In [1]:
```
import os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import sklearn
import math

import librosa
import librosa.display

import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
path= "music genera classification"
genres=os.listdir(os.path.join(path,'genres_original/'))
print(genres)
```

```
['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop', 'r
eggae', 'rock']
```

In [5]:
```
y,sr= librosa.load(os.path.join(path,'genres_original','classical','classical.000
print("y=",y,'\n')
print("Sample rate",sr,'\n')
```

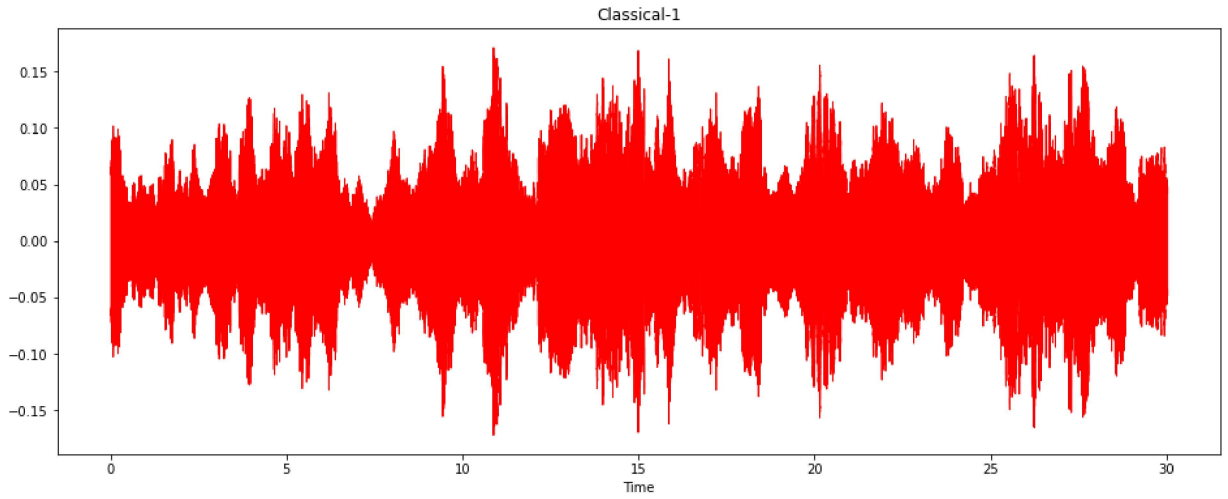
```
y= [-0.01138306 -0.00216675  0.01687622 ... -0.02954102 -0.0378418
 -0.03924561]

Sample rate 22050
```

In [6]:
```
#basic thing to do is to trim the silence
audio,_=librosa.effects.trim(y)
print("y: ",y,'\n')
print('Duration :',audio.shape[0]/sr)
```

```
y:  [-0.01138306 -0.00216675  0.01687622 ... -0.02954102 -0.0378418
 -0.03924561]

Duration : 30.013333333333332
```

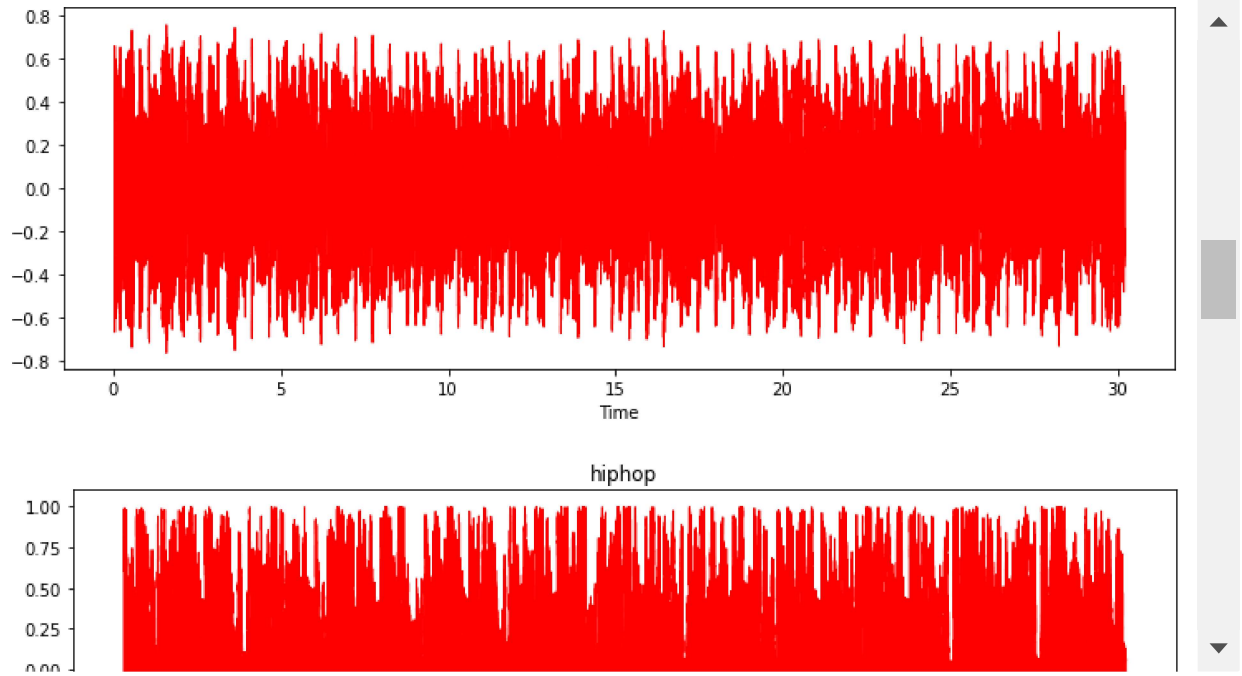
In [7]:
```
plt.figure(figsize=(16,6))
librosa.display.waveshow(y=audio,sr=sr,color='r')
plt.title('Classical-1')
```

Out[7]:
```
Text(0.5, 1.0, 'Classical-1')
```



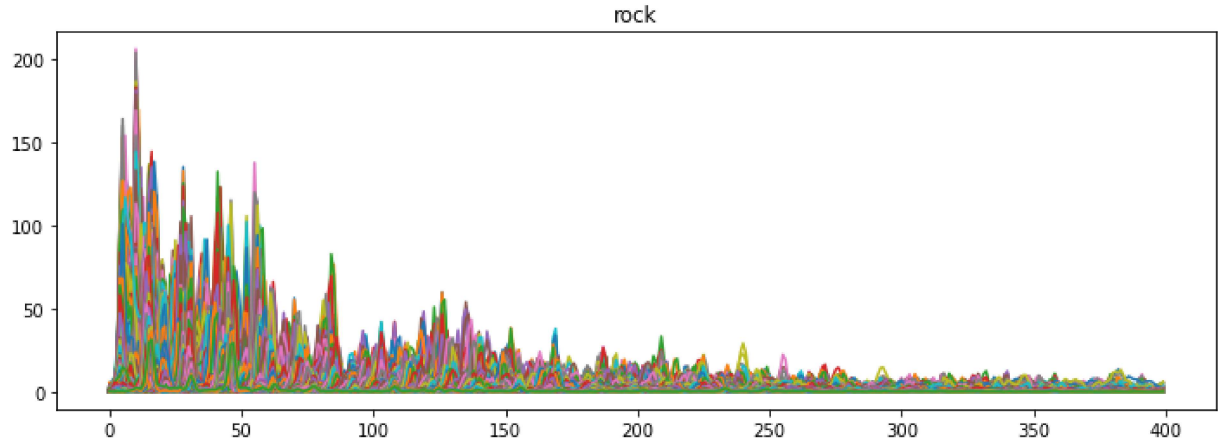
Seeing waveplot of all genres.

In [8]:
```
for i in genres:
    aud,sr=librosa.load(os.path.join(path,'genres_original',i,f'{i}.00001.wav'))
    plt.figure(figsize=(12,4))
    librosa.display.waveshow(y=aud,sr=sr,color='r')
    plt.title(f'{i}')
```

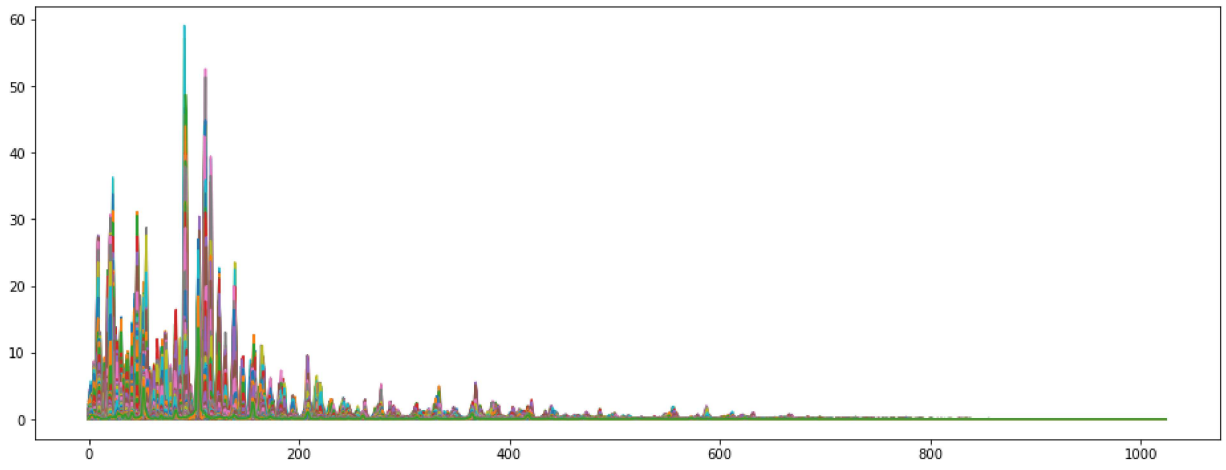


In [66]:
```
n_fft=2048 #default value recommended, n_ftt represents the number of samples tha
hop_length=512 #understandable by name
win_length=2048 #window using which samples are converted.

for i in genres:
    aud,sr=librosa.load(os.path.join(path,'genres_original',i,f'{i}.00001.wav'))
    aud_ft= np.abs(librosa.stft(aud, n_fft = n_fft, hop_length = hop_length,win_l

    # print(np.shape(aud_ft)) #(1025,1302)
plt.figure(figsize=(12,4))
plt.plot(aud_ft[:400,:])#viewing only upto 400 Hz
plt.title(f'{i}')
```

Out[66]: Text(0.5, 1.0, 'rock')

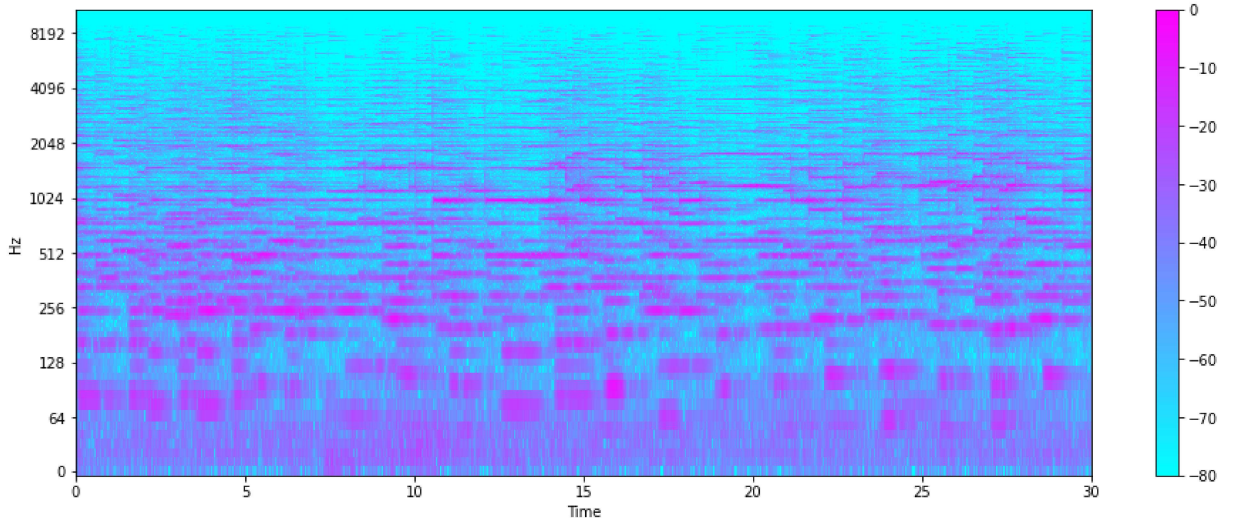


In [65]:
```
music_stft = np.abs(librosa.stft(audio,n_fft=n_fft,hop_length= hop_length))
plt.figure(figsize = (16,6))
plt.plot(music_stft);
```



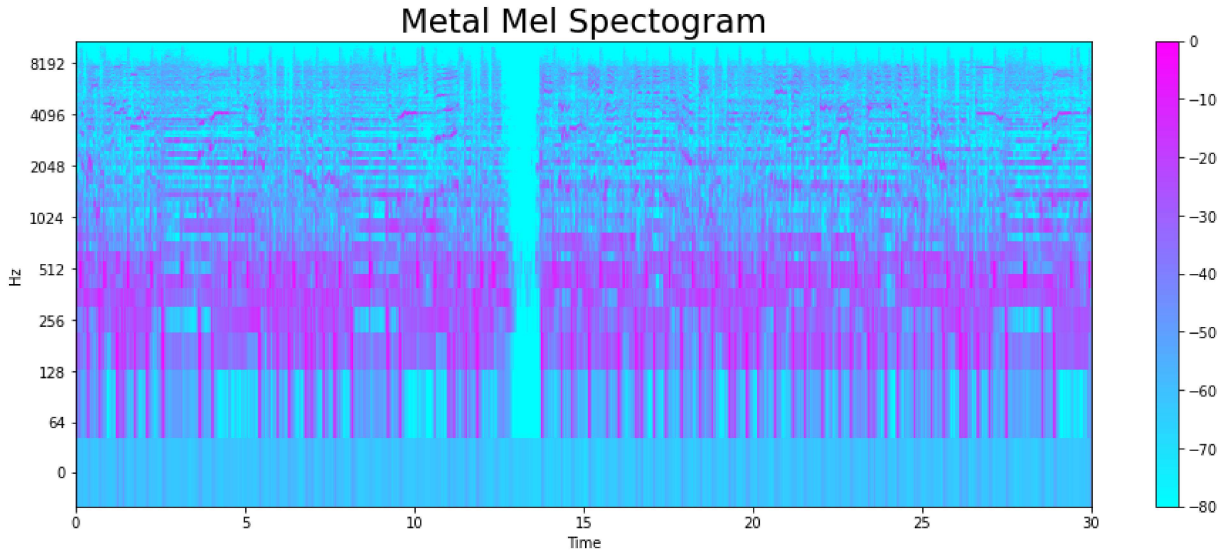
In [16]:
```
# Converting from amplitute(Linear scale) to decibels, a log scale
music_stft_decibels = librosa.amplitude_to_db(music_stft, ref= np.max)
```

In [17]:
```
# Plotting the spectogram
plt.figure(figsize=(16,6))
librosa.display.specshow(music_stft_decibels, sr = sr, hop_length= hop_length, x_
plt.colorbar();
```

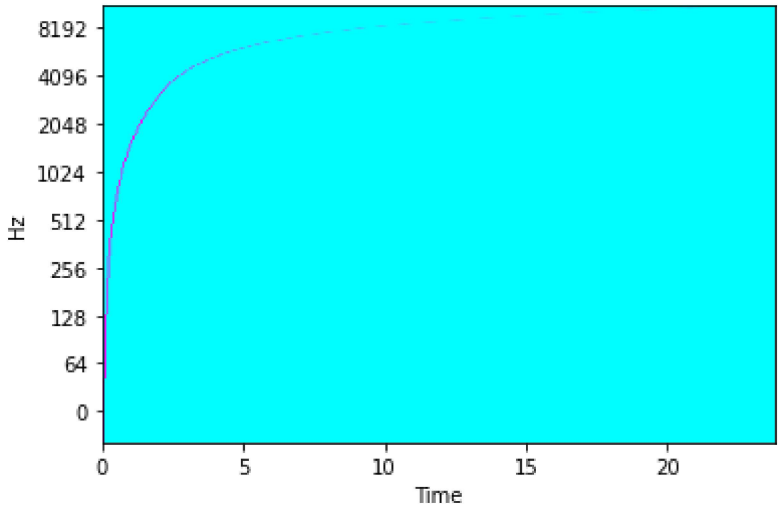


In [18]:
```
metal_sample = "metal/metal.00032.wav"
y, sr = librosa.load(os.path.join(path,"genres_original",metal_sample))
y,_ = librosa.effects.trim(y)

S = librosa.feature.melspectrogram(y, sr = sr)
S_DB = librosa.amplitude_to_db(S, ref= np.max)

plt.figure(figsize = (16,6))


librosa.display.specshow(S_DB, sr = sr, hop_length=hop_length, x_axis='time', y_a
# https://stackoverflow.com/questions/66235936/why-spectrogram-from-librosa-libra
# Why it is important to provide hop length here
plt.colorbar();
plt.title("Metal Mel Spectogram", fontsize = "23")
```

Out[18]: Text(0.5, 1.0, 'Metal Mel Spectogram')



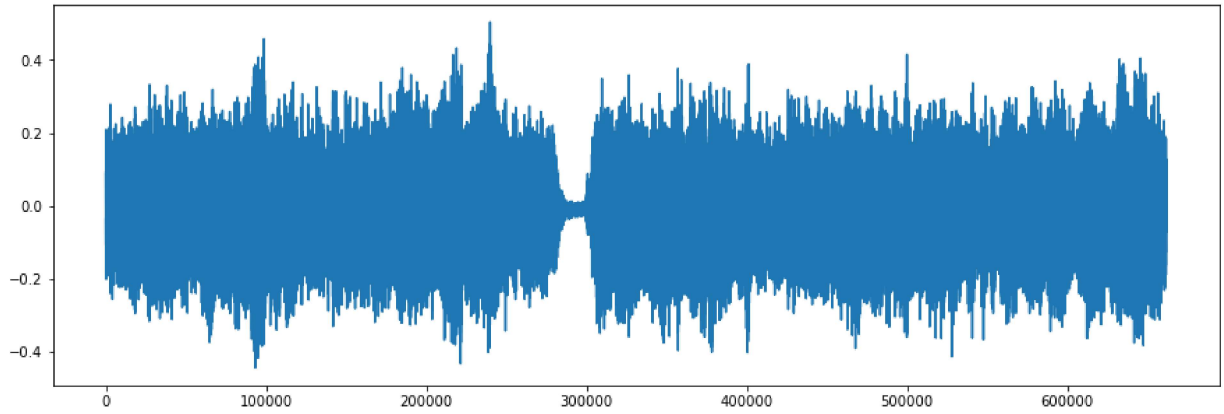
In [19]:
```
l(sr=sr, n_fft=n_fft, n_mels=n_mels)
(mel, sr = sr, hop_length=hop_length, x_axis='time', y_axis='log', cmap='cool');
```

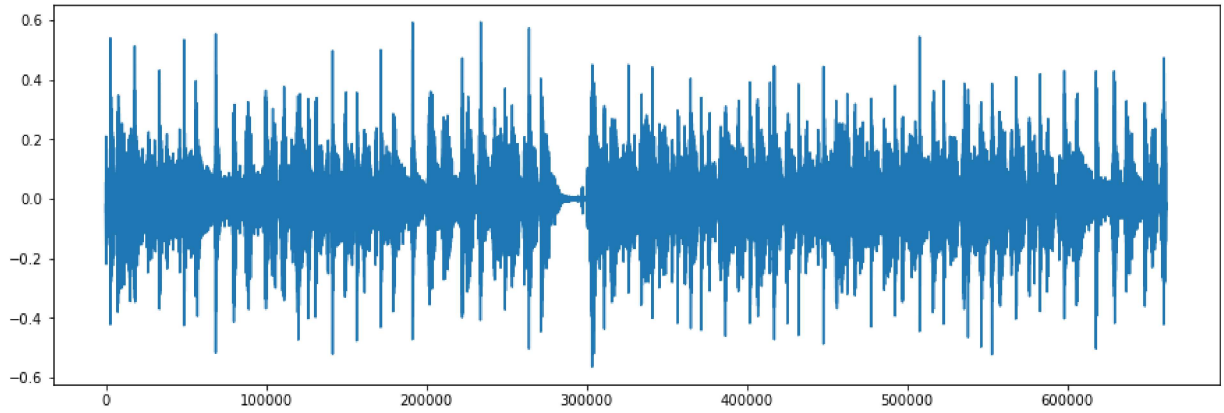


In [20]:
```
zero_crossing_rate=librosa.zero_crossings(y) # has a boolean output
sum(zero_crossing_rate)
```

Out[20]: 98145

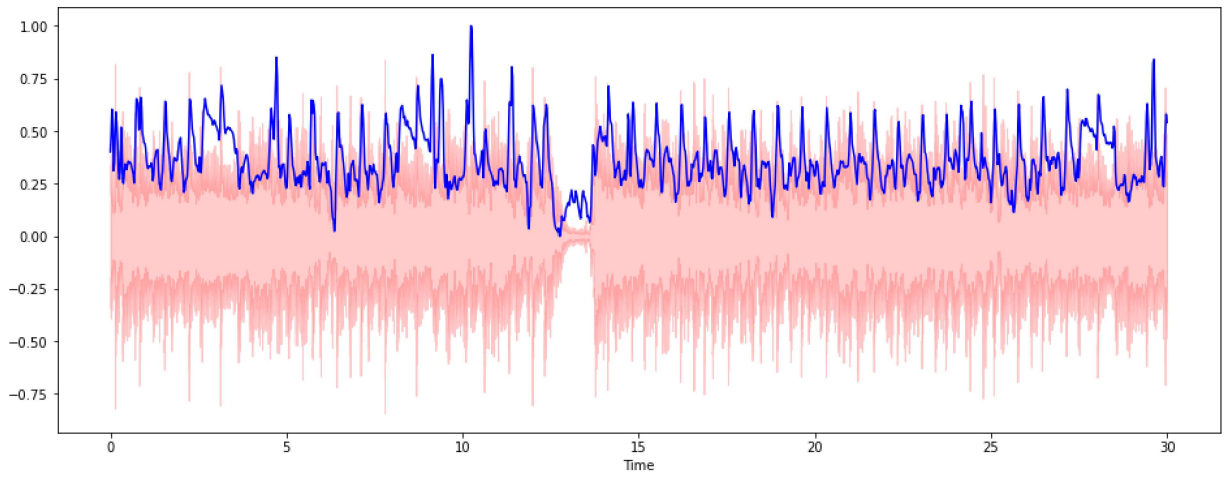
In [21]:
```
harmonics,percussive=librosa.effects.hpss(y)
plt.figure(figsize=(15,5))
plt.plot(harmonics);
plt.figure(figsize=(15,5))
plt.plot(percussive);
```



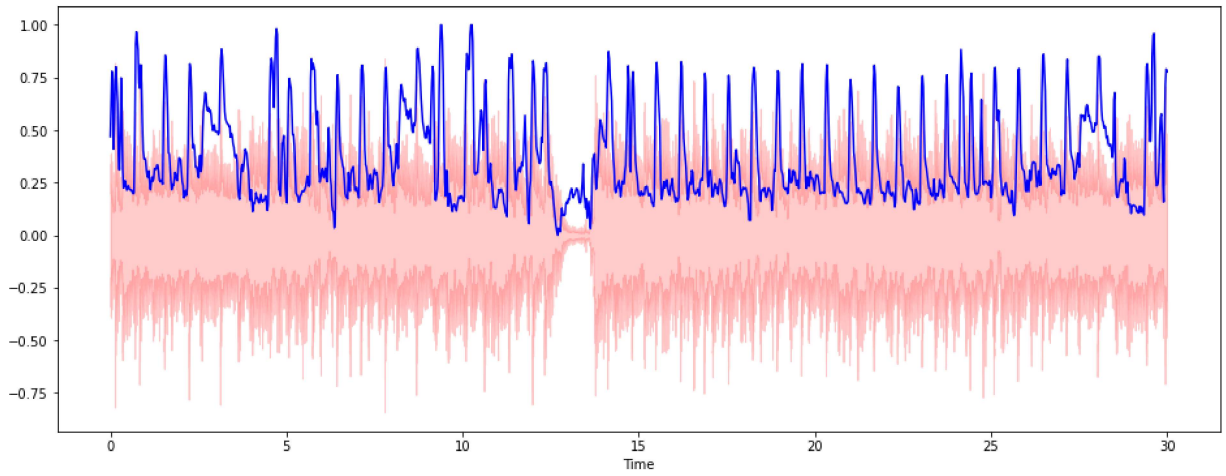


In [22]:
```
pectral_centroids = librosa.feature.spectral_centroid(y, sr=sr)[0] #returns spect
rint('Centroids:', spectral_centroids, '\n')
rint('Shape of Spectral Centroids:', spectral_centroids.shape, '\n')

 Computing the time variable for visualization
rames = range(len(spectral_centroids))

 Converts frame counts to time (seconds)
 = librosa.frames_to_time(frames)

rint('frames:', frames, '\n')
rint('t:', t)

 Function that normalizes the Sound Data
ef normalize(x, axis=0):
    return sklearn.preprocessing.minmax_scale(x, axis=axis)
lt.figure(figsize = (16, 6))
ibrosa.display.waveshow(y, sr=sr, alpha=0.2, color = 'red');
lt.plot(t, normalize(spectral_centroids), color='blue');
```

```
Centroids: [2622.73050509 2881.57344115 3268.66541681 ... 2937.61979423 3196.10
716976
 3071.26493668]

Shape of Spectral Centroids: (1293,)

frames: range(0, 1293)

t: [0.00000000e+00 2.32199546e-02 4.64399093e-02 ... 2.99537415e+01
 2.99769615e+01 3.00001814e+01]
```

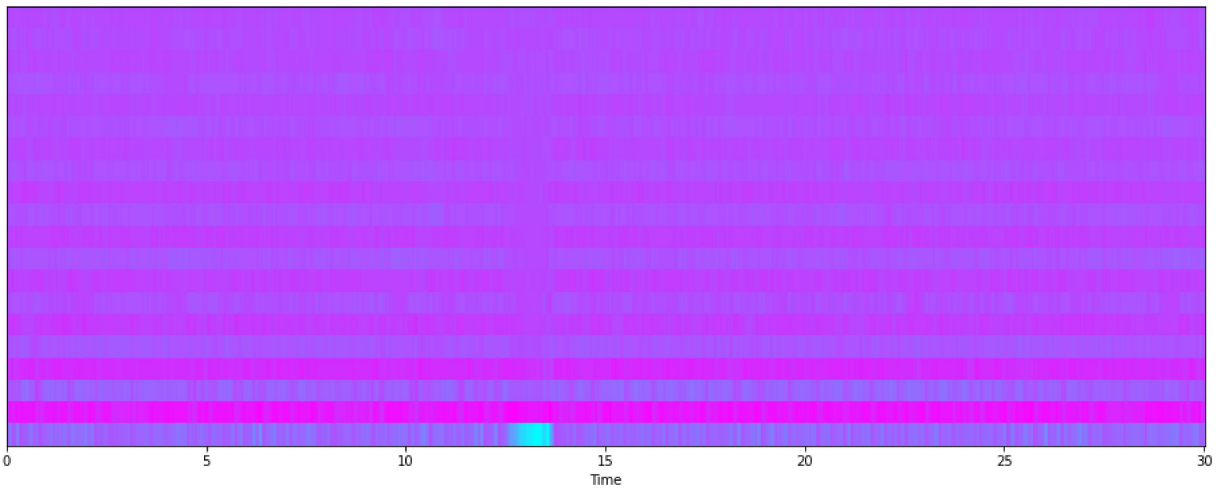


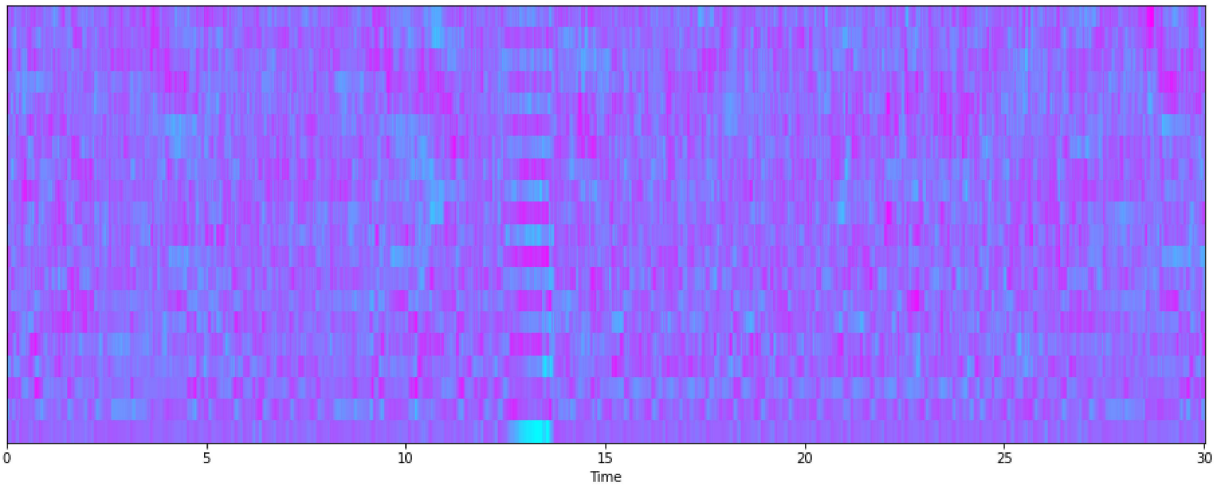
In [23]:
```
spectral_rolloff = librosa.feature.spectral_rolloff(y, sr=sr)[0]
plt.figure(figsize = (16, 6))
librosa.display.waveshow(y, sr=sr, alpha=0.2, color = 'red');
plt.plot(t, normalize(spectral_rolloff), color='blue');
```



In [24]:
```
mfccs = librosa.feature.mfcc(y, sr=sr)
print('mfccs shape:', mfccs.shape)
plt.figure(figsize = (16, 6))
librosa.display.specshow(mfccs, sr=sr, x_axis='time', cmap = 'cool');
#the data is in small range, hence need to be scaled.
mfccs = sklearn.preprocessing.scale(mfccs,axis=1)
plt.figure(figsize = (16, 6))
librosa.display.specshow(mfccs, sr=sr, x_axis='time', cmap = 'cool');
```

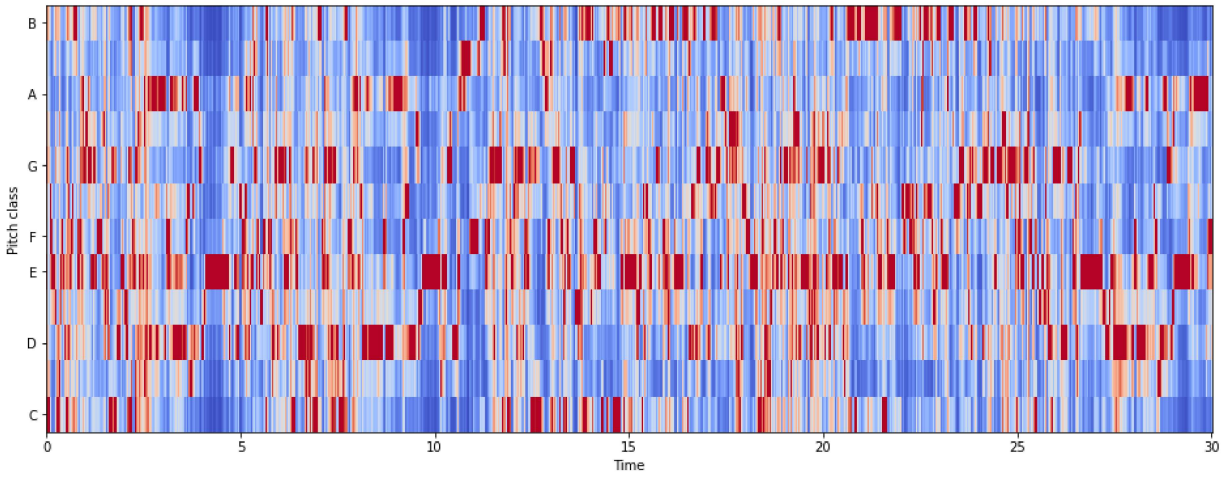
```
mfccs shape: (20, 1293)
```





In [25]:
```
chromagram = librosa.feature.chroma_stft(y, sr=sr, hop_length=hop_length)
print('Chromogram shape:', chromagram.shape)
plt.figure(figsize=(16, 6))
librosa.display.specshow(chromagram, x_axis='time', y_axis='chroma', hop_length=h
```

```
Chromogram shape: (12, 1293)
```



In [26]:
```
#reading the 30 s csv
data=pd.read_csv(os.path.join(path,'features_30_sec.csv'))
data.head()
```

Out[26]:

	filename	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_cent
0	blues.00000.wav	661794	0.350088	0.088757	0.130228	0.002827	1
1	blues.00001.wav	661794	0.340914	0.094980	0.095948	0.002373	1
2	blues.00002.wav	661794	0.363637	0.085275	0.175570	0.002746	1
3	blues.00003.wav	661794	0.404785	0.093999	0.141093	0.006346	1
4	blues.00004.wav	661794	0.308526	0.087841	0.091529	0.002303	1

5 rows × 60 columns

In [27]:
```
data.columns
```

Out[27]:
```
Index(['filename', 'length', 'chroma_stft_mean', 'chroma_stft_var', 'rms_mean',
       'rms_var', 'spectral_centroid_mean', 'spectral_centroid_var',
       'spectral_bandwidth_mean', 'spectral_bandwidth_var', 'rolloff_mean',
       'rolloff_var', 'zero_crossing_rate_mean', 'zero_crossing_rate_var',
       'harmony_mean', 'harmony_var', 'perceptr_mean', 'perceptr_var', 'tempo',
       'mfcc1_mean', 'mfcc1_var', 'mfcc2_mean', 'mfcc2_var', 'mfcc3_mean',
       'mfcc3_var', 'mfcc4_mean', 'mfcc4_var', 'mfcc5_mean', 'mfcc5_var',
       'mfcc6_mean', 'mfcc6_var', 'mfcc7_mean', 'mfcc7_var', 'mfcc8_mean',
       'mfcc8_var', 'mfcc9_mean', 'mfcc9_var', 'mfcc10_mean', 'mfcc10_var',
       'mfcc11_mean', 'mfcc11_var', 'mfcc12_mean', 'mfcc12_var', 'mfcc13_mean',
       'mfcc13_var', 'mfcc14_mean', 'mfcc14_var', 'mfcc15_mean', 'mfcc15_var',
       'mfcc16_mean', 'mfcc16_var', 'mfcc17_mean', 'mfcc17_var', 'mfcc18_mean',
       'mfcc18_var', 'mfcc19_mean', 'mfcc19_var', 'mfcc20_mean', 'mfcc20_var',
       'label'],
      dtype='object')
```

In [28]:
```
from sklearn import preprocessing
data=data.iloc[0:,2:]
Y=data.loc[:,'label']
X=data.loc[:,data.columns!='label']

cols=X.columns
min_max_scaler=preprocessing.MinMaxScaler()
scaled_X=min_max_scaler.fit_transform(X) #the column names are removed
X=pd.DataFrame(scaled_X,columns=cols)
```

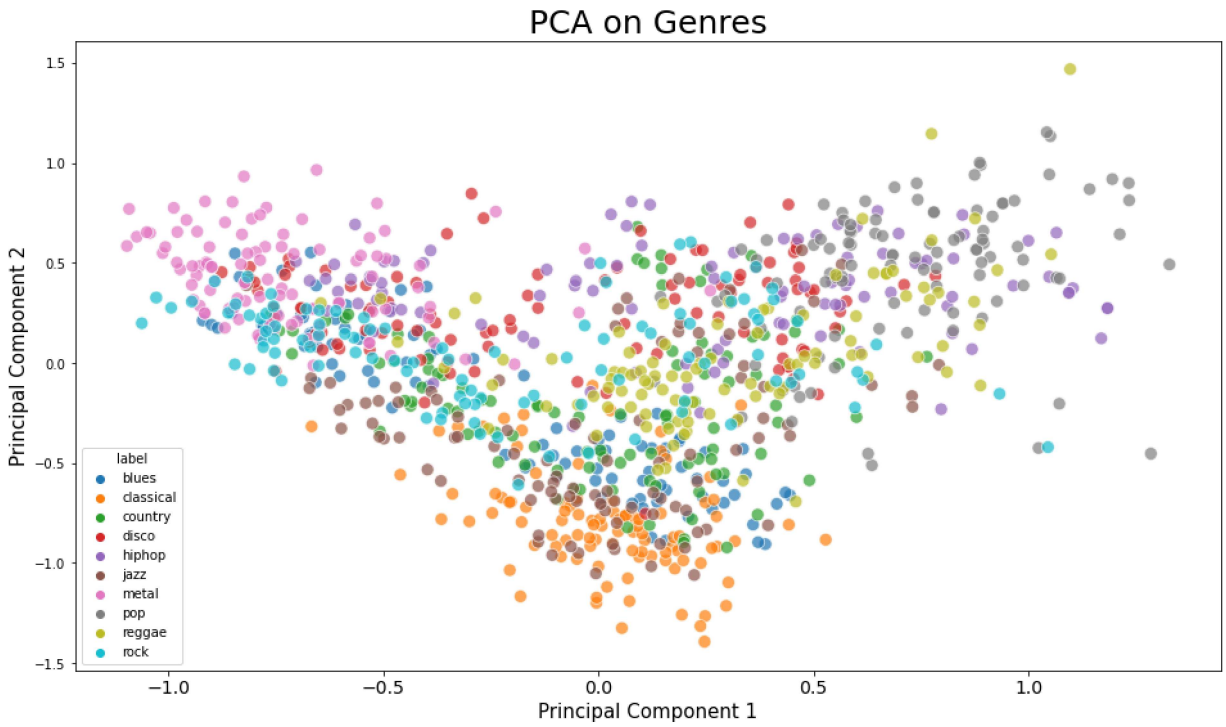
In [29]:
```
from sklearn.decomposition import PCA
n=10
pca=PCA(n_components=n)
pc=pca.fit_transform(X)
col_names=[f'PC{i}' for i in range(1,n+1)]
data_X=pd.DataFrame(data=pc,columns=col_names)

final_df=pd.concat([data_X,Y],axis=1)
final_df.head()
```

Out[29]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9
0	-0.394212	-0.116145	-0.099806	0.042689	-0.121922	0.107266	-0.017494	-0.003765	0.023459
1	0.052019	-0.270757	0.427637	-0.026008	-0.262903	-0.105095	-0.136280	-0.212256	-0.102066
2	-0.479184	-0.224616	-0.014295	-0.233705	-0.025667	0.267477	0.075904	0.190766	0.132839
3	0.017145	-0.439886	0.017747	-0.067034	-0.434105	0.239529	-0.059289	-0.271078	-0.048625
4	-0.160395	-0.508617	0.073920	-0.360533	0.480087	0.099143	0.227757	-0.001936	0.127938

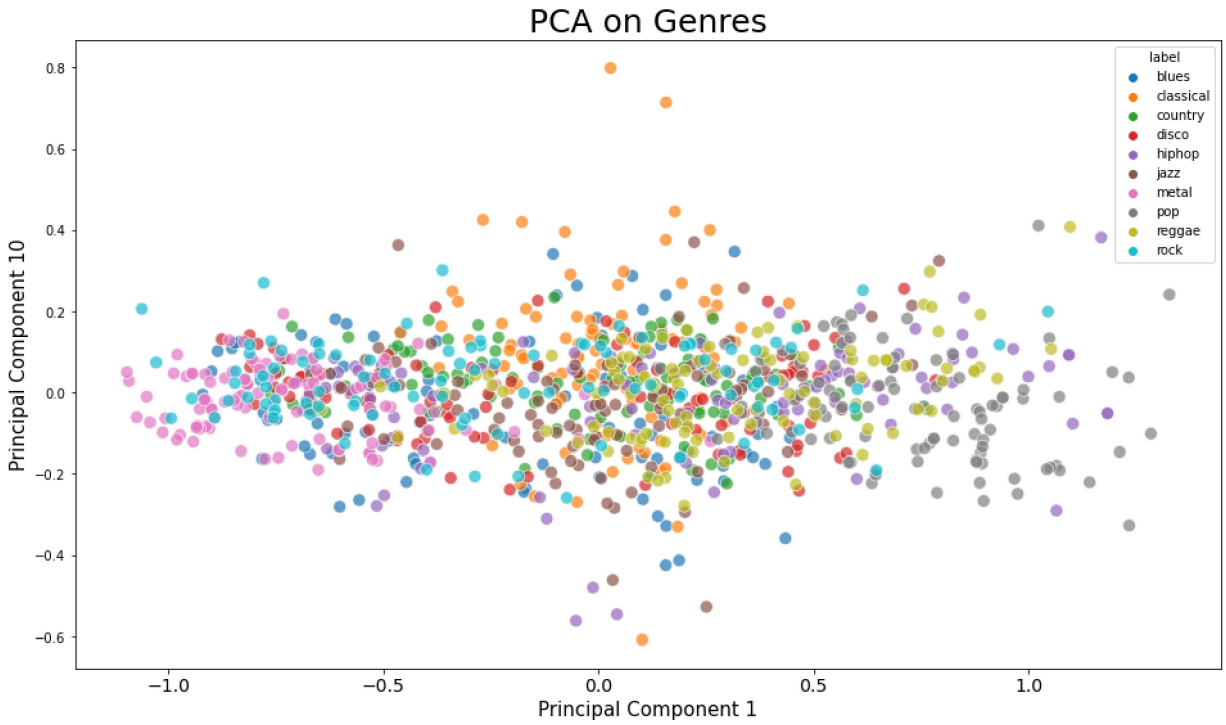
In [30]:
```
plt.figure(figsize = (16, 9))
sns.scatterplot(x = "PC1", y = "PC2", data = final_df, hue = "label", alpha = 0.7
                s = 100);

plt.title('PCA on Genres', fontsize = 25)
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 10);
plt.xlabel("Principal Component 1", fontsize = 15)
plt.ylabel("Principal Component 2", fontsize = 15)
```

Out[30]:
```
Text(0, 0.5, 'Principal Component 2')
```



In [31]:
```
plt.figure(figsize = (16, 9))
sns.scatterplot(x = "PC1", y = "PC10", data = final_df, hue = "label", alpha = 0.
                s = 100);

plt.title('PCA on Genres', fontsize = 25)
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 10);
plt.xlabel("Principal Component 1", fontsize = 15)
plt.ylabel("Principal Component 10", fontsize = 15)
```

Out[31]:
```
Text(0, 0.5, 'Principal Component 10')
```



In [32]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier, XGBRFClassifier
from xgboost import plot_tree, plot_importance

from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE
```

In [33]:
```
#reading the 3 s csv
data= pd.read_csv(os.path.join(path,'features_3_sec.csv'))
data.head()
```

Out[33]:

	filename	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_cen
0	blues.00000.0.wav	66149	0.335406	0.091048	0.130405	0.003521	1
1	blues.00000.1.wav	66149	0.343065	0.086147	0.112699	0.001450	1
2	blues.00000.2.wav	66149	0.346815	0.092243	0.132003	0.004620	1
3	blues.00000.3.wav	66149	0.363639	0.086856	0.132565	0.002448	1
4	blues.00000.4.wav	66149	0.335579	0.088129	0.143289	0.001701	1

5 rows × 60 columns

In [34]:
```
data.columns
```

Out[34]:
```
Index(['filename', 'length', 'chroma_stft_mean', 'chroma_stft_var', 'rms_mean',
       'rms_var', 'spectral_centroid_mean', 'spectral_centroid_var',
       'spectral_bandwidth_mean', 'spectral_bandwidth_var', 'rolloff_mean',
       'rolloff_var', 'zero_crossing_rate_mean', 'zero_crossing_rate_var',
       'harmony_mean', 'harmony_var', 'perceptr_mean', 'perceptr_var', 'tempo',
       'mfcc1_mean', 'mfcc1_var', 'mfcc2_mean', 'mfcc2_var', 'mfcc3_mean',
       'mfcc3_var', 'mfcc4_mean', 'mfcc4_var', 'mfcc5_mean', 'mfcc5_var',
       'mfcc6_mean', 'mfcc6_var', 'mfcc7_mean', 'mfcc7_var', 'mfcc8_mean',
       'mfcc8_var', 'mfcc9_mean', 'mfcc9_var', 'mfcc10_mean', 'mfcc10_var',
       'mfcc11_mean', 'mfcc11_var', 'mfcc12_mean', 'mfcc12_var', 'mfcc13_mean',
       'mfcc13_var', 'mfcc14_mean', 'mfcc14_var', 'mfcc15_mean', 'mfcc15_var',
       'mfcc16_mean', 'mfcc16_var', 'mfcc17_mean', 'mfcc17_var', 'mfcc18_mean',
       'mfcc18_var', 'mfcc19_mean', 'mfcc19_var', 'mfcc20_mean', 'mfcc20_var',
       'label'],
      dtype='object')
```

In [35]:
```
data=data.iloc[0:,2:]
Y=data.loc[:,'label']
X=data.loc[:,data.columns!='label']

cols=X.columns

scaler=preprocessing.MinMaxScaler()
scaled_X=scaler.fit_transform(X)

X=pd.DataFrame(scaled_X,columns=cols)
```

In [36]:
```
_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.3,random_state=21)
```

In [37]:
```
def model_assess(model,title):
    model.fit(X_train,y_train)
    preds=model.predict(X_test)
    print('Accuracy', title, ':', round(accuracy_score(y_test, preds), 5), '\n')
```

In [38]:
```
# Naive Bayes
nb = GaussianNB()
model_assess(nb, "Naive Bayes")
```

```
Accuracy Naive Bayes : 0.52386
```

In [39]:
```
# Stochastic Gradient Descent
sgd = SGDClassifier(max_iter=5000, random_state=0)
model_assess(sgd, "Stochastic Gradient Descent")
```

```
Accuracy Stochastic Gradient Descent : 0.64298
```

In [40]:
```
# KNN
knn = KNeighborsClassifier(n_neighbors=19)
model_assess(knn, "KNN")
```

```
Accuracy KNN : 0.8038
```

In [41]:
```
# Decission trees
tree = DecisionTreeClassifier()
model_assess(tree, "Decission trees")
```

```
Accuracy Decission trees : 0.64665
```

In [42]:
```
# Random Forest
rforest = RandomForestClassifier(n_estimators=1000, max_depth=10, random_state=0)
model_assess(rforest, "Random Forest")
```

```
Accuracy Random Forest : 0.79479
```

In [43]:
```
# Support Vector Machine
svm = SVC(decision_function_shape="ovo")
model_assess(svm, "Support Vector Machine")
```

```
Accuracy Support Vector Machine : 0.74274
```

In [44]:
```
# Logistic Regression
lg = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial'
model_assess(lg, "Logistic Regression")
```

```
Accuracy Logistic Regression : 0.67434
```

In [45]:
```
# Neural Nets
nn = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5000, 10), ran
model_assess(nn, "Neural Nets")
```

```
Accuracy Neural Nets : 0.70103
```

In [46]:
```
le=LabelEncoder()
labels=le.fit_transform(Y)
X_train,X_test,y_train,y_test=train_test_split(X,labels,test_size=0.3,random_stat
```

In [47]:
```
# Cross Gradient Booster
xgb = XGBClassifier(n_estimators=1000, learning_rate=0.05)
xgb.fit(X_train,y_train)
preds=xgb.predict(X_test)
print('Accuracy',"xgb", ':', round(accuracy_score(y_test, preds), 5), '\n')
```

```
Accuracy xgb : 0.89656
```

In [48]:
```
# Cross Gradient Booster (Random Forest)
xgbrf = XGBRFClassifier(objective= 'multi:softmax')
xgbrf.fit(X_train,y_train)
preds=xgbrf.predict(X_test)
print('Accuracy',"xgbrf", ':', round(accuracy_score(y_test, preds), 5), '\n')
```

```
Accuracy xgbrf : 0.74708
```

In [49]:
```
#improving xgb gradient booster
xgb1=XGBClassifier(n_estimators=1400,learning_rate=0.03)
xgb1.fit(X_train,y_train)
preds=xgb1.predict(X_test)
print('Accuracy',"xgb1", ':', round(accuracy_score(y_test, preds), 5), '\n')
```

```
Accuracy xgb1 : 0.89923
```

In [50]:
```
from sklearn.metrics.pairwise import cosine_similarity
from sklearn import preprocessing
```

In [51]:
```
data = pd.read_csv(f'{path}/features_30_sec.csv', index_col='filename')
```

In [52]:
```
labels = data[['label']]
```

In [53]:
```
# Drop labels from original dataframe
data = data.drop(columns=['length','label'])
data.head()
```

Out[53]:

filename	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean
blues.00000.wav	0.350088	0.088757	0.130228	0.002827	1784.165850
blues.00001.wav	0.340914	0.094980	0.095948	0.002373	1530.176679
blues.00002.wav	0.363637	0.085275	0.175570	0.002746	1552.811865
blues.00003.wav	0.404785	0.093999	0.141093	0.006346	1070.106615
blues.00004.wav	0.308526	0.087841	0.091529	0.002303	1835.004266

5 rows × 57 columns

In [54]:
```
# Scale the data
data_scaled=preprocessing.scale(data)
print('Scaled data type:', type(data_scaled))
```

```
Scaled data type: <class 'numpy.ndarray'>
```

In [63]:
```
# Cosine similarity
similarity = cosine_similarity(data_scaled)
print("Similarity shape:", similarity.shape)

# Convert into a dataframe and then set the row index and column names as labels
sim_df_labels = pd.DataFrame(similarity)
```

```
Similarity shape: (1000, 1000)
```

In [58]:
```
sim_df_names = sim_df_labels.set_index(labels.index)
```

In [59]:
```
sim_df_names.columns = labels.index
```

In [60]:
```
sim_df_names.head()
```

Out[60]:

filename	blues.00000.wav	blues.00001.wav	blues.00002.wav	blues.00003.wav	blues.00004.w
filename					
blues.00000.wav	1.000000	0.049231	0.589618	0.284862	0.0255
blues.00001.wav	0.049231	1.000000	-0.096834	0.520903	0.0807
blues.00002.wav	0.589618	-0.096834	1.000000	0.210411	0.4002
blues.00003.wav	0.284862	0.520903	0.210411	1.000000	0.1264
blues.00004.wav	0.025561	0.080749	0.400266	0.126437	1.0000

5 rows × 1000 columns

In [56]:
```
def find_similar_songs(name):
    # Find songs most similar to another song
    series = sim_df_names[name].sort_values(ascending = False)

    # Remove cosine similarity == 1 (songs will always have the best match with t
    series = series.drop(name)

    # Display the 5 top matches
    print("\n*******\nSimilar songs to ", name)
    print(series.head(5))
```

In [61]:
```
find_similar_songs('metal.00002.wav')
```

```
*******
Similar songs to  metal.00002.wav
filename
metal.00028.wav    0.904367
metal.00059.wav    0.896096
rock.00018.wav     0.891910
rock.00017.wav     0.886526
rock.00016.wav     0.867508
Name: metal.00002.wav, dtype: float64
```

In [ ]:

In [ ]: