from 'float' to 'np.floating' is deprecated. In future, it will be treated as 'np.float64 == np.dtype(float).type'.
  from ._conv import register_converters as _register_converters
WARNING:root:WARNING: libsvm is not installed, please refer to the documentation if you intend to use SVM classifiers
```

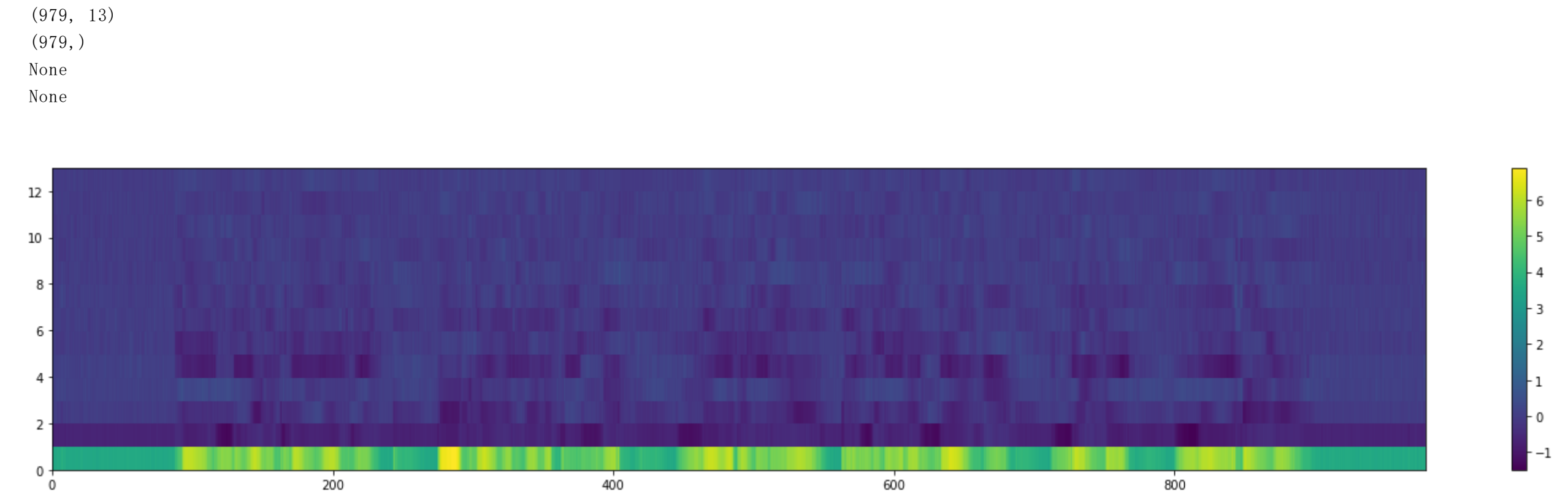
```
In [55]: plp_ = plp.get_plp(audio, sampling_freq)
        for i in range(len(plp_)):
            if plp_[i] is None:
                print("None")
            else:
                print(plp_[i].shape)
        drawer_utils.plot_colormesh(np.transpose(plp_[0]))
```



```
In [56]: drawer_utils.plot_array(plp_[1])
```



```
In [57]: plp_ = plp.get_plp(audio, sampling_freq, rasta=False)
        for i in range(len(plp_)):
            if plp_[i] is None:
                print("None")
            else:
                print(plp_[i].shape)
        drawer_utils.plot_colormesh(np.transpose(plp_[0]))
```



作用

- PLP在LPC的基础上加上了Bark域滤波，让LPC更适用于人耳的听觉习惯
- RASTA是在PLP的基础上加上RASTA滤波器，让模型对4Hz左右频率更敏感，更符合我们的正常说话的感觉

问题

- 无

发现

- 单纯的PLP特征过于凸显，两极分化严重
- 加入RASTA对提高低频作用，降低高频作用，看起来更合理。

个人理解

- 加入RASTA更符合人的听觉系统，归一化的同时还滤除掉人耳不敏感的地方，加强了敏感的地方。

Hilbert-Huang Transform

```
In [58]: imfs , ht = pretreatment.hht(audio)
        print(imfs.shape, ht.shape)

Mode 1, iteration 600
3 minima > 0; 41873 maxima < 0.
Mode 1, iteration 800
1 minima > 0; 42135 maxima < 0.
Mode 1, iteration 1000
1 minima > 0; 42309 maxima < 0.
Mode 2, iteration 1200
5 minima > 0; 25275 maxima < 0.
Mode 2, iteration 1400
2 minima > 0; 25620 maxima < 0.
Mode 2, iteration 1600
1 minima > 0; 25802 maxima < 0.
Mode 2, iteration 1800
1 minima > 0; 25894 maxima < 0.

e:\anaconda3\envs\mypython\lib\site-packages\pyhht\emd.py:368: UserWarning: Emd:warning, Forced stop of sifting - Maximum iteration limit reached.
  "Maximum iteration limit reached.")

(4, 157000) (4, 157000)
```

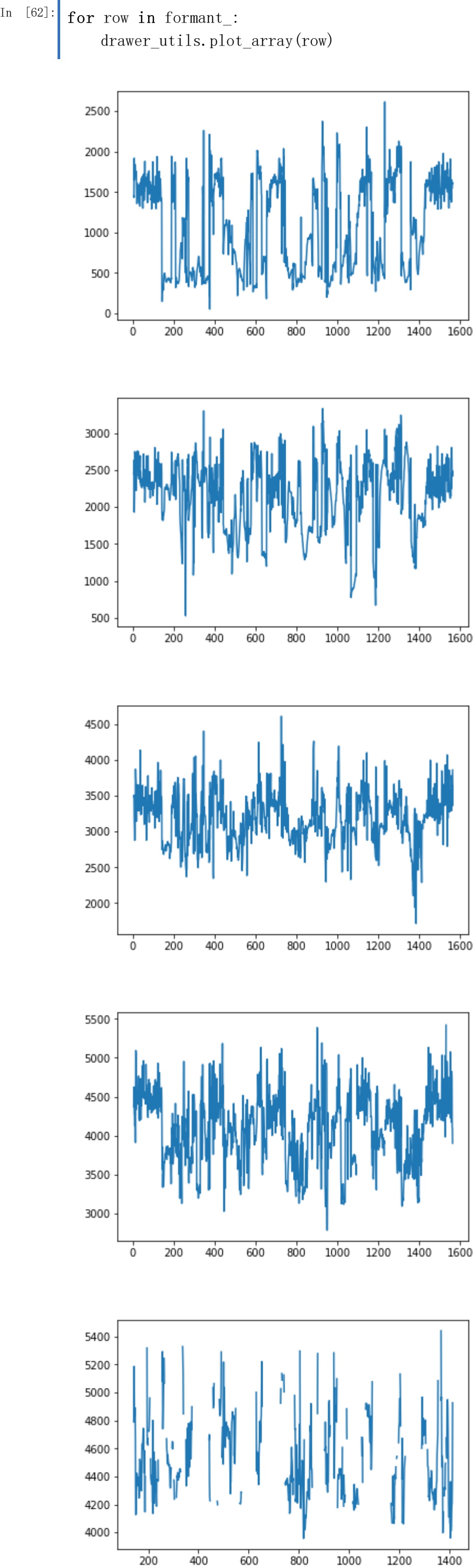
共振峰

```
In [59]: import feature_extraction.formant as formant
```

```
In [60]: formant_ = formant.get_formant(wave_file)
```

```
In [61]: print(formant_.shape)
```

(5, 1563)



**作用**

- 描述人类声道中的共振情形，不同的共振峰代表着不同的作用。
- 基频代表声带振动，共振峰代表声道振动

**问题**

- #### 发现
- 从分布来讲F1的分布方差较大，后面的差异性变小
- F5开始出现“空缺”的现象（值为0），表明往后分析价值较低，频域分解有限

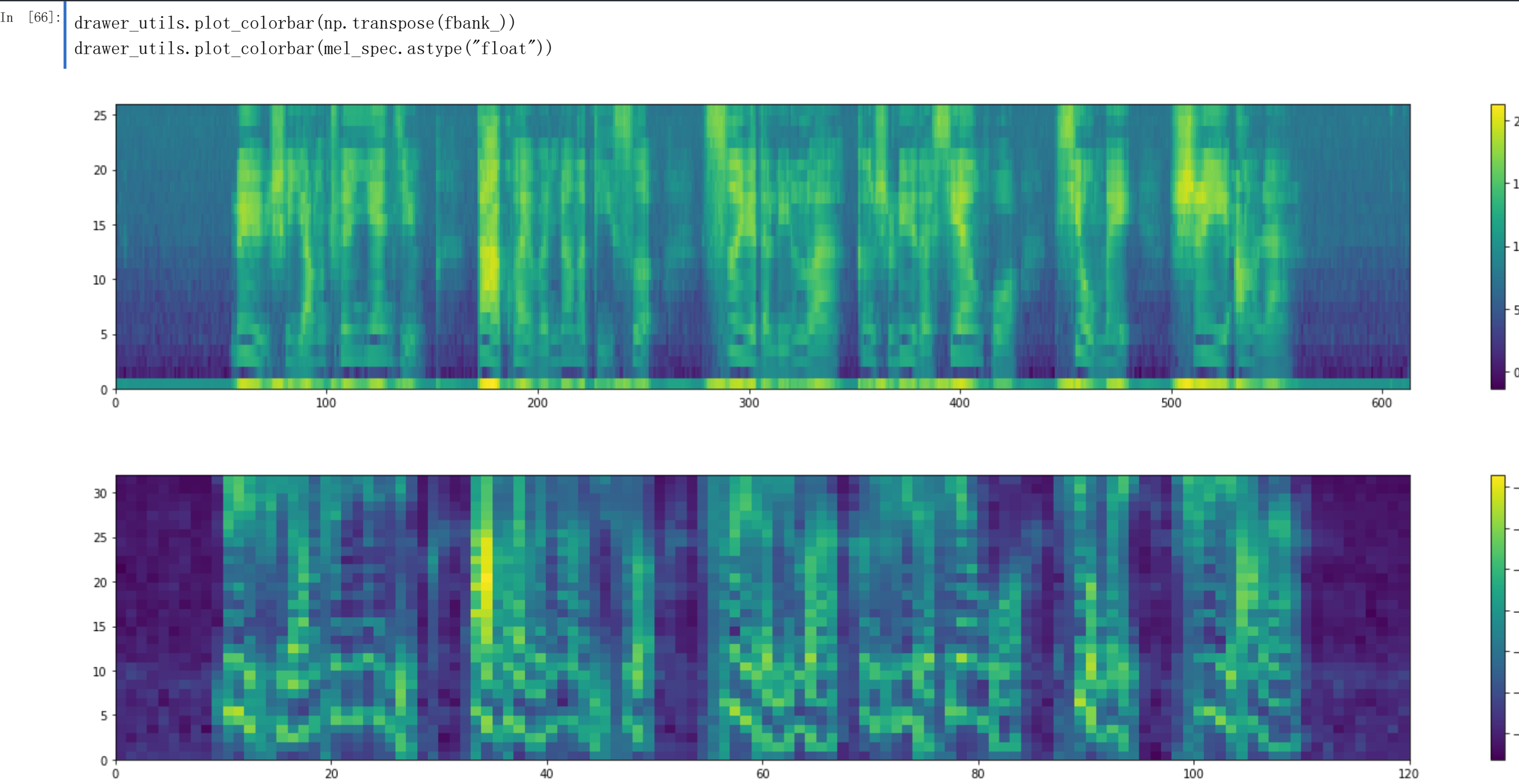
**个人理解**

- 差异性越大越有代表性。

Fbank

```
In [63]: import feature_extraction.fbank as fbank
In [64]: fbank_ = fbank.calcFbank(hanning_frames, sample_rate=sampling_freq)
In [65]: print(fbank_.shape)
```

(613, 26)



**作用**

- Fbank提取更多的是希望符合声音信号的本质，拟合人耳接受的特性

**问题**

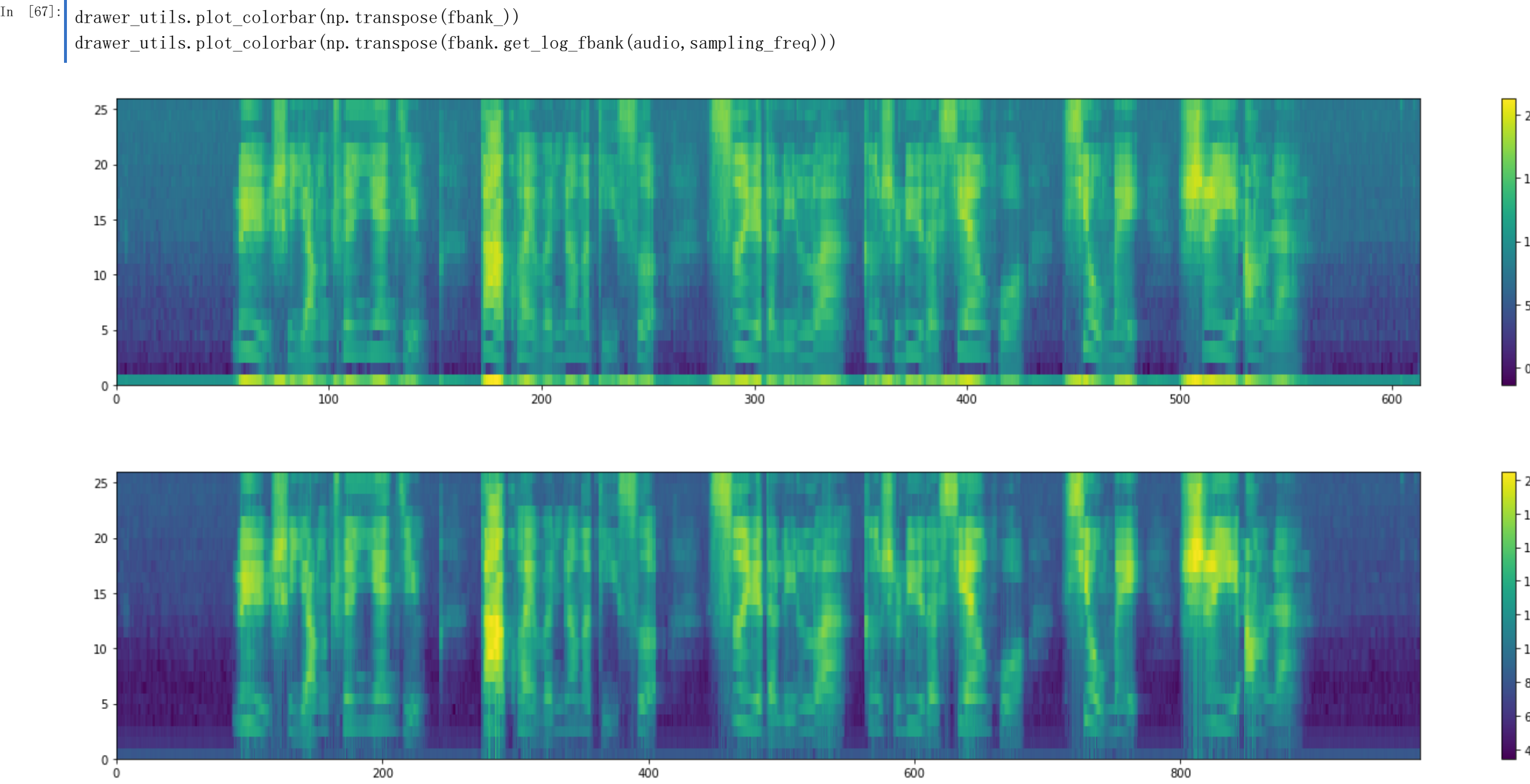
**发现**

- 与MFCC（下图）相比，明显相关性较强，颜色偏亮表示，值都偏大，包括“空白语音”部分，颜色也是从下往上渐变。

**个人理解**

- MFCC不一定是最优选择，因为神经网络对高度相关的信息不敏感，而且DCT变换是线性的，会丢失语音信号中原本的一些非线性成分。

Log Fbank



**作用**

- Fbank加了个log

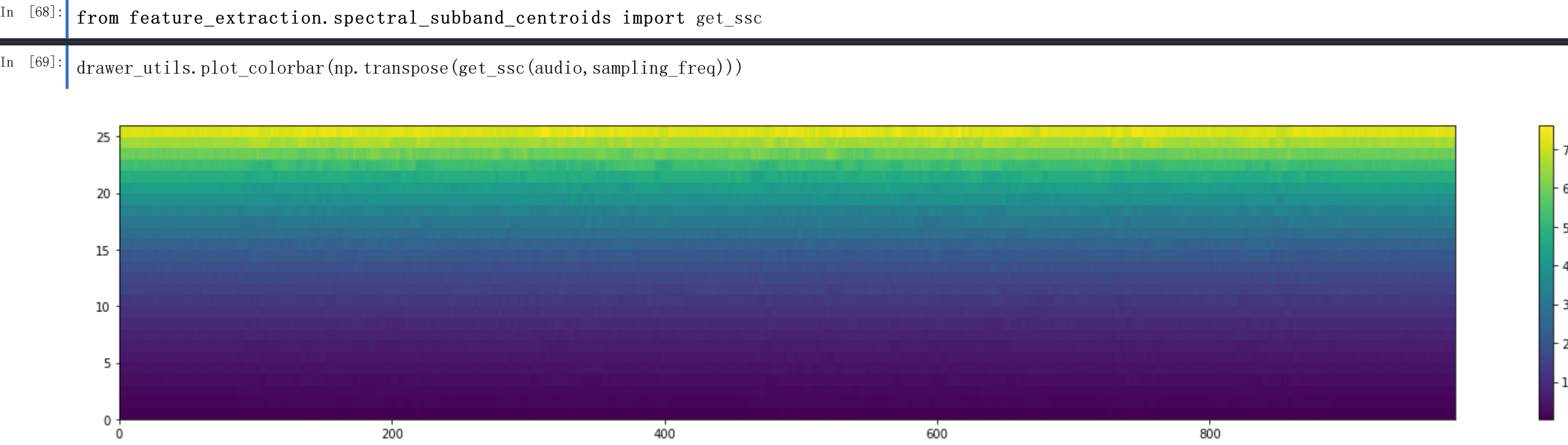
**问题**

**发现**

- 压缩，并降低了高频能量

**个人理解**

- 减弱相关性

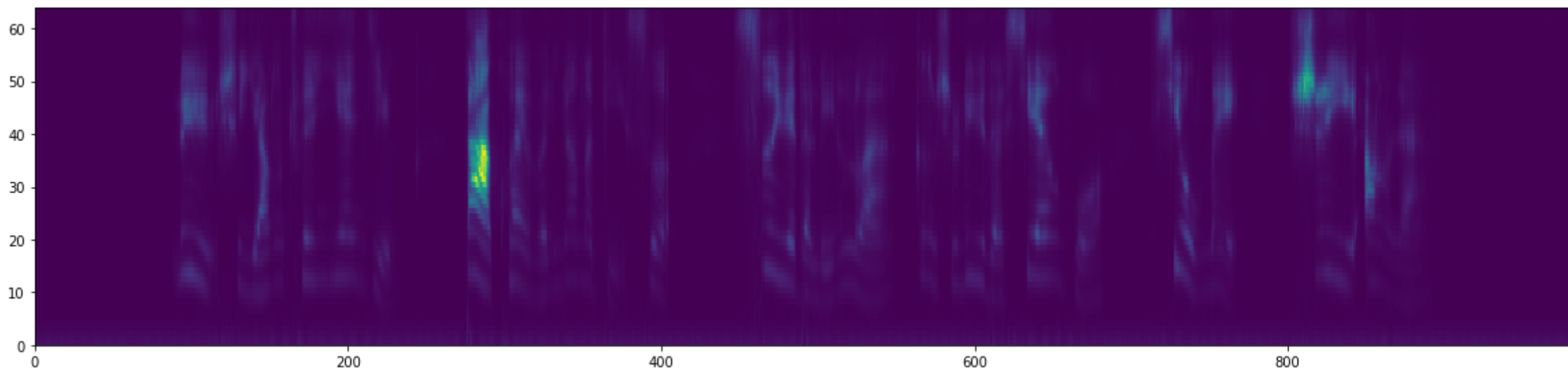


GFCC



```
In [77]: from feature_extraction.gammatone_gram import gammatonegram

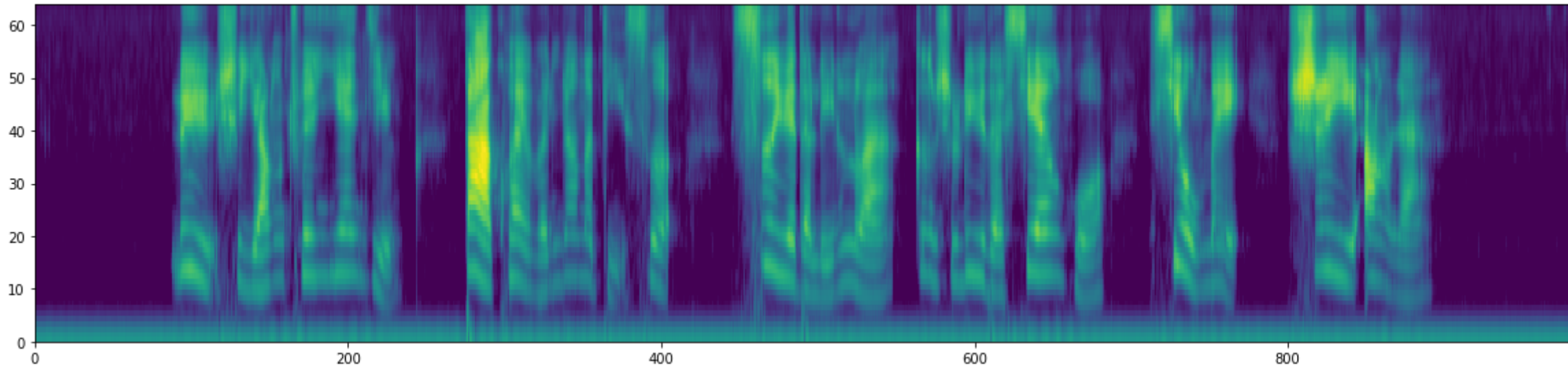
In [78]: sxx, center_freq = gammatonegram(
        audio, sr=sampling_freq, fmin=20, fmax=int(sampling_freq/2)
        )
        drawer_utils.plot_colormesh(sxx)
```



```
In [77]: from feature_extraction.gfcc import get_gfcc

In [78]: gfcc_, center_freq = get_gfcc(audio, sampling_freq)

In [79]: drawer_utils.plot_colormesh(gfcc_)
```



```
In [80]: drawer_utils.plot_colormesh(-np.log(np.abs(gfcc_)))
```

