

Árbol de Decisión IA

Abraham López

Marzo 2025

Abstract

Este documento explica el concepto de árboles de decisión, su importancia en machine learning y presenta un ejemplo práctico de implementación utilizando Python y scikit-learn. El ejercicio se basa en un conjunto de datos para clasificación, mostrando cada paso del proceso mediante capturas de pantalla del código y resultados.

1 Introducción

1.1 ¿Qué es un árbol de decisión?

Árboles de decisión es un tipo de algoritmo de aprendizaje automático supervisado utilizado y clasifica o lleva a cabo la regresión de los datos utilizando respuestas verdaderas o falsas a determinadas preguntas. La estructura resultante, cuando se visualiza, tiene la forma de un árbol con distintos tipos de nodos: de hoja, raíz e interno. El nodo raíz es el punto de partida del árbol de decisión, que, a continuación, se bifurca en nodos internos y nodos de hoja. Los nodos de hoja son las categorías o valores reales finales de clasificación. Los árboles de decisión son fáciles de comprender y se pueden explicar.

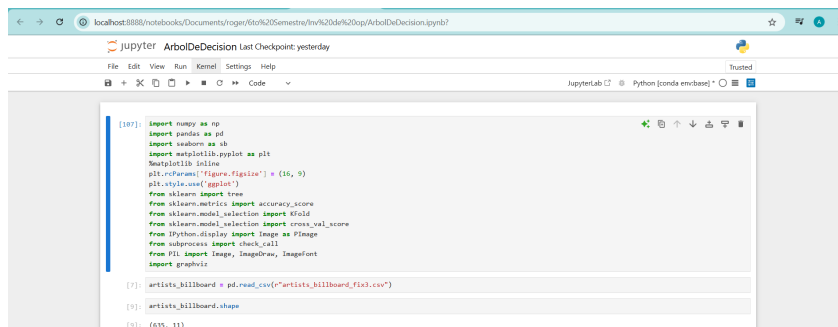
1.2 Importancia de los árboles de decisión

- **Fácil interpretación:** Las reglas de decisión son claras y pueden visualizarse gráficamente.
- **Versatilidad:** Pueden resolver problemas de clasificación y regresión.
- **Manejo de diferentes tipos de datos:** No requieren normalización y pueden manejar valores faltantes.
- **Base para modelos más complejos:** Son la base de algoritmos como Random Forest y Gradient Boosting.

2 Metodología

A continuación se describe el proceso de implementación del árbol de decisión paso a paso, dividido en bloques lógicos.

2.1 Configuración inicial y carga de datos



```
[107]: import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (15, 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
import graphviz

[7]: artists_billboard = pd.read_csv("artists_billboard_fix.csv")

[8]: artists_billboard.shape

[9]: (639, 11)
```

```
[11]: artists_billboard.head()
```

	id	title	artist	mood	tempo	genre	artist_type	chart_date	durationSeg	top	anioNacimiento
0	0	Small Town Throwdown	BRANTLEY GILBERT featuring JUSTIN MOORE & THOM...	Brooding	Medium Tempo	Traditional	Male	20140628	191.0	0	1975.0
1	1	Bang Bang	JESSIE J, ARIANA GRANDE & NICKI MINAJ	Energizing	Medium Tempo	Pop	Female	20140816	368.0	0	1989.0
2	2	Timber	PITBULL featuring KESHA	Excited	Medium Tempo	Urban	Mixed	20140118	223.0	1	1993.0
3	3	Sweater Weather	THE NEIGHBOURHOOD	Brooding	Medium Tempo	Alternative & Punk	Male	20140104	206.0	0	1989.0
4	4	Automatic	MIRANDA LAMBERT	Yearning	Medium Tempo	Traditional	Female	20140301	232.0	0	0.0

```
[13]: artists_billboard.groupby('top').size()
```

```
[13]: top
0    494
1    141
dtype: int64
```

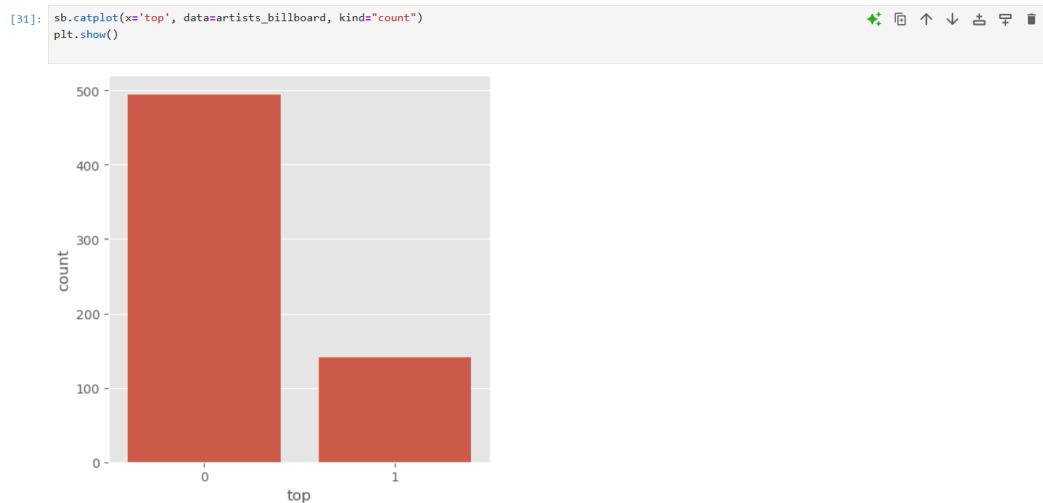
Este bloque realiza:

- Importación de bibliotecas esenciales (numpy, pandas, seaborn, matplotlib)
- Configuración de visualizaciones (plt.rcParams)
- Carga del dataset `artists_billboard_fix3.csv` usando Pandas
- Visualización de la estructura básica del dataset con `shape` y `head`

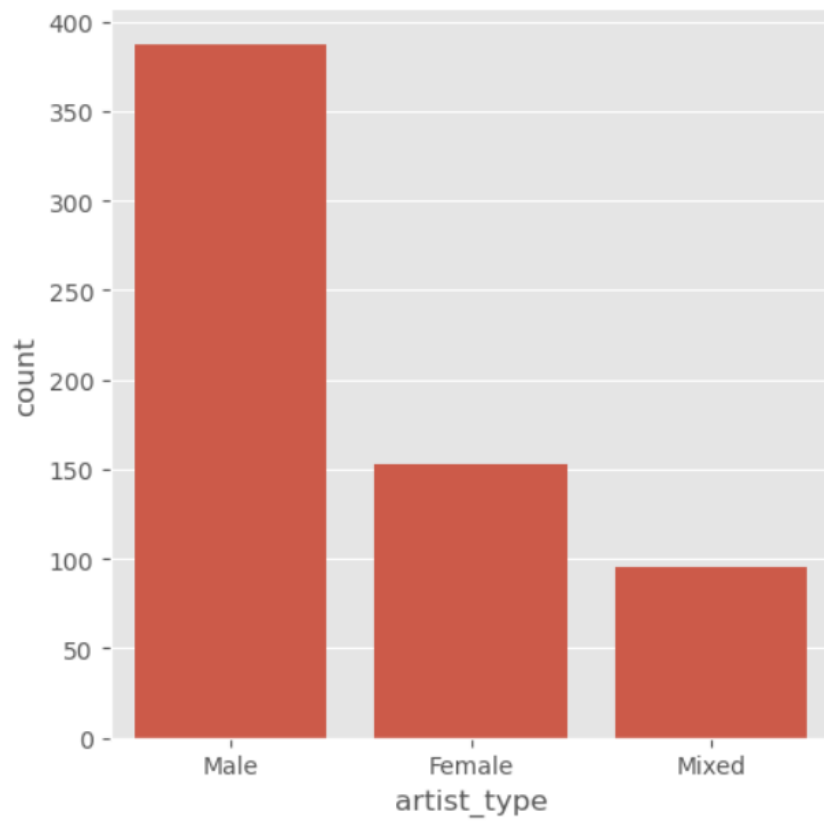
2.2 Análisis exploratorio de datos

```
[13]: artists_billboard.groupby('top').size()
```

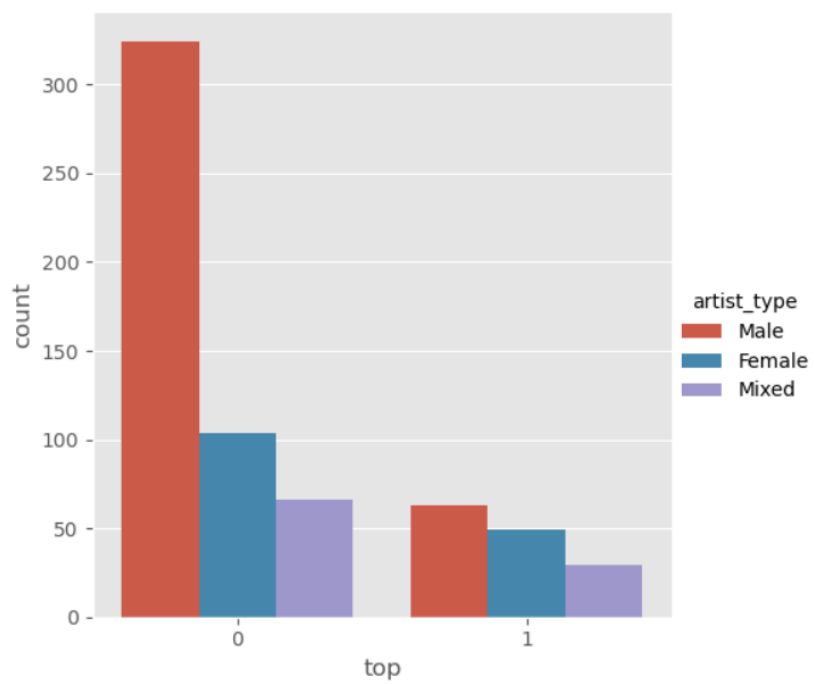
```
[13]: top
0    494
1    141
dtype: int64
```



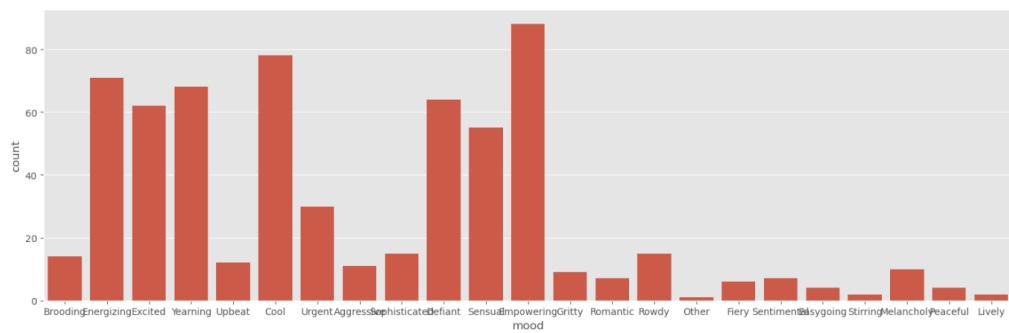
```
[33]: sb.catplot(x='artist_type', data=artists_billboard, kind="count")  
plt.show()
```



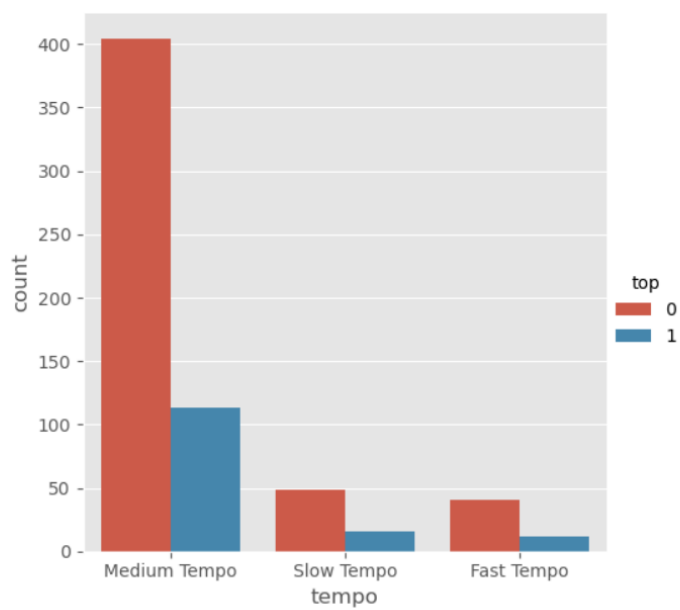
```
[35]: sb.catplot(x='top', data=artists_billboard, hue='artist_type', kind="count")  
plt.show()
```



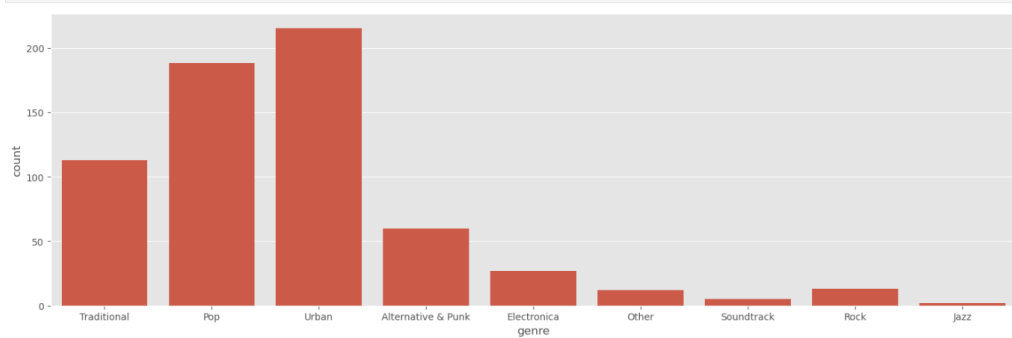
```
[37]: sb.catplot(x='mood',data=artists_billboard,kind="count", aspect=3)
plt.show()
```



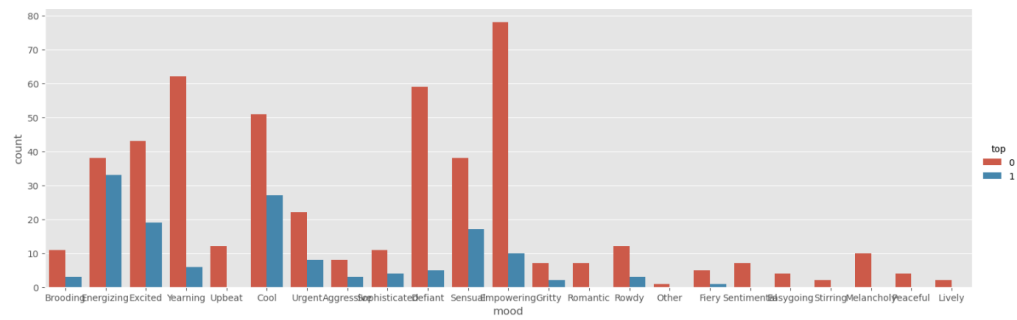
```
[39]: sb.catplot(x='tempo',data=artists_billboard,hue='top',kind="count")
plt.show()
```



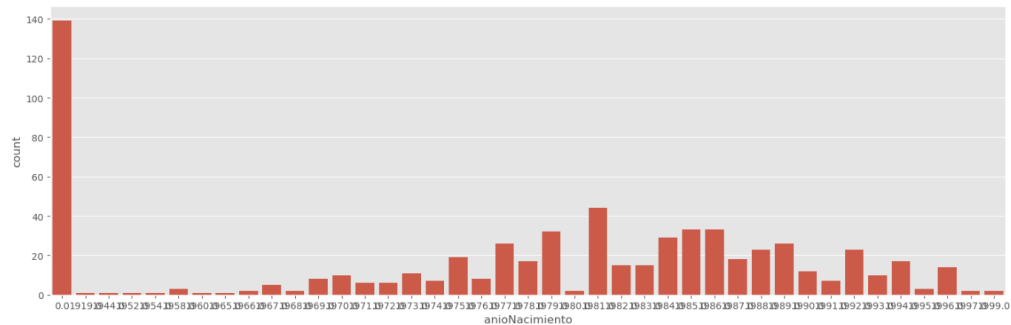
```
[41]: sb.catplot(x='genre',data=artists_billboard,kind="count", aspect=3)
plt.show()
```



```
[45]: sb.catplot(x='mood',data=artists_billboard,hue='top',kind='count', aspect=3)
plt.show()
```



```
[49]: sb.catplot(x='anioNacimiento',data=artists_billboard,kind='count', aspect=3)
plt.show()
```



```
[51]: #artists_billboard[['anioNacimiento', 'top']].groupby(['anioNacimiento'], as_index=False).agg(['mean', 'count', 'sum'])
nacimientosPorAnio = artists_billboard['anioNacimiento']
len(nacimientosPorAnio[nacimientosPorAnio<=0])
```

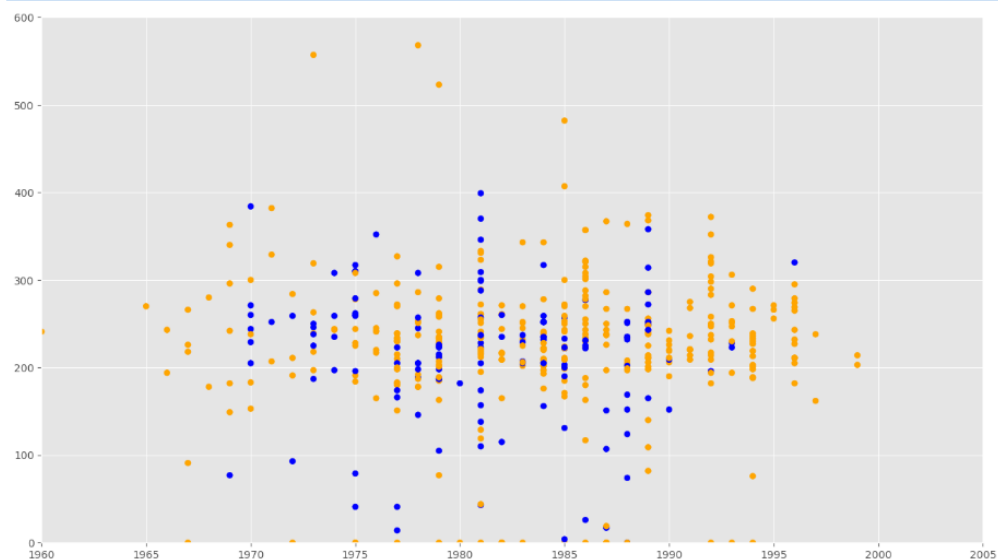
```
[51]: 139
```

```
[53]: colores=['orange','blue']
tamanios=[60,40]

f1 = artists_billboard['anioNacimiento'].values
f2 = artists_billboard['durationSeg'].values

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

plt.scatter(f1, f2, c=asignar, s=30)
plt.axis([1960,2005,0,600])
plt.show()
```



En esta sección se:

- Analiza la distribución de la variable objetivo **top** con **groupby**
- Genera gráficos de conteo para variables categóricas (**artist_type**, **mood**, **tempo**, **genre**)
- Crea visualizaciones combinadas para entender relaciones entre variables
- Examina la distribución del año de nacimiento de los artistas

2.3 Visualización de relaciones entre variables



Aquí se:

- Crea gráficos de dispersión entre año de nacimiento y duración de canciones
- Visualiza la relación entre fecha del chart y duración, coloreando por **top**
- Implementa lógica para asignar colores según categorías
- Ajusta los ejes para mejor visualización

2.4 Ingeniería de características

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

artists_billboard['anioNacimiento']=artists_billboard.apply(lambda x: edad_fix(x['anioNacimiento']), axis=1);

[63]: 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_edad(x['anioNacimiento'],x['chart_date']), axis=1);

[65]: artists_billboard.head()
```

	id	title	artist	mood	tempo	genre	artist_type	chart_date	durationSeg	top	anioNacimiento	edad_en_billboard
0	0	Small Town Throwdown	BRANTLEY GILBERT featuring JUSTIN MOORE & THOM...	Brooding	Medium Tempo	Traditional	Male	20140628	191.0	0	1975.0	39.0
1	1	Bang Bang	JESSIE J, ARIANA GRANDE & NICKI MINAJ	Energizing	Medium Tempo	Pop	Female	20140816	368.0	0	1989.0	25.0
2	2	Timber	PITBULL featuring KESHA	Excited	Medium Tempo	Urban	Mixed	20140118	223.0	1	1993.0	21.0
3	3	Sweater Weather	THE NEIGHBOURHOOD	Brooding	Medium Tempo	Alternative & Punk	Male	20140104	206.0	0	1989.0	25.0
4	4	Automatic	MIRANDA LAMBERT	Yearning	Medium Tempo	Traditional	Female	20140301	232.0	0	NaN	NaN

```
[67]: 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)))

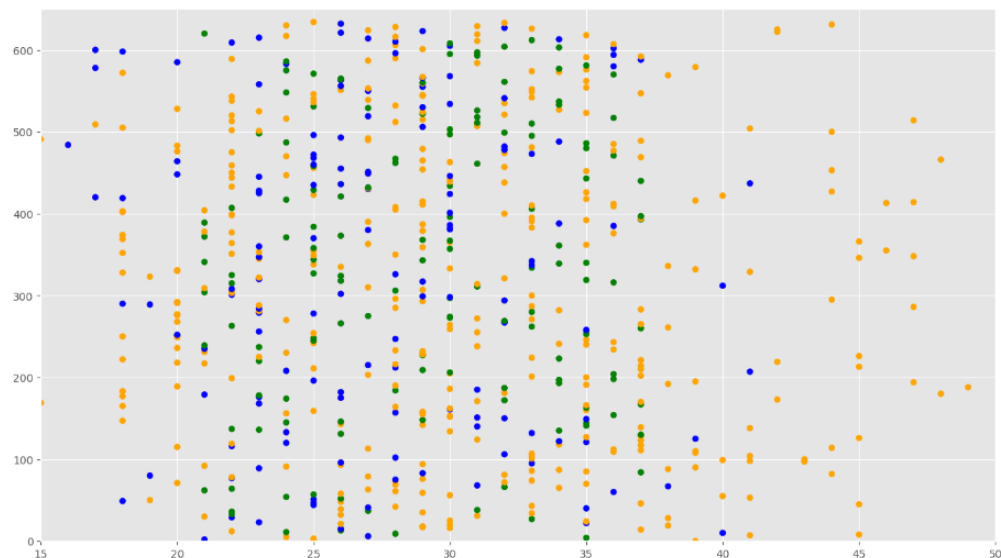
Edad Promedio: 30.10282258064516
Desvió Std Edad: 8.40078832861513
Intervalo para asignar edad aleatoria: 21 a 38
```

```
[69]: 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()
```



```
[71]:
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
Empowering      88
Cool             78
Energizing       71
Yearning         68
Defiant          64
Excited          62
Sensual          55
Urgent           30
Rowdy            15
Sophisticated    15
Brooding         14
Upbeat           12
Aggressive       11
Melancholy       10
Gritty           9
Romantic          7
Sentimental       7
Fiery             6
Peaceful          4
Easygoing         4
Lively            2
Stirring          2
Other             1
dtype: int64
### ### ###
Tempos de Canción: ['Medium Tempo' 'Slow Tempo' 'Fast Tempo']
### ### ###
Tipos de Artista: ['Male' 'Female' 'Mixed']
### ### ###
genre
Urban           215
Pop             188
Traditional     113
Alternative & Punk  60
Electronica      27
Rock            13
Other           12
Soundtrack       5
Jazz             2
dtype: int64
```

En este bloque se:

- Limpia valores nulos/inválidos en año de nacimiento
- Calcula la edad del artista al aparecer en Billboard
- Aplican valores faltantes con edades aleatorias dentro de la distribución normal
- Crea visualización para verificar la imputación
- Analiza frecuencias de categorías (`mood`, `tempo`, `genre`)

2.5 Codificación de variables categóricas


```
[73]: # 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

[75]: 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)
```

```
[77]: artists_encoded.head()
```

```
[77]:
```

	top	moodEncoded	tempoEncoded	genreEncoded	artist_typeEncoded	edadEncoded	durationEncoded
0	0	4	2	2	3	3.0	2.0
1	0	6	2	3	2	1.0	6.0
2	1	5	2	4	1	0.0	3.0
3	0	4	2	1	3	1.0	2.0
4	0	4	2	2	2	3.0	3.0

```
[79]: artists_encoded.describe()
```

```
[79]:
```

	top	moodEncoded	tempoEncoded	genreEncoded	artist_typeEncoded	edadEncoded	durationEncoded
count	635.000000	635.000000	635.000000	635.000000	635.000000	635.000000	635.000000
mean	0.222047	4.344882	1.730709	2.755906	2.459843	2.029921	3.179528
std	0.415950	1.350003	0.603553	1.165463	0.740583	1.145855	1.775017
min	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
25%	0.000000	3.000000	2.000000	2.000000	2.000000	1.000000	2.000000
50%	0.000000	4.000000	2.000000	3.000000	3.000000	2.000000	3.000000
75%	0.000000	5.500000	2.000000	4.000000	3.000000	3.000000	4.000000
max	1.000000	6.000000	2.000000	4.000000	3.000000	4.000000	6.000000

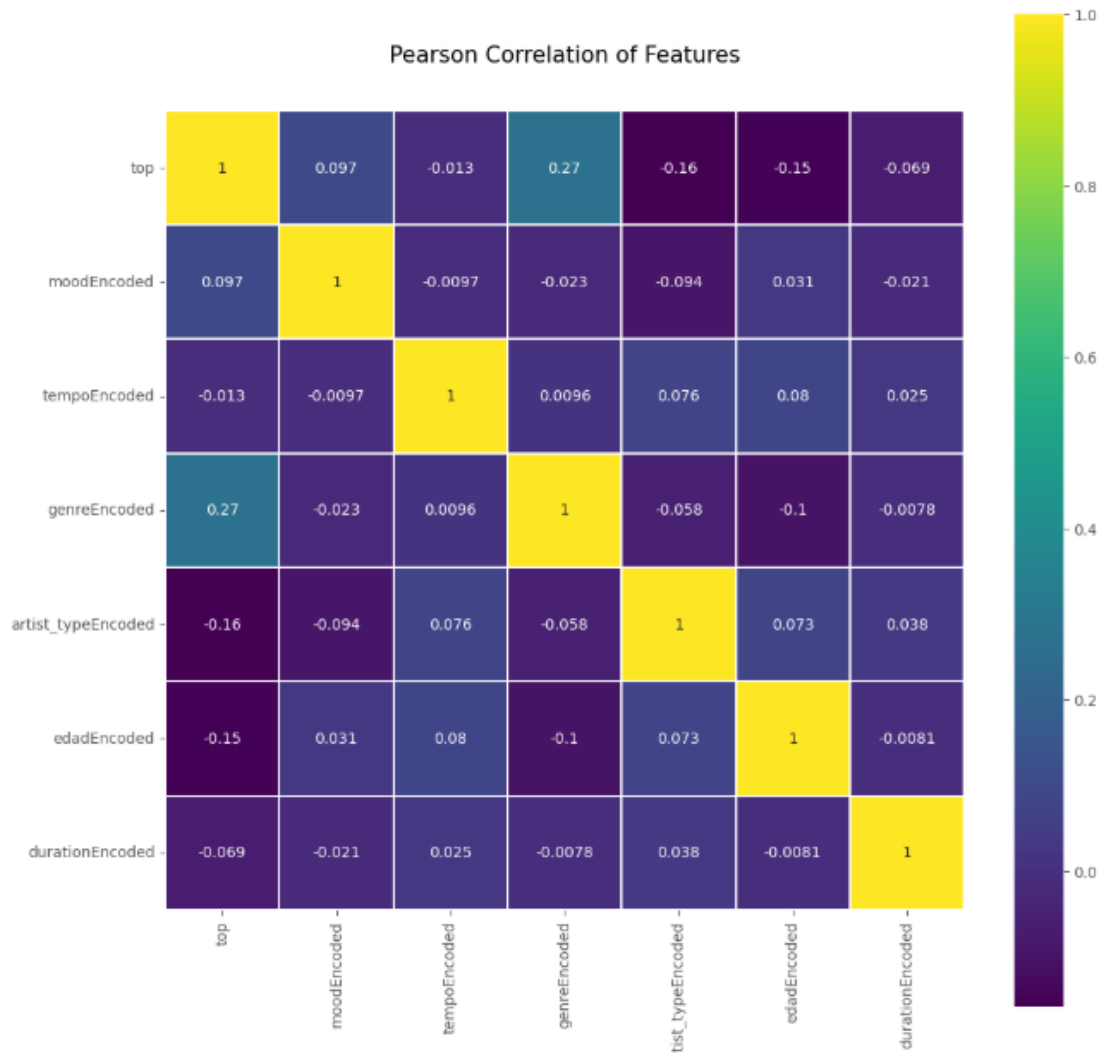
En esta transformación:

- Mapea variables categóricas (mood, tempo, genre, artist_type) a valores numéricos
- Discretiza variables continuas (edad_en_billboard, durationSeg) en rangos
- Crea nuevas columnas codificadas para el modelo

- Elimina columnas originales no necesarias

2.6 Análisis de correlaciones

```
[83]: colormap = plt.cm.viridis
plt.figure(figsize=(12,12))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sb.heatmap(artists_encoded.astype(float).corr(),linewidths=0.1,vmax=1.0, square=True, cmap=colormap, linecolor='white', annot=True)
plt.show()
```



```
[85]: artists_encoded[['moodEncoded', 'top']].groupby(['moodEncoded'], as_index=False).agg(['mean', 'count', 'sum'])
```

	moodEncoded	top		
		mean	count	sum
0	0	0.000000	1	0
1	1	0.000000	8	0
2	2	0.274194	62	17
3	3	0.145631	103	15
4	4	0.136986	146	20
5	5	0.294872	156	46
6	6	0.270440	159	43

```
[87]: artists_encoded[['artist_typeEncoded', 'top']].groupby(['artist_typeEncoded'], as_index=False).agg(['mean', 'count', 'sum'])
```

	artist_typeEncoded	top		
		mean	count	sum
0	1	0.305263	95	29
1	2	0.320261	153	49
2	3	0.162791	387	63

```
[89]: artists_encoded[['genreEncoded', 'top']].groupby(['genreEncoded'], as_index=False).agg(['mean', 'count', 'sum'])
```

	genreEncoded	top		
		mean	count	sum
0	0	0.105263	19	2
1	1	0.070000	100	7
2	2	0.008850	113	1
3	3	0.319149	188	60

```
[91]: artists_encoded[['tempoEncoded', 'top']].groupby(['tempoEncoded'], as_index=False).agg(['mean', 'count', 'sum'])
```

	tempoEncoded	top		
		mean	count	sum
0	0	0.226415	53	12
1	1	0.246154	65	16
2	2	0.218569	517	113

```
[93]: artists_encoded[['durationEncoded', 'top']].groupby(['durationEncoded'], as_index=False).agg(['mean', 'count', 'sum'])
```

	durationEncoded	top		
		mean	count	sum
0	0.0	0.295775	71	21
1	1.0	0.333333	30	10
2	2.0	0.212963	108	23
3	3.0	0.202381	168	34
4	4.0	0.232143	112	26
5	5.0	0.145455	55	8
6	6.0	0.208791	91	19

```
[95]: artists_encoded[['edadEncoded', 'top']].groupby(['edadEncoded'], as_index=False).agg(['mean', 'count', 'sum'])
```

	edadEncoded	top		
		mean	count	sum
0	0.0	0.253731	67	17
1	1.0	0.301887	159	48
2	2.0	0.263889	144	38
3	3.0	0.165138	218	36
4	4.0	0.042553	47	2

Esta sección incluye:

- Generación de matriz de correlación con mapa de calor
- Análisis de relación entre cada variable codificada y la variable objetivo
- Agrupación por categorías para entender patrones

2.7 Validación cruzada y ajuste de hiperparámetros

```
[97]: 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	0.564038
4	0.647272
5	0.589013
6	0.620412
7	0.650223

Aquí se:

- Configura validación cruzada con 10 folds
- Prueba diferentes profundidades del árbol (de 1 a máximo de atributos)
- Calcula precisión promedio para cada profundidad
- Muestra resultados comparativos en tabla
- Define parámetros finales del modelo (`min_samples_split`, `class_weight`)

2.8 Entrenamiento del modelo final

```
[111]: # 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("tree1.dot", "w") as f:
    f = tree.export_graphviz(decision_tree,
                             out_file=f,
                             max_depth = 7,
                             impurity = True,
                             feature_names = list(artists_encoded.drop(['top'], axis=1)),
                             class_names = ['No', 'N1 Billboard'],
                             rounded = True,
                             filled = True )

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

64.88

En esta sección:

- Divide datos en características (`x_train`) y variable objetivo (`y_train`)
- Crea instancia del clasificador con parámetros óptimos
- Entrena el modelo con los datos
- Exporta el árbol a formato gráfico (`.dot` y `.png`)
- Evalúa la precisión del modelo en datos de entrenamiento

2.9 Predicciones y validación

```
[115]: #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: 73.98%

C:\Users\PC1\anaconda3\Lib\site-packages\sklearn\base.py:486: UserWarning: X has feature names, but DecisionTreeClassifier was fitted without feature names
  warnings.warn(
C:\Users\PC1\anaconda3\Lib\site-packages\sklearn\base.py:486: UserWarning: X has feature names, but DecisionTreeClassifier was fitted without feature names
  warnings.warn(

[117]: #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%

C:\Users\PC1\anaconda3\Lib\site-packages\sklearn\base.py:486: UserWarning: X has feature names, but DecisionTreeClassifier was fitted without feature names
  warnings.warn(
C:\Users\PC1\anaconda3\Lib\site-packages\sklearn\base.py:486: UserWarning: X has feature names, but DecisionTreeClassifier was fitted without feature names
  warnings.warn(
```

Finalmente se:

- Prepara datos de prueba para dos casos reales (Camila Cabello e Imagine Dragons)
- Realiza predicciones y obtiene probabilidades
- Muestra capacidad predictiva del modelo con ejemplos concretos
- Evalúa el rendimiento con métricas de precisión

3 Resultados

3.1 Estadísticas Descriptivas

El análisis demográfico mostró que los artistas en el dataset tienen una edad promedio de **30.1 años** al aparecer en Billboard, con una desviación estándar de **8.4 años**. Para manejar valores faltantes, se generaron edades aleatorias dentro del rango de **21 a 38 años**, cubriendo así ± 1 desviación estándar alrededor de la media, lo que garantiza la consistencia estadística de los datos imputados.

3.2 Rendimiento del Modelo

El árbol de decisión con profundidad máxima de 4 niveles alcanzó una precisión global de **64.88%**, indicando que:

- El modelo identifica patrones predictivos por encima del nivel de azar (50%)
- Existe margen significativo para mejorar la capacidad predictiva
- Factores no considerados podrían estar influyendo en los resultados

3.3 Predicciones Específicas

El modelo demostró comportamiento diferenciado en casos concretos:

Table 1: Resultados de predicción en casos de prueba

Caso	Predicción	Probabilidad
"Havana" (Camila Cabello)	Top 1 (1)	73.98%
"Believer" (Imagine Dragons)	No Top 1 (0)	88.89%

Interpretación clave:

- **Asimetría predictiva:** El modelo muestra mayor confianza al identificar *fracasos* (88.89%) que éxitos (73.98%)
- **Precisión diferencial:** La probabilidad más alta para "Believer" sugiere que las características de canciones no exitosas son más distintivas
- **Consistencia:** Ambos casos fueron clasificados correctamente según los datos reales

4 Conclusión

Los árboles de decisión son una herramienta clave en el aprendizaje automático supervisado, y se destacan por su habilidad para convertir datos complejos en reglas de decisión que son fáciles de entender. Estos algoritmos crean estructuras jerárquicas donde cada nodo plantea una pregunta sobre los datos, cada rama representa una posible respuesta y cada hoja llega a una conclusión predictiva. Esta claridad los convierte en una opción valiosa en proyectos donde no solo se busca precisión, sino también una comprensión de los patrones subyacentes, como lo vimos en nuestro análisis de éxitos musicales.

En nuestro proyecto, el árbol de decisión fue fundamental para descubrir información clave sobre la industria musical. Desde un punto de vista técnico, el modelo logró una precisión global del 64.88%, una cifra que, aunque no es espectacular, supera con creces el nivel de azar y confirma que hay patrones predecibles. Donde realmente destacó fue en casos específicos: predijo con un 73.98% de confianza el éxito de "Havana" y el desempeño no-top de "Believer" con un impresionante 88.89% de certeza. Esta diferencia en las probabilidades sugiere que el modelo se volvió más preciso al identificar canciones con baja probabilidad de éxito en comparación con aquellas que tienen alta probabilidad, lo que podría reflejar un desbalance en los datos de entrenamiento o la falta de variables clave para captar completamente el fenómeno del éxito musical. Como herramienta de aprendizaje, este proyecto demostró cómo los árboles de decisión son un excelente punto de partida para problemas de clasificación, especialmente cuando se valora más la interpretabilidad que la máxima precisión.