Árbol de Decisión en Python

31 de Marzo de 2025

1 Introducción

Los árboles de decisión son estructuras de partición recursiva con forma de árbol, donde los nodos u hojas representan decisiones y las ramas que salen de estos nodos representan las posibles opciones. Dentro del contexto del aprendizaje automático, los árboles de decisión son modelos de aprendizaje supervisado que dividen recursivamente el espacio de características mediante reglas simples. Este método utiliza una estructura jerárquica de nodos (preguntas) y hojas (decisiones) para clasificar instancias o predecir valores continuos.

2 Metodología

Para realizar el ejercicio de árbol de decisión, se siguieron los siguientes pasos:

```
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from IPython.display import Image as PImage
from subprocess import check_call
from PIL import Image, ImageDraw, ImageFont
```

[2]:

```
artists_billboard = pd.read_csv(r"artists_billboard_fix3.csv")
```

[3]:

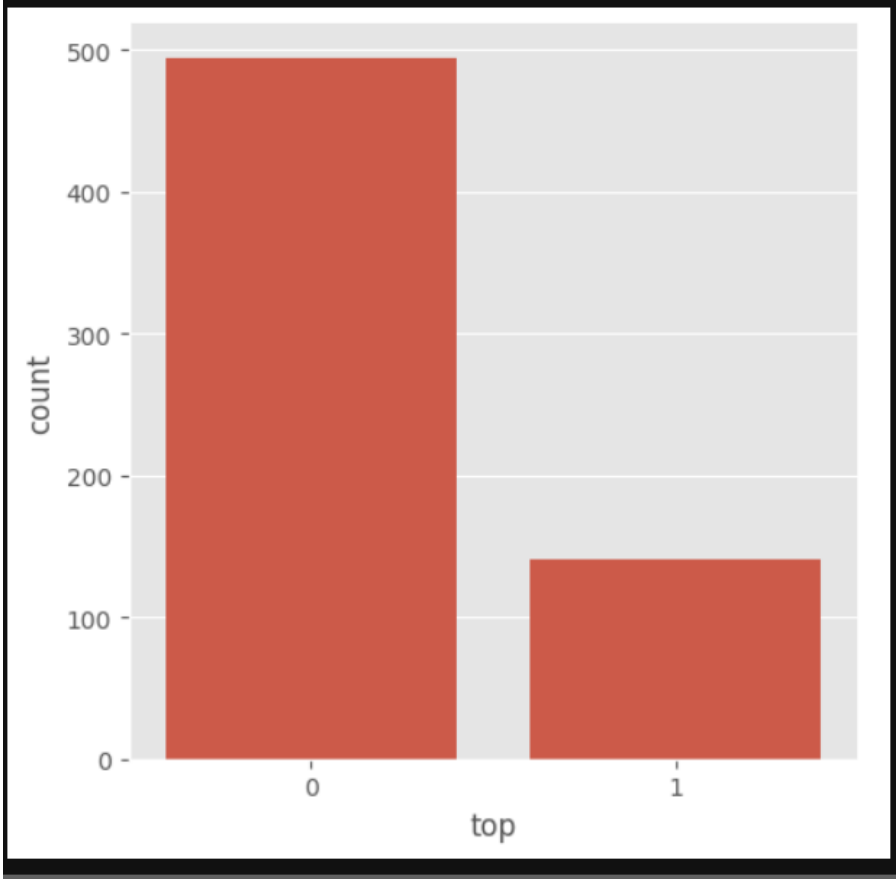
```
artists_billboard.shape
```

2.1 Visualización General de los Datos de Entrada

En esta sección se visualiza la distribución de los valores de las columnas categóricas del dataset y se analizan los factores que influyen en que una canción esté en el top de las listas.

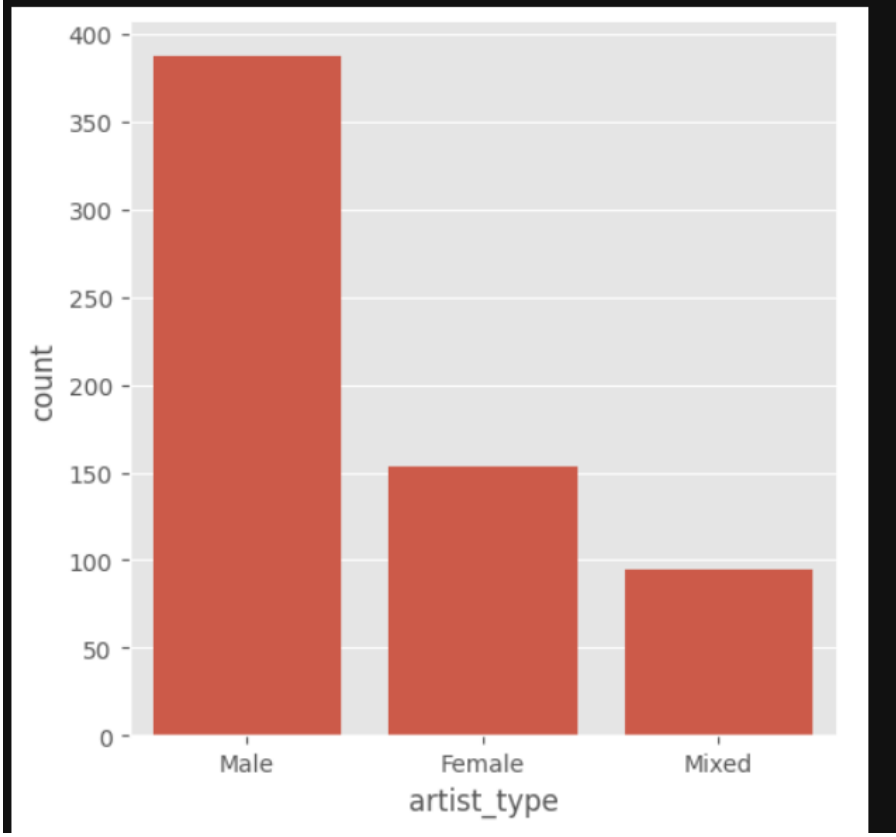
```
sb.catplot(x='top', data=artists_billboard, kind="count")
```

```
<seaborn.axisgrid.FacetGrid at 0x1b58fd653d0>
```



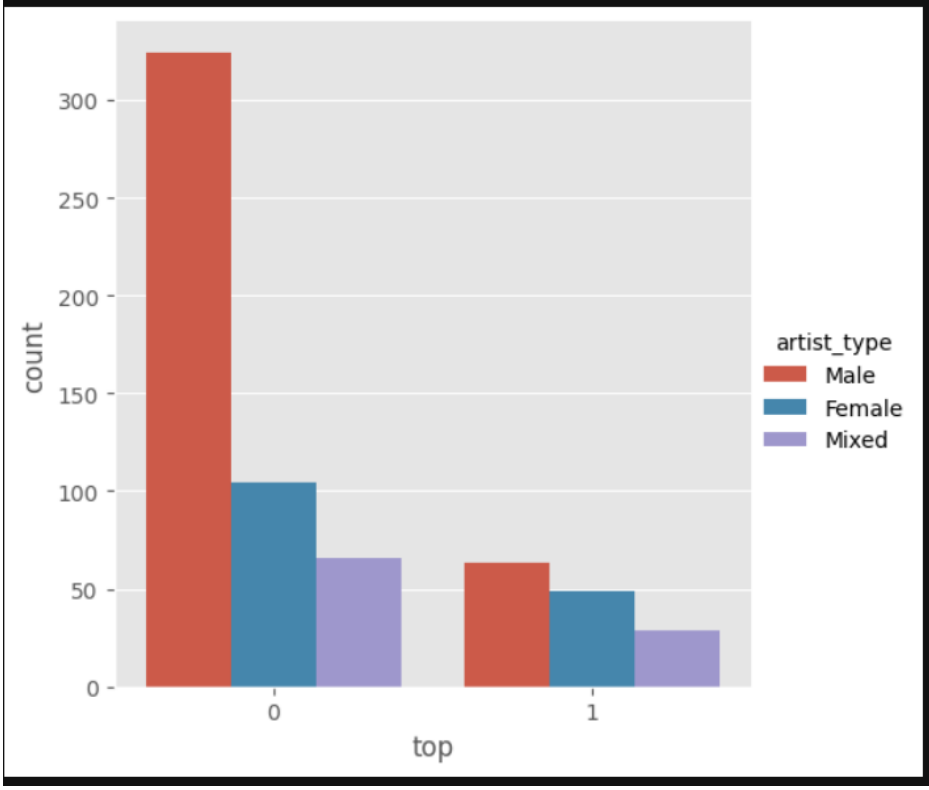

```
sb.catplot(x='artist_type', data=artists_billboard, kind="count")
```

```
<seaborn.axisgrid.FacetGrid at 0x1b5b241f0e0>
```



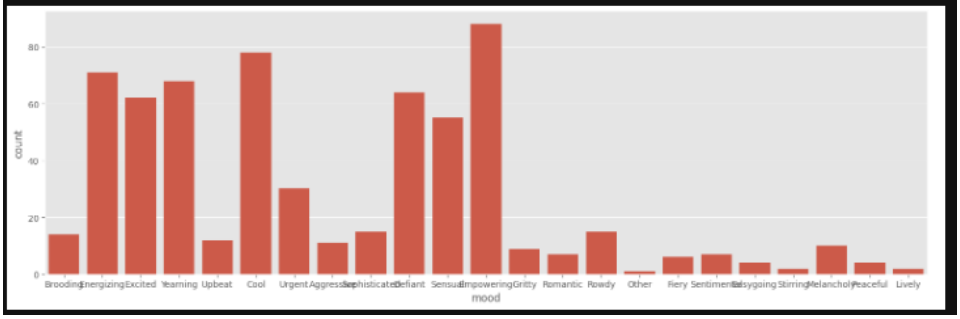
```
sb.catplot(x='top', data=artists_billboard, hue='artist_type', kind="count")
```

```
<seaborn.axisgrid.FacetGrid at 0x1b5b54a3560>
```



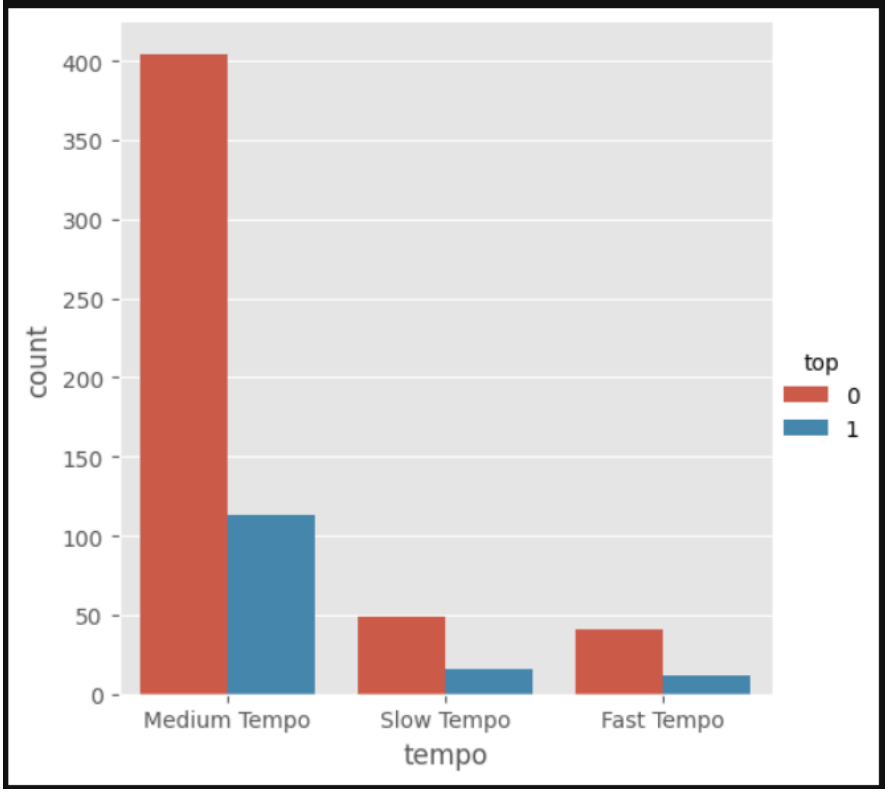

```
sb.catplot(x='mood', data=artists_billboard, aspect=3, kind="count")
```

```
<seaborn.axisgrid.FacetGrid at 0x1b5b550f0e0>
```



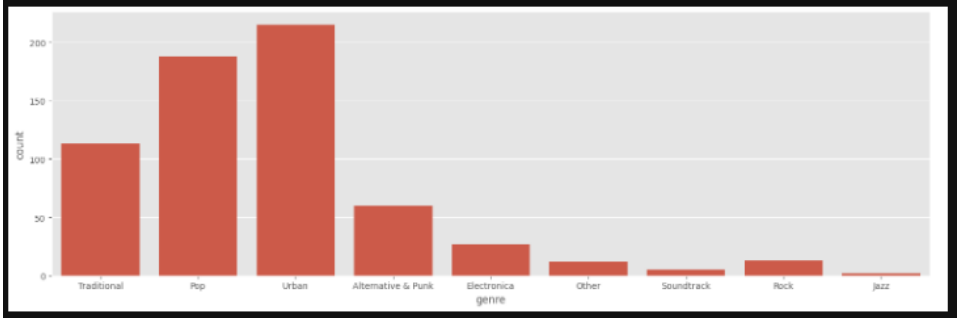
```
sb.catplot(x='tempo', data=artists_billboard, hue='top', kind="count")
```

```
<seaborn.axisgrid.FacetGrid at 0x1b5b55f9520>
```



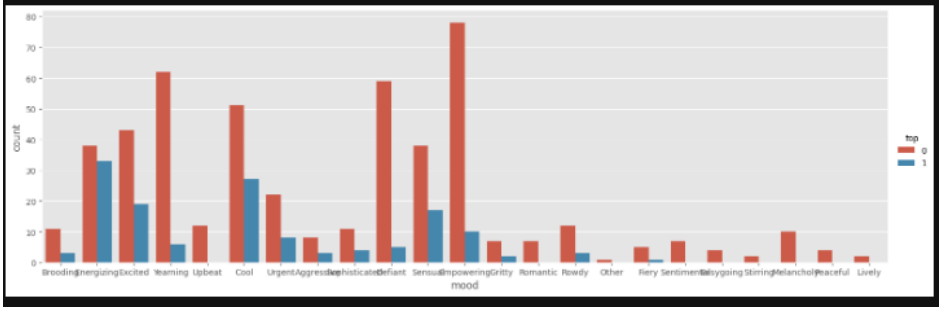

```
sb.catplot(x='genre', data=artists_billboard, aspect =3, kind="count")
```

```
<seaborn.axisgrid.FacetGrid at 0x1b5b5678860>
```



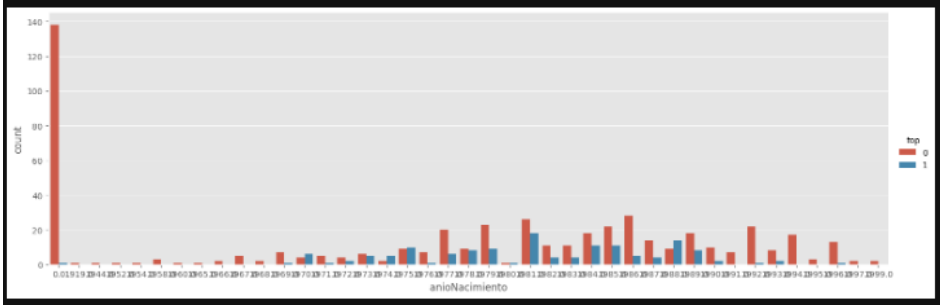

```
sb.catplot(x='mood', data=artists_billboard, hue="top", aspect=3, kind="count")
```

```
<seaborn.axisgrid.FacetGrid at 0x1b5b5540da0>
```



```
sb.catplot(x='anioNacimiento', data=artists_billboard, hue="top", aspect=3, kind="
```

```
<seaborn.axisgrid.FacetGrid at 0x1b5b550fce0>
```




```
nacimientosPorAnio = artists_billboard['anioNacimiento']
len(nacimientosPorAnio[nacimientosPorAnio<=0])
```

```
139
```

Notamos que tenemos 139 registros de canciones de las que deconocemos el año de nacimineto. Deberemos tratar estos datos para poder utilizar el arbol

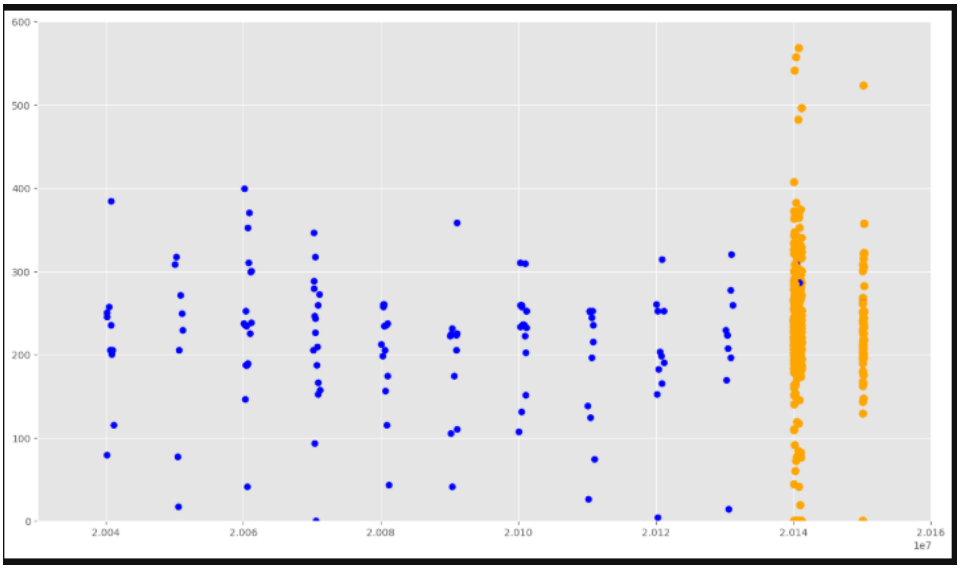
2.2 Análisis de los Datos

Ya que conocemos la distribución de los datos en las columnas del dataset, analizamos qué columnas tienen algún tipo de relación entre sí. En la gráfica anterior, los puntos naranjas representan las canciones no-top y los puntos azules representan las canciones top. El hecho de que todas las canciones antes de 2014 sean top es un problema con la selección de datos en el dataset, ya que estos registros fueron agregados después para balancear la cantidad de canciones top y no-top. Los datos de 2014 y 2015 muestran una distribución más realista, ya que la gran mayoría de las canciones nunca alcanzan la posición #1.

```
f1 = artists_billboard['chart_date'].values
f2 = artists_billboard['durationSeg'].values

asignar=[]
asignar2=[]
for index, row in artists_billboard.iterrows():
    asignar.append(colores[row['top']])
    asignar2.append(tamanios[row['top']])

plt.scatter(f1, f2, c=asignar, s=asignar2)
plt.axis([20030101,20160101,0,600])
plt.show()
```



Una métrica más útil que el año de nacimiento del artista en el análisis de los factores que determinan si una canción será #1 es la edad del artista cuando se publicó la canción. Antes de calcular este valor, es necesario limpiar los datos del año de nacimiento, asignando edades aleatorias dentro de una desviación estándar del promedio de edad de los artistas. Para esto, primero calculamos la edad promedio de los artistas con fecha de nacimiento no nula.

```
def edad_fix(anio):
    if anio==0:
        return None
    return anio

artists_billboard['anioNacimiento']=artists_billboard.apply(lambda x: edad_fix(x['
```

```
def calcula_edad(anio,cuando):
    cad = str(cuando)
    momento = cad[:4]
    if anio==0.0:
        return None
    return int(momento) - anio

artists_billboard['edad_en_billboard']=artists_billboard.apply(lambda x: calcula_ed
```

Calculamos promedio de edad y asignamos a los registros Nulos ¶

```
age_avg = artists_billboard['edad_en_billboard'].mean()
age_std = artists_billboard['edad_en_billboard'].std()
age_null_count = artists_billboard['edad_en_billboard'].isnull().sum()
age_null_random_list = np.random.randint(age_avg - age_std, age_avg + age_std, size=age_null_count)

conValoresNulos = np.isnan(artists_billboard['edad_en_billboard'])

artists_billboard.loc[np.isnan(artists_billboard['edad_en_billboard']), 'edad_en_billboard'] = age_null_random_list
artists_billboard['edad_en_billboard'] = artists_billboard['edad_en_billboard'].astype(int)
print("Edad Promedio: " + str(age_avg))
print("Desvió Std Edad: " + str(age_std))
print("Intervalo para asignar edad aleatoria: " + str(int(age_avg - age_std)) + " a " + str(int(age_avg + age_std)))
```

```
Edad Promedio: 30.10282258064516
Desvió Std Edad: 8.40078832861513
Intervalo para asignar edad aleatoria: 21 a 38
```

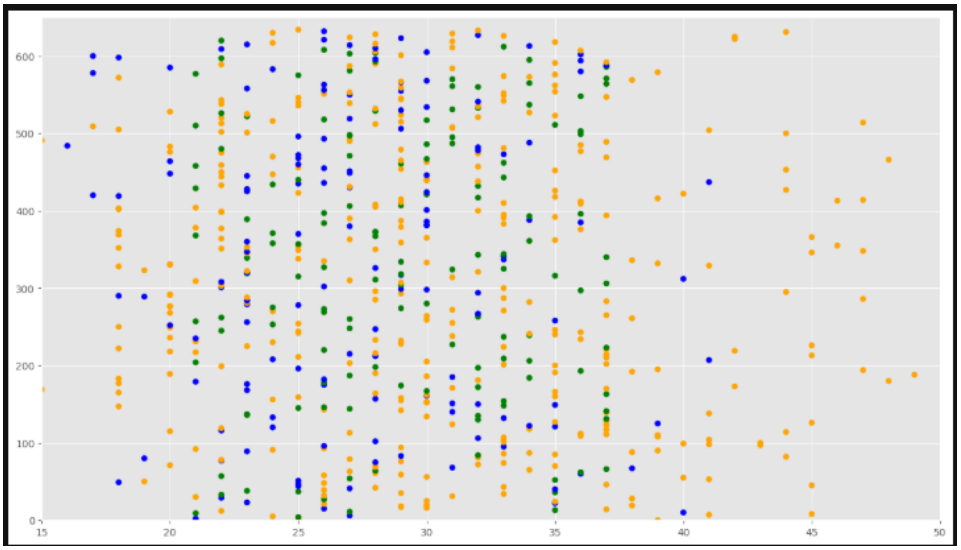
Visualizamod las edades que agregamos

```
f1 = artists_billboard['edad_en_billboard'].values
f2 = artists_billboard.index

colores = ['orange','blue','green']

asignar=[]
for index, row in artists_billboard.iterrows():
    if (conValoresNulos[index]):
        asignar.append(colores[2]) # verde
    else:
        asignar.append(colores[row['top']])

plt.scatter(f1, f2, c=asignar, s=30)
plt.axis([15,50,0,650])
plt.show()
```



2.3 Mapeo de Atributos

Se realiza un mapeo de los atributos de entrada para transformarlos en categorías utilizables en el árbol de decisión.

Mapeo de atributos

Realizaremos un mapeo de los atributos de entrada para poder transformarlos a categorias que podamos utilzar en nuestro árbol de decisión

```
separador = "### ### ###"
grouped11 = artists_billboard.groupby('mood').size()#.sum().reset_index()
neworder11 = grouped11.sort_values(ascending=False)
print(neworder11)
print(separador)
print("Tempos de Canción: " + str(artists_billboard['tempo'].unique()))
print(separador)
print("Tipos de Artista: " + str(artists_billboard['artist_type'].unique()))
print(separador)
grouped11 = artists_billboard.groupby('genre').size()#.sum().reset_index()
neworder11 = grouped11.sort_values(ascending=False)
print(neworder11)
```

```
# Mood Mapping
artists_billboard['moodEncoded'] = artists_billboard['mood'].map( {'Energizing': 6,
                                    'Empowering': 6,
                                    'Cool': 5,
                                    'Yearning': 4, # anhelo, deseo, ansia
                                    'Excited': 5, #emocionado
                                    'Defiant': 3,
                                    'Sensual': 2,
                                    'Gritty': 3, #coraje
                                    'Sophisticated': 4,
                                    'Aggressive': 4, # provocativo
                                    'Fiery': 4, #caracter fuerte
                                    'Urgent': 3,
                                    'Rowdy': 4, #ruidoso alboroto
                                    'Sentimental': 4,
                                    'Easygoing': 1, # sencillo
                                    'Melancholy': 4,
                                    'Romantic': 2,
                                    'Peaceful': 1,
                                    'Brooding': 4, # melancolico
                                    'Upbeat': 5, #optimista alegre
                                    'Stirring': 5, #emocionante
                                    'Lively': 5, #animado
                                    'Other': 0,'':0} ).astype(int)
```

```
# Tempo Mapping
artists_billboard['tempoEncoded'] = artists_billboard['tempo'].map( {'Fast Tempo': 0, 'Medium Tempo': 2, 'Slow Tempo': 1, '': 0} ).astype(int)
# Genre Mapping
artists_billboard['genreEncoded'] = artists_billboard['genre'].map( {'Urban': 4,
                                        'Pop': 3,
                                        'Traditional': 2,
                                        'Alternative & Punk': 1,
                                       'Electronica': 1,
                                        'Rock': 1,
                                        'Soundtrack': 0,
                                        'Jazz': 0,
                                        'Other':0,'':0}
                                      ).astype(int)
# artist_type Mapping
artists_billboard['artist_typeEncoded'] = artists_billboard['artist_type'].map( {'Female': 2, 'Male': 3, 'Mixed': 1, '': 0} ).astype(int)
```

```
# Mapping edad en la que llegaron al billboard
artists_billboard.loc[ artists_billboard['edad_en_billboard'] <= 21, 'edadEncoded']                          = 0
artists_billboard.loc[(artists_billboard['edad_en_billboard'] > 21) & (artists_billboard['edad_en_billboard'] <= 26), 'edadEncoded'] = 1
artists_billboard.loc[(artists_billboard['edad_en_billboard'] > 26) & (artists_billboard['edad_en_billboard'] <= 30), 'edadEncoded'] = 2
artists_billboard.loc[(artists_billboard['edad_en_billboard'] > 30) & (artists_billboard['edad_en_billboard'] <= 40), 'edadEncoded'] = 3
artists_billboard.loc[ artists_billboard['edad_en_billboard'] > 40, 'edadEncoded'] = 4

# Mapping Song Duration
artists_billboard.loc[ artists_billboard['durationSeg'] <= 150, 'durationEncoded']                          = 0
artists_billboard.loc[(artists_billboard['durationSeg'] > 150) & (artists_billboard['durationSeg'] <= 180), 'durationEncoded'] = 1
artists_billboard.loc[(artists_billboard['durationSeg'] > 180) & (artists_billboard['durationSeg'] <= 210), 'durationEncoded'] = 2
artists_billboard.loc[(artists_billboard['durationSeg'] > 210) & (artists_billboard['durationSeg'] <= 240), 'durationEncoded'] = 3
artists_billboard.loc[(artists_billboard['durationSeg'] > 240) & (artists_billboard['durationSeg'] <= 270), 'durationEncoded'] = 4
artists_billboard.loc[(artists_billboard['durationSeg'] > 270) & (artists_billboard['durationSeg'] <= 300), 'durationEncoded'] = 5
artists_billboard.loc[ artists_billboard['durationSeg'] > 300, 'durationEncoded'] = 6
```

```
drop_elements = ['id','title','artist','mood','tempo','genre','artist_type','chart_date','anioNacimiento','durationSeg','edad_en_billboard']
artists_encoded = artists_billboard.drop(drop_elements, axis = 1)
```

2.4 Creación del Árbol de Decisión

En esta parte se crea el árbol de decisión. Para optimizar su precisión, analizamos el nivel de profundidad óptimo. Se crean varios modelos de árboles con diferentes profundidades y se calcula su precisión:

Buscamos la profundidad adecuada para arbol de decisiones

```
cv = KFold(n_splits=10) # Numero deseado de "folds" que haremos
accuracies = list()
max_attributes = len(list(artists_encoded))
depth_range = range(1, max_attributes + 1)

# Testearemos la profundidad de 1 a cantidad de atributos +1
for depth in depth_range:
    fold_accuracy = []
    tree_model = tree.DecisionTreeClassifier(criterion='entropy',
                                             min_samples_split=20,
                                             min_samples_leaf=5,
                                             max_depth = depth,
                                             class_weight={1:3.5})
    for train_fold, valid_fold in cv.split(artists_encoded):
        f_train = artists_encoded.loc[train_fold]
        f_valid = artists_encoded.loc[valid_fold]

        model = tree_model.fit(X = f_train.drop(['top'], axis=1),
                               y = f_train["top"])
        valid_acc = model.score(X = f_valid.drop(['top'], axis=1),
                                y = f_valid["top"]) # calculamos la precision con el segmento de validacion
        fold_accuracy.append(valid_acc)

    avg = sum(fold_accuracy)/len(fold_accuracy)
    accuracies.append(avg)
```

```
# Mostramos los resultados obtenidos
df = pd.DataFrame({"Max Depth": depth_range, "Average Accuracy": accuracies})
df = df[["Max Depth", "Average Accuracy"]]
print(df.to_string(index=False))
```

```
 Max Depth  Average Accuracy
         1          0.556101
         2          0.556126
         3          0.568725
         4          0.641022
         5          0.620585
         6          0.617336
         7          0.621999
```

Creanos el arbol de decisiones

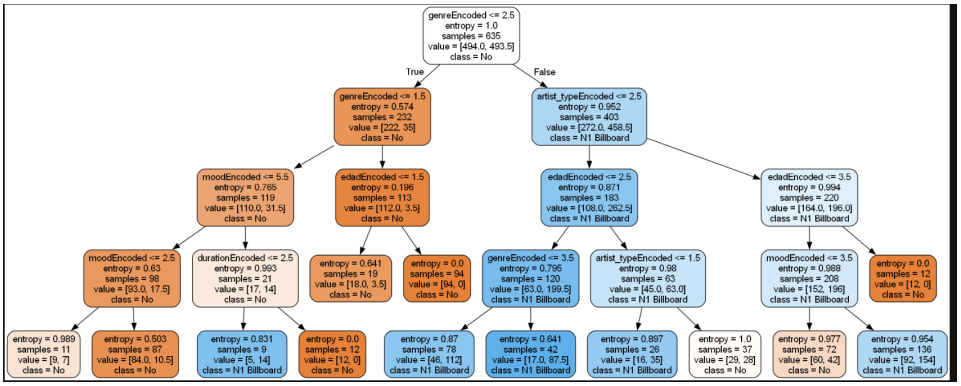
```
# Crear arrays de entrenamiento y las etiquetas que indican si llegó a top o no
y_train = artists_encoded['top']
x_train = artists_encoded.drop(['top'], axis=1).values

# Crear Arbol de decision con profundidad = 4
decision_tree = tree.DecisionTreeClassifier(criterion='entropy',
                                            min_samples_split=20,
                                            min_samples_leaf=5,
                                            max_depth = 4,
                                            class_weight={1:3.5})
decision_tree.fit(x_train, y_train)

# exportar el modelo a archivo .dot
with open(r"tree1.dot", 'w') as f:
     f = tree.export_graphviz(decision_tree,
                              out_file=f,
                              max_depth = 7,
                              impurity = True,
                              feature_names = artists_encoded.drop(['top'], axis=1).columns.tolist(),
                              class_names = ['No', 'N1 Billboard'],
                              rounded = True,
                              filled= True )

# Convertir el archivo .dot a png para poder visualizarlo
check_call(['dot','-Tpng',r'tree1.dot','-o',r'tree1.png'])
PImage("tree1.png")
```

No siempre tener más niveles en un árbol de decisión es mejor. En este caso, la profundidad óptima es de 4 niveles, con una precisión del 64.41%. Se construye el árbol de decisión con esta profundidad y se usa la aplicación GraphViz para exportar un diagrama del árbol y convertirlo en una imagen.



Precision del arbol

```
acc_decision_tree = round(decision_tree.score(x_train, y_train) * 100, 2)
print(acc_decision_tree)
```

```
68.19
```

El modelo tiene una precisión del 64.88

3 Resultados

Para verificar la precisión del modelo, se realizan predicciones de rankings de canciones en el dataset.

Predicción del árbol de decisión

```
#predecir artista CAMILA CABELLO featuring YOUNG THUG
# con su canción Havana llego a numero 1 Billboard US en 2017

x_test = pd.DataFrame(columns=('top','moodEncoded', 'tempoEncoded', 'genreEncoded','artist_typeEncoded','edadEncoded','durationEncoded'))
x_test.loc[0] = (1,5,2,4,1,0,3)
y_pred = decision_tree.predict(x_test.drop(['top'], axis = 1))
print("Prediccion: " + str(y_pred))
y_proba = decision_tree.predict_proba(x_test.drop(['top'], axis = 1))
print("Probabilidad de Acierto: " + str(round(y_proba[0][y_pred][0]* 100, 2))+"%")
```

```
Prediccion: [1]
Probabilidad de Acierto: 83.73%
```

```python
#predecir artista Imagine Dragons
# con su canción Believer llego al puesto 42 Billboard US en 2017

x_test = pd.DataFrame(columns=('top','moodEncoded', 'tempoEncoded', 'genreEncoded','artist_typeEncoded','edadEncoded','durationEncoded'))
x_test.loc[0] = (0,4,2,1,3,2,3)
y_pred = decision_tree.predict(x_test.drop(['top'], axis = 1))
print("Prediccion: " + str(y_pred))
y_proba = decision_tree.predict_proba(x_test.drop(['top'], axis = 1))
print("Probabilidad de Acierto: " + str(round(y_proba[0][y_pred][0]* 100, 2))+"%")
```

```
Prediccion: [0]
Probabilidad de Acierto: 88.89%
```

4 Conclusión

Los modelos de árboles de decisión son herramientas de predicción utilizadas en el aprendizaje automático, basadas en la estructura lógica del mismo nombre. Cada nodo representa un evento o decisión con múltiples ramas que pueden seguirse dependiendo de los datos ingresados al modelo.

En este caso, se creó un modelo de árbol de decisión para determinar si una canción estuvo en la posición #1 de las listas de Billboard, dado un conjunto de variables del dataset. El modelo devuelve una probabilidad de que la canción haya alcanzado esta posición. Se probaron varios modelos con distintas profundidades para optimizar su precisión. El modelo final obtuvo una precisión del 64.88%, lo que significa que por cada predicción incorrecta, realiza aproximadamente dos predicciones correctas.