```
clear;
% Author: Gustav Kjellberg
```

Lab 2 - Feature extraction

First, let's read the data from the three different tracks.

```
[Y1, FS1] = audioread('Songs/melody_1.wav');
[Y2, FS2] = audioread('Songs/melody_2.wav');
[Y3, FS3] = audioread('Songs/melody_3.wav');
```

Then, set the length of the window, 0.03 is default.

```
lengthOfWindow = 0.03
```

```
lengthOfWindow = 0.0300
```

Use the data yielded from audioread and extract the 3 features

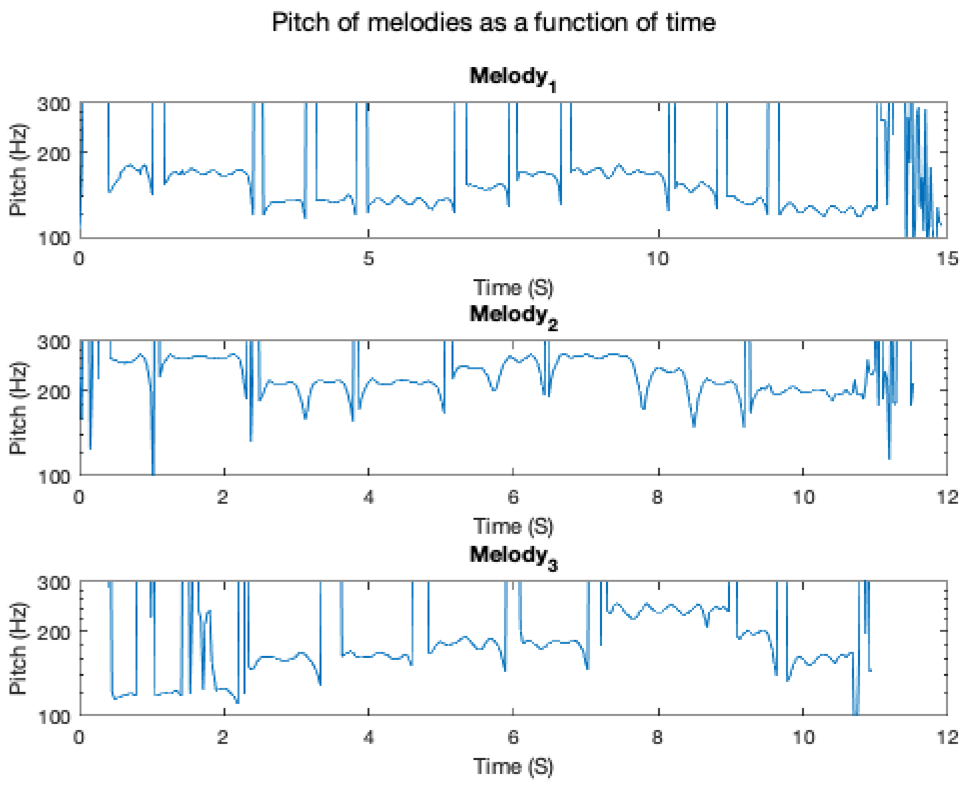
- Pitch: frIseq(1,:) 80 - 1100 Hz
- Correlation coefficient (ρ): frIseq(2,:)
- Esitmates of per-sample intensity: frIseq(3,:)

```
frIseqT1 = GetMusicFeatures(Y1,FS1,lengthOfWindow);
frIseqT2 = GetMusicFeatures(Y2,FS2,lengthOfWindow);
frIseqT3 = GetMusicFeatures(Y3,FS3,lengthOfWindow);
```

Plot the frequencies for the three different signals. Note that two melodies are from the same song and should thus be similair.

```
figure;
sgtitle('Pitch of melodies as a function of time');
fig1 = subplot(3,1,1);
t_1 = 0.00:lengthOfWindow:(length(frIseqT1(1,:))-1)*0.03;
plot(t_1, frIseqT1(1,:));
xlabel('Time (S)');
ylabel('Pitch (Hz)');
title('Melody_1')
```

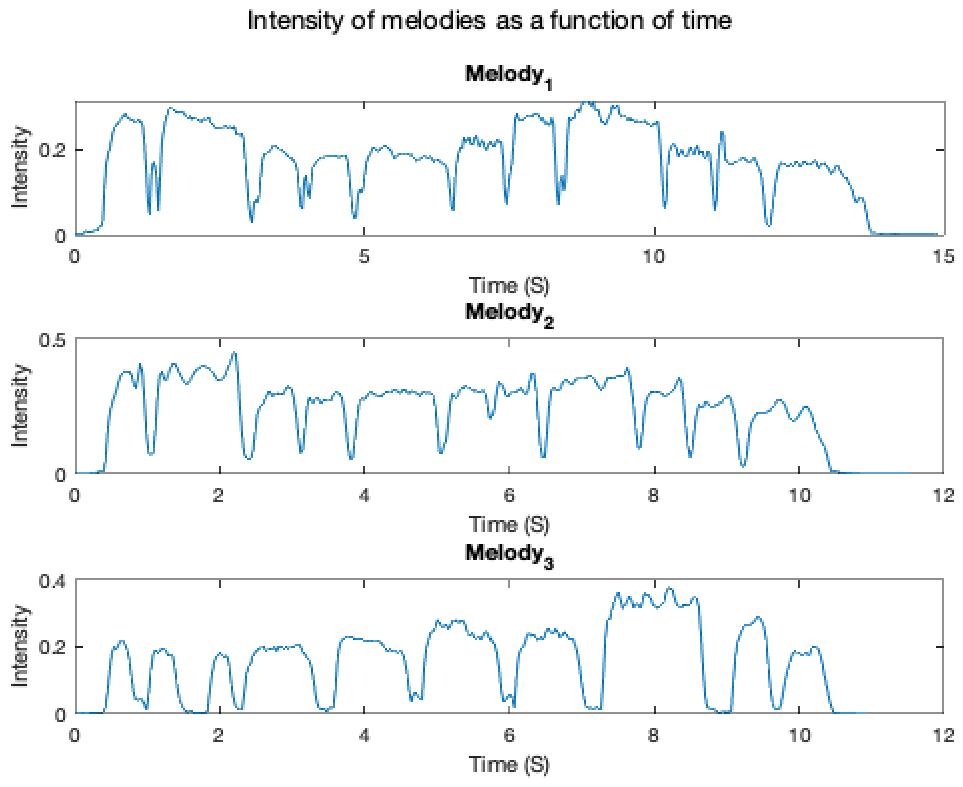
```
fig2 = subplot(3,1,2);
t_2 = 0.00:lengthOfWindow:(length(frIseqT2(1,:))-1)*0.03;
plot(t_2, frIseqT2(1,:));
xlabel('Time (S)');
ylabel('Pitch (Hz)');
title('Melody_2')

fig3 = subplot(3,1,3);
t_3 = 0.00:lengthOfWindow:(length(frIseqT3(1,:))-1)*0.03;
plot(t_3, frIseqT3(1,:));
xlabel('Time (S)');
ylabel('Pitch (Hz)');
title('Melody_3')
set([fig1 fig2 fig3], 'YScale','log');
set([fig1 fig2 fig3], 'YLim', [100 300]);
```



Let's now plot the per-sample intensity, note that the time is still the same as the above figure.

```
figure;
sgtitle('Intensity of melodies as a function of time');
subplot(3,1,1);
plot(t_1, frIseqT1(3,:));
xlabel('Time (S)');
```

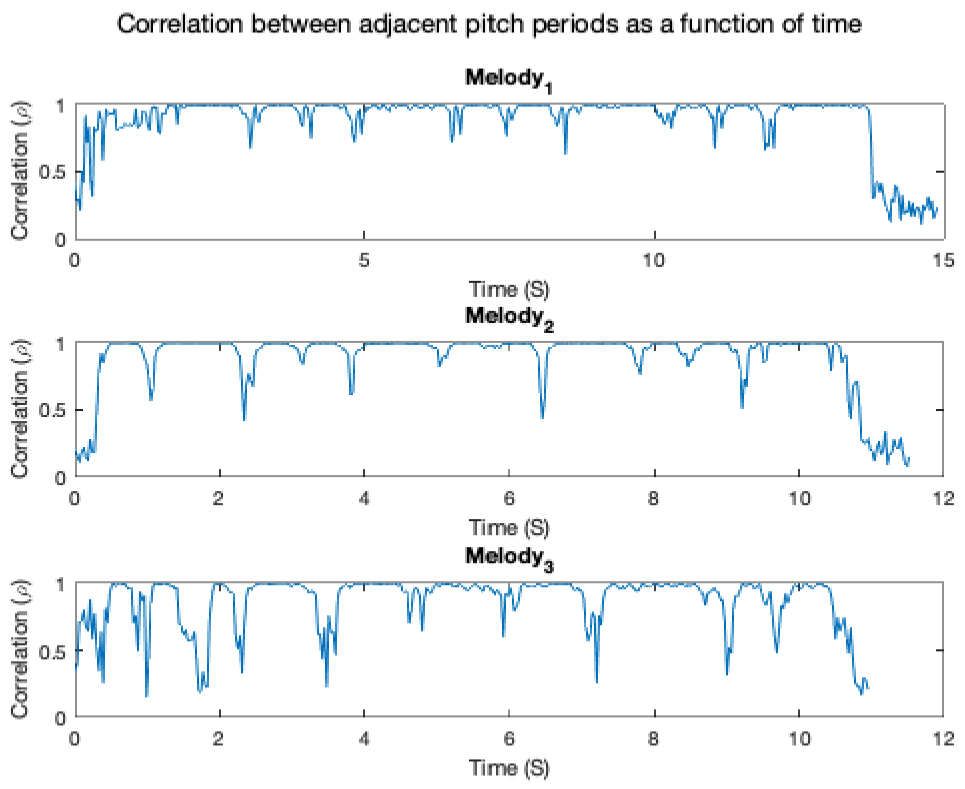
```
ylabel('Intensity');
title('Melody_1')

subplot(3,1,2);
plot(t_2, frIseqT2(3,:));
xlabel('Time (S)');
ylabel('Intensity');
title('Melody_2')

subplot(3,1,3);
plot(t_3, frIseqT3(3,:));
xlabel('Time (S)');
ylabel('Intensity');
title('Melody_3')
```



Finally let's have look at the correlation, note that the time is still the same as the above figure. (Closer to one is better). This should tell who is the better singer.

```
figure;
sgtitle('Correlation between adjacent pitch periods as a function of time');
subplot(3,1,1);
plot(t_1, frIseqT1(2,:));
xlabel('Time (S)');
ylabel('Correlation (\rho)');
title('Melody_1')
```

```
subplot(3,1,2);
plot(t_2, frIseqT2(2,:));
xlabel('Time (S)');
ylabel('Correlation (\rho)');
title('Melody_2')

subplot(3,1,3);
plot(t_3, frIseqT3(2,:));
xlabel('Time (S)');
ylabel('Correlation (\rho)');
title('Melody_3')
```



Let's now use the postprocessing created in order to see if we can extraxt features that fit the hmm.

```
% ---- Uncomment the lines below if you want to verify that it works for
% ---- modifying the pitches.

%frIseqT1 = frIseqT1*3;
%frIseqT2 = frIseqT2*3;
%frIseqT3 = frIseqT3*3;
features_song_1 = Postprocess(frIseqT1);
features_song_2 = Postprocess(frIseqT2);
features_song_3 = Postprocess(frIseqT3);
```

https://se.mathworks.com/help/signal/ref/dtw.html

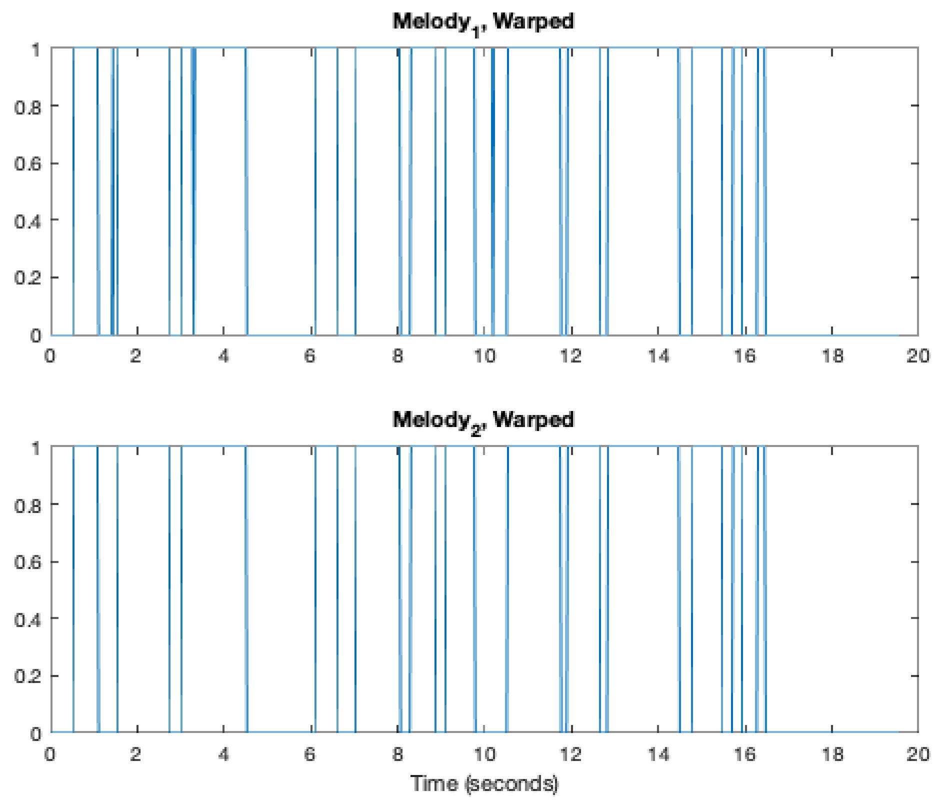
```
[d1, i11, i21] = dtw(features_song_1, features_song_2);
[d2, i12, i22] = dtw(features_song_2, features_song_3);
[d3, i13, i23] = dtw(features_song_1, features_song_3);
```

We see above that the lowest value is actually yielded for melody 1 and 2, that being those that are from the same song. Further, we could try to plot the wrapped signals.

```
a1w = features_song_1(i11);
a2w = features_song_2(i21);

t = (0:lengthOfWindow:(numel(i11)-1)*lengthOfWindow);

duration = t(end)
```

```
duration = 19.5600
```

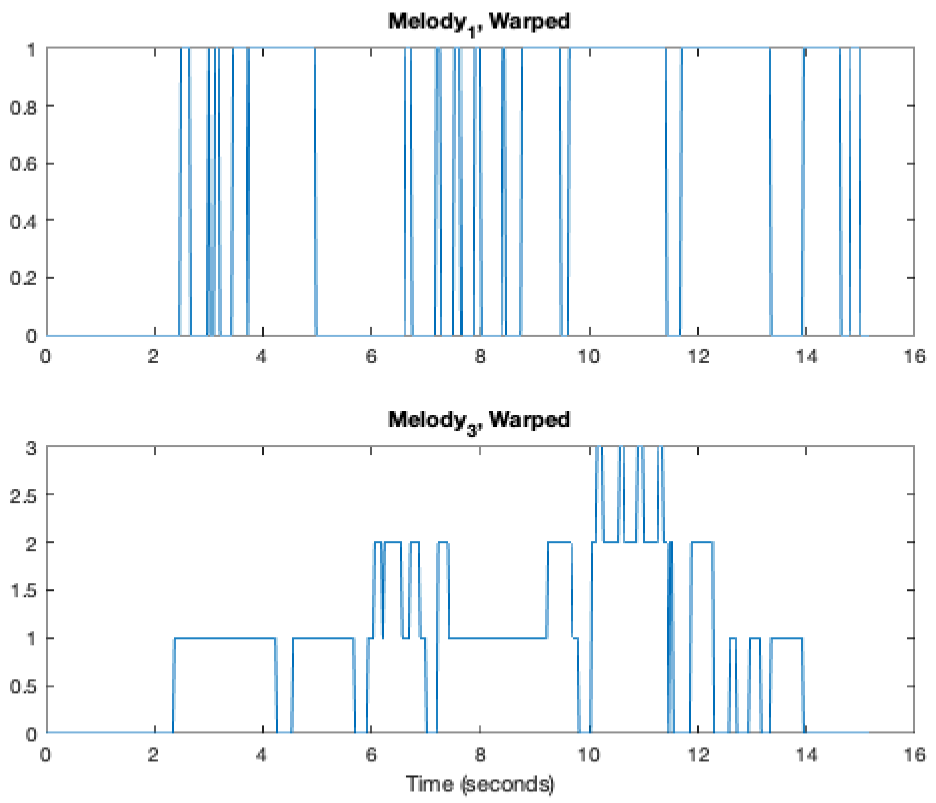
```
figure;
subplot(2,1,1)
plot(t,a1w)
title('Melody_1, Warped')
subplot(2,1,2)
plot(t,a2w)
title('Melody_2, Warped')
xlabel('Time (seconds)')
```



```
a1w = features_song_1(i12);
a2w = features_song_3(i22);

t = (0:lengthOfWindow:(numel(i12)-1)*lengthOfWindow);

duration = t(end)
```

```
duration = 15.1500
```

```
subplot(2,1,1)
plot(t,a1w)
title('Melody_1, Warped')
subplot(2,1,2)
plot(t,a2w)
title('Melody_3, Warped')
xlabel('Time (seconds)')
```

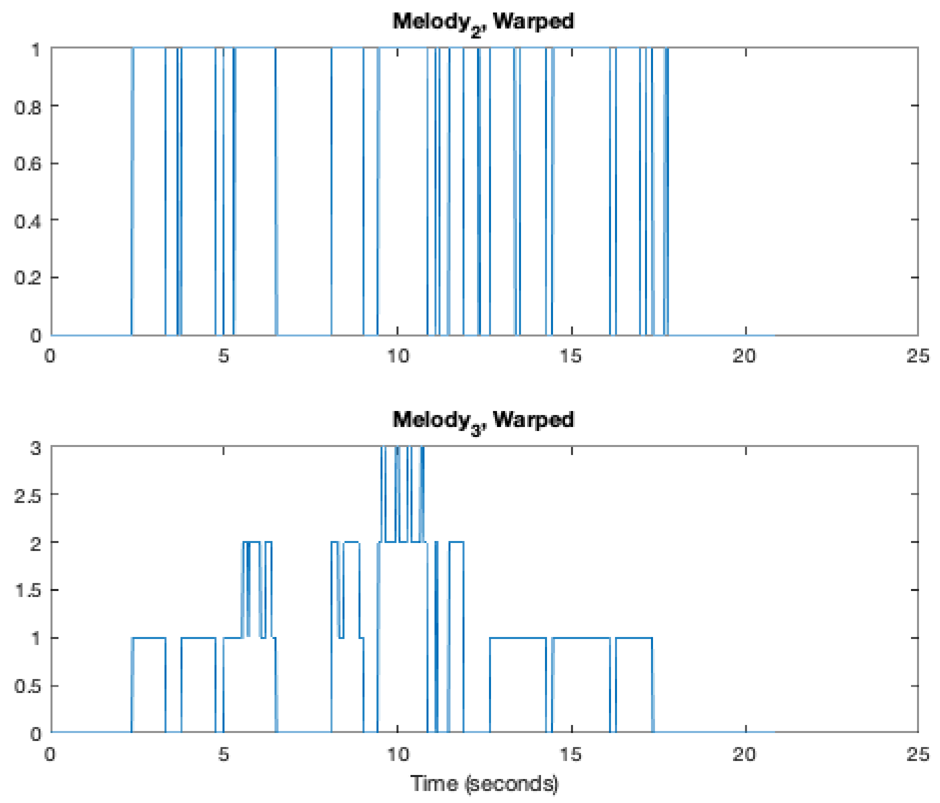


```
a1w = features_song_1(i13);
a2w = features_song_3(i23);

t = (0:lengthOfWindow:(numel(i13)-1)*lengthOfWindow);

duration = t(end)
```

```
duration = 20.8800
```

```
subplot(2,1,1)
```

```
plot(t,a1w)
title('Melody_2, Warped')
subplot(2,1,2)
plot(t,a2w)
title('Melody_3, Warped')
xlabel('Time (seconds)')
```



Finally, let's look at the distance between the signals that the dynamic time warping yielded.

```
disp('---------x = 1.0 ---------')
```

```
---------x = 1.0 ---------
```

```
Distance_Melody1_and_2 = d1
```

```
Distance_Melody1_and_2 = 4
```

```
Distance_Melody1_and_3 = d2
```

```
Distance_Melody1_and_3 = 138
```

```
Distance_Melody2_and_3 = d3
```

```
Distance_Melody2_and_3 = 141
```

As we can clearly see, the distance between the signals of the first and second melody in lower than that of melody 1 and 3 and significantly lower than that of melody 2 and 3. Further, by visual inspection of the graphs above, it is clear that melody 1 & 2 are more simililair in shape than the others.

Conclusion

By taking the logarithmic value of the frequency and intensity but leaving the relation values alone, we can calculate the mean and variance of the three features. By visualizing the data as a normal-distribution, we assume that we can use the mean and variance to identify noise and silent segments. This is done by marking values of frequency that falls outside of the interval $[\overline{frequency} - std(frequency), \overline{frequency} + std(frequency)]$ as noise, since these don't match the more common values and are thus regerded as outliers. Further, we mark the relationship values that falls below the mean as outliers (noise) and the same for intensity. As a stronger relation is better, we want to keep those values. As mentioned in the assignment, the model should not be sensitive to increased loadness (intensity), therefore we chose not to set an upper bound for outliers there.

There are a few properties defined in the assignment that the feature extractor must have

1. The possibility to distinguish between diffrent melodier. - This is fulfilled as shown by warped melodies above.
2. Should allow to distinguish between note sequences with the same pitch track, but where note or pause durations differ. - This can be seen in how the time warped plots if melody_1 and melody_2 is very similair.
3. The features are created by utalizing the Twelfth root of two, which looks at the quotient between the base frequency (f_0) and any other frequency at time t (f_t) $12 * log_2(\frac{f_t}{f_0})$ so any scalar transformation wont have effect.
4. Noises are filtered out with a hueristic that identifies outliers in the data and sets all those datapoints to 0, as the chosen feature output is discrete to out HMM.

Robustness

To verify that the model (3) above, let us look at some transformations, where we multiply all values in melody_1's vector with a value x.

x = 0.5

```
features_song_1 = Postprocess(frIseqT1*0.5);
features_song_2 = Postprocess(frIseqT2);
features_song_3 = Postprocess(frIseqT3);

[d1, i11, i21] = dtw(features_song_1, features_song_2);
[d2, i12, i22] = dtw(features_song_2, features_song_3);
[d3, i13, i23] = dtw(features_song_1, features_song_3);
disp('---------x = 0.5---------')
```

```
---------x = 0.5---------
```

```
Distance_Melody1_and_2 = d1
```

```
Distance_Melody1_and_2 = 44
```

```
Distance_Melody1_and_3 = d2
```

```
Distance_Melody1_and_3 = 138
```

```
Distance_Melody2_and_3 = d3
```

```
Distance_Melody2_and_3 = 107
```

x = 2 (one octave jump)

```
features_song_1 = Postprocess(frIseqT1*2);
features_song_2 = Postprocess(frIseqT2);
features_song_3 = Postprocess(frIseqT3);

[d1, i11, i21] = dtw(features_song_1, features_song_2);
[d2, i12, i22] = dtw(features_song_2, features_song_3);
[d3, i13, i23] = dtw(features_song_1, features_song_3);
disp('---------x = 2.0---------')
```

```
---------x = 2.0---------
```

```
Distance_Melody1_and_2 = d1
```

```
Distance_Melody1_and_2 = 1
```

```
Distance_Melody1_and_3 = d2
```

```
Distance_Melody1_and_3 = 138
```

```
Distance_Melody2_and_3 = d3
```

```
Distance_Melody2_and_3 = 130
```

x = 2.5

```
features_song_1 = Postprocess(frIseqT1*2.5);
features_song_2 = Postprocess(frIseqT2);
features_song_3 = Postprocess(frIseqT3);

[d1, i11, i21] = dtw(features_song_1, features_song_2);
[d2, i12, i22] = dtw(features_song_2, features_song_3);
[d3, i13, i23] = dtw(features_song_1, features_song_3);
disp('---------x = 2.5---------')
```

```
---------x = 2.5---------
```

```
Distance_Melody1_and_2 = d1
```

```
Distance_Melody1_and_2 = 8
```

```
Distance_Melody1_and_3 = d2
```

```
Distance_Melody1_and_3 = 138
```

```
Distance_Melody2_and_3 = d3
```

```
Distance_Melody2_and_3 = 125
```

```
% Author: Gustav Kjellberg
```

Brief summary of this function.

This function takes the input from GetMusicFeatures and post processes the data so that it may better work with our HMM.

Detailed explanation of this function.

Input:

Array-like (3xT) shape, i.e the data yielded from GetMusicFeatures.

Returns:

Array-like (1xT) shape, i.e the features.

```
function features = Postprocess(musicFeatures)
```

Let first assume that the data we obtain from `GetMusicFeatures` is normally distributed. This is a simple assumption and the hueristic might not be valid but it will hopefully enable us to easier remove nosie and identify silent sections. This will be done by verifying if a datapoint is an outlier or not.

1.Divide and parse the data

The relation between sub-notes does obviously not need to be taken as a log while the both frequency and intensity should be, as stated in the assignment.

```
frequency = log(musicFeatures(1,:));
relation = musicFeatures(2,:);
intensity = log(musicFeatures(3,:));
```

2.Let's now calculate mean and variance in order to utlize that to find outlier that thus oculd be silent or noise sections.

```
frequencyMean = mean(frequency);
frequencyStd = std(frequency);

frequencyUpperBound = frequencyMean+frequencyStd;
frequencyLowerBound = frequencyMean-frequencyStd;
frequencyNormalDist = makedist('Normal', "mu",frequencyMean, "sigma",frequencyStd);
frequencyTargetDist = fitdist(transpose(frequency), "Normal");

relationMean = mean(relation);
relationStd = std(relation);

relationUpperBound = relationMean+relationStd;
relationLowerBound = relationMean-relationStd;
relationNormalDist = makedist('Normal', "mu",relationMean, "sigma",relationStd);
relationTargetDist = fitdist(transpose(relation), 'Normal');
```

```
intensityMean = mean(intensity);
intensityStd = std(intensity);

intensityUpperBound = intensityMean+intensityStd;
intensityLowerBound = intensityMean-intensityStd;
intensityNormalDist = makedist('Normal', "mu",intensityMean, "sigma",intensityStd);
intensityTargetDist = fitdist(transpose(intensity), 'Normal');
```

Starting off by testing ony looking at outliers that are outside of the lower bounds. We never want to look at high relation as noise, since that is good, intensity is unknown to me as of now and the high frequencies might be pitches when singing... Testing to set diffrent bounds in order to see if we get a big difference in results. Changed to that the relation can't be weaker than the mean, which was good and quite reasonalbe that we are hard on that specific feature. I'm not capping intensity as that can be an increase in volume.

```
noiseOrQuite = (frequency<frequencyLowerBound)|(frequency>frequencyUpperBound)|(relatio

nonNoiseIndecies = find(frequency == 0);
```

Not to find the lower octave, let's try to find the smallest frequency value that isn't calssified as noise:

```
f_0 = min(frequency(find(noiseOrQuite == 0)));
```

Calculate features, this builds on Twelfth root of two

```
features = 12*log2(frequency/f_0);
```

Set the noisy regions to zero

```
features(find(noiseOrQuite==1)) = 0;
```

Transform the contignoues values into scalars:

```
features = round(features);



end
```