.mp3 as input..

As I was playing with my first Colab notebook, I realized there was a problem with how I was setting up my model. The inputs I was prepared to feed in were the 518 attributes per track in the FMA dataset. In addition to be being a very complex and large input, it's not how I want the model to function.

My goal is to input an `.mp3` file and have the model output a genre prediction. I feel I have not framed my model/problem this way and will start fresh setting it up here.

Information from Audio Files

In the FMA dataset, the 9 features are the following:

Number of Instances	`106,574` *tracks*	
Audio Features *[9 total]*	**Feature Definition**	**Attributes per Feature** *[518 total]*
Chroma	*A quality of pitch class which refers to the "color" of a musical pitch* (src)	84
Tonnetz	*"Tonal grid;" planar array of pitches along three simplicial axes, corresponding to the three consonant interval* (src)	42
Mel Frequency Cepstral Coefficient (MFCC)	*Cepstrum: The info of rate of spectral bands* *Mel scale: scale that relates perceived frequency of a tone to the actual measured frequency* *MFCC features represent distinct units of sound as the shape of the vocal tract* (src)	140
Spectral centroid	*Characterizes spectrum (where center of mass of the spectrum is located). Connected to impression of brightness of sound* (src)	7
Spectral bandwidth	*Determines the resolution of a signal..* (src)	7
Spectral contrast	*Level difference between peaks and valleys in the spectrum* (src)	49
Spectral rolloff	*Measure of the amount of the right-skewedness of the power spectrum* (src)	7
Root Mean Square Energy	...have not found audio-related definition yet, just AC voltage/current here	7
Zero-crossing rate	*The rate of sign-changes along a signal* (src)	7

Extracting these features from an `.mp3` using `Librosa`

Python's `Librosa` library includes many feature extraction methods, including the following spectral features:

`chroma_stft` ([y, sr, S, norm, n_fft, ...])	Compute a chromagram from a waveform or power spectro
`chroma_cqt` ([y, sr, C, hop_length, fmin, ...])	Constant-Q chromagram
`chroma_cens` ([y, sr, C, hop_length, fmin, ...])	Computes the chroma variant "Chroma Energy Normalized"
`melspectrogram` ([y, sr, S, n_fft, ...])	Compute a mel-scaled spectrogram.
`mfcc` ([y, sr, S, n_mfcc, dct_type, norm, lifter])	Mel-frequency cepstral coefficients (MFCCs)
`rms` ([y, S, frame_length, hop_length, ...])	Compute root-mean-square (RMS) value for each frame, eit
`spectral_centroid` ([y, sr, S, n_fft, ...])	Compute the spectral centroid.
`spectral_bandwidth` ([y, sr, S, n_fft, ...])	Compute p'th-order spectral bandwidth.
`spectral_contrast` ([y, sr, S, n_fft, ...])	Compute spectral contrast [R6ffcc01153df-1]
`spectral_flatness` ([y, S, n_fft, hop_length, ...])	Compute spectral flatness
`spectral_rolloff` ([y, sr, S, n_fft, ...])	Compute roll-off frequency.
`poly_features` ([y, sr, S, n_fft, hop_length, ...])	Get coefficients of fitting an nth-order polynomial to the co
`tonnetz` ([y, sr, chroma])	Computes the tonal centroid features (tonnetz), following t
`zero_crossing_rate` (y[, frame_length, ...])	Compute the zero-crossing rate of an audio time series.

These features correspond ***perfectly*** with what's in the FMA dataset; they're all here:

- chroma
- tonnetz
- mfcc
    - it also computes melspectrograms, which I hope can be a cool/useful visualization
- spectral_centroid
- spectral_bandwidth
- spectral_contrast
- spectral_rolloff
- rms (root mean square energy)
- zero_crossing_rate

Let's explore this data:

[Helpful article link here (https://towardsdatascience.com/extract-features-of-music-75a3f9bc265d)](https://towardsdatascience.com/extract-features-of-music-75a3f9bc265d)

In [2]:

```
import librosa
audio_path = "audio_files/Don't Run With Pizzas.mp3"
x, sr = librosa.load(audio_path) # decodes audio file to 1D array of time serie
s x, and sample rate of x, sr
print(type(x), type(sr))
```

```
/Users/mkarroqe/anaconda3/lib/python3.5/importlib/_bootstrap_extern
al.py:415: ImportWarning: Not importing directory /Users/mkarroqe/a
naconda3/lib/python3.5/site-packages/virtualenvwrapper: missing __i
nit__
  _warnings.warn(msg.format(portions[0]), ImportWarning)
/Users/mkarroqe/anaconda3/lib/python3.5/importlib/_bootstrap_extern
al.py:415: ImportWarning: Not importing directory /Users/mkarroqe/a
naconda3/lib/python3.5/site-packages/sphinxcontrib: missing __init_
_
  _warnings.warn(msg.format(portions[0]), ImportWarning)
/Users/mkarroqe/anaconda3/lib/python3.5/site-packages/numba/errors.
py:137: UserWarning: Insufficiently recent colorama version found.
Numba requires colorama >= 0.3.9
  warnings.warn(msg)
/Users/mkarroqe/anaconda3/lib/python3.5/site-packages/librosa/core/
audio.py:146: UserWarning: PySoundFile failed. Trying audioread ins
tead.
  warnings.warn('PySoundFile failed. Trying audioread instead.')

<class 'numpy.ndarray'> <class 'int'>
```

In [3]:

```
# Playing audio
import IPython.display as ipd
ipd.Audio(audio_path)
```

```
/Users/mkarroqe/anaconda3/lib/python3.5/site-packages/IPython/core/
formatters.py:92: DeprecationWarning: DisplayFormatter._ipython_dis
play_formatter_default is deprecated: use @default decorator instea
d.
  def _ipython_display_formatter_default(self):
/Users/mkarroqe/anaconda3/lib/python3.5/site-packages/IPython/core/
formatters.py:98: DeprecationWarning: DisplayFormatter._formatters_
default is deprecated: use @default decorator instead.
  def _formatters_default(self):
/Users/mkarroqe/anaconda3/lib/python3.5/site-packages/IPython/core/
formatters.py:677: DeprecationWarning: PlainTextFormatter._deferred
_printers_default is deprecated: use @default decorator instead.
  def _deferred_printers_default(self):
/Users/mkarroqe/anaconda3/lib/python3.5/site-packages/IPython/core/
formatters.py:669: DeprecationWarning: PlainTextFormatter._singleto
n_printers_default is deprecated: use @default decorator instead.
  def _singleton_printers_default(self):
/Users/mkarroqe/anaconda3/lib/python3.5/site-packages/IPython/core/
formatters.py:672: DeprecationWarning: PlainTextFormatter._type_pri
nters_default is deprecated: use @default decorator instead.
  def _type_printers_default(self):
```

Out[3]:

0:00 / 3:38

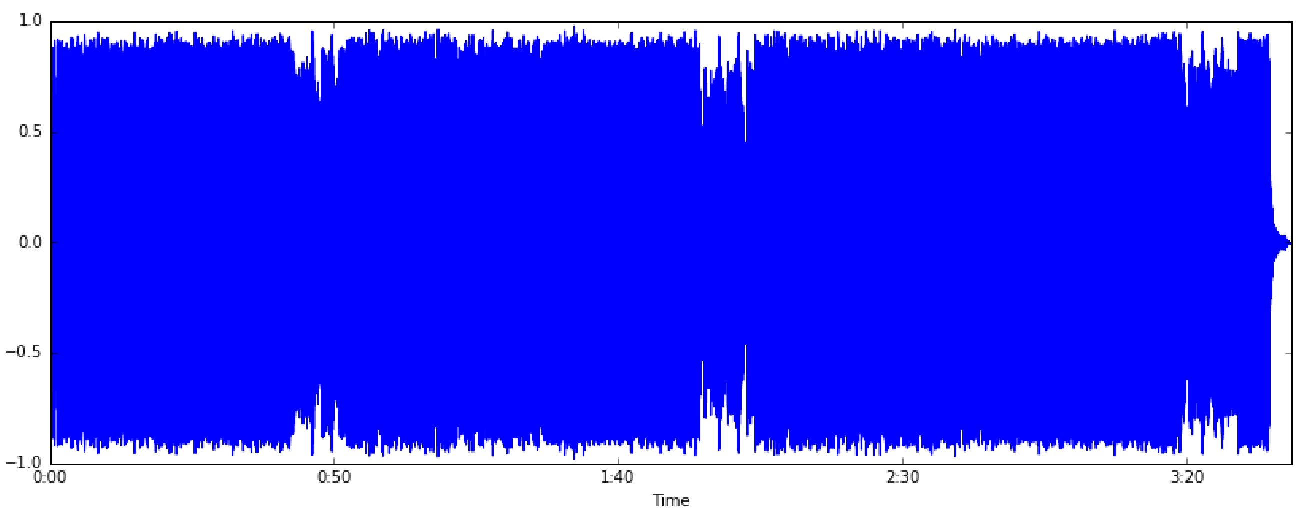
In [4]:

```
#display waveform
%matplotlib inline
import matplotlib.pyplot as plt
import librosa.display
plt.figure(figsize=(14, 5))
librosa.display.waveplot(x, sr=sr)
```

```
/Users/mkarroqe/anaconda3/lib/python3.5/site-packages/ipykernel/pyl
ab/config.py:66: DeprecationWarning: metadata {'config': True} was
set from the constructor.  Metadata should be set using the .tag()
method, e.g., Int().tag(key1='value1', key2='value2')
  inline backend."""
/Users/mkarroqe/anaconda3/lib/python3.5/site-packages/ipykernel/pyl
ab/config.py:44: DeprecationWarning: InlineBackend._config_changed
is deprecated: use @observe and @unobserve instead.
  def _config_changed(self, name, old, new):
/Users/mkarroqe/anaconda3/lib/python3.5/site-packages/traitlets/tra
itlets.py:770: DeprecationWarning: A parent of InlineBackend._confi
g_changed has adopted the new @observe(change) API
  clsname, change_or_name), DeprecationWarning)
```

Out[4]:

```
<matplotlib.collections.PolyCollection at 0x1057acc18>
```



Spectrogram

A spectrogram shows the spectrum of frequencies of sound over time. Here's a very pleasant video explaining how to read a spectrogram: https://www.youtube.com/watch?v=_FatxGN3vAM (https://www.youtube.com/watch?v=_FatxGN3vAM)

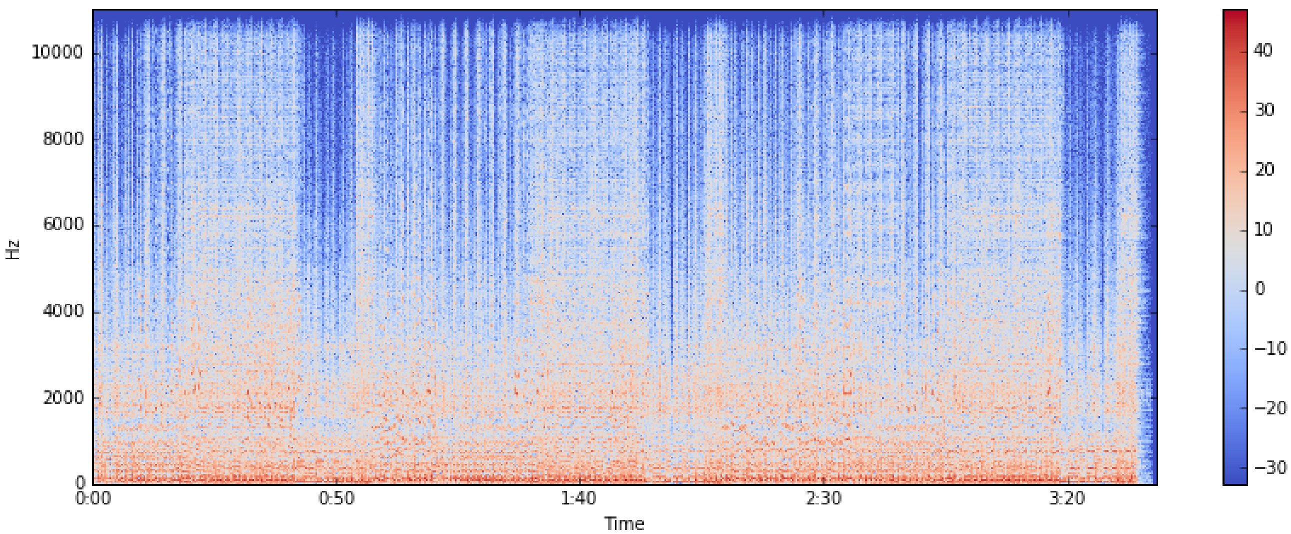
Here it is for this track:

In [5]:

```
X = librosa.stft(x)
Xdb = librosa.amplitude_to_db(abs(X))
plt.figure(figsize=(14, 5))
librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
#If to pring log of frequencies
#librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='log')
plt.colorbar()
```

Out[5]:

<matplotlib.colorbar.Colorbar at 0x125848080>



That's a noisy song!

Zero-crossing rate

The rate of sign-changes along a signal; usually has higher values for highly percussive sounds (like in metal/rock)

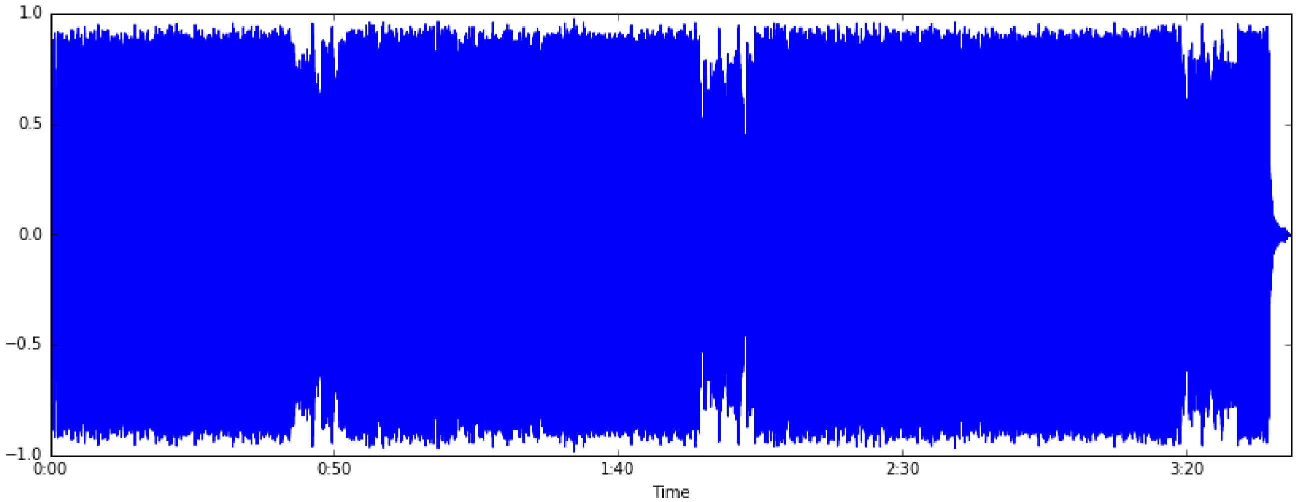
In [6]:

```
x, sr = librosa.load(audio_path)
#Plot the signal:
plt.figure(figsize=(14, 5))
librosa.display.waveplot(x, sr=sr)
```

```
/Users/mkarroqe/anaconda3/lib/python3.5/site-packages/librosa/core/
audio.py:146: UserWarning: PySoundFile failed. Trying audioread ins
tead.
  warnings.warn('PySoundFile failed. Trying audioread instead.')
```

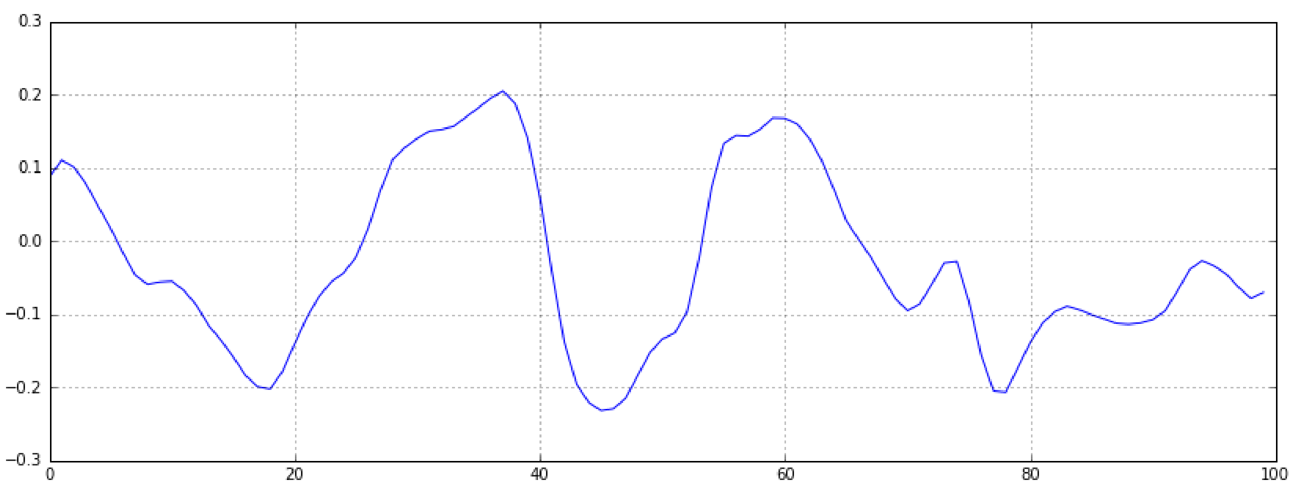
Out[6]:

```
<matplotlib.collections.PolyCollection at 0x125892e10>
```



In [7]:

```
# Zooming in
n0 = 9000
n1 = 9100
plt.figure(figsize=(14, 5))
plt.plot(x[n0:n1])
plt.grid()
```



We can see here that the signal crosses zero 5 times in the above graph, which looks at the first 100 columns in our 1D array. It can also be calculated as follows:

In [9]:

```
zero_crossings = librosa.zero_crossings(x[n0:n1], pad=False)
zero_crossings
```

Out[9]:

```
array([False, False, False, False, False, False,  True, False, Fals
e,
       False, False, False, False, False, False, False, False, Fals
e,
       False, False, False, False, False, False, False, False,  Tru
e,
       False, False, False, False, False, False, False, False, Fals
e,
       False, False, False, False, False,  True, False, False, Fals
e,
       False, False, False, False, False, False, False, False, Fals
e,
        True, False, False, False, False, False, False, False, Fals
e,
       False, False, False, False,  True, False, False, False, Fals
e,
       False, False, False, False, False, False, False, False, Fals
e,
       False, False, False, False, False, False, False, False, Fals
e,
       False, False, False, False, False, False, False, False, Fals
e,
       False])
```

In [10]:

```
# for entire song
zero_crossings = librosa.zero_crossings(x, pad=False)
print(sum(zero_crossings))
```

```
419596
```

In [13]:

```
zero_crossings = librosa.zero_crossings(x[n0:n1], pad=False)
print(sum(zero_crossings))
```

```
5
```

The zero crossing rate is represented in our dataset, and I think will be a good indicator of the "roughness" of a track (as high percussion is probably more likely to be rock)

Spectral Centroid

This feature shows where the "center of mass" for a sound is; it is the weighted mean of all present frequencies.

In [18]:

```
# spectral centroid
import sklearn
spectral_centroids = librosa.feature.spectral_centroid(x, sr=sr)[0]
spectral_centroids.shape # shape of array with column for each frame in sample
```

Out[18]:

```
(9402,)
```

In [21]:

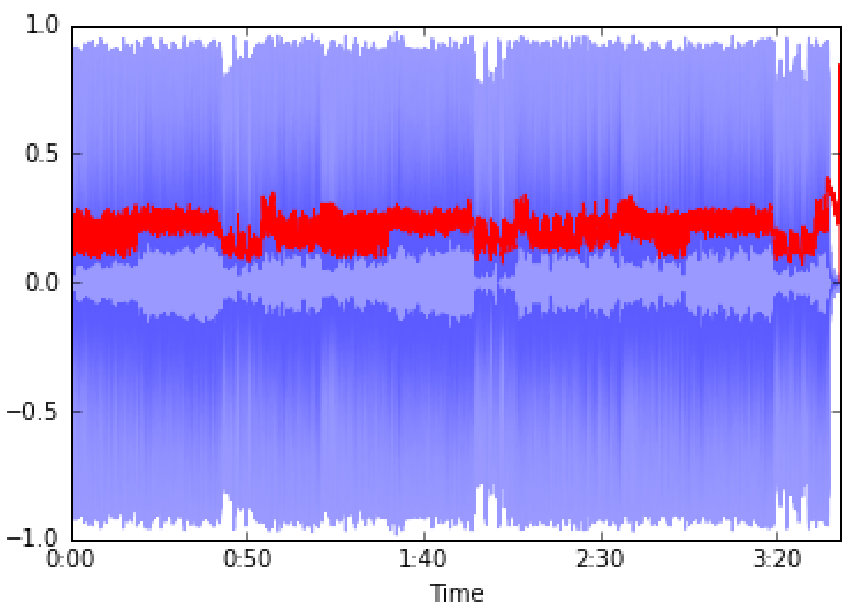
```
# Computing the time variable for visualization
frames = range(len(spectral_centroids))
t = librosa.frames_to_time(frames)

# Normalizing viz
def normalize(x, axis=0):
    return sklearn.preprocessing.minmax_scale(x, axis=axis)

# Plotting the Spectral Centroid along the waveform
librosa.display.waveplot(x, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_centroids), color='r')
```

Out[21]:

```
[<matplotlib.lines.Line2D at 0x12ff392e8>]
```

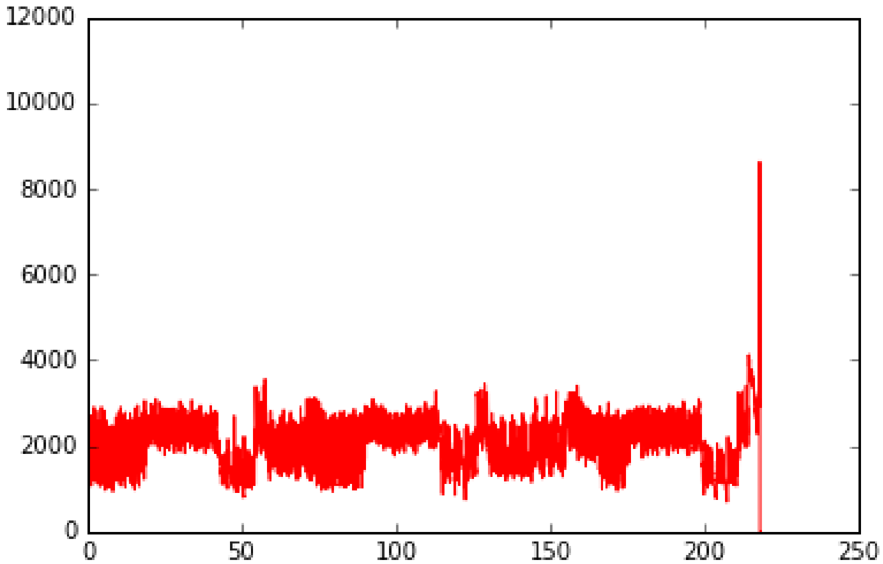


In [23]:

```
plt.plot(t, spectral_centroids, color='r') # without normalizing
```

Out[23]:

```
[<matplotlib.lines.Line2D at 0x1258f79b0>]
```



Mel-Frequency Cepstral Coefficient (MFCC)

These coefficients describe the "shape" of a sound signal. It can also be used to describe the timbre. Timbre, or tone is described with words like sharp, round, reedy, brassy, bright, etc.

In [26]:

```
mfccs = librosa.feature.mfcc(x, sr=sr)
print(mfccs.shape)
```

```
(20, 9402)
```

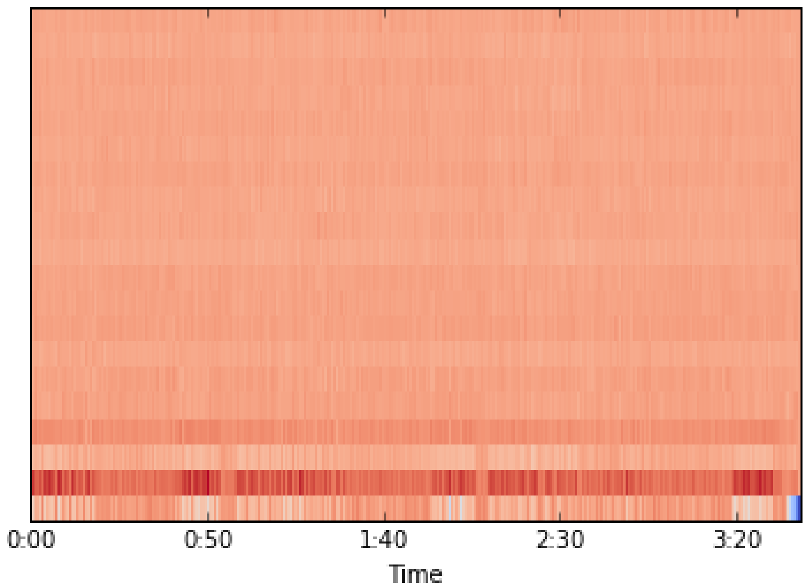
The tuple above gives us the total number of MFCCs calculated (index 0) and the total number of frames available.

In [32]:

```
#Displaying  the MFCCs:
librosa.display.specshow(mfccs, sr=sr, x_axis='time')
```

Out[32]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1248af3c8>
```

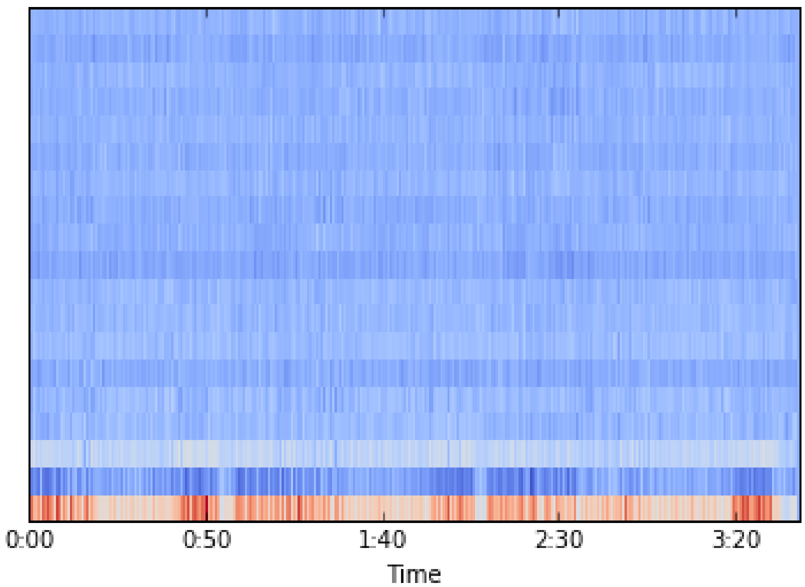


In [34]:

```
#Displaying  the MFCCs:
librosa.display.specshow(mfccs[1:20], sr=sr, x_axis='time')
```

Out[34]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x127b9d4e0>
```



This one is without the first feature, which contains a constant offset; it is often discarded according to [this (https://musicinformationretrieval.com/mfcc.html)](https://musicinformationretrieval.com/mfcc.html)

In [ ]:

```

```