Task4

Step 1.Pre-emphasis of the speech signal x with windowing and FFT processing.The window function is a Hamming window with a length of 320. the sound sampling frequency is 8000Hz, and the FFT length defaults to 65536.
Step 2.Taking the inverse spectrum of X (k).
Step 3.Window-added to the cepstrum signal.
Step 4.Find the envelope and find the extreme values on the envelope to obtain the corresponding resonance peak parameters.
Step 5.Comparing known resonance peaks with files of speech.
Step 6.Output string.

```
from scipy.signal import lfilter
import librosa
import numpy as np
import matplotlib.pyplot as plt


def local_maxium(x):
    """
    Find the extreme value of a sequence
    :param x:
    :return:
    """
    d = np.diff(x)
    l_d = len(d)
    maxium = []
    loc = []
    for i in range(l_d - 1):
        if d[i] > 0 and d[i + 1] <= 0:
            maxium.append(x[i + 1])
            loc.append(i + 1)
    return maxium, loc


def Formant_Cepst(u, cepstL):
    """
    Resonance peak estimation function by inverse spectroscopy
    :param u:Input signal
    :param cepstL:Width of the window function on frequency
    :return: val resonance peak amplitude
    :return: loc resonance peak position
    :return: spec envelope
    """
    wlen2 = len(u) // 2
    u_fft=np.fft.fft(u)                          # Step 1
    U = np.log(np.abs( u_fft[:wlen2]))
    Cepst = np.fft.ifft(U)                       # Step 2
    cepst = np.zeros(wlen2, dtype=np.complex)
    cepst[:cepstL] = Cepst[:cepstL]              # Step 3
```

```
    cepst[-cepstL + 1:] = Cepst[-cepstL + 1:]   #Take the opposite of the second
equation
    spec = np.real(np.fft.fft(cepst))
    val, loc = local_maxium(spec)               #Finding extreme values on the
envelope
    return val, loc, spec


def voweldetector(wavfile):
    path1="data_vowel_a_.wav"
    path2="data_vowel_ae.wav"
    path3=wavfile

    #sr=None Sound maintains original sampling frequency,  mono=False Sound
maintains original number of channels
    data1, fs1 = librosa.load(path1, sr=None, mono=False)
    data2, fs2 = librosa.load(path2, sr=None, mono=False)
    data3, fs3 = librosa.load(path3, sr=None, mono=False)

    # Pre-treatment - pre-emphasis
    u_1 = lfilter([1, -0.99], [1], data1)
    u_2 = lfilter([1, -0.99], [1], data2)
    u_3 = lfilter([1, -0.99], [1], data3)

    cepstL = 7
    wlen1 = len(u_1)
    wlen2 = len(u_2)
    wlen3 = len(u_3)
    wlenn1 = wlen1 // 2
    wlenn2 = wlen2 // 2
    wlenn3 = wlen3 // 2

    # Pre-treatment - window-added
    freq1 = [i * fs1 / wlen1 for i in range(wlenn1)]
    freq2 = [i * fs2 / wlen2 for i in range(wlenn2)]
    freq3 = [i * fs3/ wlen3 for i in range(wlenn3)]

    #val （resonance peak amplitude）, loc （resonance peak position）, spec
（envelope）
    val1, loc1, spec1 = Formant_Cepst(u_1, cepstL)
    val2, loc2, spec2 = Formant_Cepst(u_2, cepstL)
    val3, loc3, spec3 = Formant_Cepst(u_3, cepstL)


    #Resonance peak frequency
    f_a = [freq1[loc1[0]],freq1[loc1[1]]]
    f_ae =[freq2[loc2[0]],freq2[loc2[1]]]
    f_unk =[freq3[loc3[0]],freq3[loc3[1]]]

    if(f_unk == f_a):
        return "a"
    if(f_unk == f_ae):
        return"ae"
    else:
        return "unknown"
```

```
if __name__ == '__main__':
    wavfile = 'data_vowel_ae.wav'
    ReturnValue = voweldetector(wavfile)
    print("The vowel is "+ReturnValue+"  according to the vowel detector")
```

```
The vowel is ae  according to the vowel detector
```