# Reto_modelacion_estadisitica

November 30, 2023

## 1 Reto entregable 1


- Guillermo Villegas Morales A01637169
- Adara Luisa Pulido Sánchez A01642450
- Jorge Eduardo Guijarro Márquez A01563113
- Alan Rojas López A01706146
- Gabriel Eduardo Meléndez Zavala A01638293


## 2 Introducción

En este entregable realizamos una fase exploratoria de una base de datos de canciones donde cada entrada es una canción y sus atributos constan del nombre del artista/s, nombre de la canción, tonalidad, popularidad, duración entre otros. La base de datos tiene problemas como datos faltantes, limpiearemos la base de datos para obtener un análisis propio, además de que realizamos diferentes estadísticas descriptivas buscando contestar preguntas sobre la base de datos en cuestión. Algunas de las preguntas que buscamos responder en este trabajo incluyen ¿Qué canciones son las más populares?, ¿Qué caracterísitcas tienen?, ¿Qué rasgos tienen generalmente las canciones de tal género? ¿Qué palabras son las más comunes en el nombre de artistas y canciones? entre otras.

## 3 Objetivos

Un análisis detallado de las características musicales revela sorprendentes patrones y diferencias significativas en la popularidad y géneros. El objetivo es explorar la base de datos de Spotify, y encontar conclusiones sobre el comportamiento de los datos con respecto a la variable popularidad, con la finalidad de comprender los factores que influyen en la preferencia de las canciones por parte de los usuarios.

## 4 Métodos

El análisis se centra en un conjunto de datos que contiene información sobre canciones, con variables como popularidad, género musical, modo, valencia, tempo, duración, entre otras. El objetivo es comprender las relaciones y patrones presentes en estos datos. Se aborda la presencia de datos faltantes mediante imputación simple, utilizando la media, mediana o moda según sea apropiado. Se clasifican las variables entre numéricas y categóricas. La variable “Class” se recodifica para indicar

el género de la canción. Además, se recodifica el tono de las canciones tomando un orden alfabético. Se crea un histograma para visualizar la distribución de la duración de las canciones, sin embargo se encuentra que algunos datos están en minutos y otros en milisegundos, haciendo necesaria una conversión. Se genera una nueva variable que indica si una canción es una colaboración entre artistas, proporcionando información adicional sobre las dinámicas de colaboración en la música.

### 4.0.1 Import Libraries

Empezaremos por incluir en el programa las herramientas que utilizaremos para realizar las operaciones, las gráficas y los análisis. La biblioteca numpy nos da funciones para operar con matrices, Pandas no será util para manipular la base de datos. Matplotlib, Seaborn y Plotly serán necesarias para generar las gráficas. Finalmente el módulo Stats de Scipy ofrece herramientas de cáclulos estádisticos más avanzados.

[149]:
```
import numpy as np # lots of math operations and matrices
import pandas as pd # data structures
import matplotlib.pyplot as plt # plot charts. More on this later
from scipy import stats as st
import seaborn as sns
from wordcloud import WordCloud
import plotly.graph_objects as go

df=pd.read_csv("music.csv")
```

[150]:
```
#from google.colab import drive
#drive.mount('/content/drive')
```

### 4.0.2 Información básica de la base de datos

Primer vistazo a la base de datos

[151]:
```
df.head()
```

[151]:

| | Artist Name | Track Name | Popularity | danceability | energy | key | loudness | mode | speechiness | acousticness | instrumentalness | liveness | valence | tempo | duration_in min/ms |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Bruno Mars | That's What I Like (feat. Gucci Mane) | 60.0 | 0.854 | 0.564 | 1.0 | -4.964 | 1 | 0.0485 | 0.017100 | | | | | |
| 1 | Boston | Hitch a Ride | 54.0 | 0.382 | 0.814 | 3.0 | -7.230 | 1 | 0.0406 | 0.001100 | | | | | |
| 2 | The Raincoats | No Side to Fall In | 35.0 | 0.434 | 0.614 | 6.0 | -8.334 | 1 | 0.0525 | 0.486000 | | | | | |
| 3 | Deno | Lingo (feat. J.I & Chunkz) | 66.0 | 0.853 | 0.597 | 10.0 | -6.528 | 0 | 0.0555 | 0.021200 | | | | | |
| 4 | Red Hot Chili Peppers | Nobody Weird Like Me - Remastered | 53.0 | 0.167 | 0.975 | 2.0 | -4.279 | 1 | 0.2160 | 0.000169 | | | | | |

```
0               NaN    0.0849   0.8990  134.071            234596.0
1          0.004010    0.1010   0.5690  116.454            251733.0
2          0.000196    0.3940   0.7870  147.681            109667.0
3               NaN    0.1220   0.5690  107.033            173968.0
4          0.016100    0.1720   0.0918  199.060            229960.0

   time_signature  Class
0               4      5
1               4     10
2               4      6
3               4      5
4               4     10
```

Buscamos los datos nulos dentro de la base de datos con la función info(), esto nos ofrece adicionalmente otros datos sobre la base de datos como su tamaño y los tipos de datos que manejamos

[152]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17996 entries, 0 to 17995
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Artist Name         17996 non-null  object
 1   Track Name          17996 non-null  object
 2   Popularity          17568 non-null  float64
 3   danceability        17996 non-null  float64
 4   energy              17996 non-null  float64
 5   key                 15982 non-null  float64
 6   loudness            17996 non-null  float64
 7   mode                17996 non-null  int64
 8   speechiness         17996 non-null  float64
 9   acousticness        17996 non-null  float64
 10  instrumentalness    13619 non-null  float64
 11  liveness            17996 non-null  float64
 12  valence             17996 non-null  float64
 13  tempo               17996 non-null  float64
 14  duration_in min/ms  17996 non-null  float64
 15  time_signature      17996 non-null  int64
 16  Class               17996 non-null  int64
dtypes: float64(12), int64(3), object(2)
memory usage: 2.3+ MB
```

Análisis rápido de cada columna donde podemos ver la cantidad de datos, media, desviación estándar extremos y cuartiles

### 4.0.3 Clasificación de variables

- Artist name: categórica

- Track name: categórica
- Popularity: numérica
- danceability: numérica
- energy: numérica
- key: numérica
- loudness: numérica
- mode: numérica
- speechiness: numérica
- acousticness: numérica
- instrumentalness: numérica
- liveness: numérica
- valence: numérica
- tempo: numérica
- duration_in min/ms: numérica
- time_signature:numérica
- Class: categórica

[153]:
```
df.describe()
```

[153]:

| | Popularity | danceability | energy | key | loudness | \ |
|---|---|---|---|---|---|---|
| count | 17568.000000 | 17996.000000 | 17996.000000 | 15982.000000 | 17996.000000 | |
| mean | 44.512124 | 0.543433 | 0.662777 | 5.952447 | -7.910660 | |
| std | 17.426928 | 0.166268 | 0.235373 | 3.196854 | 4.049151 | |
| min | 1.000000 | 0.059600 | 0.000020 | 1.000000 | -39.952000 | |
| 25% | 33.000000 | 0.432000 | 0.509000 | 3.000000 | -9.538000 | |
| 50% | 44.000000 | 0.545000 | 0.700000 | 6.000000 | -7.016000 | |
| 75% | 56.000000 | 0.659000 | 0.860000 | 9.000000 | -5.189000 | |
| max | 100.000000 | 0.989000 | 1.000000 | 11.000000 | 1.355000 | |

| | mode | speechiness | acousticness | instrumentalness | \ |
|---|---|---|---|---|---|
| count | 17996.000000 | 17996.000000 | 17996.000000 | 13619.000000 | |
| mean | 0.636753 | 0.079707 | 0.247082 | 0.177562 | |
| std | 0.480949 | 0.083576 | 0.310632 | 0.304048 | |
| min | 0.000000 | 0.022500 | 0.000000 | 0.000001 | |
| 25% | 0.000000 | 0.034800 | 0.004300 | 0.000089 | |
| 50% | 1.000000 | 0.047400 | 0.081400 | 0.003910 | |
| 75% | 1.000000 | 0.083000 | 0.434000 | 0.200000 | |
| max | 1.000000 | 0.955000 | 0.996000 | 0.996000 | |

| | liveness | valence | tempo | duration_in min/ms | \ |
|---|---|---|---|---|---|

```
count  17996.000000  17996.000000  17996.000000        1.799600e+04
mean       0.196170      0.486208    122.623294        2.007445e+05
std        0.159212      0.240195     29.571527        1.119891e+05
min        0.011900      0.018300     30.557000        5.016500e-01
25%        0.097500      0.297000     99.620750        1.663370e+05
50%        0.129000      0.481000    120.065500        2.091600e+05
75%        0.258000      0.672000    141.969250        2.524900e+05
max        1.000000      0.986000    217.416000        1.477187e+06

       time_signature         Class
count    17996.000000  17996.000000
mean         3.924039      6.695821
std          0.361618      3.206073
min          1.000000      0.000000
25%          4.000000      5.000000
50%          4.000000      8.000000
75%          4.000000     10.000000
max          5.000000     10.000000
```

Tamaño de la matríz

[154]:
```
df.shape
```

[154]:
```
(17996, 17)
```

#### 4.0.4 Imputación Simple

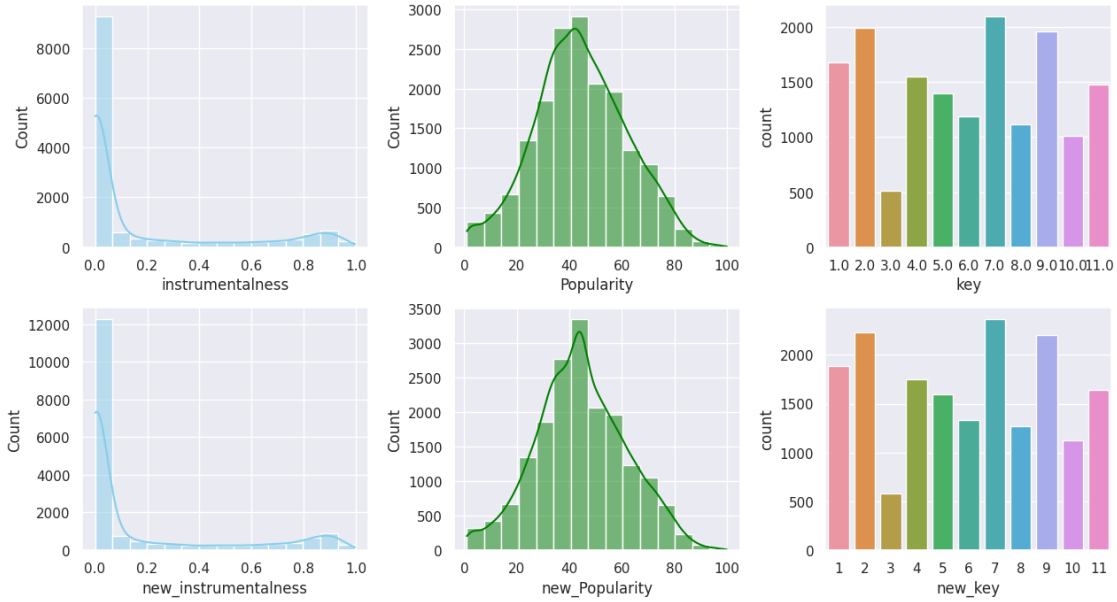
Se detectaron valores faltantes en las columnas de “instrumentalness”, “key” y “Popularity”. Utilizando media, moda y imputacion de k-vecinos más cercanos, se realizó una amputación simple de valores. Para los datos faltantes de ‘popularity’ introducimos el promedio de la columna ya que no cambia la distribucion. Para la columna de valores discretos ‘key’ introducimos el 0 donde faltaran valores ya que las columnas sin estos valores estaban en la tonalidad de C, para saber esto buscamos casos de prueba dentro de la base de datos y los comparamos con la tonalidad que se declaraba en sitios web, así concluimos que las celdas vacías estaban en el C. Finalmente para la variable ‘instrumentalness’ imputamos con el valor anterior para que la distribucion no cambie significativamente. Adicionalmente, no se pueden borrar los datos ya que las variables incluyen un porecentaje significativo de datos, 0.0237, 0.1120, y 0.2432 respectivamente.

[155]:
```
#Imprime el porcentaje de valores faltantes
print('Porcentaje de valores faltantes "Popularity": ', 1-(df['Popularity'].
 ↪count()/17996))
print('Porcentaje de valores faltantes "Key": ', 1-(df['key'].count()/17996))
print('Porcentaje de valores faltantes "Instrumentalness": ',␣
 ↪1-(df['instrumentalness'].count()/17996))

#Creamos nuevas variables para mantener las originales
df['new_instrumentalness'] = df['instrumentalness'].ffill()
```

```
df['new_instrumentalness'].fillna(np.mean(df.instrumentalness), inplace = True)␣
 ↪#Changes the last value

df['new_Popularity'] = df['Popularity']
df['new_Popularity'].fillna(np.mean(df.Popularity),inplace=True)

df['new_key'] = df['key'].ffill()
df['new_key'].fillna(np.mean(df.key), inplace = True) #Changes the last value
df['new_key'] = df['new_key'].astype(int)
```

```
Porcentaje de valores faltantes "Popularity":  0.023783062902867358
Porcentaje de valores faltantes "Key":  0.1119137586130251
Porcentaje de valores faltantes "Instrumentalness":  0.24322071571460324
```

[156]:
```
fig, axs = plt.subplots(2,3, figsize=(15, 8))

bin_count = int(np.ceil(np.log2(len(df)))) #sturges law to figure out␣
 ↪appropiate bin count

sns.histplot(data=df, x="instrumentalness", kde=True, color="skyblue",␣
 ↪ax=axs[0, 0], bins = bin_count)
sns.histplot(data=df, x="new_instrumentalness", kde=True, color="skyblue",␣
 ↪ax=axs[1, 0], bins = bin_count)

sns.histplot(data=df, x="Popularity", kde=True, color="green", ax=axs[0, 1],␣
 ↪bins = bin_count)
sns.histplot(data=df, x="new_Popularity", kde=True, color="green", ax=axs[1,␣
 ↪1], bins = bin_count)

sns.countplot(data=df, x="key", ax=axs[0, 2])
sns.countplot(data=df, x="new_key", ax=axs[1, 2])

fig.subplots_adjust(wspace=0.3, hspace=0.25)
```




[157]:
```
df.info() #Show the changes that were made
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17996 entries, 0 to 17995
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Artist Name           17996 non-null  object
 1   Track Name            17996 non-null  object
 2   Popularity            17568 non-null  float64
 3   danceability          17996 non-null  float64
 4   energy                17996 non-null  float64
 5   key                   15982 non-null  float64
 6   loudness              17996 non-null  float64
 7   mode                  17996 non-null  int64
 8   speechiness           17996 non-null  float64
 9   acousticness          17996 non-null  float64
 10  instrumentalness      13619 non-null  float64
 11  liveness              17996 non-null  float64
 12  valence               17996 non-null  float64
 13  tempo                 17996 non-null  float64
 14  duration_in min/ms    17996 non-null  float64
 15  time_signature        17996 non-null  int64
 16  Class                 17996 non-null  int64
 17  new_instrumentalness  17996 non-null  float64
 18  new_Popularity        17996 non-null  float64
 19  new_key               17996 non-null  int64
```

```
dtypes: float64(14), int64(4), object(2)
memory usage: 2.7+ MB
```

Como podemos ver, todas las columnas nuevas que se crearon tienen 17996 datos no nulos (se crearon columnas nuevas para preservar las orginales)

### 4.0.5 Creando nueva clase de Género

Al estar codificada la variable "Class" en números del 1 al 10 es necesario interpretar los números con respecto a cada uno de los géneros musicales. En Base a la tabla proporcionada se crea una nueva variable llamada "Genre" que representa explicítamente el género al que pertenece cada canción.

[158]:
```
#Create a function that relates the numerical values of class to its␣
 ↪corresponding genre
def class_to_genre(row):
    if row == 0:
        return 'Acoustic/Folk'
    elif row == 1:
        return 'Alternative'
    elif row == 2:
       return 'Blues'
    elif row == 3:
       return 'Bollywood'
    elif row == 4:
       return 'Country'
    elif row == 5:
        return 'Hip-Hop'
    elif row == 6:
       return 'Indie'
    elif row == 7:
        return 'Instrumental'
    elif row == 8:
        return 'Metal'
    elif row == 9:
        return 'Pop'
    elif row == 10:
        return 'Rock'

df['Genre'] = df['Class'].apply(class_to_genre)
df.head()
```

[158]:

| | Artist Name | Track Name | Popularity | \ |
|---|---|---|---|---|
| 0 | Bruno Mars | That's What I Like (feat. Gucci Mane) | 60.0 | |
| 1 | Boston | Hitch a Ride | 54.0 | |
| 2 | The Raincoats | No Side to Fall In | 35.0 | |
| 3 | Deno | Lingo (feat. J.I & Chunkz) | 66.0 | |
| 4 | Red Hot Chili Peppers | Nobody Weird Like Me - Remastered | 53.0 | |

```
   danceability  energy   key  loudness  mode  speechiness  acousticness  …  \
0         0.854   0.564   1.0    -4.964     1       0.0485      0.017100  …
1         0.382   0.814   3.0    -7.230     1       0.0406      0.001100  …
2         0.434   0.614   6.0    -8.334     1       0.0525      0.486000  …
3         0.853   0.597  10.0    -6.528     0       0.0555      0.021200  …
4         0.167   0.975   2.0    -4.279     1       0.2160      0.000169  …

   liveness  valence    tempo  duration_in min/ms  time_signature  Class  \
0    0.0849   0.8990  134.071            234596.0               4      5
1    0.1010   0.5690  116.454            251733.0               4     10
2    0.3940   0.7870  147.681            109667.0               4      6
3    0.1220   0.5690  107.033            173968.0               4      5
4    0.1720   0.0918  199.060            229960.0               4     10

   new_instrumentalness  new_Popularity  new_key    Genre
0              0.177562            60.0        1  Hip-Hop
1              0.004010            54.0        3     Rock
2              0.000196            35.0        6    Indie
3              0.000196            66.0       10  Hip-Hop
4              0.016100            53.0        2     Rock

[5 rows x 21 columns]
```

#### 4.0.6 New class Key

Similar al proceso anterior, decodificamos el atributo 'key' de los registros donde el 0.0 recibe el la calsificación de C, 1.0 de C#, … y 11.0 de B. dentro de una nueva variable categórica 'Key'.

[159]:
```
def class_to_Key(row):
    if row == 0.0:
        return 'C'
    elif row == 1.0:
        return 'C#'
    elif row == 2.0:
       return 'D'
    elif row == 3.0:
       return 'D#'
    elif row == 4.0:
       return 'E'
    elif row == 5.0:
        return 'F'
    elif row == 6.0:
       return 'F#'
    elif row == 7.0:
        return 'G'
    elif row == 8.0:
```

```
        return 'G#'
    elif row == 9.0:
        return 'A'
    elif row == 10.0:
        return 'A#'
    elif row == 11.0:
        return 'B'

df['Key'] = df['key'].apply(class_to_Key)
df.head()
```

[159]:

| | Artist Name | Track Name | Popularity | \ |
|---|---|---|---|---|
| 0 | Bruno Mars | That's What I Like (feat. Gucci Mane) | 60.0 | |
| 1 | Boston | Hitch a Ride | 54.0 | |
| 2 | The Raincoats | No Side to Fall In | 35.0 | |
| 3 | Deno | Lingo (feat. J.I & Chunkz) | 66.0 | |
| 4 | Red Hot Chili Peppers | Nobody Weird Like Me - Remastered | 53.0 | |

| | danceability | energy | key | loudness | mode | speechiness | acousticness | … | \ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.854 | 0.564 | 1.0 | -4.964 | 1 | 0.0485 | 0.017100 | … | |
| 1 | 0.382 | 0.814 | 3.0 | -7.230 | 1 | 0.0406 | 0.001100 | … | |
| 2 | 0.434 | 0.614 | 6.0 | -8.334 | 1 | 0.0525 | 0.486000 | … | |
| 3 | 0.853 | 0.597 | 10.0 | -6.528 | 0 | 0.0555 | 0.021200 | … | |
| 4 | 0.167 | 0.975 | 2.0 | -4.279 | 1 | 0.2160 | 0.000169 | … | |

| | valence | tempo | duration_in min/ms | time_signature | Class | \ |
|---|---|---|---|---|---|---|
| 0 | 0.8990 | 134.071 | 234596.0 | 4 | 5 | |
| 1 | 0.5690 | 116.454 | 251733.0 | 4 | 10 | |
| 2 | 0.7870 | 147.681 | 109667.0 | 4 | 6 | |
| 3 | 0.5690 | 107.033 | 173968.0 | 4 | 5 | |
| 4 | 0.0918 | 199.060 | 229960.0 | 4 | 10 | |

| | new_instrumentalness | new_Popularity | new_key | Genre | Key |
|---|---|---|---|---|---|
| 0 | 0.177562 | 60.0 | 1 | Hip-Hop | C# |
| 1 | 0.004010 | 54.0 | 3 | Rock | D# |
| 2 | 0.000196 | 35.0 | 6 | Indie | F# |
| 3 | 0.000196 | 66.0 | 10 | Hip-Hop | A# |
| 4 | 0.016100 | 53.0 | 2 | Rock | D |

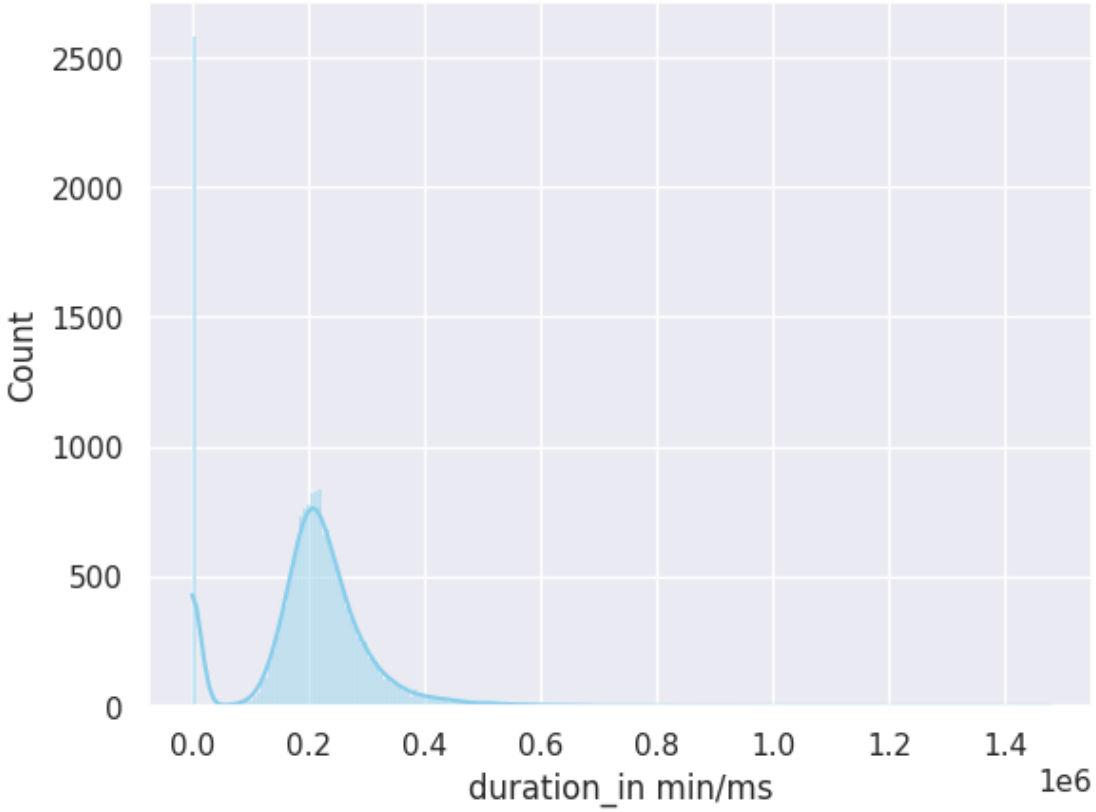
[5 rows x 22 columns]

#### 4.0.7 Histogramas

En esta parte analizaremos el atributo de duración en las canciones. Se nota que los datos es tan en min/ms es por ello que se debe hacer el ajuste necesario para mejor comprender los datos.

[160]:
```
sns.set(style="darkgrid")
sns.histplot(data=df, x="duration_in min/ms", kde=True, color="skyblue")
```

[160]:
```
<Axes: xlabel='duration_in min/ms', ylabel='Count'>
```



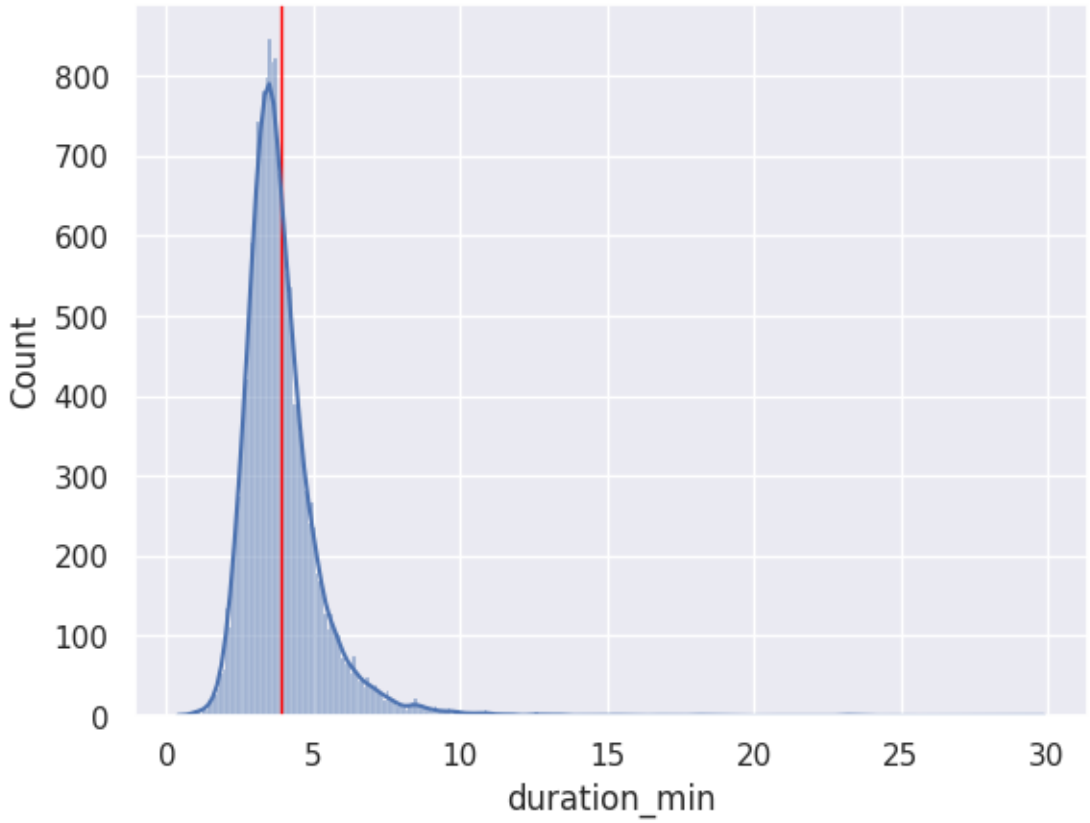

Al revisar la gráfica se observa una gran cantidad de datos en la duración 3 min, al corroborar con la base de datos se encontró que algunas canciones estaban en minutos mientras que otras estaban escritas como milisegundos. Por lo tanto se comprueba la medida de la duración en cada una de las canciones, aquellas con valores menores a 100 se multiplican por 60000 para convertirlos a minutos. Una vez se tiene todos los datos de la duración en minutos se vuelve a generar un histograma de la misma variable con los datos correctamente medidos.

[161]:
```
condition = df['duration_in min/ms'] < 100 #condicion que discrimina los datos␣
 ↪con nuestra medida incorrecta

df.loc[condition, 'duration_in min/ms']=df.loc[condition, 'duration_in min/
 ↪ms']*60000
df['duration_min'] = df['duration_in min/ms']/60000
```

Con el fin de observar la distribución de frecuencias del tiempo correcta en cada una de las canciones se genera un histograma de la variable “duration_min” ya corregida.

[162]:
```
plt.axvline(df['duration_min'].mean(), color='red',linewidth=1)

sns.histplot(data=df,x="duration_min", kde=True)
```

[162]:
```
<Axes: xlabel='duration_min', ylabel='Count'>
```




[163]:
```
df['duration_min'].describe()
```

[163]:
```
count    17996.000000
mean         3.930388
std          1.427917
min          0.388667
25%          3.127496
50%          3.666667
75%          4.384704
max         29.886000
Name: duration_min, dtype: float64
```

Se observa que al generar esta nueva columna de datos que describe la duracion en minutos de las canciones resalta que la maxima de los datos es de 29.89 mientras la media es de 3.93 con una std de 1.42. Por lo tanto seria importante hacer un analisis para determinar si todos los datos son

relevantes para el analisis ya que existen valores extraordinarios.

#### 4.0.8 Nueva variable "collab"

Aquí creamos una nueva variable booleana 'collab' donde 1 significa que la canción es una colaboración entre artistas y 0 es que no lo es. Para hacer esto definimos que hay dos posible casos que indiquen esto: cuando la canción tiene una ',' en el atributo 'Artist Name' o cuando contiene la palabra 'feat.' dentro de 'Track Name'. En total encontramos 1202 canciones con colaboración.

[164]:
```
df['collab'] = df['Artist Name'].str.contains(',') + df['Track Name'].str.
 ↪contains('feat.')
df['collab']
```

[164]:
```
0         True
1        False
2        False
3         True
4        False
         …
17991    False
17992    False
17993    False
17994    False
17995    False
Name: collab, Length: 17996, dtype: bool
```

## 5 Fase 2. Exploración de los datos

En esta fase desarrollaremos análisis estadísticos detallados enfocándonos en la distribución de los atributos y sus caracterísiticas. Adicionalmente realizaremos un radar chart para conocer las características de las 10 canciones más escuchadas en nuestra base de datos

[165]:
```
df2=df[['new_Popularity','danceability','energy',␣
 ↪'mode','loudness','speechiness','acousticness','new_instrumentalness','liveness','valence',
df2.head()
```

[165]:

| | new_Popularity | danceability | energy | mode | loudness | speechiness | \ |
|---|---|---|---|---|---|---|---|
| 0 | 60.0 | 0.854 | 0.564 | 1 | -4.964 | 0.0485 | |
| 1 | 54.0 | 0.382 | 0.814 | 1 | -7.230 | 0.0406 | |
| 2 | 35.0 | 0.434 | 0.614 | 1 | -8.334 | 0.0525 | |
| 3 | 66.0 | 0.853 | 0.597 | 0 | -6.528 | 0.0555 | |
| 4 | 53.0 | 0.167 | 0.975 | 1 | -4.279 | 0.2160 | |

| | acousticness | new_instrumentalness | liveness | valence | tempo | \ |
|---|---|---|---|---|---|---|
| 0 | 0.017100 | 0.177562 | 0.0849 | 0.8990 | 134.071 | |
| 1 | 0.001100 | 0.004010 | 0.1010 | 0.5690 | 116.454 | |
| 2 | 0.486000 | 0.000196 | 0.3940 | 0.7870 | 147.681 | |

```
3      0.021200              0.000196    0.1220   0.5690  107.033
4      0.000169              0.016100    0.1720   0.0918  199.060

   duration_min
0      3.909933
1      4.195550
2      1.827783
3      2.899467
4      3.832667
```

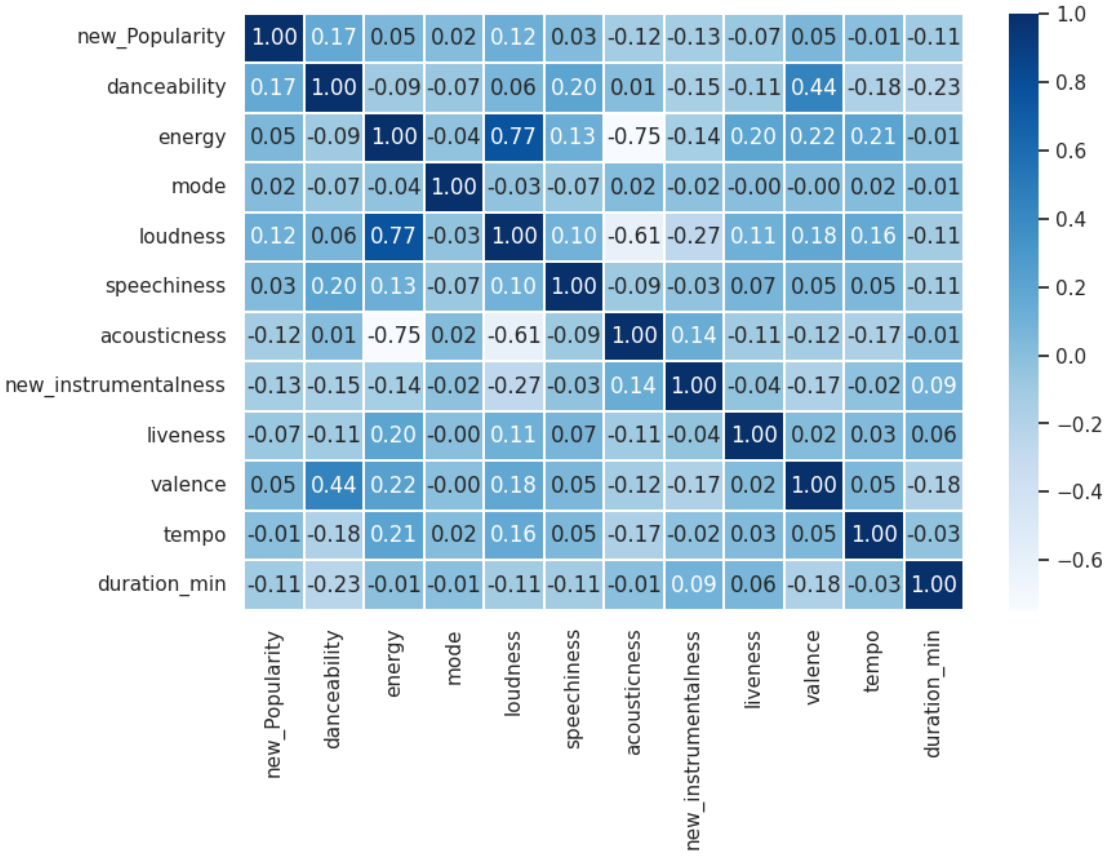
Estos serán las variables cuantitativas que se usaran para el análisis de estadística descriptiva.

### 5.0.1 Correlation Heatmap

En cuanto a la visualización de la matriz de correlación que tienen entre las variables de "Popularity", "danceability", "energy", "loudness", "speechiness", "acousticness", "instrumentalness", "liveness", "valence", "tempo" y "duration_in min", y se produce un mapa de calor con el fin de encontrar alguna relación entre "Popularity" y las demas variables.

[166]:
```
fig, ax = plt.subplots(figsize = (9, 6))
sns.heatmap(data = df2.corr(), cmap ='Blues', linewidths = 0.30, annot␣
 ↪=True,fmt='.2f')
```

[166]:
```
<Axes: >
```

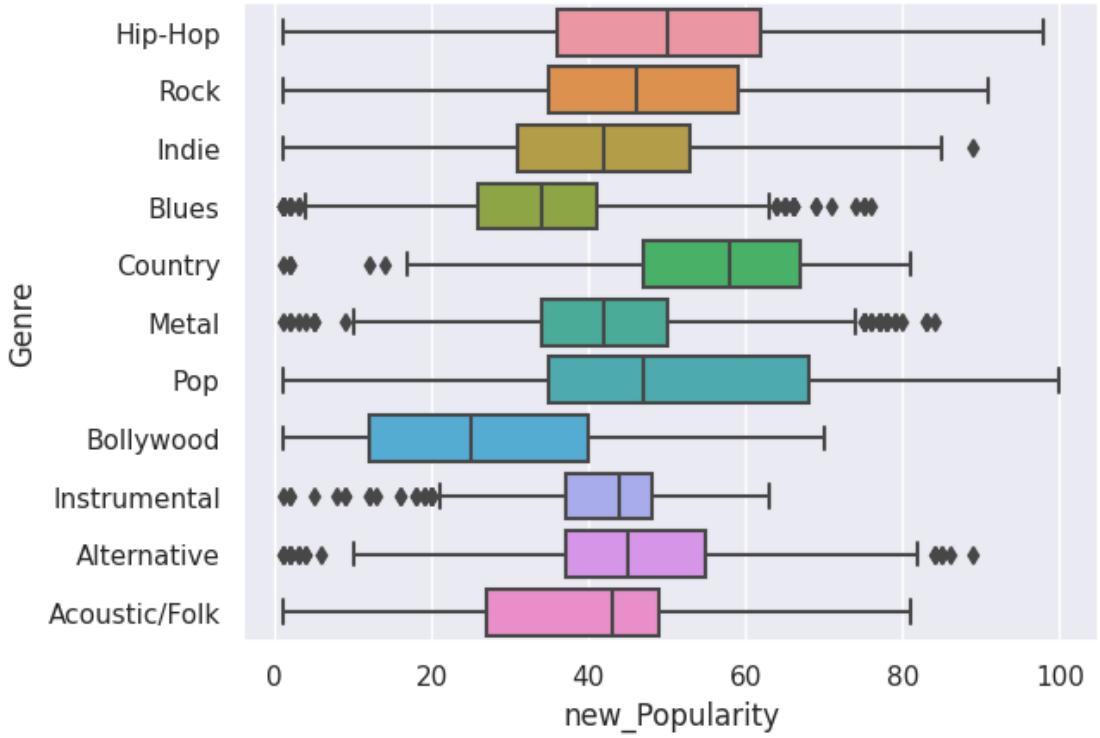



Se denotan la variables “loudness” y “energy” con una relacion positiva moderadamente fuerte de 0.77 que nos indica entre más ruidosa la canción la energia de ella suele aumentar también. Seguida por “valence” y “danceability” con una relación moderademente debil de 0.44. Por otro lado, “acousticness” y “energy” tienen una relación negativa moderadamente fuerte lo que sugiere que entre más acústica menos energía en la canción. Además, la variable “Popularity” tiene relaciones muy poco significativas o neutras con las demás variables cuantitativas.

### 5.0.2 Boxplot

Con el objetivo de ver la relación que tienen el género de las canciones con su popularidad se genera un boxplot. En el eje horizontal de la visualización se representa la popularidad de las canciones, mientras que en el eje vertical se observan cada una de las categorías de género.

[167]:
```
sns.boxplot(x=df['new_Popularity'],y=df['Genre']) #Distribución de popularidad␣
↪por Género musical
plt.show()
```




Esta gráfica presenta información acerca del rango intercuartil, la mediana, la cual indica la variabilidad en la popularidad dentro de cada género, así como los valores atípicos. Al examinar los datos proporcionados por el boxplot se resalta que el género "Country" tiende a ser más popular, pues presenta una mediana más alta que el resto, mientras que "Indie" y "Alternative" tienen canciones excepcionalmente populares. Por otro lado, el género con menor popularidad es el de "Bollywood" con una mediana menor.

### 5.0.3 Worldcloud de artistas

Generamos un Wordcloud de artistas para visualizar las palabras más recurridas.

[168]:
```
# Create the wordcloud object
artist_array = ''.join(df['Artist Name'])
wordcloud = WordCloud(width=480, height=480, margin=0).generate(artist_array)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.margins(x=0, y=0)
plt.show()
#sns.violinplot(x=df["species"], y=df["sepal_length"])
```


Palabras más comunes en los nombres de artistas, resalta “Ben”, “Blue”, “Black”, “Rolling”, entre otros.

### 5.0.4 Worldcloud de nombres de canciones

De misma manera se generó un wordcloud para los nombres de canciones y buscamos por patrones o relaciones.

[169]:
```
# Create the wordcloud object
track_array = ''.join(df['Track Name'])
wordcloud = WordCloud(width=480, height=480, margin=0).generate(track_array)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.margins(x=0, y=0)
plt.show()
```

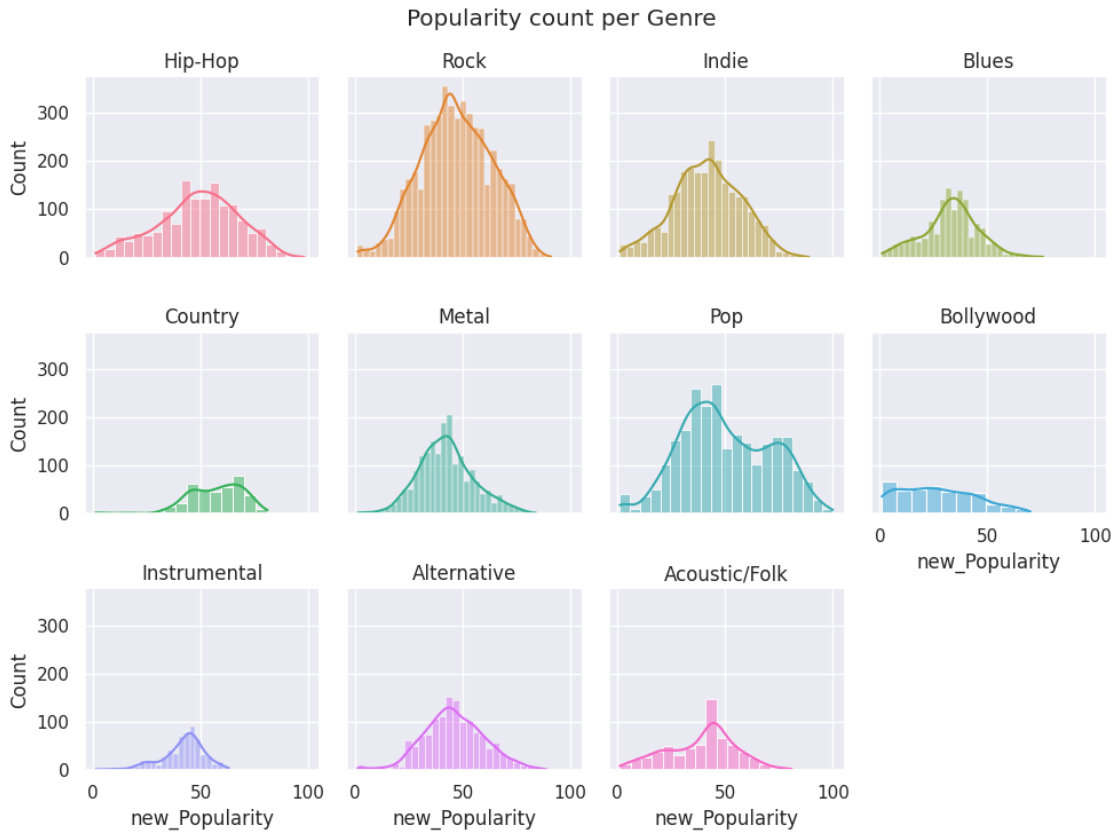

Palabras más comunes en nombres de canciones, resaltan feat, Love ,remastered y live. A su vez, ecnontramos caracteres inusuales que corresponden en buena parte a los datos irrelevantes dentro de la columna

### 5.0.5 Comparativo entre "Popularity"y "Genre"

Se realizaron histogramas de la popularidad de cada género con fin de visualizar la popularidad de cada uno en la musica.

[170]:
```
g = sns.FacetGrid(df, col='Genre',hue = 'Genre', col_wrap=4, height=2.5)
g.map(sns.histplot,'new_Popularity', kde = True)
g.set_titles("{col_name}")
g.fig.subplots_adjust(top=0.9)
g.fig.suptitle('Popularity count per Genre')
```

[170]:
```
Text(0.5, 0.98, 'Popularity count per Genre')
```




En el gráfico anterior, se puede observar las diferencias de popularidad, según cada género musical, siendo rock, pop, Indie y Metal los más populares. Además la linea suave en cada diagrama, ayuda a darnos una idea de la forma en que están distribuidos los datos en cada género,siendo la forma más popular la de una distribución normal

### 5.0.6 Estadística básica de las variables relevantes

[171]:
```
#Pairplot with the most relevant or significant variables
'''
df3=df[['new_Popularity','danceability','energy',␣
 ↪'mode','loudness','liveness','duration_in min/ms']]
g = sns.pairplot(df3, kind="reg", diag_kind = 'kde',height=2,corner = True,␣
 ↪plot_kws={'line_kws':{'color':'red'}, 'scatter_kws': {'alpha': 0.1}})
#makes the lower half have a sort of heat map density
g.map_lower(sns.kdeplot, color=".2",levels=5)
plt.show()
'''
```

[171]:
```
'\ndf3=df[[\'new_Popularity\',\'danceability\',\'energy\',
\'mode\',\'loudness\',\'liveness\',\'duration_in min/ms\']]\ng =
sns.pairplot(df3, kind="reg", diag_kind = \'kde\',height=2,corner = True,
```

```
plot_kws={\'line_kws\':{\'color\':\'red\'}, \'scatter_kws\': {\'alpha\':
0.1}})\n#makes the lower half have a sort of heat map
density\ng.map_lower(sns.kdeplot, color=".2",levels=5)\nplt.show()\n'
```

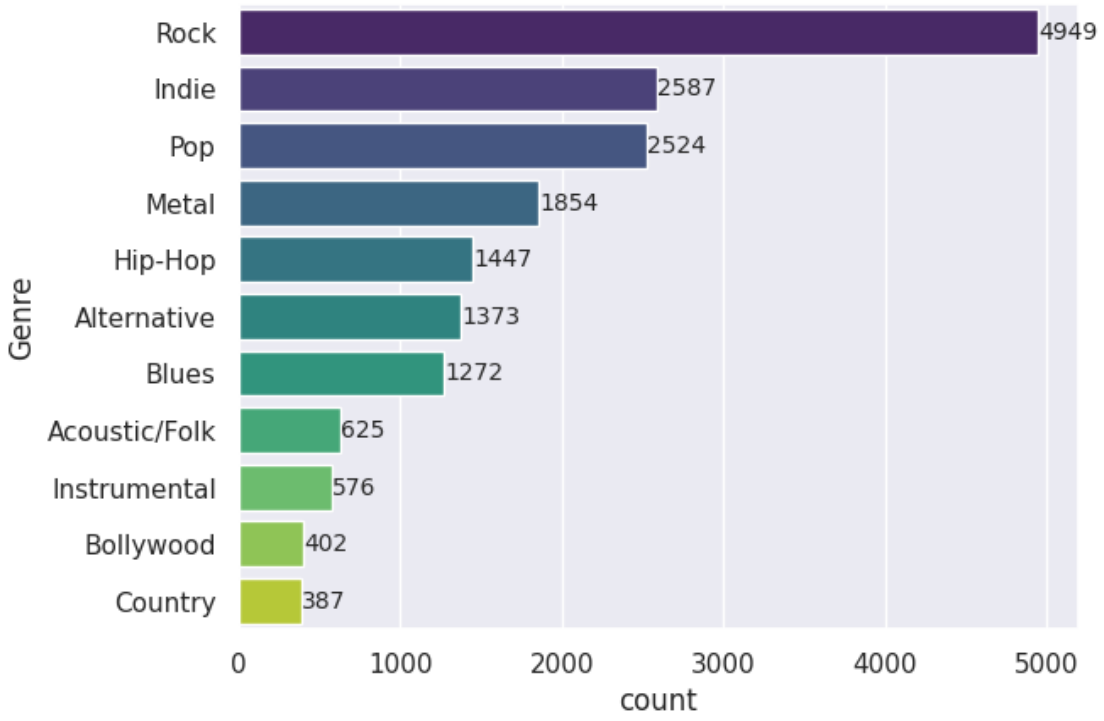
- Selección de Variables: Se eligieron específicamente 'new_Popularity', 'danceability', 'energy', 'loudness', 'liveness' y 'duration_in min/ms' para explorar sus relaciones.
- Líneas de Regresión y Scatter Plots: Las líneas rojas en los gráficos de dispersión indican las tendencias lineales entre las variables. Observa la dirección y pendiente de estas líneas para inferir la dirección y fuerza de las relaciones.
- Densidad Kernel en la Diagonal Principal: Los gráficos de densidad kernel en la diagonal principal ofrecen una visión de la distribución univariante de cada variable.
- Mapa de Calor de Densidad: En la mitad inferior, se utiliza un mapa de calor de densidad para resaltar áreas de alta densidad de puntos, proporcionando información adicional sobre las concentraciones de observaciones.

Algo interesante que se puede observar es que todas las variables demuestran tener una fuerte relación con la variable de duracion, las concentraciones de datos están muy cerca de la línea de regresión, lo que significa que existe una correlación fuerte

### 5.0.7 Bar Plot de "Genre"

Con el fin de comparar los géneros de música, a se representa su frecuencia dentro de los datos por medio de un diagrama de barras. Además se incluye la descripción de sus porcentajes.

[172]:
```
ax = sns.countplot(y = 'Genre', data = df, palette = 'viridis', order =␣
  ↪df['Genre'].value_counts().index)
for bars in ax.containers:
  ax.bar_label(bars,size= 10)
```




[173]:
```
df['Genre'].value_counts(normalize=True)
```

[173]:
```
Rock             0.275006
Indie            0.143754
Pop              0.140253
Metal            0.103023
Hip-Hop          0.080407
Alternative      0.076295
Blues            0.070682
Acoustic/Folk    0.034730
Instrumental     0.032007
Bollywood        0.022338
Country          0.021505
Name: Genre, dtype: float64
```

Tras analizar los resultados, se observa que el género "Rock" es aquel que sobresale, acumulando 4949 canciones, el equivalente al 27% del total. Es seguido por "Indie", "Pop", "Metal", etc.

[174]:
```
#Estadística Básica
columnas_numericas = df2.select_dtypes(include=[int, float])

#Coeficiente de asimetria
coeficiente_asimetria_dict = {}
```

```
for i in columnas_numericas.columns:
    coeficiente = columnas_numericas[i].skew()
    coeficiente_asimetria_dict[i] = coeficiente

print(f"coeficiente de asimetría: {coeficiente_asimetria_dict}")

#Coeficiente de variación (%)
coeficiente_variacion_dict = {}

for i in columnas_numericas.columns:
    media = columnas_numericas[i].mean()
    desviacion_estandar = columnas_numericas[i].std()
    coeficiente_variacion = (desviacion_estandar / media) * 100
    coeficiente_variacion_dict[i] = coeficiente_variacion

print(f"coeficiente de variación: {coeficiente_variacion_dict}")
```

```
coeficiente de asimetría: {'new_Popularity': 0.07662169457966905,
'danceability': -0.08352192347287282, 'energy': -0.6611691117532402, 'mode':
-0.5687418793933883, 'loudness': -1.7613834605630743, 'speechiness':
3.088002356652607, 'acousticness': 1.1054970459533517, 'new_instrumentalness':
1.5257937663230359, 'liveness': 2.176072140966749, 'valence':
0.08992812736275954, 'tempo': 0.37961889582629255, 'duration_min':
4.028886798341066}
coeficiente de variación: {'new_Popularity': 38.68257565171272, 'danceability':
30.595935969574263, 'energy': 35.51322160007704, 'mode': 75.53147216805436,
'loudness': -51.186008881905785, 'speechiness': 104.85411233918515,
'acousticness': 125.72046929178262, 'new_instrumentalness': 171.13801210900118,
'liveness': 81.16008924316768, 'valence': 49.401699003315144, 'tempo':
24.115750260034265, 'duration_min': 36.330182897292474}
```

[175]:
```
#bs = columnas_numericas.describe()
bs = columnas_numericas.describe()
bs.loc["Mode"] = [df['new_Popularity'].mode()[0],df['danceability'].
 ↪mode()[0],df['energy'].mode()[0], df['mode'].mode()[0],df['loudness'].
 ↪mode()[0],df['speechiness'].mode()[0],df['acousticness'].
 ↪mode()[0],df['new_instrumentalness'].mode()[0],df['liveness'].
 ↪mode()[0],df['valence'].mode()[0],df['tempo'].mode()[0],df['duration_in min/
 ↪ms'].mode()[0]]
bs.loc["Variance"] = [df['new_Popularity'].var(),df['danceability'].
 ↪var(),df['energy'].var(), df['mode'].var(),df['loudness'].
 ↪var(),df['speechiness'].var(),df['acousticness'].
 ↪var(),df['new_instrumentalness'].var(),df['liveness'].var(),df['valence'].
 ↪var(),df['tempo'].var(),df['duration_in min/ms'].var()]
IQR = [bs.iat[6, 0]-bs.iat[4, 0]
       ,bs.iat[6, 1]-bs.iat[4, 1]
       ,bs.iat[6, 2]-bs.iat[4, 2]
```

```
       ,bs.iat[6, 3]-bs.iat[4, 3]
       ,bs.iat[6, 4]-bs.iat[4, 4]
       ,bs.iat[6, 5]-bs.iat[4, 5]
       ,bs.iat[6, 6]-bs.iat[4, 6]
       ,bs.iat[6, 7]-bs.iat[4, 7]
       ,bs.iat[6, 8]-bs.iat[4, 8]
       ,bs.iat[6, 9]-bs.iat[4, 9]
       ,bs.iat[6, 10]-bs.iat[4, 10]
       ,bs.iat[6, 11]-bs.iat[4, 11]]
bs.loc["IQR"] = IQR
rango = [bs.iat[7, 0]-bs.iat[3, 0]
       ,bs.iat[7, 1]-bs.iat[3, 1]
       ,bs.iat[7, 2]-bs.iat[3, 2]
       ,bs.iat[7, 3]-bs.iat[3, 3]
       ,bs.iat[7, 4]-bs.iat[3, 4]
       ,bs.iat[7, 5]-bs.iat[3, 5]
       ,bs.iat[7, 6]-bs.iat[3, 6]
       ,bs.iat[7, 7]-bs.iat[3, 7]
       ,bs.iat[7, 8]-bs.iat[3, 8]
       ,bs.iat[7, 9]-bs.iat[3, 9]
       ,bs.iat[7, 10]-bs.iat[3, 10]
       ,bs.iat[7, 11]-bs.iat[3, 11]]
bs.loc["Range"] = rango
bs.loc["Kurtosis"] = [df['new_Popularity'].kurtosis(),df['danceability'].
 ↪kurtosis(),df['energy'].kurtosis(), df['mode'].kurtosis(),df['loudness'].
 ↪kurtosis(),df['speechiness'].kurtosis(),df['acousticness'].
 ↪kurtosis(),df['new_instrumentalness'].kurtosis(),df['liveness'].
 ↪kurtosis(),df['valence'].kurtosis(),df['tempo'].kurtosis(),df['duration_in␣
 ↪min/ms'].kurtosis()]
bs
```

[175]:

| | new_Popularity | danceability | energy | mode | \ |
|---|---|---|---|---|---|
| count | 17996.000000 | 17996.000000 | 17996.000000 | 17996.000000 | |
| mean | 44.512124 | 0.543433 | 0.662777 | 0.636753 | |
| std | 17.218436 | 0.166268 | 0.235373 | 0.480949 | |
| min | 1.000000 | 0.059600 | 0.000020 | 0.000000 | |
| 25% | 33.000000 | 0.432000 | 0.509000 | 0.000000 | |
| 50% | 44.000000 | 0.545000 | 0.700000 | 1.000000 | |
| 75% | 56.000000 | 0.659000 | 0.860000 | 1.000000 | |
| max | 100.000000 | 0.989000 | 1.000000 | 1.000000 | |
| Mode | 42.000000 | 0.527000 | 0.872000 | 1.000000 | |
| Variance | 296.474544 | 0.027645 | 0.055401 | 0.231312 | |
| IQR | 23.000000 | 0.227000 | 0.351000 | 1.000000 | |
| Range | 99.000000 | 0.929400 | 0.999980 | 1.000000 | |
| Kurtosis | -0.147601 | -0.283735 | -0.316936 | -1.676719 | |

| | loudness | speechiness | acousticness | new_instrumentalness | \ |
|---|---|---|---|---|---|

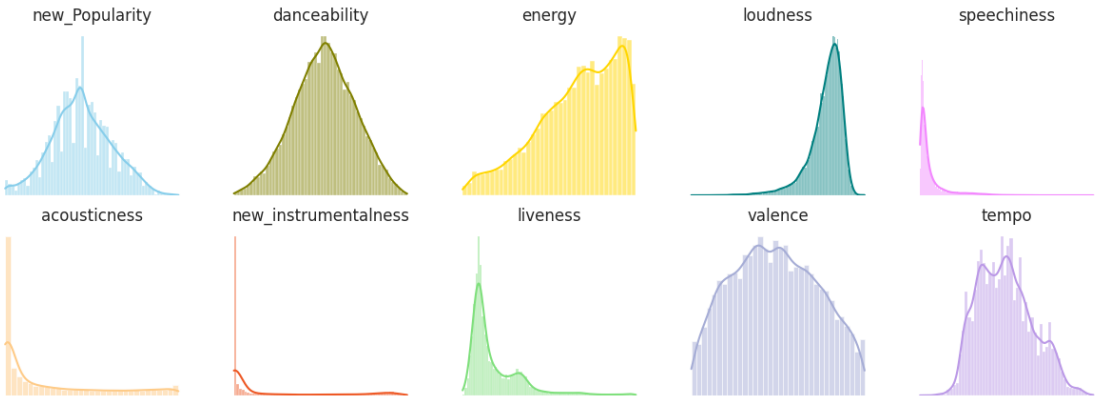
| | | | | |
|---|---|---|---|---|
| count | 17996.000000 | 17996.000000 | 17996.000000 | 17996.000000 |
| mean | -7.910660 | 0.079707 | 0.247082 | 0.177977 |
| std | 4.049151 | 0.083576 | 0.310632 | 0.304586 |
| min | -39.952000 | 0.022500 | 0.000000 | 0.000001 |
| 25% | -9.538000 | 0.034800 | 0.004300 | 0.000089 |
| 50% | -7.016000 | 0.047400 | 0.081400 | 0.003870 |
| 75% | -5.189000 | 0.083000 | 0.434000 | 0.199000 |
| max | 1.355000 | 0.955000 | 0.996000 | 0.996000 |
| Mode | -5.576000 | 0.031700 | 0.102000 | 0.929000 |
| Variance | 16.395624 | 0.006985 | 0.096492 | 0.092773 |
| IQR | 4.349000 | 0.048200 | 0.429700 | 0.198911 |
| Range | 41.307000 | 0.932500 | 0.996000 | 0.995999 |
| Kurtosis | 5.037741 | 12.668128 | -0.179139 | 0.702757 |

| | liveness | valence | tempo | duration_min |
|---|---|---|---|---|
| count | 17996.000000 | 17996.000000 | 17996.000000 | 1.799600e+04 |
| mean | 0.196170 | 0.486208 | 122.623294 | 3.930388e+00 |
| std | 0.159212 | 0.240195 | 29.571527 | 1.427917e+00 |
| min | 0.011900 | 0.018300 | 30.557000 | 3.886667e-01 |
| 25% | 0.097500 | 0.297000 | 99.620750 | 3.127496e+00 |
| 50% | 0.129000 | 0.481000 | 120.065500 | 3.666667e+00 |
| 75% | 0.258000 | 0.672000 | 141.969250 | 4.384704e+00 |
| max | 1.000000 | 0.986000 | 217.416000 | 2.988600e+01 |
| Mode | 0.110000 | 0.389000 | 119.993000 | 1.920000e+05 |
| Variance | 0.025348 | 0.057694 | 874.475229 | 7.340211e+09 |
| IQR | 0.160500 | 0.375000 | 42.348500 | 1.257208e+00 |
| Range | 0.988100 | 0.967700 | 186.859000 | 2.949733e+01 |
| Kurtosis | 5.633397 | -0.915963 | -0.447204 | 3.939116e+01 |

###Transformación de Box-Cox

[176]:
```
fig, axs = plt.subplots(2,5, figsize=(15, 5))

sns.histplot(data=df2, x="new_Popularity", kde=True, color="skyblue", ax=axs[0,␣
↪0]).set_title('new_Popularity')
sns.histplot(data=df2, x="danceability", kde=True, color="olive", ax=axs[0, 1]).
↪set_title('danceability')
sns.histplot(data=df2, x="energy", kde=True, color="gold", ax=axs[0, 2]).
↪set_title('energy')
sns.histplot(data=df2, x="loudness", kde=True, color="teal", ax=axs[0, 3]).
↪set_title('loudness')
sns.histplot(data=df2, x="speechiness", kde=True, color="#F387FE", ax=axs[0,␣
↪4]).set_title('speechiness')
sns.histplot(data=df2, x="acousticness", kde=True, color="#FECB87", ax=axs[1,␣
↪0]).set_title('acousticness')
sns.histplot(data=df2, x="new_instrumentalness", kde=True, color="#EE5824",␣
↪ax=axs[1, 1]).set_title('new_instrumentalness')
```

```
sns.histplot(data=df2, x="liveness", kde=True, color="#7FE07C", ax=axs[1, 2]).
 ↪set_title('liveness')
sns.histplot(data=df2, x="valence", kde=True, color="#A6ADD6", ax=axs[1, 3]).
 ↪set_title('valence')
sns.histplot(data=df2, x="tempo", kde=True, color="#BB99E7", ax=axs[1, 4]).
 ↪set_title('tempo')

for ax in axs.flat:
    ax.label_outer()

axs = axs.flatten()
for ax in axs:
    ax.set_axis_off()
    ax.set_xticks([])
    ax.set_yticks([])

plt.xticks(visible=False)

plt.show()
```

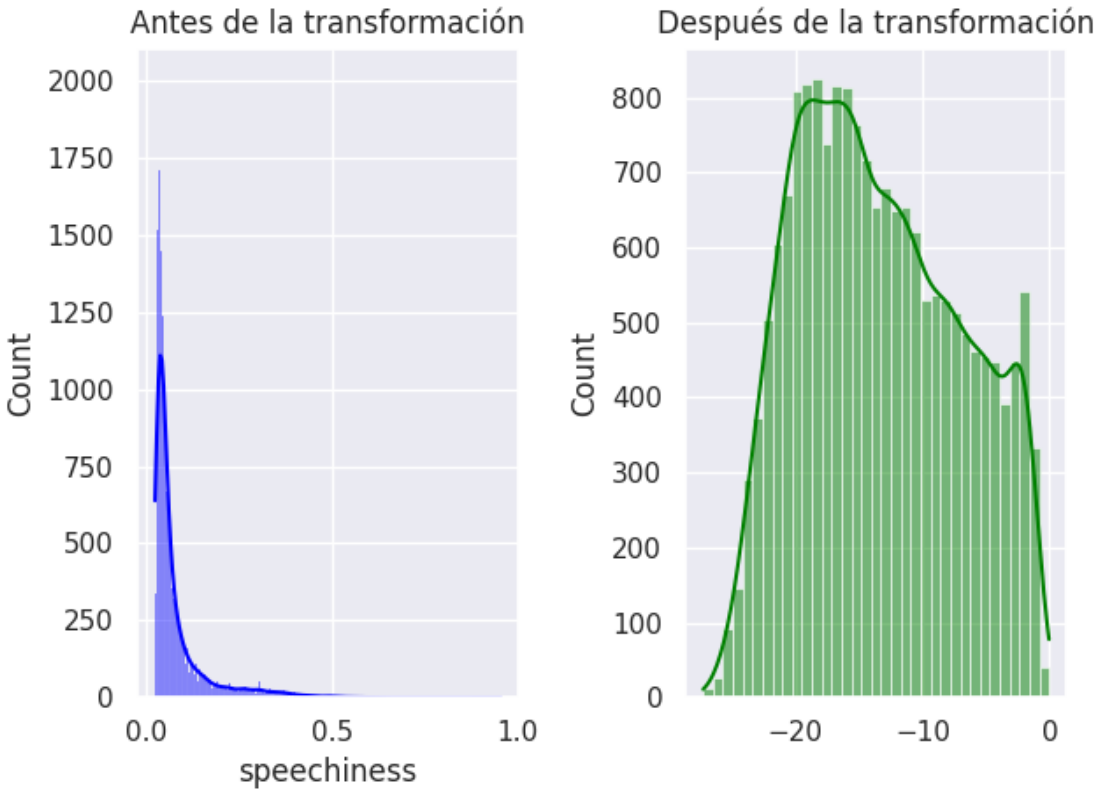



Las variables de 'loudness', 'speechiness', 'acousticness', 'new_instrumentalness' y 'liveness' son las principales variables que presentan un exceso de sesgo, algunos hacia la izquierda, algunos hacia de la derecha.

La transformación de Box-Cox puede tratar con el sesgo de los datos, esta transformación nos puede ayudar a estabilizar las varianzas de los datos para asemejar más su distribución a una normal. Este metodo será util más adelante ya que utiizaremos métodos estadísticos en los que se asume la normalidad de los datos

[177]:
```
data_speechiness, lambda_speechiness = st.boxcox(df['speechiness'])

# Histograma antes de la transformación
plt.subplot(1, 2, 1)
```

```
sns.histplot(df['speechiness'], kde=True, color='blue')
plt.title('Antes de la transformación')

# Histograma después de la transformación
plt.subplot(1, 2, 2)
sns.histplot(data_speechiness, kde=True, color='green')
plt.title('Después de la transformación')

plt.tight_layout()
plt.show()
```




Se optó por aplicar la transformación a la variable 'speechiness' debido a su marcado sesgo hacia la izquierda. Todos los valores de esta variable son mayores a cero, lo que la hace adecuada para la transformación de Box-Cox.

En los resultados se observa el cambio en el sesgo, aunque no se ha alcanzado una distribución normal perfecta, la variable ahora es más facil de manipular y trabajar con ella al aporximarse a una distribución normal.

### 5.0.8 Test de Grubbs

Con el objetivo de encontrar valores atípicos en los datos presentados se busca realizar un Test de Grubbs, el cual tiene la finalidad de encontrar un valor atípico en aquellos datos que tienen una

distribución normal. Es por esto que se aplica esta prueba aquellos datos que se observan que tienen una distribución normal o parecida como es el caso de “speechiness”, “liveness” y “duration_in min/ms”

[178]:
```
!pip install outlier_utils
from outliers import smirnov_grubbs as grubbs
```

```
Requirement already satisfied: outlier_utils in /usr/local/lib/python3.10/dist-
packages (0.0.5)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from outlier_utils) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from outlier_utils) (1.11.3)
```

###“speechiness”

[179]:
```
grubbs.test(df["speechiness"], alpha=.05) #array sin valores atípicos
```

[179]:
```
0        0.0485
1        0.0406
2        0.0525
3        0.0555
4        0.2160
          …
17991    0.0413
17992    0.0329
17993    0.0712
17994    0.1340
17995    0.0591
Name: speechiness, Length: 17757, dtype: float64
```

###“liveness”

[180]:
```
grubbs.test(df["liveness"], alpha=.05)
```

[180]:
```
0        0.0849
1        0.1010
2        0.3940
3        0.1220
4        0.1720
          …
17991    0.0984
17992    0.0705
17993    0.6660
17994    0.2560
17995    0.3340
Name: liveness, Length: 17800, dtype: float64
```

###“duration_min”

[181]:
```
grubbs.test(df["duration_min"], alpha=.05)
```

[181]:
```
0        3.909933
1        4.195550
2        1.827783
3        2.899467
4        3.832667
           …
17991    3.224167
17992    4.284450
17993    3.603700
17994    3.661550
17995    3.037117
Name: duration_min, Length: 17852, dtype: float64
```

Una vez instalados los paquetes, se utiliza la función para regresar el arreglo sin los datos atípicos. Posteriormente se localizan la posición en la que se encuentran, por último, regresan los valores atípicos. En el caso de la variable "speechiness" son 0.995 y 0.937 que se encuentran en el 11358 y 1301 respectivamente. Por el otro lado, la variable "liveness" tiene dos datos atípicos que son 1.0 y 0.992, localizados en 13405 y 460 respectivamente. Finalmente la varible "duration_in min/ms" tiene 1793160.0 y 1767000.0 como valores atípicos ubicados en 14934 y 1219.

[182]:
```
df2.describe()
```

[182]:

| | new_Popularity | danceability | energy | mode | loudness | \ |
|---|---|---|---|---|---|---|
| count | 17996.000000 | 17996.000000 | 17996.000000 | 17996.000000 | 17996.000000 | |
| mean | 44.512124 | 0.543433 | 0.662777 | 0.636753 | -7.910660 | |
| std | 17.218436 | 0.166268 | 0.235373 | 0.480949 | 4.049151 | |
| min | 1.000000 | 0.059600 | 0.000020 | 0.000000 | -39.952000 | |
| 25% | 33.000000 | 0.432000 | 0.509000 | 0.000000 | -9.538000 | |
| 50% | 44.000000 | 0.545000 | 0.700000 | 1.000000 | -7.016000 | |
| 75% | 56.000000 | 0.659000 | 0.860000 | 1.000000 | -5.189000 | |
| max | 100.000000 | 0.989000 | 1.000000 | 1.000000 | 1.355000 | |

| | speechiness | acousticness | new_instrumentalness | liveness | \ |
|---|---|---|---|---|---|
| count | 17996.000000 | 17996.000000 | 17996.000000 | 17996.000000 | |
| mean | 0.079707 | 0.247082 | 0.177977 | 0.196170 | |
| std | 0.083576 | 0.310632 | 0.304586 | 0.159212 | |
| min | 0.022500 | 0.000000 | 0.000001 | 0.011900 | |
| 25% | 0.034800 | 0.004300 | 0.000089 | 0.097500 | |
| 50% | 0.047400 | 0.081400 | 0.003870 | 0.129000 | |
| 75% | 0.083000 | 0.434000 | 0.199000 | 0.258000 | |
| max | 0.955000 | 0.996000 | 0.996000 | 1.000000 | |

| | valence | tempo | duration_min |
|---|---|---|---|
| count | 17996.000000 | 17996.000000 | 17996.000000 |
| mean | 0.486208 | 122.623294 | 3.930388 |

```
std         0.240195     29.571527      1.427917
min         0.018300     30.557000      0.388667
25%         0.297000     99.620750      3.127496
50%         0.481000    120.065500      3.666667
75%         0.672000    141.969250      4.384704
max         0.986000    217.416000     29.886000
```

Radar Chart of Top 10 popular songs: https://www.data-to-viz.com/caveat/spider.html

[183]:
```
# parameters we will be using

top10 = pd.DataFrame(df.nlargest(10,'new_Popularity'))

top_10_song_names = [] # Create array for song names in Radar Chart
for name in (pd.merge(top10,df)['Track Name']):
  top_10_song_names.append(name)

Radar_Chart1 = go.Figure() #Creating Chart1
for i in range(0,10):
  Radar_Chart1.add_trace(go.Scatterpolar(
        r= top10.iloc[i,2:],
        theta=columnas_numericas.columns,
        fill='toself',
        name = top_10_song_names[i]
  ))
Radar_Chart1.show()

Radar_Chart2 = go.Figure() #Creating Chart2
for i in range(0,10):
  Radar_Chart2.add_trace(go.Scatterpolar(
        r= top10.iloc[i,[3,12]],
        theta=columnas_numericas.columns,
        fill='toself',
        name = top_10_song_names[i]
  ))
Radar_Chart2.show()
```

# 6 Fase 3

## 6.1 Análisis estadístico

En esta fase del análisis, se modelaran las variables de interés con distribuciones de probabilidad. Este proceso permitirá entender mejor las variables y proporcionará un marco para realizar predicciones y cálculos de probabilidad. Se revisarán los análisis anteriores para generar hipótesis sobre qué distribuciones de probabilidad podrían modelar adecuadamente las variables, además de estimar los parámetros de las distribuciones seleccionadas por medio del método de máxima verosimilitud. Todo esto para explorar la utilidad de los modelos de probabilidad en la comprensión y predicción

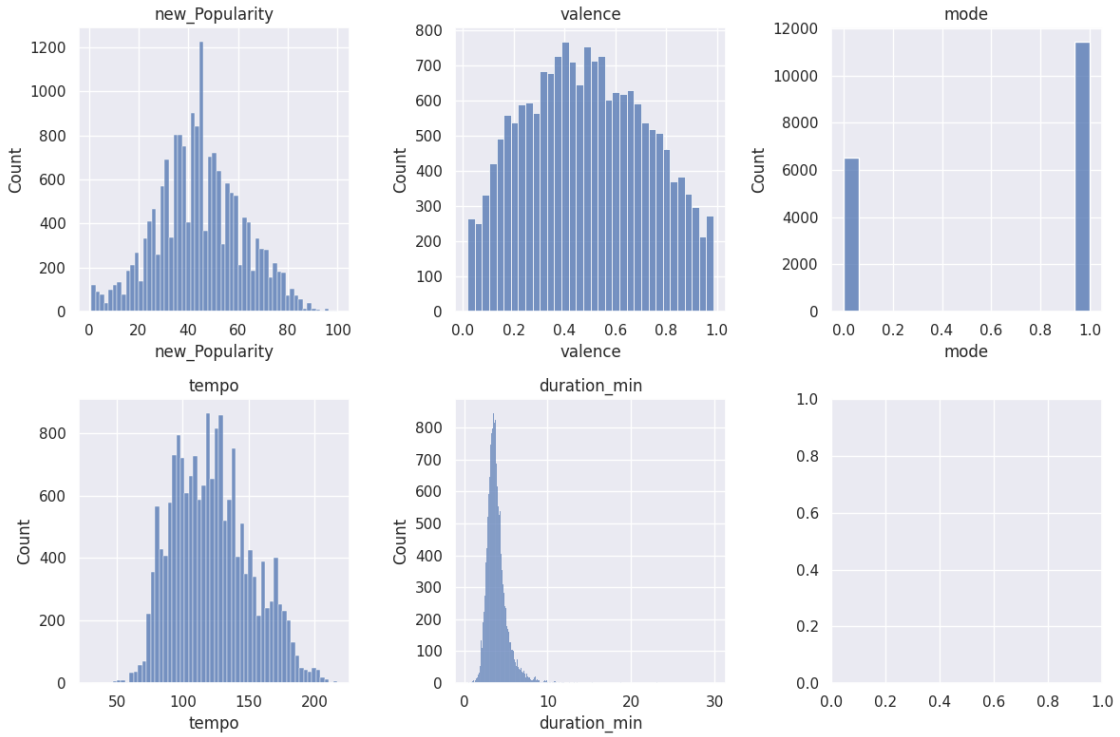
de las variables clave.

[184]:
```
df2.head()
```

[184]:
```
   new_Popularity  danceability  energy  mode  loudness  speechiness  \
0            60.0         0.854   0.564     1    -4.964       0.0485
1            54.0         0.382   0.814     1    -7.230       0.0406
2            35.0         0.434   0.614     1    -8.334       0.0525
3            66.0         0.853   0.597     0    -6.528       0.0555
4            53.0         0.167   0.975     1    -4.279       0.2160

   acousticness  new_instrumentalness  liveness  valence    tempo  \
0      0.017100              0.177562    0.0849   0.8990  134.071
1      0.001100              0.004010    0.1010   0.5690  116.454
2      0.486000              0.000196    0.3940   0.7870  147.681
3      0.021200              0.000196    0.1220   0.5690  107.033
4      0.000169              0.016100    0.1720   0.0918  199.060

   duration_min
0      3.909933
1      4.195550
2      1.827783
3      2.899467
4      3.832667
```

[185]:
```
#  Determina qué distribución de probabilidad sería adecuada para modelar la␣
 ↪popularidad, el modo, la valence, el tempo y la duración de las canciones.
fig, axs = plt.subplots(2, 3, figsize=(12, 8))

# Histograma 1
sns.histplot(data=df2, x="new_Popularity", ax=axs[0, 0])
axs[0, 0].set_title('new_Popularity')

# Histograma 2
sns.histplot(data=df2, x="valence", ax=axs[0, 1])
axs[0, 1].set_title('valence')

# Histograma 3
sns.histplot(data=df2, x="tempo", ax=axs[1, 0])
axs[1, 0].set_title('tempo')

# Histograma 4
sns.histplot(data=df2, x="duration_min", ax=axs[1, 1])
axs[1, 1].set_title('duration_min')

# Histograma 5
sns.histplot(data=df2, x="mode", ax=axs[0, 2])
axs[0, 2].set_title('mode')
```

```
plt.tight_layout()
plt.show()
```




Podemos observar que las variables como mode no siguen una distribución continua, por lo que no serán utilizada para el análisis

[186]:
```
!pip install distfit
from distfit import distfit
```

```
Requirement already satisfied: distfit in /usr/local/lib/python3.10/dist-
packages (1.7.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from distfit) (23.2)
Requirement already satisfied: matplotlib>=3.5.2 in
/usr/local/lib/python3.10/dist-packages (from distfit) (3.7.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from distfit) (1.23.5)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(from distfit) (1.5.3)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-
packages (from distfit) (0.14.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from distfit) (1.11.3)
```

```
Requirement already satisfied: pypickle in /usr/local/lib/python3.10/dist-
packages (from distfit) (1.1.0)
Requirement already satisfied: colourmap>=1.1.10 in
/usr/local/lib/python3.10/dist-packages (from distfit) (1.1.16)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages
(from distfit) (1.3.2)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.5.2->distfit)
(1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib>=3.5.2->distfit) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.5.2->distfit)
(4.44.3)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.5.2->distfit)
(1.4.5)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib>=3.5.2->distfit) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.5.2->distfit)
(3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.5.2->distfit)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->distfit) (2023.3.post1)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-
packages (from statsmodels->distfit) (0.5.3)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from patsy>=0.5.2->statsmodels->distfit) (1.16.0)
```

### 6.2 Método de Suma residual de cuadrados (RSS)

Se utiliza el método de suma residual de cuadrados (RSS) para determinar el tipo de distribución que mejor puede ajustarse a los datos.

El método de suma residual de cuadrados mide la disparidad entre los datos y un modelo de predicción. En este caso se comparan los datos de cada variable con una lista de posibles distribuciones usando el valor de RSS.

Al igual que en una regresión un valor pequeño, indica una relación estrecha al tipo de distribución.

[187]:
```python
# NEW POPULARITY
dist = distfit()

df_popularity = df2['new_Popularity']
dist.fit_transform(df_popularity)
print(dist.model)
```

```
[distfit] >INFO> fit
[distfit] >INFO> transform
[distfit] >INFO> [norm      ] [0.00 sec] [RSS: 0.000975352] [loc=44.512
scale=17.218]
[distfit] >INFO> [expon     ] [0.00 sec] [RSS: 0.00646717] [loc=1.000
scale=43.512]
[distfit] >INFO> [pareto    ] [0.05 sec] [RSS: 0.00646717] [loc=-8589934591.000
scale=8589934592.000]
[distfit] >INFO> [dweibull  ] [0.20 sec] [RSS: 0.00123336] [loc=43.490
scale=14.481]
[distfit] >INFO> [t         ] [2.34 sec] [RSS: 0.00097536] [loc=44.513
scale=17.218]
[distfit] >INFO> [genextreme] [1.32 sec] [RSS: 0.000978078] [loc=38.205
scale=16.999]
[distfit] >INFO> [gamma     ] [0.28 sec] [RSS: 0.000961013] [loc=-362.668
scale=0.728]
[distfit] >INFO> [lognorm   ] [0.02 sec] [RSS: 0.000960917] [loc=-569.051
scale=613.321]
[distfit] >INFO> [beta      ] [0.25 sec] [RSS: 0.000984547] [loc=-38.420
scale=182.077]
[distfit] >INFO> [uniform   ] [0.00 sec] [RSS: 0.00461376] [loc=1.000
scale=99.000]
[distfit] >INFO> [loggamma  ] [0.25 sec] [RSS: 0.000985294] [loc=-3226.396
scale=489.515]
```

```
{'name': 'lognorm', 'score': 0.0009609168439831747, 'loc': -569.0506838001525,
'scale': 613.3213434714281, 'arg': (0.028058195490132925,), 'params':
(0.028058195490132925, -569.0506838001525, 613.3213434714281), 'model':
<scipy.stats._distn_infrastructure.rv_continuous_frozen object at
0x7e2f056a7730>, 'bootstrap_score': 0, 'bootstrap_pass': None, 'color':
'#e41a1c', 'CII_min_alpha': 16.60812849625711, 'CII_max_alpha':
73.23978030579269}
```

De acuerdo al score obtenido, se muestra una relación estrecha de los datos con los parametros de una distribución log normal. Se puede asumir que los datos siguen una distribución log normal

[188]:
```
#MODE
df_mode = df2['mode']
dist.fit_transform(df_mode)
print(dist.model)
```

```
{'name': 'genextreme', 'score': 73.89540241926905, 'loc': 0.8038834884578729,
'scale': 0.3891482569675242, 'arg': (1.9842707475649375,), 'params':
(1.9842707475649375, 0.8038834884578729, 0.3891482569675242), 'model':
<scipy.stats._distn_infrastructure.rv_continuous_frozen object at
0x7e2f597b6e90>, 'bootstrap_score': 0, 'bootstrap_pass': None, 'color':
'#e41a1c', 'CII_min_alpha': -0.7299164161380245, 'CII_max_alpha':
0.9994593388940588}
```

La variable mode es una variable no continua, por lo que se obtuvo un score demasiado grande para asumir que la variable puede seguir una distribución "genextreme"

[189]:
```
#VALENCE
df_valence = df2['valence']
dist.fit_transform(df_valence)
print(dist.model)
```

```
{'name': 'beta', 'score': 0.33778723506375574, 'loc': 0.01819155931057474,
'scale': 0.9688430807295921, 'arg': (1.4509419060173188, 1.5400169734440536),
'params': (1.4509419060173188, 1.5400169734440536, 0.01819155931057474,
0.9688430807295921), 'model':
<scipy.stats._distn_infrastructure.rv_continuous_frozen object at
0x7e2f04b73400>, 'bootstrap_score': 0, 'bootstrap_pass': None, 'color':
'#e41a1c', 'CII_min_alpha': 0.10387134975814072, 'CII_max_alpha':
0.8845022817658035}
```

La variable valence muestra un score suficientemente bajo para decir que los datos pueden estar distribuidos como una distribución beta

[190]:
```
#TEMPO
df_tempo = df2['tempo']
dist.fit_transform(df_tempo)
print(dist.model)
```

```
{'name': 'beta', 'score': 0.00013404483815240552, 'loc': 28.91008544933102,
'scale': 360.5746210881764, 'arg': (7.27496001215548, 20.711944233576943),
'params': (7.27496001215548, 20.711944233576943, 28.91008544933102,
360.5746210881764), 'model':
<scipy.stats._distn_infrastructure.rv_continuous_frozen object at
0x7e2f0492d210>, 'bootstrap_score': 0, 'bootstrap_pass': None, 'color':
'#e41a1c', 'CII_min_alpha': 78.0402267525354, 'CII_max_alpha':
174.38185578768338}
```

La variable tempo sigue una distribución beta de acuerdo a la prueba RSS

[191]:
```
#duration_min
df_duration = df2['duration_min']
dist.fit_transform(df_duration)
print(dist.model)
```

```
{'name': 'genextreme', 'score': 0.07710354872856083, 'loc': 3.3549316032984517,
'scale': 0.9288974735259921, 'arg': (-0.04667568374141821,), 'params':
(-0.04667568374141821, 3.3549316032984517, 0.9288974735259921), 'model':
<scipy.stats._distn_infrastructure.rv_continuous_frozen object at
0x7e2f0ebea6b0>, 'bootstrap_score': 0, 'bootstrap_pass': None, 'color':
'#e41a1c', 'CII_min_alpha': 2.3614129758004054, 'CII_max_alpha':
6.314340144106264}
```

La variable tempo sigue una distribución genextreme de acuerdo a la prueba RSS

###Calcular parámetros de distribuciones Para probar si las variables siguen las distribuciones estimadas, podemos generar una función

Popularity (lognormal)

[192]:
```
param_popularity = st.lognorm.fit(df['new_Popularity'])
print(param_popularity)
```

```
(0.028058195490132925, -569.0506838001525, 613.3213434714281)
```

Mode (bernoulli)

[193]:
```
param_mode = df['mode'].sum()/df['mode'].count()
print(param_mode)
```

```
0.636752611691487
```

Valence (Beta)

[194]:
```
param_valence = st.beta.fit(df['valence'])
print(param_valence)
```

```
(1.4509419060173188, 1.5400169734440536, 0.01819155931057474,
0.9688430807295921)
```

Tempo (Beta)

[195]:
```
param_tempo = st.beta.fit(df['tempo'])
print(param_tempo)
```

```
(7.27496001215548, 20.711944233576943, 28.91008544933102, 360.5746210881764)
```

Duration (genextreme)

[196]:
```
param_duration = st.genextreme.fit(df['duration_min'])
print(param_duration)
```

```
(-0.04667568374141821, 3.3549316032984517, 0.9288974735259921)
```

###Prueba Kolmogorov-Smirnov A partir de los histrogramas, podemos suponer que todos los atributos anteriores tienen una distribución normal (menos 'mode'). Para confirmar esto, haremos una prueba de hipótesis usando la prueba de Kolmogorov-Smirnov. Esta prueba de hipotesis plantea la hipotesis nula de que la distribución standard normal y la distribución de los datos estandarizados son igual, la hipótesis alternativa al aplicar un test de dos colas es que las distribuciones son diferentes. Aplicaremos este test para los 5 atributos planteando que las 5 son normales, esto con una significancia de 95% empezaremos creando nuevas variables con nuestros atributos estandarizados

Popularity

El p-value<0.05, por lo tanto rechazamos la hipótesis nula, lo que significa que no hay suficiente evidencia para asumir que los datos siguen una distribucion normal

[197]:
```
st.kstest(df['new_Popularity'], st.norm.cdf)
```

[197]:
```
KstestResult(statistic=0.9919819534909304, pvalue=0.0, statistic_location=3.0,
statistic_sign=-1)
```

Valence

[198]:
```
st.kstest(df['valence'], st.norm.cdf)
```

[198]:
```
KstestResult(statistic=0.5120657712541103, pvalue=0.0,
statistic_location=0.0322, statistic_sign=-1)
```

El p-value<0.05, por lo tanto rechazamos la hipótesis nula, lo que significa que no hay suficiente evidencia para asumir que los datos siguen una distribucion normal

Tempo

[199]:
```
st.kstest(df['tempo'], st.norm.cdf)
```

[199]:
```
KstestResult(statistic=1.0, pvalue=0.0, statistic_location=30.557,
statistic_sign=-1)
```

El p-value<0.05, por lo tanto rechazamos la hipótesis nula, lo que significa que no hay suficiente evidencia para asumir que los datos siguen una distribucion normal

Duration

[200]:
```
st.kstest(df['duration_min'], st.norm.cdf, alternative='less')
```

[200]:
```
KstestResult(statistic=0.9647603177702158, pvalue=0.0,
statistic_location=2.0191333333333334, statistic_sign=-1)
```

El p-value<0.05, por lo tanto rechazamos la hipótesis nula, lo que significa que no hay suficiente evidencia para asumir que los datos siguen una distribucion normal

### 6.3 Intervalos de Confianza Método Bootstrap

Intervalo de confianza de mediana con método boostrap con percentil para

[201]:
```
from scipy.stats import bootstrap
rng = np.random.default_rng()

print('CI para media de popularidad (95% percentil)=␣
 ↪',bootstrap((df2['new_Popularity'].values,),np.mean , confidence_level=0.95,␣
 ↪random_state=rng, method = 'percentile').confidence_interval)
print('CI para media de valencia (95% percentil)= ',bootstrap((df2['valence'].
 ↪values,),np.mean , confidence_level=0.95, random_state=rng, method =␣
 ↪'percentile').confidence_interval)
```

```
print('CI para media de tempo (95% percentil)= ', bootstrap((df2['tempo'].
  ↪values,),np.mean , confidence_level=0.95, random_state=rng, method =␣
  ↪'percentile').confidence_interval)
print('CI para media de duración (95% percentil)= ',␣
  ↪bootstrap((df2['duration_min'].values,),np.mean , confidence_level=0.95,␣
  ↪random_state=rng, method = 'percentile').confidence_interval)
```

```
CI para media de popularidad (95% percentil)=
ConfidenceInterval(low=44.25704092361675, high=44.76109600896434)
CI para media de valencia (95% percentil)=
ConfidenceInterval(low=0.48279163980884643, high=0.48977172482773945)
CI para media de tempo (95% percentil)=
ConfidenceInterval(low=122.19954685207824, high=123.06717163536341)
CI para media de duración (95% percentil)=
ConfidenceInterval(low=3.9095042545050007, high=3.9515965700730975)
```

Intervalo de confianza con Bias-corrected an accelerated bootstrap para media, mediana y desviación estandard

[202]:
```
#POPULARITY
print(bootstrap((df2['new_Popularity'].values,),np.mean , confidence_level=0.
  ↪95, random_state=rng, method = 'BCa').confidence_interval)
#print(bootstrap((df2['new_Popularity'].values,),np.median , confidence_level=0.
  ↪95, random_state=rng).confidence_interval)
print(bootstrap((df2['new_Popularity'].values,),np.std , confidence_level=0.95,␣
  ↪random_state=rng, method = 'BCa').confidence_interval)
```

```
ConfidenceInterval(low=44.269763875708016, high=44.77677602378677)
ConfidenceInterval(low=17.04055014242953, high=17.387815278433624)
```

La distribución para la variable popularidad está degenerada, lo que significa que está distribuida en un solo punto, la media. No existe variabilidad, por lo que no es posible calcular un intervalo para la mediana

[203]:
```
#VALENCE
print(bootstrap((df2['valence'].values,),np.mean , confidence_level=0.95,␣
  ↪random_state=rng, method = 'BCa').confidence_interval)
print(bootstrap((df2['valence'].values,),np.median , confidence_level=0.95,␣
  ↪random_state=rng, method = 'BCa').confidence_interval)
print(bootstrap((df2['valence'].values,),np.std , confidence_level=0.95,␣
  ↪random_state=rng, method = 'BCa').confidence_interval)
```

```
ConfidenceInterval(low=0.48268349698429475, high=0.48969962443633547)
ConfidenceInterval(low=0.476, high=0.487)
ConfidenceInterval(low=0.23836824344570096, high=0.24201316563516603)
```

[204]:
```
#TEMPO
print(bootstrap((df2['tempo'].values,),np.mean , confidence_level=0.95,␣
  ↪random_state=rng, method = 'BCa').confidence_interval)
```

```
print(bootstrap((df2['tempo'].values,),np.median , confidence_level=0.95,␣
 ↪random_state=rng, method = 'BCa').confidence_interval)
print(bootstrap((df2['tempo'].values,),np.std , confidence_level=0.95,␣
 ↪random_state=rng, method = 'BCa').confidence_interval)
```

```
ConfidenceInterval(low=122.17437385336714, high=123.04068584611646)
ConfidenceInterval(low=120.007, high=120.9575)
ConfidenceInterval(low=29.29901116547916, high=29.8355125040195)
```

[205]:
```
#DURATION_MIN
print(bootstrap((df2['duration_min'].values,),np.mean , confidence_level=0.95,␣
 ↪random_state=rng, method = 'BCa').confidence_interval)
print(bootstrap((df2['duration_min'].values,),np.median , confidence_level=0.
 ↪95, random_state=rng, method = 'BCa').confidence_interval)
print(bootstrap((df2['duration_min'].values,),np.std , confidence_level=0.95,␣
 ↪random_state=rng, method = 'BCa').confidence_interval)
```

```
ConfidenceInterval(low=3.9100009556088255, high=3.9519883963634657)
ConfidenceInterval(low=3.654, high=3.6808831056286913)
ConfidenceInterval(low=1.3693343319556324, high=1.5066262074432857)
```

##Método de Shapiro Wilk

Verificar si los datos de popularidad siguen una distribución normal.

[206]:
```
# Sabemos que el método de Shapiro Wilk es sensible cuando n > 5000, por lo que␣
 ↪veremos cuantos datos tenemos.
df['new_Popularity'].count()
```

[206]:
```
17996
```

Desde aqui vemos que el tamaño de la base de datos es muy grande para decir con certeza que la prueba de Shapiro no es precisa. Al realizar la prueba de Shapiro Wilks el p-valor es menor a <0.050 lo que indica que la variable “new popularity” no sigue una distribucion normal

[207]:
```
# La función usa 95% de confianza por default.
popularity_normal_dist = st.shapiro(df2['new_Popularity'])
print(popularity_normal_dist.statistic)
print(popularity_normal_dist.pvalue) # retorna un valor entre 0 y 1; entre más␣
 ↪cercano a 1 mejor proximación de una distribución normal sigue.
# Observamos que el valor de la prueba fue de 0.9966, por lo que con un 95%␣
 ↪confianza decimos que popularidad con colaboraciones sigue una distribución␣
 ↪normal.
```

```
0.9966110587120056
8.542267690206121e-20
```

[208]:
```
# Mismo caso, para canciones sin colaboraciones.
print(st.shapiro(df.loc[df['collab'] == False, 'new_Popularity']).statistic)
print(st.shapiro(df.loc[df['collab'] == False, 'new_Popularity']).pvalue) # Se␣
 ↪usa df pues df2 no tiene atributo collab.
# Observamos que con un intervalo de confianza del 95%, las canciones sin␣
 ↪colaboración siguen una distribución normal en cuanto a popularidad.
```

```
0.9965856671333313
3.3469436228848305e-19
```

### 6.4 Verifica si la popularidad de las canciones con y sin colaboración sigue una distribución normal.

Para verificar normalidad en los datos de la popularidad de las canciones con y sin colaboraciónse realizaron pruebas de normalidad en minitab por metodo de Anderson Darling. Abordamos la posibilidad que la estandarización de los datos afectara el resultado de la prueba de hipotesis, es por ello que se realizaron ambas pruebas con y sin estandarización. A continuacion se muestra la grafica de los datos estandarizados.

Se concluyo que en ningnuna de las pruebas la populariad de las canciones con y sin colaboración sigue una distribución normal ya que el P-valor en ambas es <0.010 lo que indicica que rechaza hipotesis nula sugiriendo como alternativa que los datos no siguen distribuciones normales.

### 6.5 Prueba si tener o no una colaboración hace una diferencia para que una canción sea más o menos popular.

Para evaluar si la presencia de una colaboración hace una diferencia en la popularidad de una canción mediante un test t de dos muestras con prueba de varianzas, se seguirían estos pasos:

Nuestro P-valor de la prueba de varianzas muestra que las varianzas no son iguales y se asumiran como diferentes para la siguiente prueba de medias:

La prueba t de dos muestras verifica si una de nuestras muestras es mayor o no a la otra. Con un valor critico de 7.07 para t y un P-valor <0.05 hay suficiente evidencia estadisitca para rechazar hipotesis nula y sugerir que una cancion con colaboración tiene mayor en popularidad.

### 6.6 Utiliza un ANOVA para determinar si existen diferencias significativas en la popularidad promedio de canciones de diferentes géneros musicales.

Se uso ANOVA para determinar la popularidad promedio entre generos es diferente. La Anova tendra 11 nivfeles ya que la base de datos tiene 11 difernectes generos.

Los resultados del analisis concluyen que por lo menos una media es diferente. Al observar los supuestos los datos no cumplen ninguno, ni con normalidad, ni homocedesticidad, ni linealidad. Para futuros analisis ya que no se cumplen los supuestos la prueba de Kruskal–Wallis seria lo proximo para continuar.

##Determinar si el tener una colaboración en la canción hace una diferencia en cuanto a la popularidad.

El primer enfoque que le daremos a la comparación, será comparar promedios de canciones con y sin colaboración.

[209]:
```
colab_mean = df.loc[df['collab']==True, 'new_Popularity'].mean()
wo_colab_mean = df.loc[df['collab']==False, 'new_Popularity'].mean()

print(f'popularity without colab: {wo_colab_mean}\n popularity with colab:␣
 ↪{colab_mean}')
```

```
popularity without colab: 44.22741697081835
 popularity with colab: 48.48997387664457
```

El segundo enfoque que daremos, será un análisis de cuartiles.

[210]:
```
colab_Quartiles = np.percentile(df.loc[df['collab']==True, 'new_Popularity'],␣
 ↪[25, 50, 75])
wo_colab_Quartiles = np.percentile(df.loc[df['collab']==False,␣
 ↪'new_Popularity'], [25, 50, 75])
for i in range(len(colab_Quartiles)):
  print(f'El cuartil {i*25+25} con colaboración es: {colab_Quartiles[i]}\n El␣
 ↪cuartil {i*25+25} sin colaboración es: {wo_colab_Quartiles[i]}')
```

```
El cuartil 25 con colaboración es: 34.0
 El cuartil 25 sin colaboración es: 33.0
El cuartil 50 con colaboración es: 48.0
 El cuartil 50 sin colaboración es: 44.0
```

```
El cuartil 75 con colaboración es: 63.0
 El cuartil 75 sin colaboración es: 55.0
```

Viendo los resultados de ambos análisis, vemos que en el análisis de media las canciones con colaboración son más populares en promedio. Posteriormente, en el análisis de cuartiles, vemos que las diferencias entre los cuartiles 50 (mediana) y 75 son considerables, por lo que se concluye que sí hay diferencia entre una canción con colaboración y una sin colaboración en cuanto a popularidad.

[211]:
```
df2.to_csv('df2_exported.csv')
df.to_csv('df_exported.csv')
```

##Conclusiones

La exploración de una base de datos de canciones en Spotify reveló varios *insights*, como lo fueron las correlaciones entre atributos de la base de datos, distribucion de los datos al comparar la popularidad según el género entre otras. Tras un preprocesamiento que incluyó imputacion, se llevó a cabo un análisis para determinar la distribución de estas. Se descartó la posibilidad de que alguna variable siguiera una distribución normal y se estimó que dos podrían seguir una distribución beta y una lognormal. Se tomó popularidad como variable de respuesta en el analisis de varianza para evaluar diferencias entre géneros.