all librosa code and feature explanation from the [CCRMA Music Information Retrieval summer intensive github repo (https://github.com/bmcfee/stanford-mir)](https://github.com/bmcfee/stanford-mir)

In [104]:
```
%matplotlib inline
import numpy, scipy, matplotlib.pyplot as plt, IPython.display as ipd
import librosa, librosa.display
import seaborn as sns
import sklearn
import pandas as pd
plt.rcParams['figure.figsize'] = (14, 5)
```

In [295]:
```
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
```

In [2]:
```
sns.set_style('whitegrid')
```

load audio

load the audio from the "forest" video:

In [3]:
```
sns.set_style('whitegrid')
```

In [4]:
```
x, sr = librosa.load('../../audioFiles/forest.wav')
```

In [5]:
```
ipd.Audio(x, rate=sr)
```

Out[5]:

0:00 / 2:51

define helper functions to normalize

In [205]:
```
# normalize data between 0 and 1 for visualization
def normalize(x, axis=0):
    return sklearn.preprocessing.minmax_scale(x, axis=axis)
```

In [206]:
```
# normalize data to mean of 0 and unit variance
```

extract root mean square energy

In [224]:
```
hop_length = 512
frame_length = 1024
```

In [225]:
```
energy = numpy.array([
    sum(abs(x[i:i+frame_length]**2))
    for i in range(0, len(x), hop_length)
])
```

In [226]:
```
energy.shape
```

Out[226]:
```
(7383,)
```

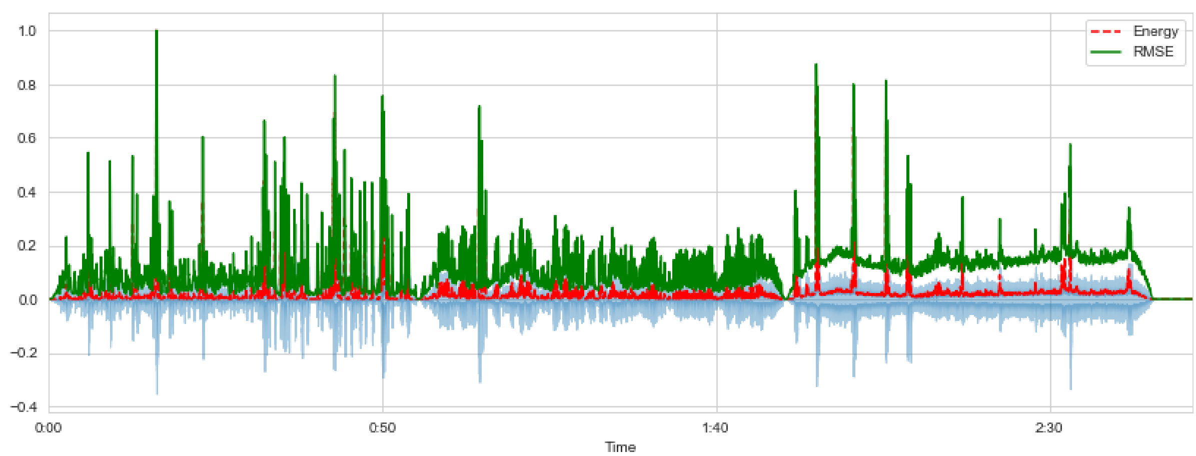
In [227]:
```
rms = librosa.feature.rms(x, frame_length=frame_length, hop_length=hop_l
ength)[0]
```

In [228]:
```
rms.shape
```

Out[228]:
```
(7383,)
```

In [229]:
```
frames = range(len(energy))
t = librosa.frames_to_time(frames, sr=sr, hop_length=hop_length)
```

In [230]:
```
librosa.display.waveplot(x, sr=sr, alpha=0.4)
plt.plot(t, energy/energy.max(), 'r--')             # normalized for vis
ualization
plt.plot(t[:len(rms)], rms/rms.max(), color='g') # normalized for visual
ization
plt.legend(('Energy', 'RMSE'))
plt.show()
```



In [232]:
```
rms = sklearn.preprocessing.scale(rms)
```

In [233]:
```
rms.mean()
```

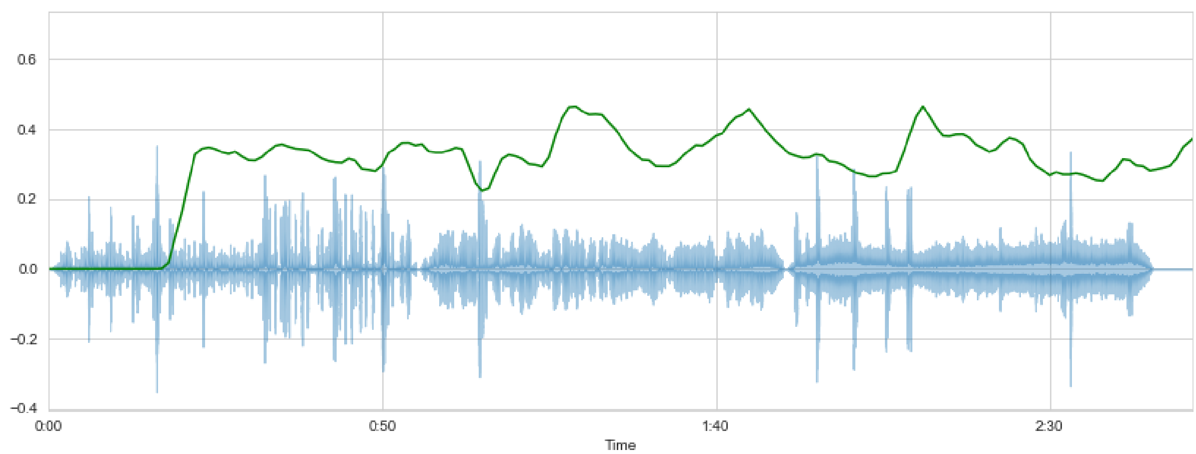
Out[233]:
```
-8.266986e-09
```

In [235]:
```
rms.var()
```
Out[235]: 0.99999994

extract zero line crossing rate

Add a small constant to avoid oscillation around silence triggering high zero-crossing rate:

In [243]:
```
librosa.display.waveplot(x, sr=sr, alpha=0.4)
zcrs = librosa.feature.zero_crossing_rate(x + 0.0001)
plt.plot(zcrs[0], 'g')
plt.show()
```



In [244]:
```
zcrs.shape
```
Out[244]: (1, 7383)

extract spectral features

extract spectral centroid

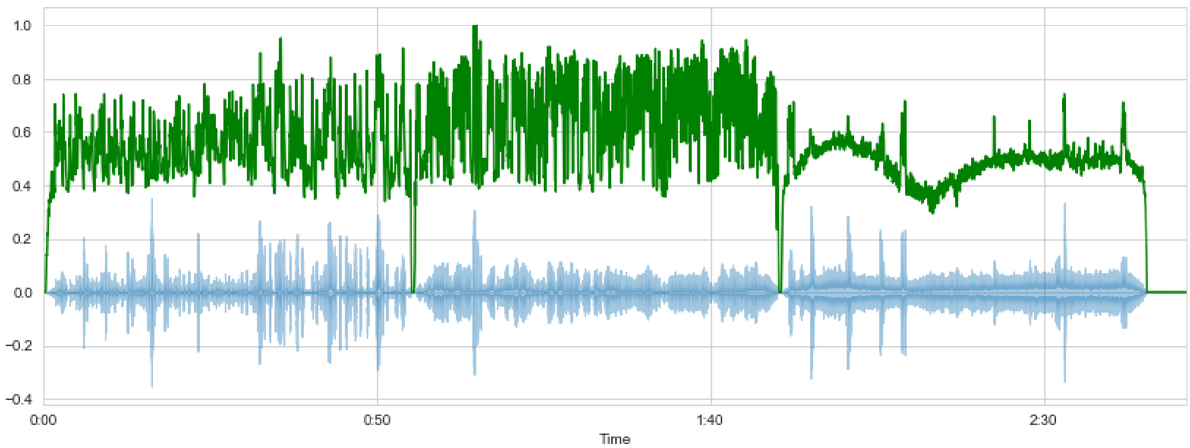
The **spectral centroid** ([Wikipedia (https://en.wikipedia.org/wiki/Spectral_centroid)](https://en.wikipedia.org/wiki/Spectral_centroid)) indicates at which frequency the energy of a spectrum is centered upon. This is like a weighted mean:
$$ f_c = \frac{\sum_k S(k) f(k)}{\sum_k S(k)} $$

where $S(k)$ is the spectral magnitude at frequency bin k, $f(k)$ is the frequency at bin k.

[`librosa.feature.spectral_centroid` (https://librosa.github.io/librosa/generated/librosa.feature.spectral_centroid.html#librosa.feature.spectral_centroid)](https://librosa.github.io/librosa/generated/librosa.feature.spectral_centroid.html#librosa.feature.spectral_centroid) computes the spectral centroid for each frame in a signal:

In [198]:
```
# calculate time variable for plotting
frames = range(len(spectral_centroids))
t = librosa.frames_to_time(frames)
```

In [245]:
```
spectral_centroids = librosa.feature.spectral_centroid(x+0.01, sr=sr)[0]
# add constant to correct value at silence
librosa.display.waveplot(x, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_centroids), color='g') # normalize for vi
sualization purposes
plt.show()
```



In [246]:
```
spectral_centroids = sklearn.preprocessing.scale(spectral_centroids)
```

In [248]:
```
spectral_centroids.mean()
```

Out[248]: 6.159384408593582e-17

In [249]:
```
spectral_centroids.var()
```

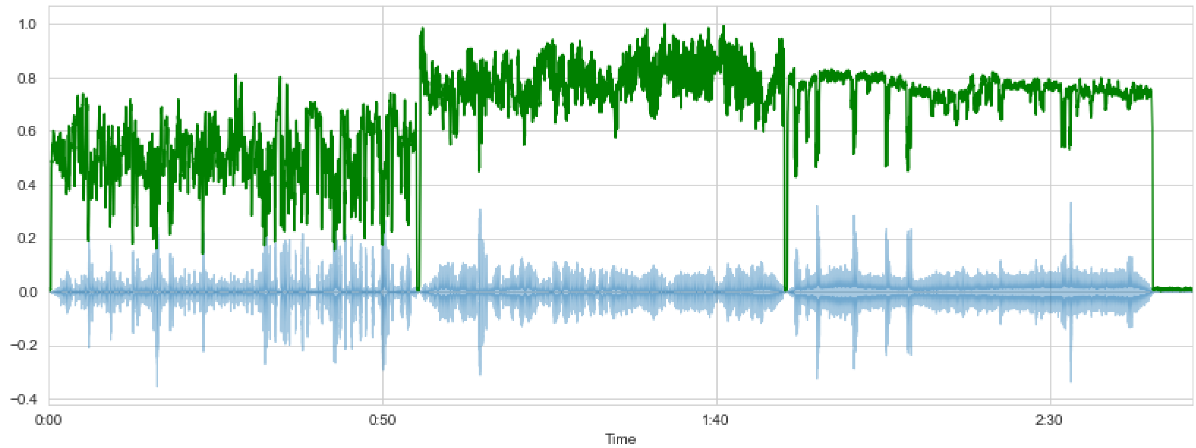
Out[249]: 1.0000000000000002

extract spectral bandwidth

`librosa.feature.spectral_bandwidth` (https://librosa.github.io/librosa/generated/librosa.feature.spectral_bandwidth.html#librosa.feature.spectral_bandw
computes the order-p spectral bandwidth:
$$ \left(\sum_k S(k) \left(f(k) - f_c \right)^p \right)^{\frac{1}{p}} $$

where $S(k)$ is the spectral magnitude at frequency bin k, $f(k)$ is the frequency at bin k, and f_c is the spectral centroid. When $p = 2$, this is like a weighted standard deviation.

In [250]:
```
spectral_bandwidth_2 = librosa.feature.spectral_bandwidth(x+0.01, sr=sr)
[0]
librosa.display.waveplot(x, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_bandwidth_2), color='g')
plt.show()
```



In [251]:
```
spectral_bandwidth_2 = sklearn.preprocessing.scale(spectral_bandwidth_2)
```

In [252]:
```
spectral_bandwidth_2.mean()
```

Out[252]: -1.5398461021483954e-16

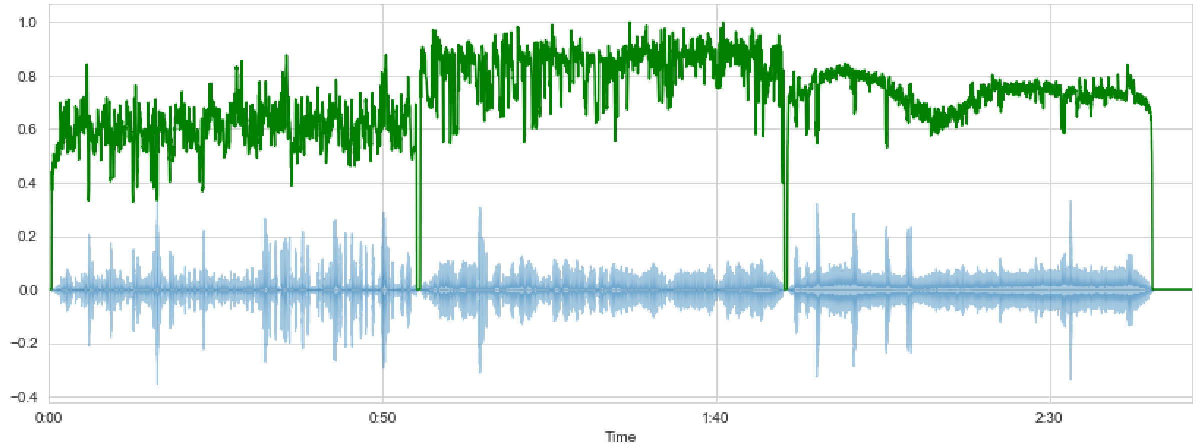
In [253]:
```
spectral_bandwidth_2.var()
```

Out[253]: 1.0000000000000002

extract spectral rolloff

Spectral rolloff is the frequency below which a specified percentage of the total spectral energy, e.g. 85%, lies.

In [254]:
```
spectral_rolloff = librosa.feature.spectral_rolloff(x+0.01, sr=sr)[0]
librosa.display.waveplot(x, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_rolloff), color='g')
plt.show()
```



In [255]:
```
spectral_rolloff = sklearn.preprocessing.scale(spectral_rolloff)
```

In [256]:
```
spectral_rolloff.mean()
```

Out[256]: 1.3858614919335558e-16

In [257]:
```
spectral_rolloff.var()
```

Out[257]: 0.9999999999999997

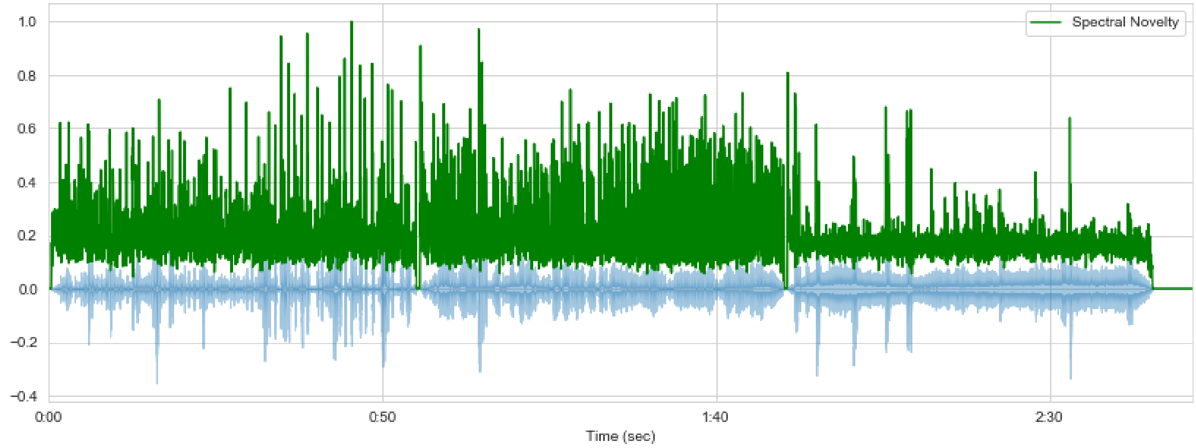
extract spectral novelty

We will compute a **spectral novelty function** ([FMP (https://www.audiolabs-erlangen.de/resources/MIR/FMP/C0/C0.html)](https://www.audiolabs-erlangen.de/resources/MIR/FMP/C0/C0.html), p. 309):

1. Compute the log-amplitude spectrogram.
2. Within each frequency bin, k, compute the energy novelty function as shown earlier, i.e. (a) first-order difference, and (b) half-wave rectification.
3. Sum across all frequency bins, k.

In [258]:
```
spectral_novelty = librosa.onset.onset_strength(x, sr=sr)
```

In [259]:
```
librosa.display.waveplot(x, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_novelty), 'g')
plt.xlim(0, t.max())
plt.xlabel('Time (sec)')
plt.legend(('Spectral Novelty',))
plt.show()
```



In [260]:
```
spectral_novelty = sklearn.preprocessing.scale(spectral_novelty)
```

```
/Users/trqk-data/.local/share/virtualenvs/nature-nurtures--FcyQ97q/lib/
python3.7/site-packages/sklearn/preprocessing/data.py:189: UserWarning:
Numerical issues were encountered when scaling the data and might not b
e solved. The standard deviation of the data is probably very close to
0.
  warnings.warn("Numerical issues were encountered "
```

In [261]:
```
spectral_novelty.mean()
```

Out[261]: 0.0

In [262]:
```
spectral_novelty.var()
```

Out[262]: 1.0000001

Extract Mel Frequency Cepstral Coefficients (MFCCs)

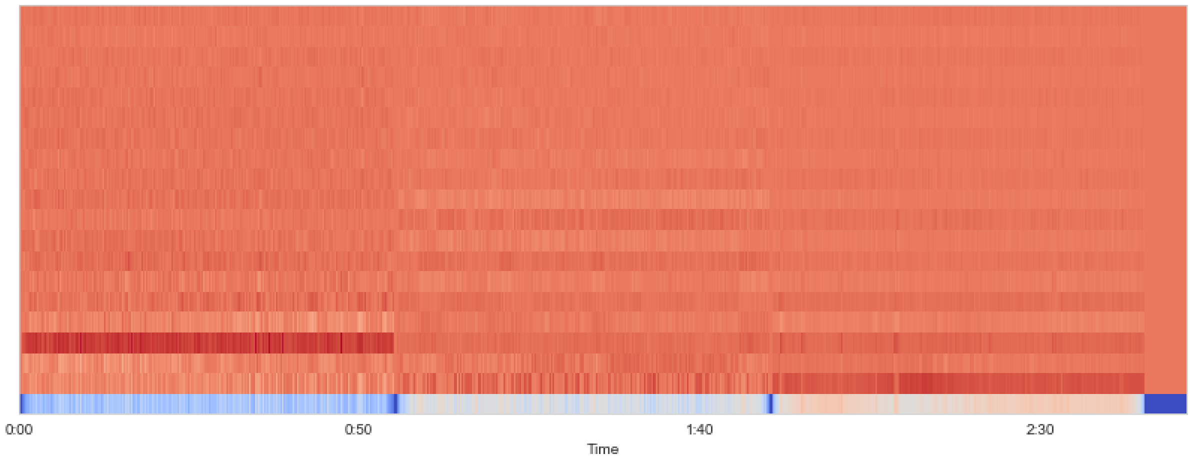
MFCCs

In [263]:
```
mfccs = librosa.feature.mfcc(x, sr=sr)
print(mfccs.shape)
```

```
(20, 7383)
```

display MFCCs:

In [264]:
```
librosa.display.specshow(mfccs, sr=sr, x_axis='time')
plt.show()
```



Let's scale the MFCCs such that each coefficient dimension has zero mean and unit variance:

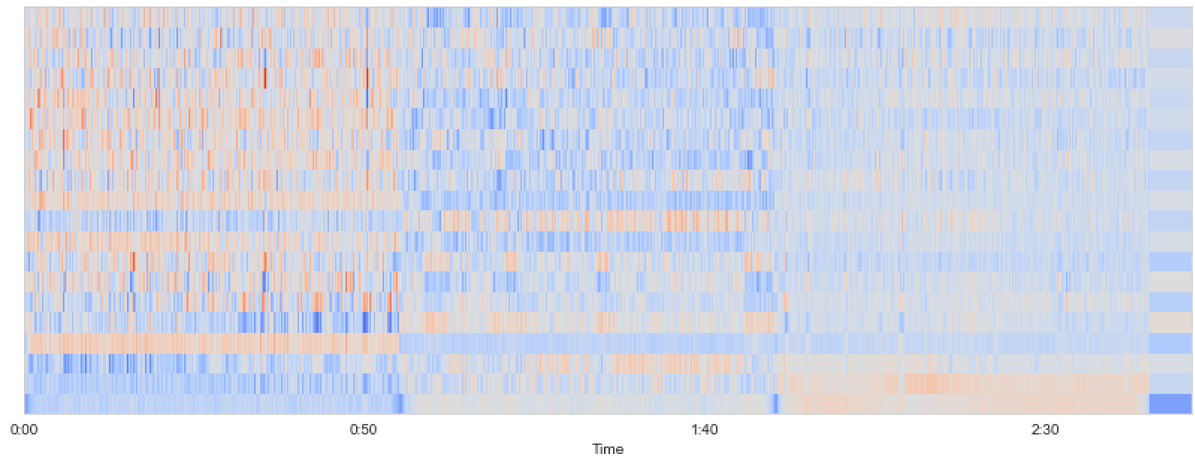
In [265]:
```
mfccs = sklearn.preprocessing.scale(mfccs, axis=1)
print(mfccs.mean(axis=1))
print(mfccs.var(axis=1))
```

```
[ 2.4478028e-08 -1.7526979e-08 -3.9790925e-07 -1.9843995e-07
 -1.8713743e-08 -4.1657859e-09 -1.2973678e-08 -2.5059300e-08
  2.6544775e-08 -3.5457617e-08  2.5083521e-08  3.5360741e-09
  3.8117747e-08 -4.5484569e-08  5.3156153e-08  2.2451648e-08
 -3.4957583e-08 -3.3407019e-08 -1.0148048e-08  5.8999152e-08]
[0.99999684 0.9999978  1.0000142  0.9999928  1.0000222  1.0000057
 1.0000012  1.0000045  0.99999297 0.999995   0.9999906  1.0000012
 1.0000141  0.9999916  0.9999927  1.0000049  0.99999785 1.0000064
 0.99999017 1.0000147 ]

/Users/trqk-data/.local/share/virtualenvs/nature-nurtures--FcyQ97q/lib/
python3.7/site-packages/sklearn/preprocessing/data.py:172: UserWarning:
Numerical issues were encountered when centering the data and might not
be solved. Dataset may contain too large values. You may need to presca
le your features.
  warnings.warn("Numerical issues were encountered "
/Users/trqk-data/.local/share/virtualenvs/nature-nurtures--FcyQ97q/lib/
python3.7/site-packages/sklearn/preprocessing/data.py:189: UserWarning:
Numerical issues were encountered when scaling the data and might not b
e solved. The standard deviation of the data is probably very close to
0.
  warnings.warn("Numerical issues were encountered "
```

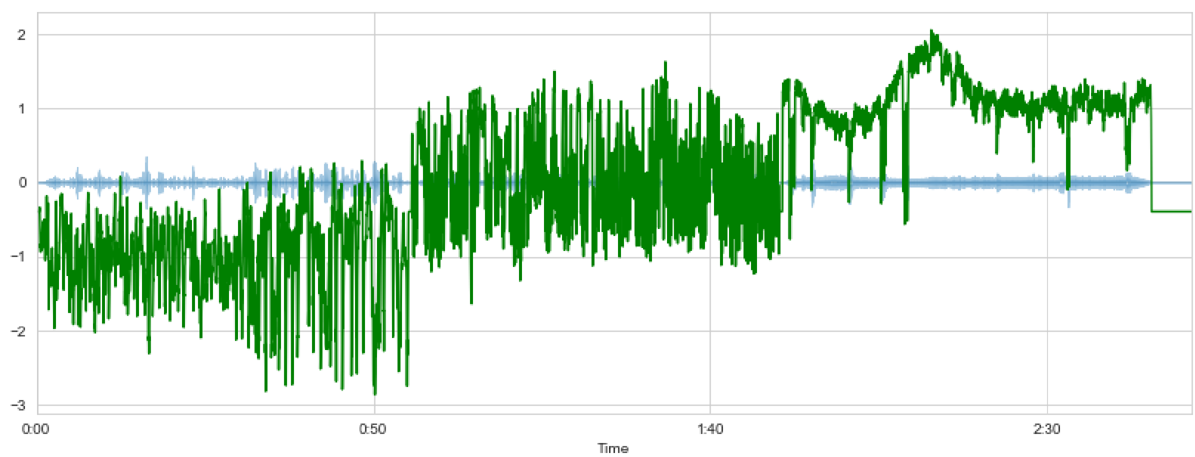
Display all the scaled MFCCs:

In [266]:
```
librosa.display.specshow(mfccs, sr=sr, x_axis='time')
plt.show()
```



Pull out one MFCC:

In [267]:
```
librosa.display.waveplot(x, sr=sr, alpha=0.4)
plt.plot(t, mfccs[1], color='g')
plt.show()
```



Create Feature Vector

Append spectral features to MFCCs as new rows of the array:

In [283]:
```
def add_second_dimension_to_array(the_array):
    cols = the_array.shape[0]
    return numpy.reshape(the_array, (-1, 7383))
```

In [284]:
```
mfccs.shape
```

Out[284]:
```
(20, 7383)
```

In [285]:
```
spectral_features = [spectral_centroids,
spectral_bandwidth_2,
spectral_rolloff,
spectral_novelty,
]
```

In [286]:
```
X = mfccs
for feature in spectral_features:
    X = numpy.append(X, add_second_dimension_to_array(feature), axis=0)
```

In [287]:
```
X.shape
```

Out[287]:
```
(24, 7383)
```

Apply Principal Component Analysis to See Which Features Predict the Signal Most

In [288]:
```
X.mean()
```

Out[288]:
```
-4.024665683990886e-09
```

In [289]:
```
model = sklearn.decomposition.PCA(n_components=2, whiten=True)
```

In [290]:
```
model.fit(X.T)
```

Out[290]:
```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=Non
e,
    svd_solver='auto', tol=0.0, whiten=True)
```

In [291]:
```
Y = model.transform(X.T)
```

In [292]:
```
print(Y.shape)
```

```
(7383, 2)
```

In [293]:
```
model.components_.shape
```

Out[293]:
```
(2, 24)
```

In [296]:
```
pd.DataFrame(model.components_, index = ['PC-1','PC-2'])
```

Out[296]:

	0	1	2	3	4	5	6	7	
PC-1	-0.260838	-0.208214	-0.267103	0.315148	-0.163073	0.004098	0.106031	0.034172	0.3063
PC-2	0.163501	-0.296752	0.049073	0.243706	-0.294038	0.326705	-0.173676	0.278562	-0.0152

Component 1 depends substantially on MFCCs 4, 9, and 11, while component 2 depends largely on MFCCs 4-6, as well as spectral rolloff and spectral novelty.

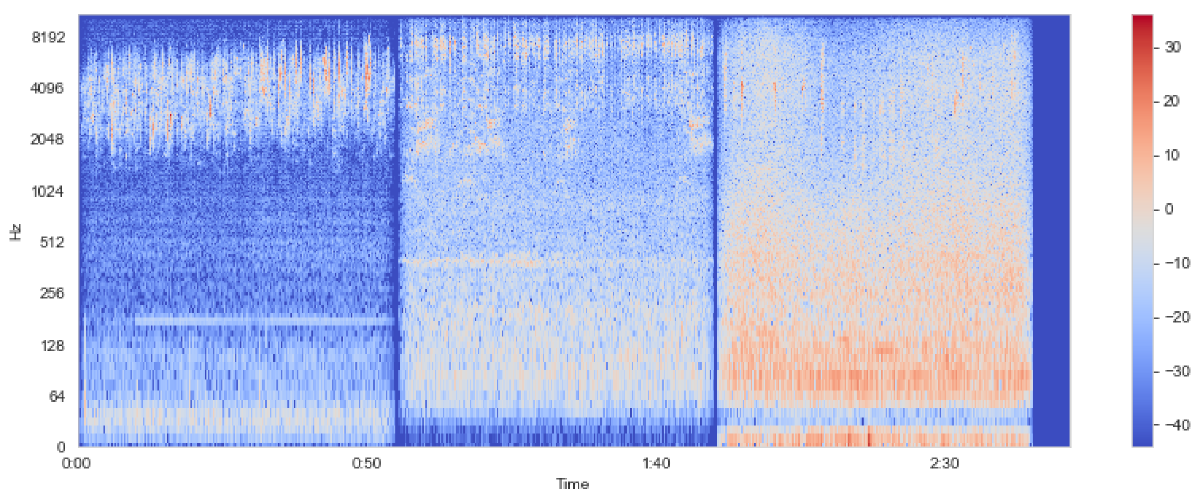
Apply Non-Negative Matrix Factorization to Decribe Signal as a Sum of Components

In [297]:
```
S = librosa.stft(x)
print(S.shape)
```

```
(1025, 7383)
```

In [298]:
```
Smag = librosa.amplitude_to_db(S)
librosa.display.specshow(Smag, sr=sr, x_axis='time', y_axis='log')
plt.colorbar()
plt.show()
```

```
/Users/trqk-data/.local/share/virtualenvs/nature-nurtures--FcyQ97q/lib/python3.7/site-packages/librosa/core/spectrum.py:1700: UserWarning: amplitude_to_db was called on complex input so phase information will be discarded. To suppress this warning, call amplitude_to_db(np.abs(S)) instead.
  warnings.warn('amplitude_to_db was called on complex input so phase '
```



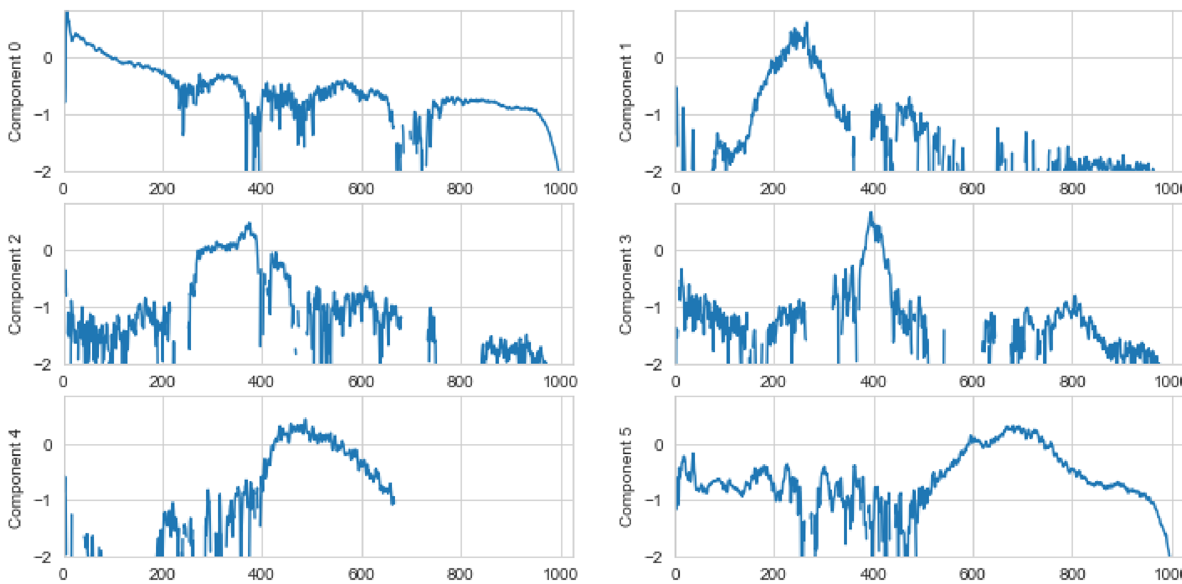
In [113]:
```
X = numpy.absolute(S)
n_components = 6
W, H = librosa.decompose.decompose(X, n_components=n_components, sort=True)
print(W.shape)
print(H.shape)
```

```
(1025, 6)
(6, 7383)
```

Plot each spectral component:

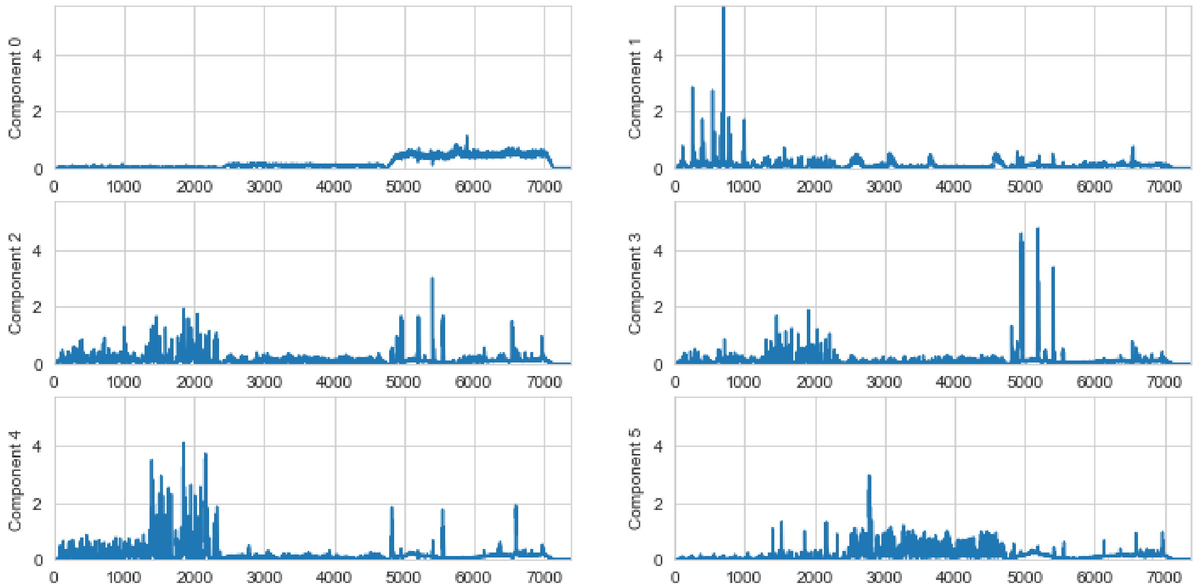
In [114]:
```
plt.figure(figsize=(12, 6))
logW = numpy.log10(W)
for n in range(n_components):
    plt.subplot(numpy.ceil(n_components/2.0), 2, n+1)
    plt.plot(logW[:,n])
    plt.ylim(-2, logW.max())
    plt.xlim(0, W.shape[0])
    plt.ylabel('Component %d' % n)
```

```
/Users/trqk-data/.local/share/virtualenvs/nature-nurtures--FcyQ97q/lib/python3.7/site-packages/ipykernel_launcher.py:2: RuntimeWarning: divide by zero encountered in log10
```



temporal activations of each component:

In [115]:
```
plt.figure(figsize=(12, 6))
for n in range(n_components):
    plt.subplot(numpy.ceil(n_components/2.0), 2, n+1)
    plt.plot(H[n])
    plt.ylim(0, H.max())
    plt.xlim(0, H.shape[1])
    plt.ylabel('Component %d' % n)
```



Listen to components:

In [116]:
```
reconstructed_signal = scipy.zeros(len(x))
for n in range(n_components):
    Y = scipy.outer(W[:,n], H[n])*numpy.exp(1j*numpy.angle(S))
    y = librosa.istft(Y)
    reconstructed_signal[:len(y)] += y
    ipd.display( ipd.Audio(y, rate=sr) )
```

0:00 / 2:51

0:00 / 2:51

0:00 / 2:51

0:00 / 2:51

0:00 / 2:51

0:00 / 0:00