Tarea de Audio

Alumno:Irving Daniel Estrada López

Matrícula: 1739907

Las empresas hoy en día utilizan la clasificación de música, ya sea para poder colocar recomendaciones a sus clientes (como Spotify, Soundcloud) o simplemente como producto (por ejemplo, Shazam). Determinar los géneros musicales es el primer paso en esa dirección. Las técnicas de aprendizaje automático han demostrado ser bastante exitosas para extraer tendencias y patrones de un gran conjunto de datos. Los mismos principios se aplican también en el análisis musical.

Usualmente para la clasificación de los géneros musicales se necesita apoyo de expertos, que esto se conocería com una clasificación por reglas, sin embargo, el Machine Learning nos da la oportunidad de poder clasificar dichos géneros por la identificación de sus características. Además, la velocidad es algo que se puede diferencias de la clasificación manual que un humano podría hacer. Un punto que tienen en ventaja las personas que clasifican las piezas musicales en géneros, es que no es necesario contar con un gran volumen de datos, a diferencia del Machine Learning y el Deep Learning, que estos necesita un gran volumen de datos para extraer y aprender correctamente las características de cada uno de los géneros.

Existen diferente tipos de formatos de audio:

- MP3
- WMA (Windows Media Audio)
- WAV (Waveform Audio File)

Al igual que en las imágenes y en el texto existen librerías que nos ayudan en el análisis de audio como:

- **Librosa:** es un paquete de python para análisis de música y audio. Proporciona los componentes básicos necesarios para crear sistemas de recuperación de información musical.
- **IPython.display.Audio:** Es un objeto audio, nos permite reproducir audio directamente en jupyter notebook

Código

En este notebook estudiaremos el análisis de audio, clasificando géneros musicales. Utilizaremos principalmente la librería librosa para extraer características relevantes de un documento de audio. A través del código se irá describiendo para que sirve cada una de las funciones, y estudiando atributos importantes al trabajar con audio. Evaluaremos 7 modelos de Machine Learning, comparándolos en la parte final, identificando los mejor resultados y su desempeño con este conjunto de audios de música.

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

# Librosa (the mother of audio files)
import librosa
import librosa.display
import IPython.display as ipd
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
import os
general_path = '/Users/irvingestrada/Documents/Maestría/9- Procesamiento y C
print(list(os.listdir(f'{general_path}/genres_original/')))
```

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

librosa.load: Carga un archivo de audio como una serie de tiempo de punto flotante.

El sample rate es el número de muestras de audio transportadas por segundo, medido en Hz o kHz.

In [3]:
```
# Importing 1 file
y, sr = librosa.load(f'{general_path}/genres_original/reggae/reggae.00036.wa

print('y:', y, '\n')
print('y shape:', np.shape(y), '\n')
print('Sample Rate (KHz):', sr, '\n')

# Verify length of the audio
print('Check Len of Audio:', 661794/22050)
```

```
y: [0.02072144 0.04492188 0.05422974 ... 0.06912231 0.08303833 0.08572388]

y shape: (661794,)

Sample Rate (KHz): 22050

Check Len of Audio: 30.013333333333332
```

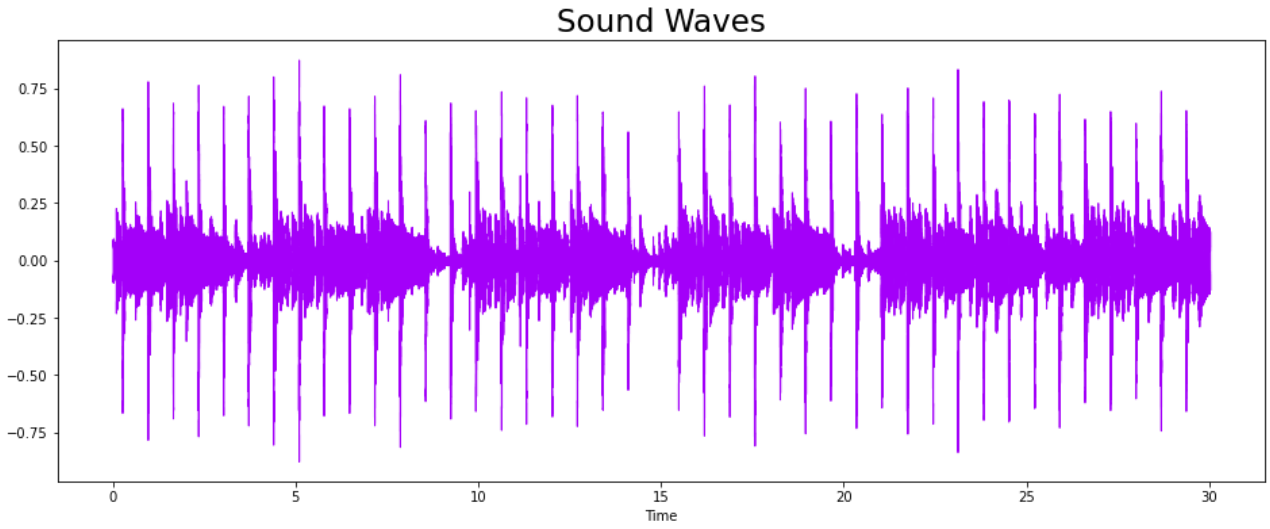
librosa.effect.trim: Divide una señal de audio en intervalos no silenciosos.

In [4]:
```
# Trim leading and trailing silence from an audio signal (silence before and
audio_file, _ = librosa.effects.trim(y)

# the result is an numpy ndarray
print('Audio File:', audio_file, '\n')
print('Audio File shape:', np.shape(audio_file))
```

```
Audio File: [0.02072144 0.04492188 0.05422974 ... 0.06912231 0.08303833 0.08
572388]

Audio File shape: (661794,)
```

librosa.display.waveshow: permite visualizar una forma de onda en el dominio del tiempo.

In [5]:
```
plt.figure(figsize = (16, 6))
librosa.display.waveshow(y = audio_file, sr = sr, color = "#A300F9");
plt.title("Sound Waves", fontsize = 23);
```

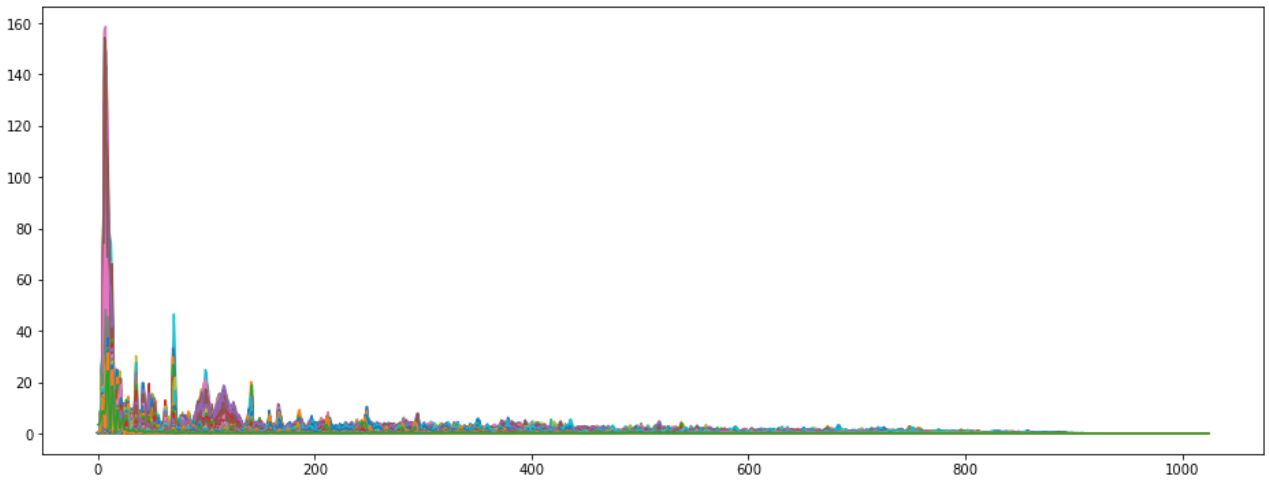


librosa.stft: Short-time Fourier transform (STFT). La STFT representa una señal en el dominio de tiempo-frecuencia mediante el cálculo de transformadas discretas de Fourier (DFT) en ventanas superpuestas cortas.

In [6]:
```
# Default FFT window size
n_fft = 2048 # FFT window size
hop_length = 512 # number audio of frames between STFT columns (looks like a

# Short-time Fourier transform (STFT)
D = np.abs(librosa.stft(audio_file, n_fft = n_fft, hop_length = hop_length))

print('Shape of D object:', np.shape(D))
```

```
Shape of D object: (1025, 1293)
```

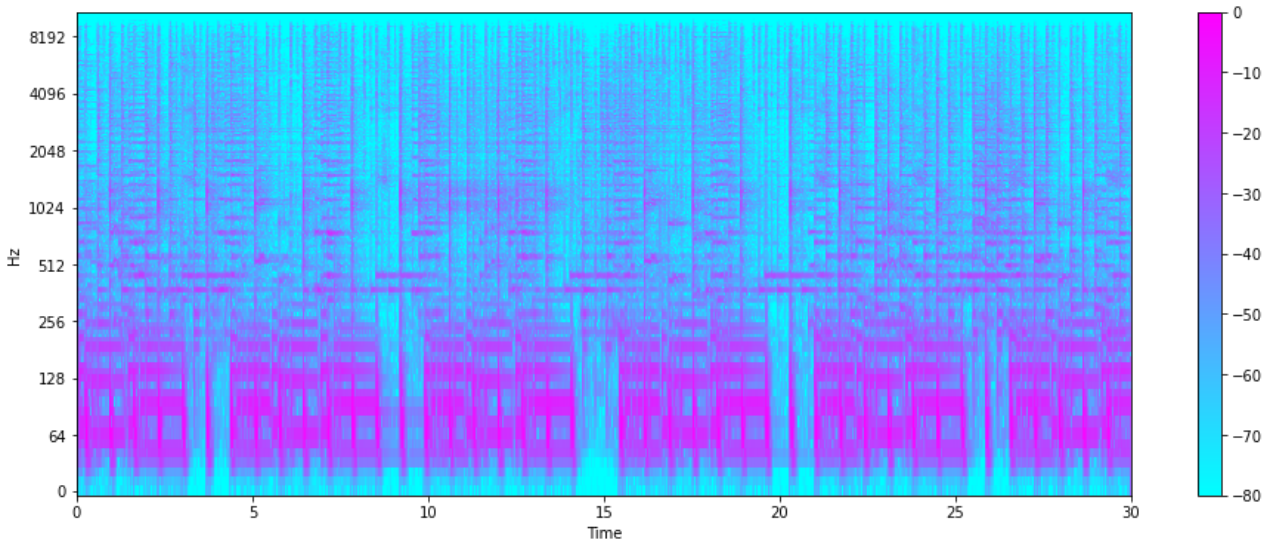
In [7]:
```
plt.figure(figsize = (16, 6))
plt.plot(D);
```



Un **espectrograma** es una representación visual del espectro de frecuencias de sonido u otras señales a medida que varían con el tiempo. Los espectrogramas a veces se denominan ecografías, huellas de voz o diagramas de voz. Cuando los datos se representan en un gráfico 3D, se les puede llamar cascadas. En matrices bidimensionales, el primer eje es la frecuencia mientras que el segundo eje es el tiempo.

librosa.amplitude_to_db: Convierte un espectrograma de amplitud en un espectrograma con escala dB.

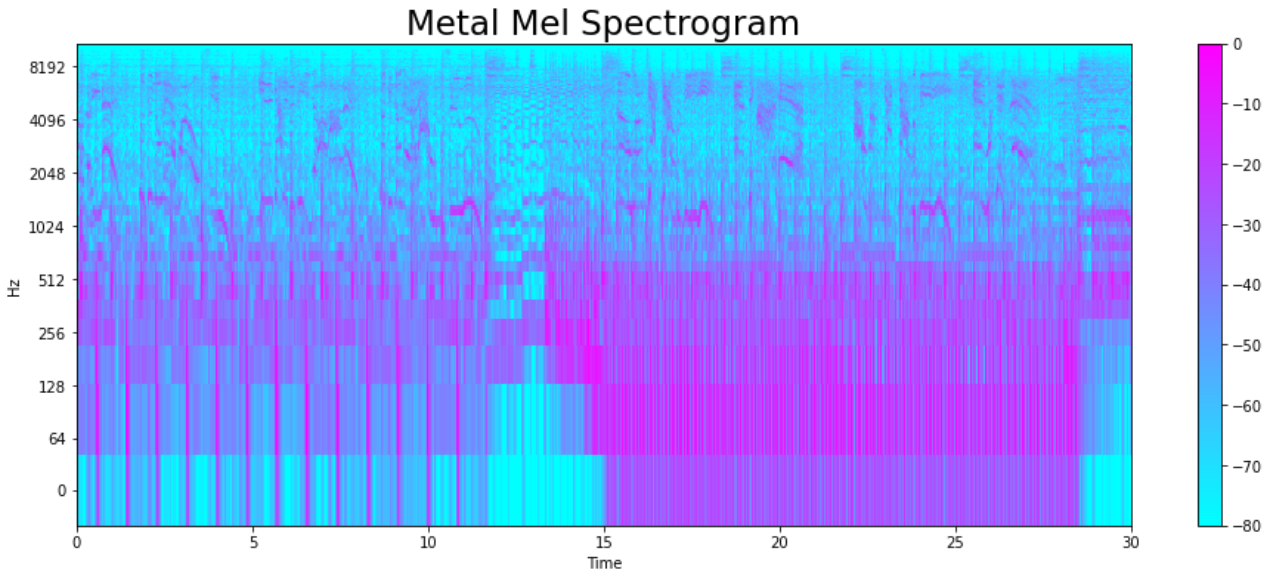
In [8]:
```
# Convert an amplitude spectrogram to Decibels-scaled spectrogram.
DB = librosa.amplitude_to_db(D, ref = np.max)

# Creating the Spectogram
plt.figure(figsize = (16, 6))
librosa.display.specshow(DB, sr = sr, hop_length = hop_length, x_axis = 'tim
                         cmap = 'cool')
plt.colorbar();
```



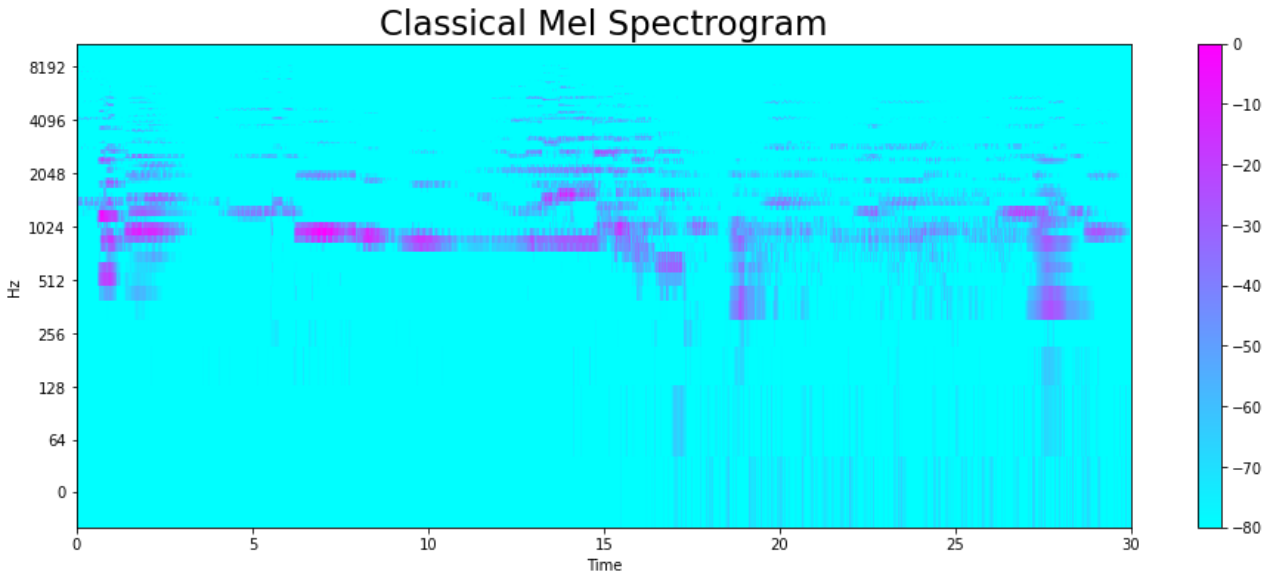
El eje vertical muestra las frecuencias y el eje horizontal muestra el tiempo del clip. Dado que toda la acción tiene lugar en la parte inferior del espectro, podemos convertir el eje de frecuencia en uno logarítmico. A continuación se presenta lo anteriormente mencionado.

librosa.feature.melspectrogram: Calcula un espectrograma a escala de mel

In [9]:
```
y, sr = librosa.load(f'{general_path}/genres_original/metal/metal.00036.wav'
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'
                         cmap = 'cool');
plt.colorbar();
plt.title("Metal Mel Spectrogram", fontsize = 23);
```



In [10]:
```
y, sr = librosa.load(f'{general_path}/genres_original/classical/classical.00
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'
                         cmap = 'cool');
plt.colorbar();
plt.title("Classical Mel Spectrogram", fontsize = 23);
```



Cada señal de audio consta de muchas características. Sin embargo, debemos extraer las características que son relevantes para el problema que estamos tratando de resolver. El proceso de extracción de características para utilizarlas en el análisis se denomina extracción de características.

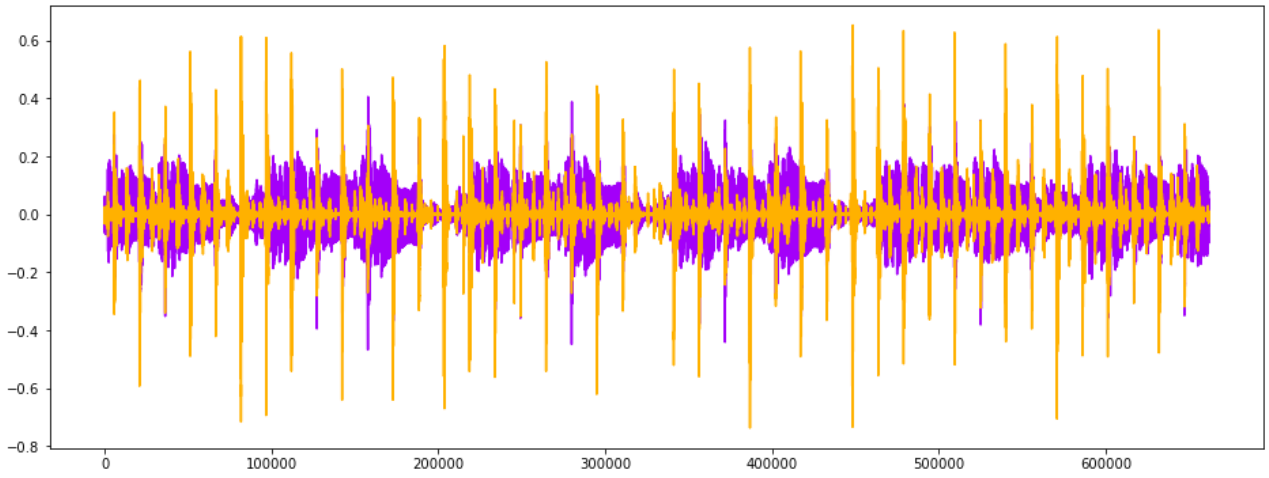
El **zero-crossing rate** es la tasa de cambios de signo junto con una señal, es decir, la tasa a la que la señal cambia de positiva a negativa o viceversa. Esta función se ha utilizado mucho tanto en el reconocimiento de voz como en la recuperación de información musical. Por lo general, tiene valores más altos para sonidos de gran percusión como los del metal y el rock.

In [11]:
```
# Total zero_crossings in our 1 song
zero_crossings = librosa.zero_crossings(audio_file, pad=False)
print(sum(zero_crossings))
```

```
39232
```

librosa.effects.hpss: Descompone una serie temporal de audio en componentes armónicos y de percusión.

In [12]:
```
y_harm, y_perc = librosa.effects.hpss(audio_file)

plt.figure(figsize = (16, 6))
plt.plot(y_harm, color = '#A300F9');
plt.plot(y_perc, color = '#FFB100');
```



librosa.beat.beat_track: Los beats se detectan en tres etapas, siguiendo el método de:

1. Medir la fuerza de inicio
2. Estimar el tempo desde la correlación de inicio
3. Se eligen picos en la fuerza de inicio aproximadamente consistentes con el tempo estimado

In [13]:
```
tempo, _ = librosa.beat.beat_track(y, sr = sr)
tempo
```

Out[13]:
```
107.666015625
```

Spectral Centroid

Indica dónde se encuentra el "centro de masa" de un sonido y se calcula como la media ponderada de las frecuencias presentes en el sonido. Si consideramos dos canciones, una de un género de blues y la otra perteneciente al metal. Ahora, en comparación con la canción de género blues, que es la misma en toda su duración, la canción de metal tiene más frecuencias hacia el final. Por lo tanto, el centroide espectral de una canción de blues estará cerca de la mitad de su espectro, mientras que el de una canción de metal estaría hacia el final.

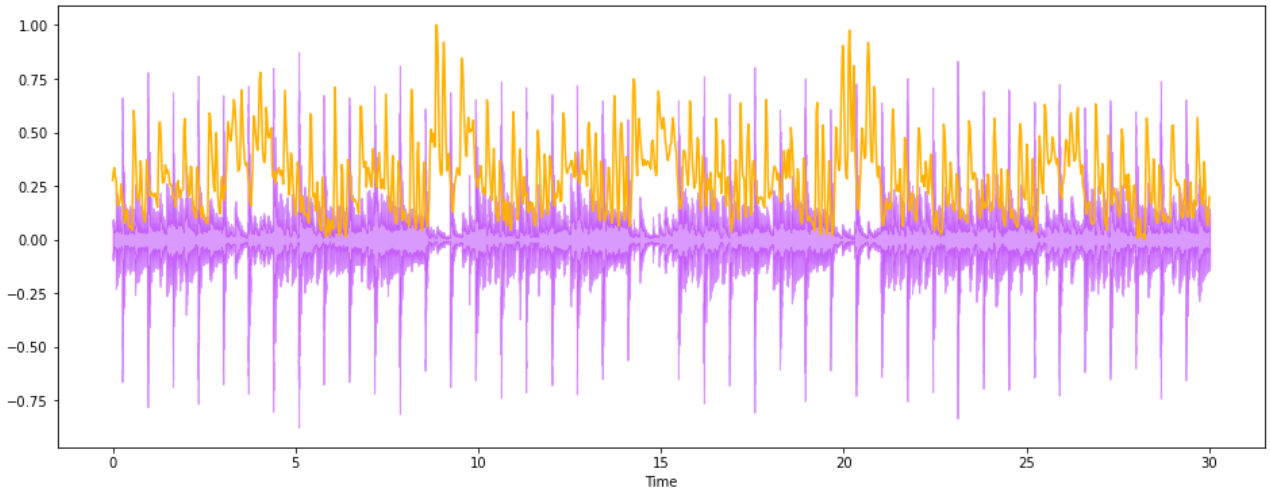
librosa.feature.spectral_centroid: Calcula el centroide espectral. Cada cuadro de un espectrograma de magnitud se normaliza y se trata como una distribución sobre intervalos de frecuencia, de los cuales se extrae la media (centroide) por cuadro.

librosa.frames_to_time: convierte la cuenta de frames en tiempo (en segundos).

In [14]:
```
# Calculate the Spectral Centroids
spectral_centroids = librosa.feature.spectral_centroid(audio_file, sr=sr)[0]

# Shape is a vector
print('Centroids:', spectral_centroids, '\n')
print('Shape of Spectral Centroids:', spectral_centroids.shape, '\n')

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

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

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

# Function that normalizes the Sound Data
def normalize(x, axis=0):
    return sklearn.preprocessing.minmax_scale(x, axis=axis)
```

```
Centroids: [1758.29476432 1946.74243678 2038.8113414  ...  766.50416352 1041
.07728901
 1391.05145642]

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 [15]:
```
#Plotting the Spectral Centroid along the waveform
plt.figure(figsize = (16, 6))
librosa.display.waveshow(audio_file, sr=sr, alpha=0.4, color = '#A300F9');
plt.plot(t, normalize(spectral_centroids), color='#FFB100');
```

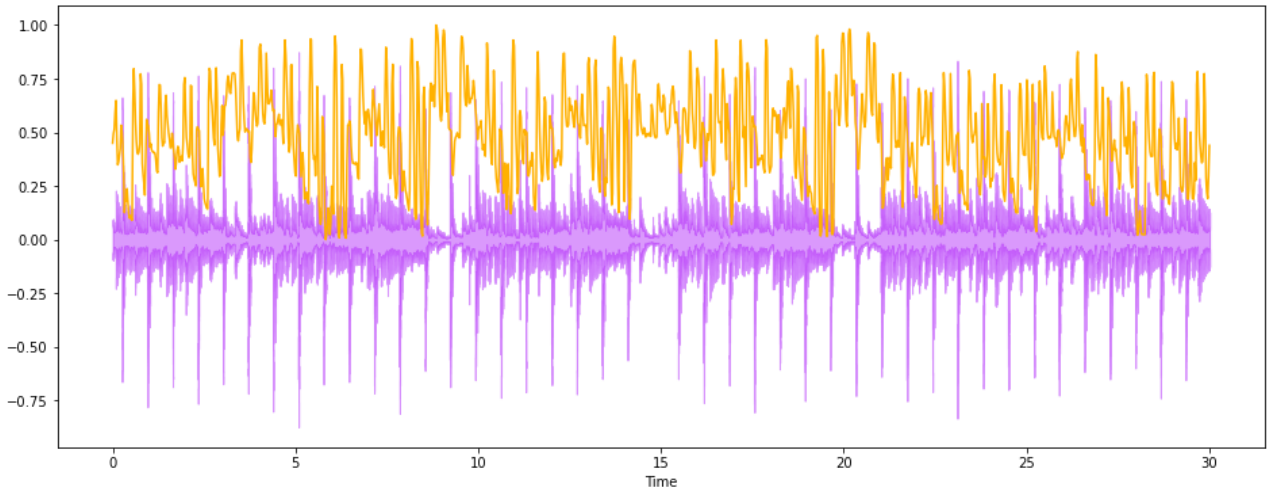


Spectral Rolloff

Es una medida de la forma de la señal. Representa la frecuencia por debajo de la cual un porcentaje específico de la energía espectral total.

librosa.feature.spectral_rolloff: La frecuencia de caída se define para cada fotograma como la frecuencia central de un contenedor de espectrograma de modo que al menos el porcentaje de caída de la energía del espectro en este marco esté contenido en este contenedor y los contenedores inferiores. Esto se puede usar para, por ejemplo, aproximar la frecuencia máxima (o mínima) configurando roll_percent en un valor cercano a 1.

In [16]:
```
# Spectral RollOff Vector
spectral_rolloff = librosa.feature.spectral_rolloff(audio_file, sr=sr)[0]

# The plot
plt.figure(figsize = (16, 6))
librosa.display.waveshow(audio_file, sr=sr, alpha=0.4, color = '#A300F9');
plt.plot(t, normalize(spectral_rolloff), color='#FFB100');
```



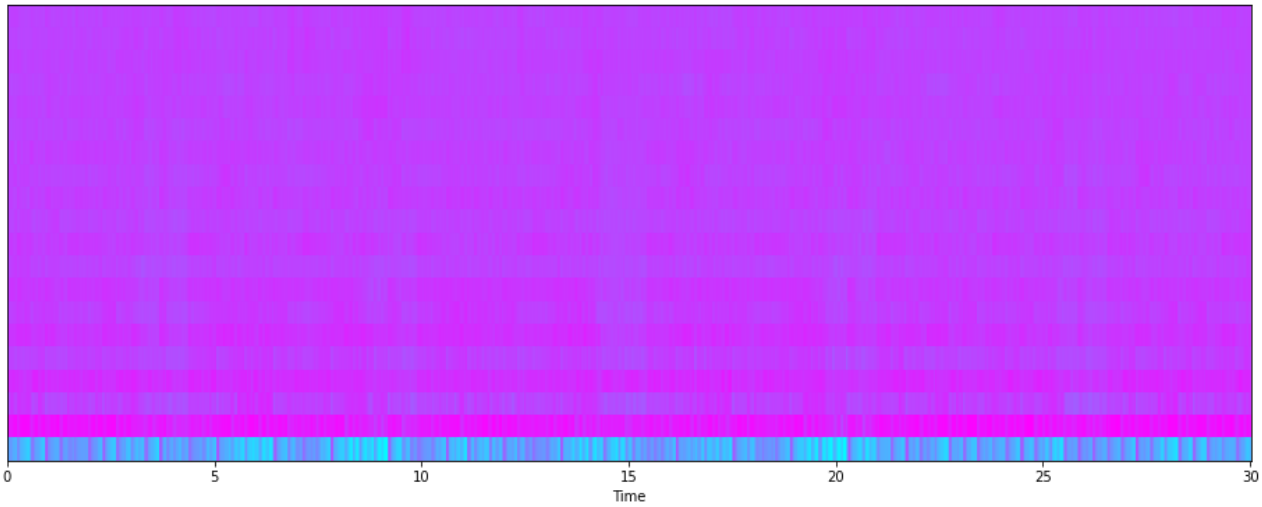
Mel-Frequency Cepstral Coefficients

Son un pequeño conjunto de características (generalmente entre 10 y 20) que describen de manera concisa la forma general de una envolvente espectral. Modela las características de la voz humana.

librosa.feature.mfcc: Obtiene los coeficientes MFCC.

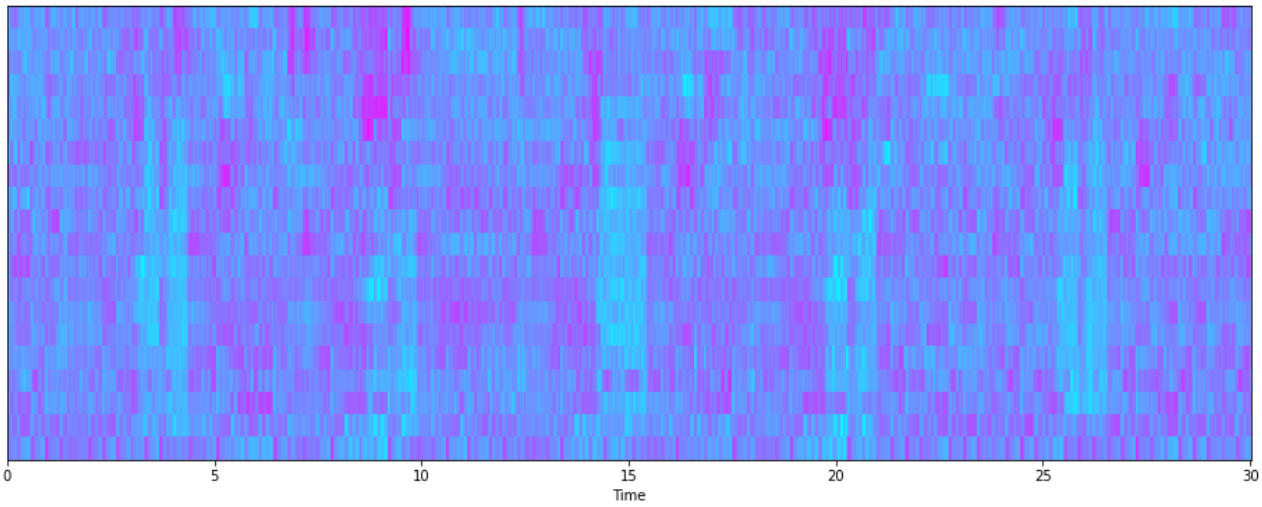
In [17]:
```
mfccs = librosa.feature.mfcc(audio_file, sr=sr)
print('mfccs shape:', mfccs.shape)

#Displaying  the MFCCs:
plt.figure(figsize = (16, 6))
librosa.display.specshow(mfccs, sr=sr, x_axis='time', cmap = 'cool');
```

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



In [18]:
```
# Perform Feature Scaling
mfccs = sklearn.preprocessing.scale(mfccs, axis=1)
print('Mean:', mfccs.mean(), '\n')
print('Var:', mfccs.var())

plt.figure(figsize = (16, 6))
librosa.display.specshow(mfccs, sr=sr, x_axis='time', cmap = 'cool');
```

```
Mean: -2.9502685e-09

Var: 1.0000001
```



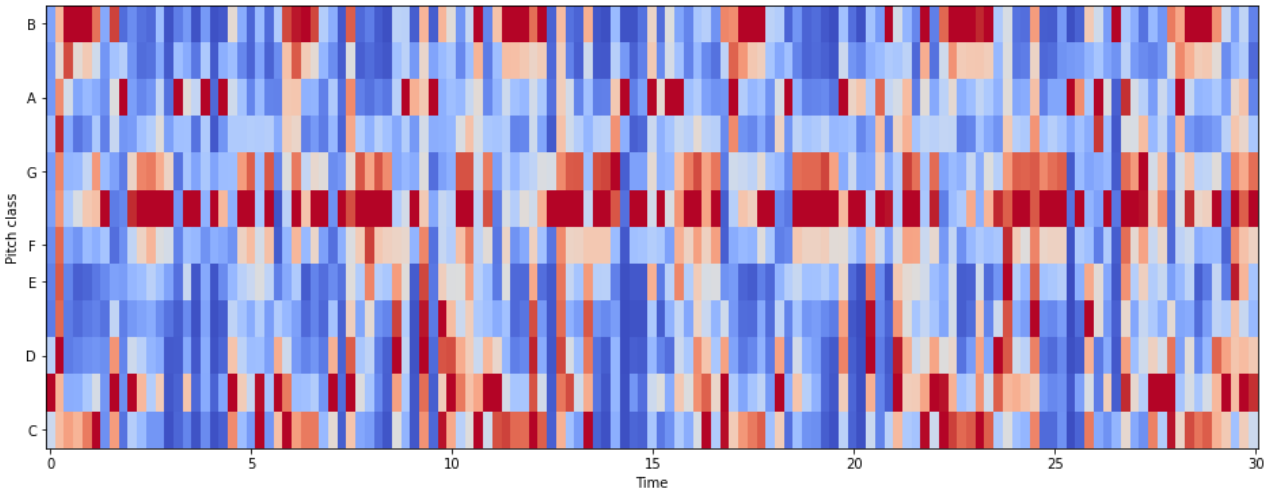
Chroma Frequencies

Son una representación interesante y poderosa para el audio de música, en la que todo el espectro se proyecta en 12 contenedores que representan los 12 semitonos distintos (o croma) de la octava musical.

librosa.feature.chroma_stft: Calcula un cromagrama a partir de una forma de onda o un espectrograma de potencia.

In [19]:
```
# Increase or decrease hop_length to change how granular you want your data
hop_length = 5000

# Chromogram
chromagram = librosa.feature.chroma_stft(audio_file, sr=sr, hop_length=hop_l
print('Chromogram shape:', chromagram.shape)

plt.figure(figsize=(16, 6))
librosa.display.specshow(chromagram, x_axis='time', y_axis='chroma', hop_len
```

```
Chromogram shape: (12, 133)
```



In [20]:
```
data = pd.read_csv(f'{general_path}/features_30_sec.csv')
data.head()
```

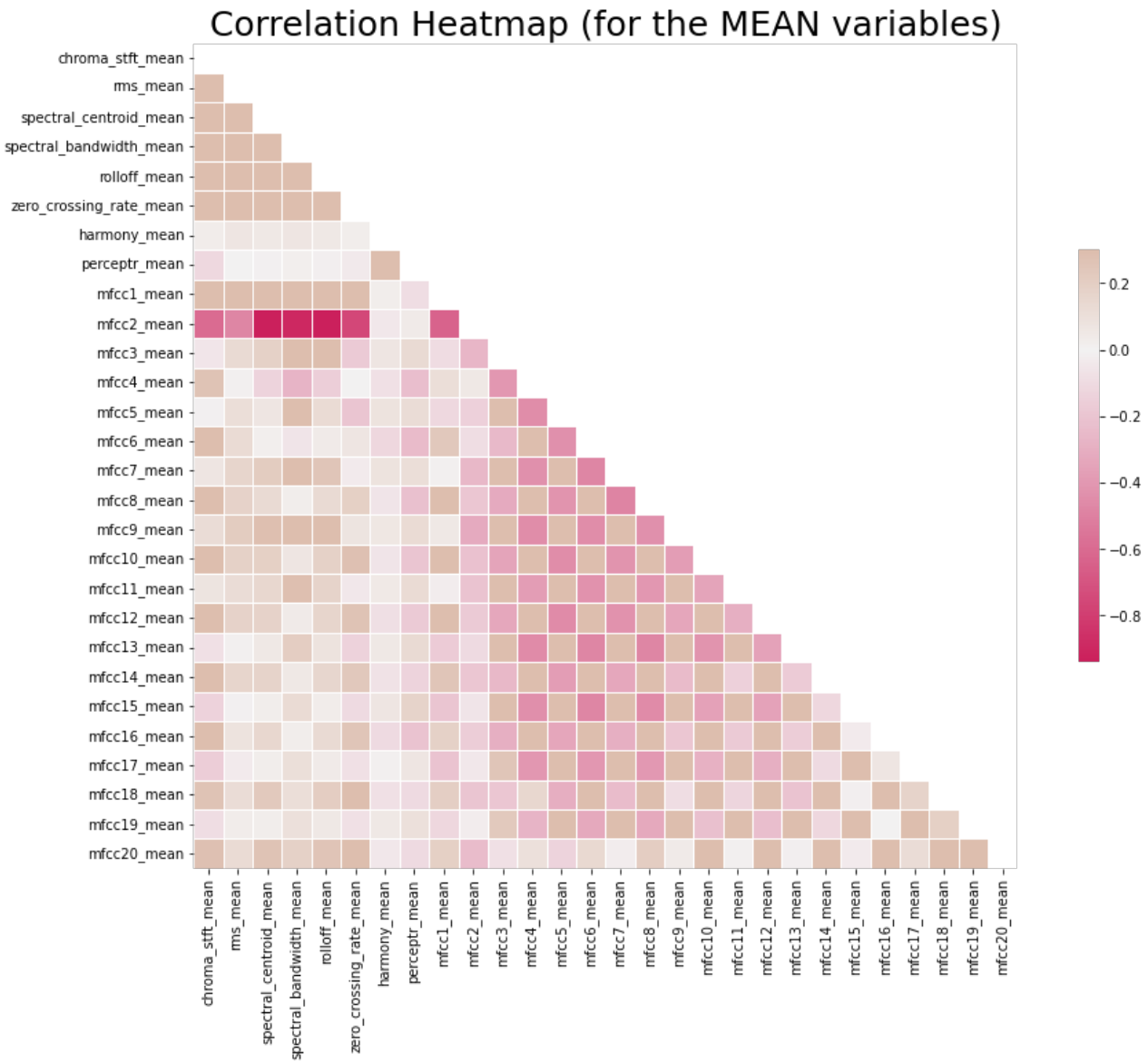
Out[20]:

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

5 rows × 60 columns

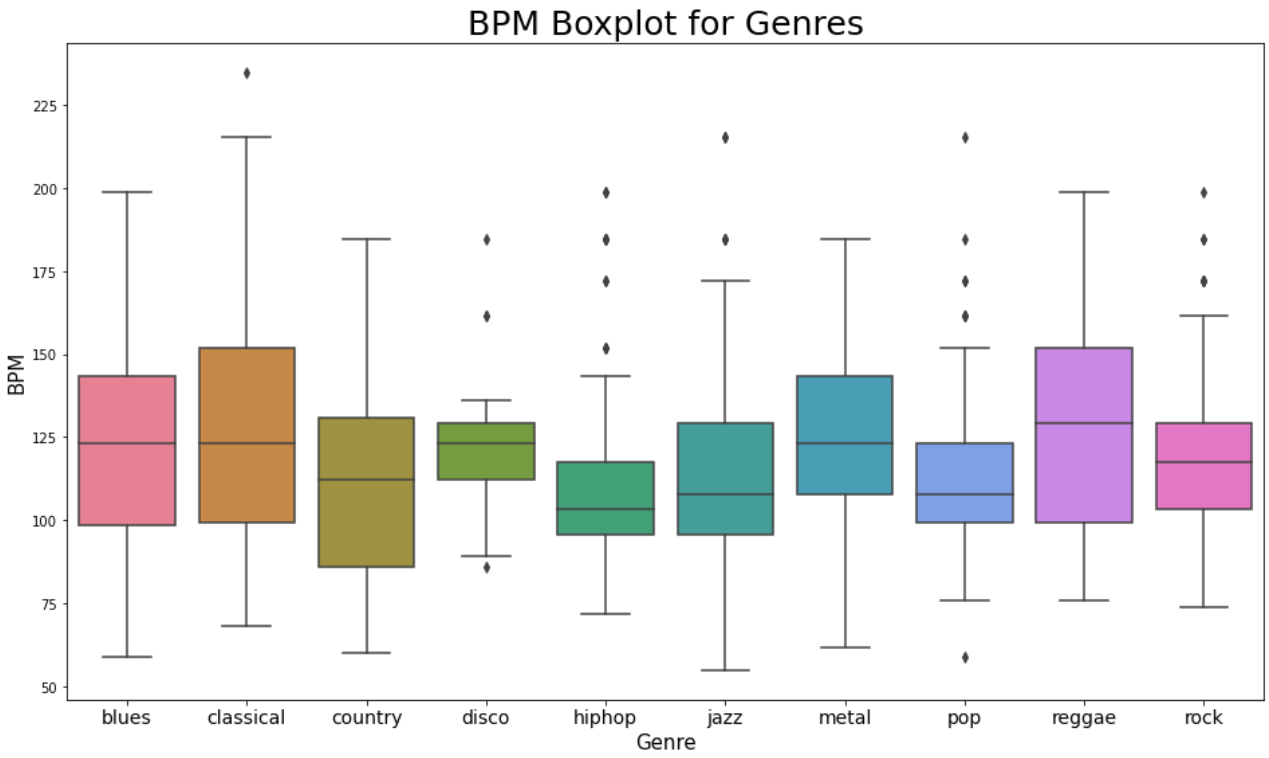
In [21]:

```
# Computing the Correlation Matrix
spike_cols = [col for col in data.columns if 'mean' in col]
corr = data[spike_cols].corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=np.bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(16, 11));

# Generate a custom diverging colormap
cmap = sns.diverging_palette(0, 25, as_cmap=True, s = 90, l = 45, n = 5)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})

plt.title('Correlation Heatmap (for the MEAN variables)', fontsize = 25)
plt.xticks(fontsize = 10)
plt.yticks(fontsize = 10);
plt.savefig("Corr Heatmap.jpg")
```



A continuación tenemos los diagramas de cajas de cada uno de los diferentes generos, podemos darnos cuenta que los géneros hiphop y pop parecen ser los que más tienen datos atípicos.

In [22]:
```
x = data[["label", "tempo"]]

f, ax = plt.subplots(figsize=(16, 9));
sns.boxplot(x = "label", y = "tempo", data = x, palette = 'husl');

plt.title('BPM Boxplot for Genres', fontsize = 25)
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 10);
plt.xlabel("Genre", fontsize = 15)
plt.ylabel("BPM", fontsize = 15)
plt.savefig("BPM Boxplot.jpg")
```

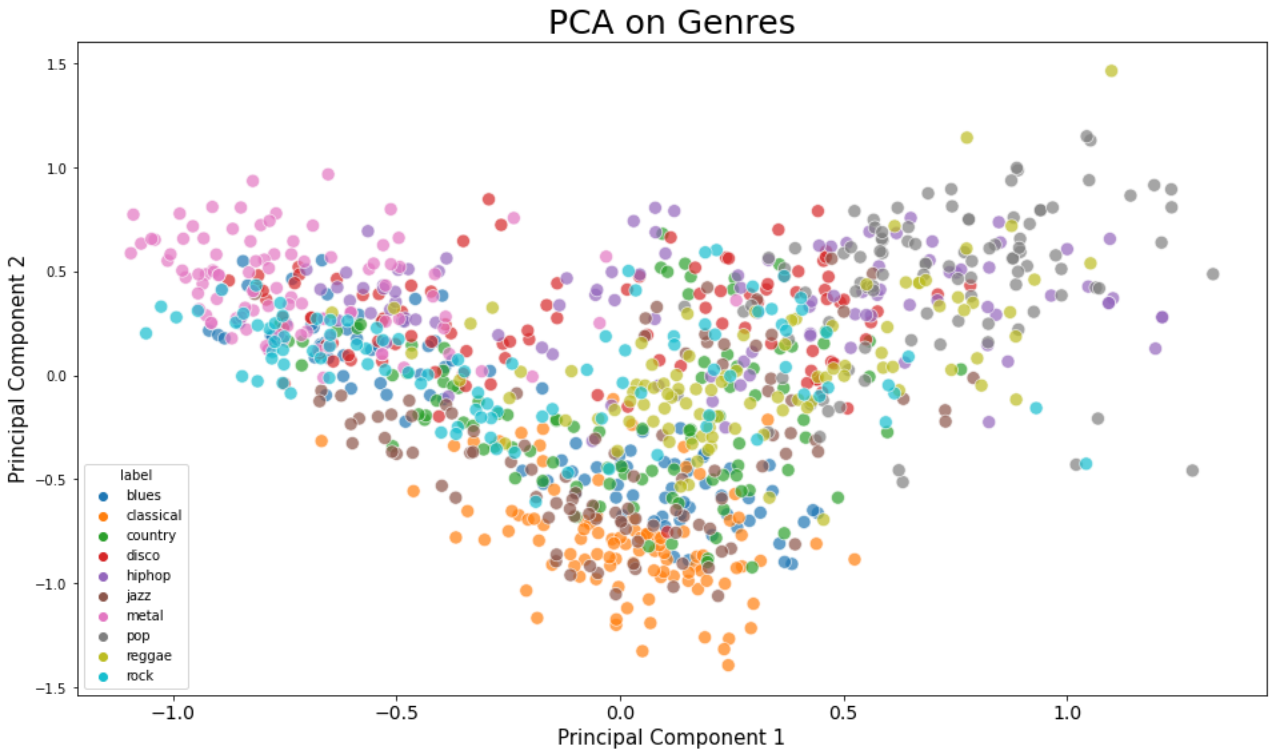


In [23]:
```
from sklearn import preprocessing

data = data.iloc[0:, 1:]
y = data['label']
X = data.loc[:, data.columns != 'label']

#### NORMALIZE X ####
cols = X.columns
min_max_scaler = preprocessing.MinMaxScaler()
np_scaled = min_max_scaler.fit_transform(X)
X = pd.DataFrame(np_scaled, columns = cols)


#### PCA 2 COMPONENTS ####
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
principalComponents = pca.fit_transform(X)
principalDf = pd.DataFrame(data = principalComponents, columns = ['principal

# concatenate with target label
finalDf = pd.concat([principalDf, y], axis = 1)

pca.explained_variance_ratio_

# 44.93 variance explained
```

Out[23]:
```
array([0.2439355 , 0.21781804])
```

A continuación tenemos el scatter plot de nuestros datos, sin embargo, **este gráfico solamente explica el 44.93% de la variación**, así que se podría decir que no es del todo significativo.

In [24]:
```
plt.figure(figsize = (16, 9))
sns.scatterplot(x = "principal component 1", y = "principal component 2", da
                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)
plt.savefig("PCA Scattert.jpg")
```



In [25]:
```
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.metrics import confusion_matrix, accuracy_score, roc_auc_score,
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE
```

In [26]:
```
data = pd.read_csv(f'{general_path}/features_3_sec.csv')
data = data.iloc[0:, 1:]
data.head()
```

Out[26]:

	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mea
0	66149	0.335406	0.091048	0.130405	0.003521	1773.06503
1	66149	0.343065	0.086147	0.112699	0.001450	1816.69377
2	66149	0.346815	0.092243	0.132003	0.004620	1788.53971
3	66149	0.363639	0.086856	0.132565	0.002448	1655.28904
4	66149	0.335579	0.088129	0.143289	0.001701	1630.65619

5 rows × 59 columns

In [27]:
```
y = data['label'] # genre variable.
X = data.loc[:, data.columns != 'label'] #select all columns but not the lab

#### NORMALIZE X ####

# Normalize so everything is on the same scale.

cols = X.columns
min_max_scaler = preprocessing.MinMaxScaler()
np_scaled = min_max_scaler.fit_transform(X)

# new data frame with the new scaled data.
X = pd.DataFrame(np_scaled, columns = cols)
```

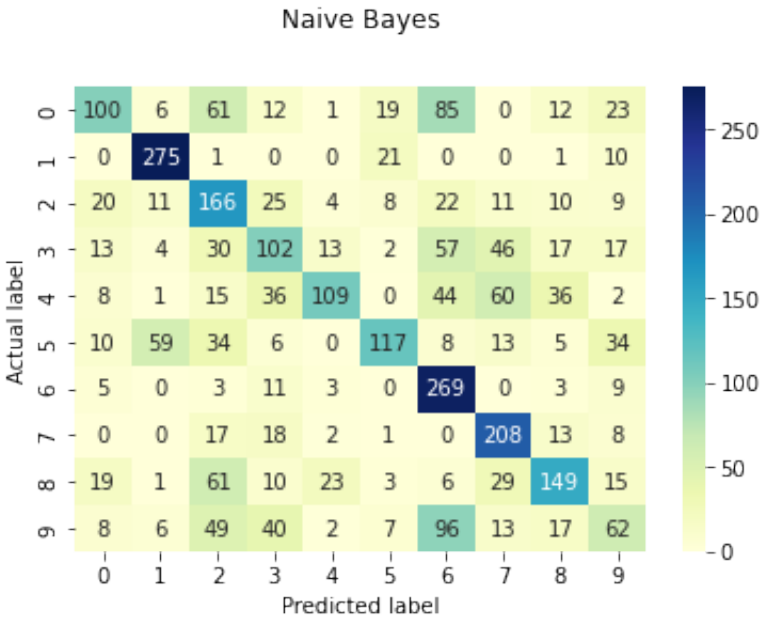
In [28]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ran
```

In [29]:
```
import seaborn as sns
from sklearn.metrics import accuracy_score,f1_score,confusion_matrix,classif
```

In [30]:
```
def model_assess(model, title = "Default"):
    model.fit(X_train, y_train)
    preds = model.predict(X_test)

    cm = confusion_matrix(y_test, preds)
    p = sns.heatmap(pd.DataFrame(cm), annot=True, cmap="YlGnBu" ,fmt='g')
    plt.title(title, y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()
    print(classification_report(y_test, preds))
    print('Accuracy', title, ':', round(accuracy_score(y_test, preds), 5), '
```

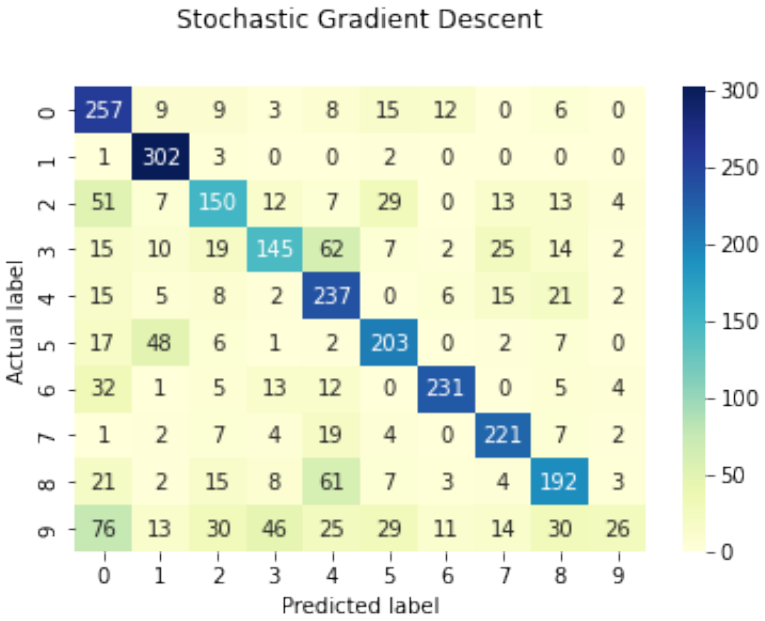
Modelos

In [31]:
```
# Naive Bayes
nb = GaussianNB()
model_assess(nb, "Naive Bayes")

# Stochastic Gradient Descent
sgd = SGDClassifier(max_iter=5000, random_state=0)
model_assess(sgd, "Stochastic Gradient Descent")

# KNN
knn = KNeighborsClassifier(n_neighbors=19)
model_assess(knn, "KNN")

# Decission trees
tree = DecisionTreeClassifier()
model_assess(tree, "Decission trees")

# Random Forest
rforest = RandomForestClassifier(n_estimators=1000, max_depth=10, random_sta
model_assess(rforest, "Random Forest")

# Support Vector Machine
svm = SVC(decision_function_shape="ovo")
model_assess(svm, "Support Vector Machine")

# Logistic Regression
lg = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinc
model_assess(lg, "Logistic Regression")
```



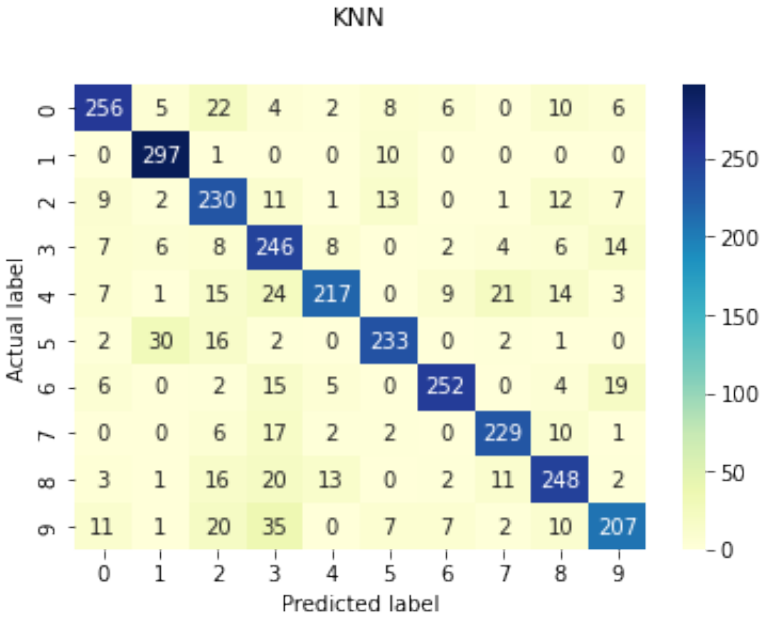
	precision	recall	f1-score	support
blues	0.55	0.31	0.40	319
classical	0.76	0.89	0.82	308
country	0.38	0.58	0.46	286
disco	0.39	0.34	0.36	301
hiphop	0.69	0.35	0.47	311
jazz	0.66	0.41	0.50	286
metal	0.46	0.89	0.60	303
pop	0.55	0.78	0.64	267
reggae	0.57	0.47	0.51	316
rock	0.33	0.21	0.25	300
accuracy			0.52	2997
macro avg	0.53	0.52	0.50	2997
weighted avg	0.53	0.52	0.50	2997

Accuracy Naive Bayes : 0.51952



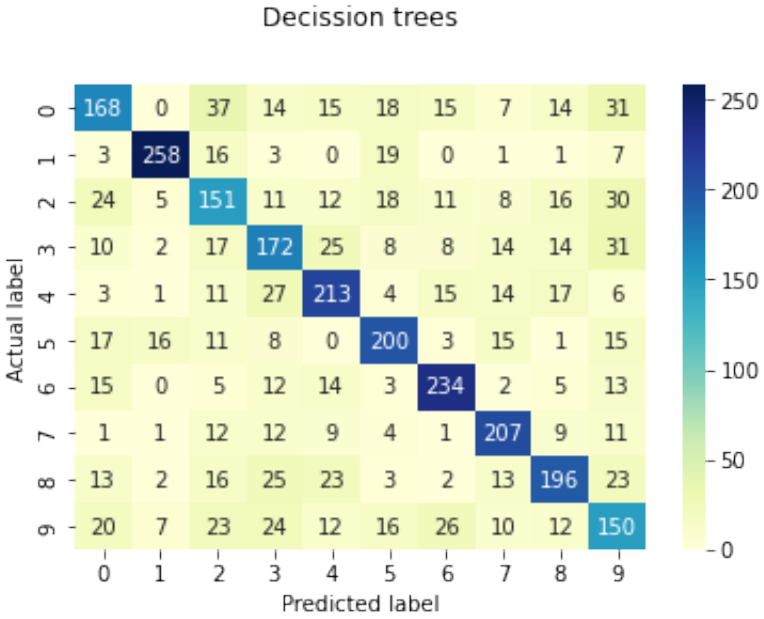
	precision	recall	f1-score	support
blues	0.53	0.81	0.64	319
classical	0.76	0.98	0.85	308
country	0.60	0.52	0.56	286
disco	0.62	0.48	0.54	301
hiphop	0.55	0.76	0.64	311
jazz	0.69	0.71	0.70	286
metal	0.87	0.76	0.81	303
pop	0.75	0.83	0.79	267
reggae	0.65	0.61	0.63	316
rock	0.60	0.09	0.15	300
accuracy			0.66	2997
macro avg	0.66	0.65	0.63	2997
weighted avg	0.66	0.66	0.63	2997

Accuracy Stochastic Gradient Descent : 0.65532



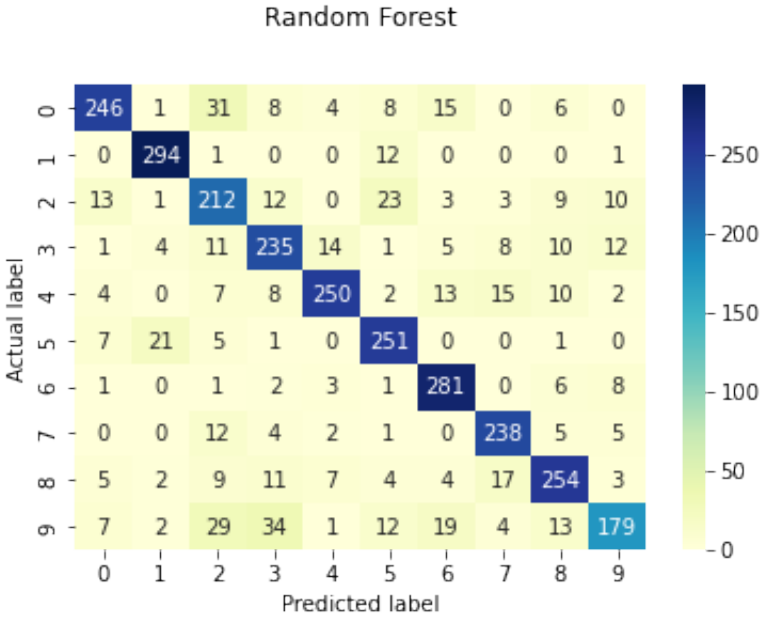
	precision	recall	f1-score	support
blues	0.85	0.80	0.83	319
classical	0.87	0.96	0.91	308
country	0.68	0.80	0.74	286
disco	0.66	0.82	0.73	301
hiphop	0.88	0.70	0.78	311
jazz	0.85	0.81	0.83	286
metal	0.91	0.83	0.87	303
pop	0.85	0.86	0.85	267
reggae	0.79	0.78	0.79	316
rock	0.80	0.69	0.74	300
accuracy			0.81	2997
macro avg	0.81	0.81	0.81	2997
weighted avg	0.81	0.81	0.81	2997

Accuracy KNN : 0.80581



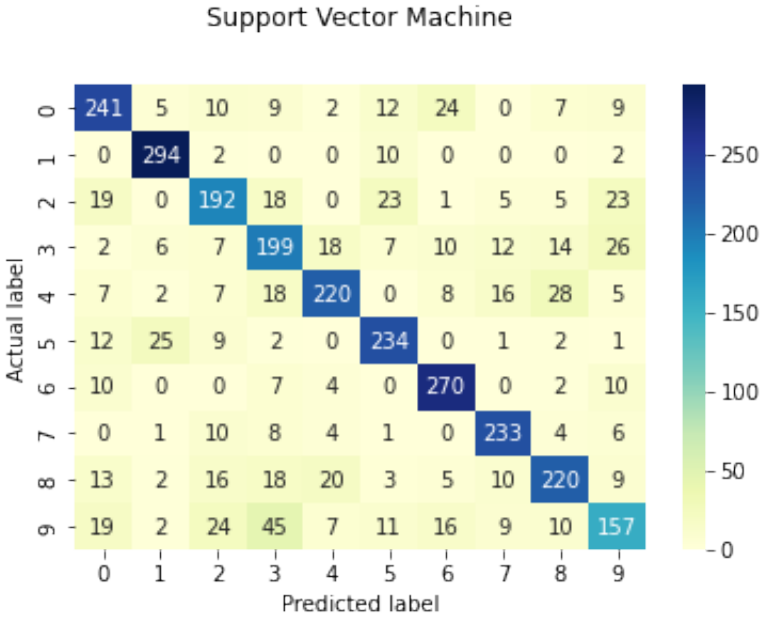
	precision	recall	f1-score	support
blues	0.61	0.53	0.57	319
classical	0.88	0.84	0.86	308
country	0.51	0.53	0.52	286
disco	0.56	0.57	0.56	301
hiphop	0.66	0.68	0.67	311
jazz	0.68	0.70	0.69	286
metal	0.74	0.77	0.76	303
pop	0.71	0.78	0.74	267
reggae	0.69	0.62	0.65	316
rock	0.47	0.50	0.49	300
accuracy			0.65	2997
macro avg	0.65	0.65	0.65	2997
weighted avg	0.65	0.65	0.65	2997

Accuracy Decission trees : 0.65032



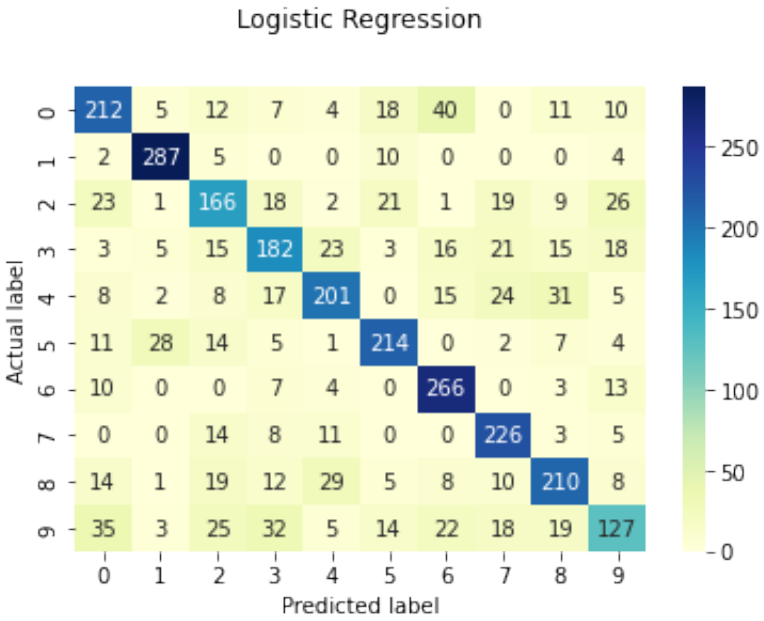
	precision	recall	f1-score	support
blues	0.87	0.77	0.82	319
classical	0.90	0.95	0.93	308
country	0.67	0.74	0.70	286
disco	0.75	0.78	0.76	301
hiphop	0.89	0.80	0.84	311
jazz	0.80	0.88	0.84	286
metal	0.83	0.93	0.87	303
pop	0.84	0.89	0.86	267
reggae	0.81	0.80	0.81	316
rock	0.81	0.60	0.69	300
accuracy			0.81	2997
macro avg	0.82	0.81	0.81	2997
weighted avg	0.82	0.81	0.81	2997

Accuracy Random Forest : 0.81415



	precision	recall	f1-score	support
blues	0.75	0.76	0.75	319
classical	0.87	0.95	0.91	308
country	0.69	0.67	0.68	286
disco	0.61	0.66	0.64	301
hiphop	0.80	0.71	0.75	311
jazz	0.78	0.82	0.80	286
metal	0.81	0.89	0.85	303
pop	0.81	0.87	0.84	267
reggae	0.75	0.70	0.72	316
rock	0.63	0.52	0.57	300
accuracy			0.75	2997
macro avg	0.75	0.76	0.75	2997
weighted avg	0.75	0.75	0.75	2997

Accuracy Support Vector Machine : 0.75409



	precision	recall	f1-score	support
blues	0.67	0.66	0.67	319
classical	0.86	0.93	0.90	308
country	0.60	0.58	0.59	286
disco	0.63	0.60	0.62	301
hiphop	0.72	0.65	0.68	311
jazz	0.75	0.75	0.75	286
metal	0.72	0.88	0.79	303
pop	0.71	0.85	0.77	267
reggae	0.68	0.66	0.67	316
rock	0.58	0.42	0.49	300
accuracy			0.70	2997
macro avg	0.69	0.70	0.69	2997
weighted avg	0.69	0.70	0.69	2997

Accuracy Logistic Regression : 0.6977

Comparación de modelos

Modelo	F1-Score										Accuracy
	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock	
Naive Bayes	0.4	0.82	0.46	0.36	0.47	0.5	0.6	0.64	0.51	0.25	0.52
SGD	0.64	0.85	0.56	0.54	0.64	0.7	0.81	0.79	0.63	0.15	0.66
KNN	0.83	0.91	0.74	0.73	0.78	0.83	0.87	0.85	0.79	0.74	0.81
Decision Tree	0.57	0.86	0.52	0.56	0.67	0.69	0.76	0.74	0.65	0.49	0.65
Random Forest	0.82	0.93	0.7	0.76	0.84	0.84	0.87	0.86	0.81	0.69	0.81
SVM	0.75	0.91	0.68	0.64	0.75	0.8	0.85	0.84	0.72	0.57	0.75
Logistic Regression	0.67	0.9	0.59	0.62	0.68	0.75	0.79	0.77	0.67	0.49	0.70

En la parte de arriba tenemos una tabla comparativa de los distintos modelos que probamos en la clasificación de géneros de audio. Todos los modelos clasificaron con un excelente desempeño la música clásica, esto debido a que debe de tener ciertas características muy distintivas entre las demás. A diferencia de esta, los modelos tuvieron problemas clasificando la música rock, la cual debe de tener alguna implicación la cual los modelos no alcanzan a distinguir de la mejor manera dicha clase. El modelo con el peor accuracy es el Naive Bayes, recordando que este modelo trabaja con la probabilidad condicional, existe la posibilidad que este método de clasificación de probabilidades de pertenencia a clases, no sea lo suficiente robusto como para alcanzar dicha distinción entre géneros.

Los dos mejores modelos para la clasificación de audio fueron el KNN y el Random Forest. Estos dos modelos tienen principios de clasificación muy distintos. El KNN coloca las observaciones en un plano de dimensión N y según sus vecinos cercanos es la clasificación que adquiere. El Random Forest es un modelo de aprendizaje en conjunto el cual está formado por un conjunto de árboles de decisión, los cuales se complementan entre sí para alcanzar el mejor desempeño posible y evitando el sobreajuste.

Conclusión

Al igual que en las otras dos etapas ya existen distintas librerías que nos permiten trabajar con audio, es sorprendente como a través del tiempo se han desarrollado librerías en Python para trabajar con datos no estructurados, las cuales tienen en común esta parte de extracción de características para pasar de un problema no estructurado a tener valores numéricos y así poder entrenar a nuestros modelos. El análisis de audio al igual que los otros 2, va cobrando relevancia en la actualidad, estos archivos multimedia cada vez son más populares debido a las nuevas tecnologías que permiten su distribución y consumo. Algunas de las aplicaciones del análisis de audio pueden revolucionar al mundo, como la detección de gritos o sonidos que representen peligro, el análisis de interacciones de llamadas, el poder detectar el canto de aves en peligro de extinción para poder ponerlas en un santuario, entre otras cosas. A pesar de que existen modelos más complejos para analizar audio, los utilizados en este notebook obtuvieron resultados aceptables. El tener modelos más complejos no garantiza un buen desempeño.

Referencias

- McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. . (2015). Librosa Doc.. 10 de Julio 2022, de Librosa Sitio web: https://librosa.org/doc/latest/index.html
- Parul Pandey. (Dec 12, 2018). Music Genre Classification with Python. 18 de Julio del 2022, de Towards Data Science Sitio web: https://towardsdatascience.com/music-genre-classification-with-python-c714d032f0d8
- ANDRADA OLTEANU. (2020). GTZAN Dataset - Music Genre Classification. 18 de Julio del 2022, de Kaggle Sitio web: https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification
- Irving Estrada. Github. 2022,Sitio web: https://github.com/Irving-Estrada/Procesamiento