Libraries

In [1]:
```
import os

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

import sklearn
from sklearn.preprocessing import MinMaxScaler

from scipy import stats

import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

Called Dataset

In [2]:
```
# Get the current working directory
directorio_actual = os.getcwd()

# Specify the relative path from the current directory
ruta_csv_relativa = os.path.join('..', 'data', '01_raw','spotify.csv')

# Load the CSV file
spotify = pd.read_csv(ruta_csv_relativa)

# Show the first rows of the DataFrame
spotify.head(10)
```

Out[2]:

	Unnamed: 0	track_id	artists	album_name	track_name	popularity	duration_ms	explicit	danceabi
0	0	5SuOikwiRyPMVoIQDJUgSV	Gen Hoshino	Comedy	Comedy	73	230666	False	0.(
1	1	4qPNDBW1i3p13qLCt0Ki3A	Ben Woodward	Ghost (Acoustic)	Ghost - Acoustic	55	149610	False	0.4
2	2	1iJBSr7s7jYXzM8EGcbK5b	Ingrid Michaelson;ZAYN	To Begin Again	To Begin Again	57	210826	False	0.4
3	3	6lfxq3CG4xtTiEg7opyCyx	Kina Grannis	Crazy Rich Asians (Original Motion Picture Sou...	Can't Help Falling In Love	71	201933	False	0.2
4	4	5vjLSffimiIP26QG5WcN2K	Chord Overstreet	Hold On	Hold On	82	198853	False	0.(
5	5	01MVOl9KtVTNfFiBU9I7dc	Tyrone Wells	Days I Will Remember	Days I Will Remember	58	214240	False	0.(
6	6	6Vc5wAMmXdKIAM7WUoEb7N	A Great Big World;Christina Aguilera	Is There Anybody Out There?	Say Something	74	229400	False	0.4
7	7	1EzrEOXmMH3G43AXT1y7pA	Jason Mraz	We Sing. We Dance. We Steal Things.	I'm Yours	80	242946	False	0.7
8	8	0IktbUcnAGrvD03AWnz3Q8	Jason Mraz;Colbie Caillat	We Sing. We Dance. We Steal Things.	Lucky	74	189613	False	0.(
9	9	7k9GuJYLp2AzqokyEdwEw2	Ross Copperman	Hunger	Hunger	56	205594	False	0.4

10 rows × 21 columns

Initial Data Information

In [3]:
```
spotify.head(10)
```

Out[3]:

	Unnamed: 0	track_id	artists	album_name	track_name	popularity	duration_ms	explicit	danceabi
0	0	5SuOikwiRyPMVoIQDJUgSV	Gen Hoshino	Comedy	Comedy	73	230666	False	0.(
1	1	4qPNDBW1i3p13qLCt0Ki3A	Ben Woodward	Ghost (Acoustic)	Ghost - Acoustic	55	149610	False	0.4
2	2	1iJBSr7s7jYXzM8EGcbK5b	Ingrid Michaelson;ZAYN	To Begin Again	To Begin Again	57	210826	False	0.4
3	3	6lfxq3CG4xtTiEg7opyCyx	Kina Grannis	Crazy Rich Asians (Original Motion Picture Sou...	Can't Help Falling In Love	71	201933	False	0.2
4	4	5vjLSffimiIP26QG5WcN2K	Chord Overstreet	Hold On	Hold On	82	198853	False	0.(
5	5	01MVOl9KtVTNfFiBU9I7dc	Tyrone Wells	Days I Will Remember	Days I Will Remember	58	214240	False	0.(
6	6	6Vc5wAMmXdKIAM7WUoEb7N	A Great Big World;Christina Aguilera	Is There Anybody Out There?	Say Something	74	229400	False	0.4
7	7	1EzrEOXmMH3G43AXT1y7pA	Jason Mraz	We Sing. We Dance. We Steal Things.	I'm Yours	80	242946	False	0.7
8	8	0IktbUcnAGrvD03AWnz3Q8	Jason Mraz;Colbie Caillat	We Sing. We Dance. We Steal Things.	Lucky	74	189613	False	0.(
9	9	7k9GuJYLp2AzqokyEdwEw2	Ross Copperman	Hunger	Hunger	56	205594	False	0.4

10 rows × 21 columns

Dataset information

In [4]:
```
spotify.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114000 entries, 0 to 113999
Data columns (total 21 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   Unnamed: 0        114000 non-null  int64  
 1   track_id          114000 non-null  object 
 2   artists           113999 non-null  object 
 3   album_name        113999 non-null  object 
 4   track_name        113999 non-null  object 
 5   popularity        114000 non-null  int64  
 6   duration_ms       114000 non-null  int64  
 7   explicit          114000 non-null  bool   
 8   danceability      114000 non-null  float64
 9   energy            114000 non-null  float64
 10  key               114000 non-null  int64  
 11  loudness          114000 non-null  float64
 12  mode              114000 non-null  int64  
 13  speechiness       114000 non-null  float64
 14  acousticness      114000 non-null  float64
 15  instrumentalness  114000 non-null  float64
 16  liveness          114000 non-null  float64
 17  valence           114000 non-null  float64
 18  tempo             114000 non-null  float64
 19  time_signature    114000 non-null  int64  
 20  track_genre       114000 non-null  object 
dtypes: bool(1), float64(9), int64(6), object(5)
memory usage: 17.5+ MB
```

Descriptive statistics of the DataSet

In [5]:
```
spotify.describe()
```

Out[5]:

	Unnamed: 0	popularity	duration_ms	danceability	energy	key	loudness	mode
count	114000.000000	114000.000000	1.140000e+05	114000.000000	114000.000000	114000.000000	114000.000000	114000.000000
mean	56999.500000	33.238535	2.280292e+05	0.566800	0.641383	5.309140	-8.258960	0.637553
std	32909.109681	22.305078	1.072977e+05	0.173542	0.251529	3.559987	5.029337	0.480709
min	0.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000	-49.531000	0.000000
25%	28499.750000	17.000000	1.740660e+05	0.456000	0.472000	2.000000	-10.013000	0.000000
50%	56999.500000	35.000000	2.129060e+05	0.580000	0.685000	5.000000	-7.004000	1.000000
75%	85499.250000	50.000000	2.615060e+05	0.695000	0.854000	8.000000	-5.003000	1.000000
max	113999.000000	100.000000	5.237295e+06	0.985000	1.000000	11.000000	4.532000	1.000000

Total rows and columns

In [6]:
```
print ('Spotify data contains a total of ' + str(spotify.shape[0]) + ' rows and ' + str(spotify.shape[1]) + ' co
```

```
Spotify data contains a total of 114000 rows and 21 columns
```

In [7]:
```
print ('Of which Spotify data, the total number of songs is ' + str(spotify['track_id'].nunique()) + ' and the '
```

```
Of which Spotify data, the total number of songs is 89741 and the total number of genres is 114
```

DataSet Size

In [8]:
```
spotify.shape
```

Out[8]:
```
(114000, 21)
```

DataSet Columns

In [9]:
```
spotify.columns
```

Out[9]:
```
Index(['Unnamed: 0', 'track_id', 'artists', 'album_name', 'track_name',
       'popularity', 'duration_ms', 'explicit', 'danceability', 'energy',
       'key', 'loudness', 'mode', 'speechiness', 'acousticness',
       'instrumentalness', 'liveness', 'valence', 'tempo', 'time_signature',
       'track_genre'],
      dtype='object')
```

Column data type

In [10]:
```
spotify.dtypes
```

Out[10]:
```
Unnamed: 0            int64
track_id             object
artists              object
album_name           object
track_name           object
popularity            int64
duration_ms           int64
explicit               bool
danceability        float64
energy              float64
key                   int64
loudness            float64
mode                  int64
speechiness         float64
acousticness        float64
instrumentalness    float64
liveness            float64
valence             float64
tempo               float64
time_signature        int64
track_genre          object
dtype: object
```

Total amount of data

In [11]:
```
spotify.dtypes.value_counts()
```

Out[11]:
```
float64    9
int64      6
object     5
bool       1
Name: count, dtype: int64
```

Null values

In [12]:
```
missing_values = spotify.isnull().sum()

print(missing_values[missing_values > 0])
```

```
artists       1
album_name    1
track_name    1
dtype: int64
```

Review row with null data

In [13]:
```
spotify[spotify.isnull().any(axis=1)]
```

Out[13]:

	Unnamed: 0	track_id	artists	album_name	track_name	popularity	duration_ms	explicit	danceability	energ
65900	65900	1kR4gIb7nGxHPI3D2ifs59	NaN	NaN	NaN	0	0	False	0.501	0.58

1 rows × 21 columns

- Since this is a column where the data is not as important as "artist" - "album_name" - "track_name" - "popularity" - "duration_ms" it was decided to keep this data and impute the popularity and duration_ms with the average of the columns.

Impute data with the mean

In [14]:
```
spotify['popularity'] = spotify['popularity'].replace(0, spotify['popularity'].mean())
spotify['duration_ms'] = spotify['duration_ms'].replace(0, spotify['duration_ms'].mean())

#Verification
spotify[spotify['popularity'] == 0]
spotify[spotify['duration_ms'] == 0]
```

Out[14]:

Unnamed: 0	track_id	artists	album_name	track_name	popularity	duration_ms	explicit	danceability	energy	...	loudness	moc

0 rows × 21 columns

Duplicate Values

Reviewing duplicate values

In [15]:
```
spotify.duplicated().value_counts()
```

Out[15]:
```
False    114000
Name: count, dtype: int64
```

Histogramas

In [16]:
```
# Create subplots with 4 rows and 4 columns to accommodate all variables
fig = make_subplots(rows=4, cols=4,
                    subplot_titles=('<i>popularity', '<i>duration_ms', '<i>danceability',
                                    '<i>energy', '<i>key', '<i>loudness',
                                    '<i>mode', '<i>speechiness', '<i>acousticness',
                                    '<i>instrumentalness', '<i>liveness', '<i>valence',
                                    '<i>tempo', '<i>time_signature'))

# Add histogram traces to each subplot
fig.add_trace(go.Histogram(x=spotify['popularity'], name='popularity'), row=1, col=1)
fig.add_trace(go.Histogram(x=spotify['duration_ms'], name='duration_ms'), row=1, col=2)
fig.add_trace(go.Histogram(x=spotify['danceability'], name='danceability'), row=1, col=3)
fig.add_trace(go.Histogram(x=spotify['energy'], name='energy'), row=1, col=4)
fig.add_trace(go.Histogram(x=spotify['key'], name='key'), row=2, col=1)
fig.add_trace(go.Histogram(x=spotify['loudness'], name='loudness'), row=2, col=2)
fig.add_trace(go.Histogram(x=spotify['mode'], name='mode'), row=2, col=3)
```

```
fig.add_trace(go.Histogram(x=spotify['speechiness'], name='speechiness'), row=2, col=4)
fig.add_trace(go.Histogram(x=spotify['acousticness'], name='acousticness'), row=3, col=1)
fig.add_trace(go.Histogram(x=spotify['instrumentalness'], name='instrumentalness'), row=3, col=2)
fig.add_trace(go.Histogram(x=spotify['liveness'], name='liveness'), row=3, col=3)
fig.add_trace(go.Histogram(x=spotify['valence'], name='valence'), row=3, col=4)
fig.add_trace(go.Histogram(x=spotify['tempo'], name='tempo'), row=4, col=1)
fig.add_trace(go.Histogram(x=spotify['time_signature'], name='time_signature'), row=4, col=2)

# Update layout
fig.update_layout(height=1000, width=1000, title_text='<b>Distribución de caracteristicas', template='plotly_da

# Show the graph
fig.show()
```

The histogram shown serves to display the distribution of the most valuable fields in the Dataset, where the following conclusions have been drawn:

1. popularity

- Has a skewed distribution to the left, indicating that most songs in the Dataset have low popularity.

2. duration_ms

- Has a skewed distribution to the left, indicating that most songs have a shorter duration.

3. Danceability

- Has a normal (bell-shaped) distribution with most songs, suggesting that most songs are danceable.

4. Energy

- Has an asymmetric distribution to the right, indicating that most songs have high energy values and therefore fewer songs with low energy.

5. Key

- Has a uniform distribution with several peaks at different values, suggesting that there is no predominant key in the songs in the Dataset.

6. Loudness

- Has a high distribution to the right, indicating that most songs are loud in terms of volume.

7. Mode

- Most songs in this dataset have a mode value of 1, indicating that most songs are in major mode, which generally conveys a more upbeat or positive feeling.

8. Speechiness

- Has a high distribution towards the left side, indicating that most songs have no spoken content.

9. Acousticness

- This distribution similar to the above would indicate that most songs have low acoustic levels (i.e. digitally produced).

10. Instrumentalness

- Has a distribution skewed heavily towards the left side, indicating that most songs have a low level of instrumentality. Many songs contain a vocal or are vocal-centric.

11. Liveness

- This distribution with high values on the left side suggests that most songs have little live sound presence.

12. Valence

- This distribution is fairly even, indicating that the sample of emotions spans a wide range conveyed in the songs.

13. Tempo

- This distribution, which is notable for having several peaks (mostly between 120-130 BPM), would indicate that the common range between popular genres such as pop or rock falls within this tempo.

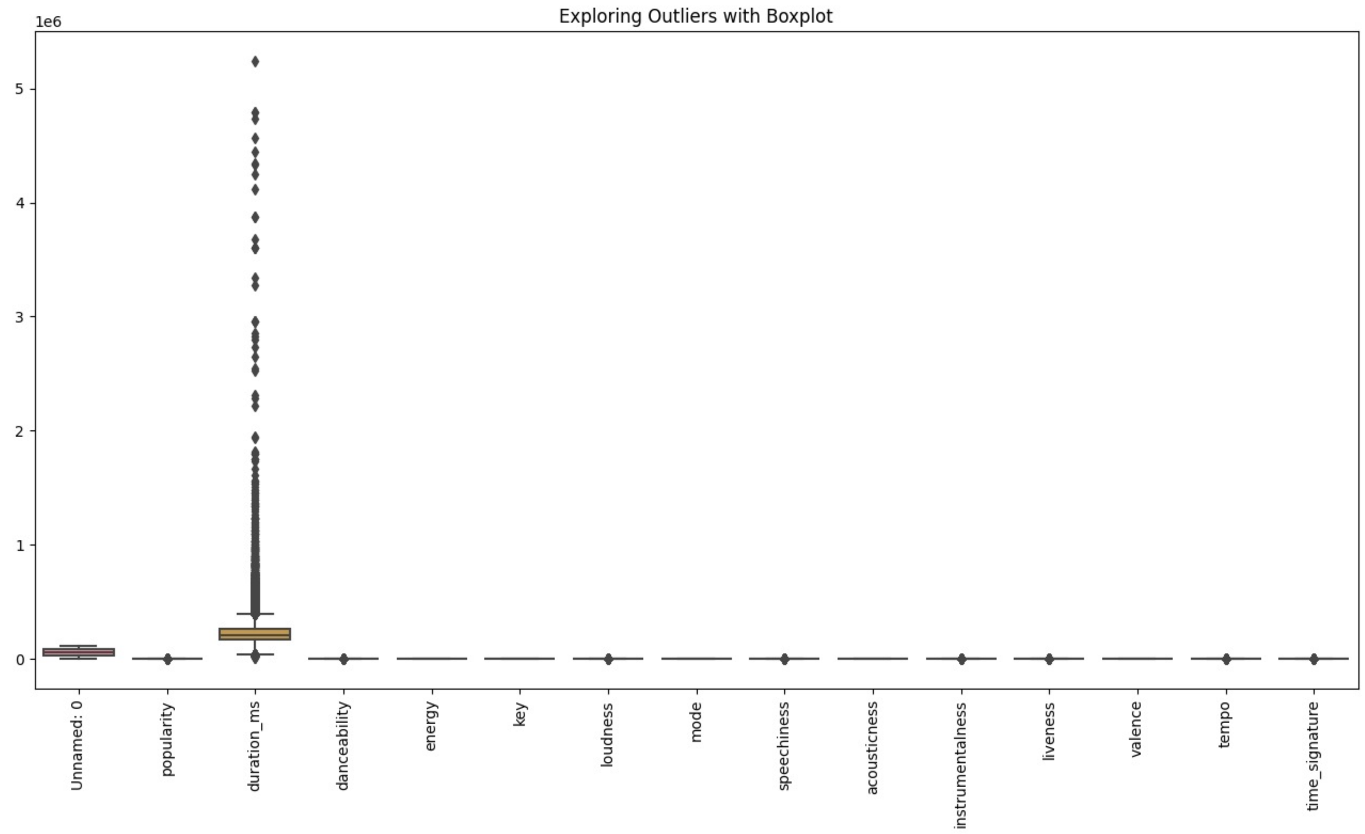
14. time_signature

- Most songs have a 4-beat time signature, which is common in popular music. This indicates that most songs follow standard rhythmic structures.

Outliers

Boxplots exploration

In [17]:
```
plt.figure(figsize=(16, 8))
sns.boxplot(data=spotify.select_dtypes(include=['float64', 'int64']))
plt.title("Exploring Outliers with Boxplot")
plt.xticks(rotation=90)
plt.show()
```



Outliers Interquartile Range (IQR)

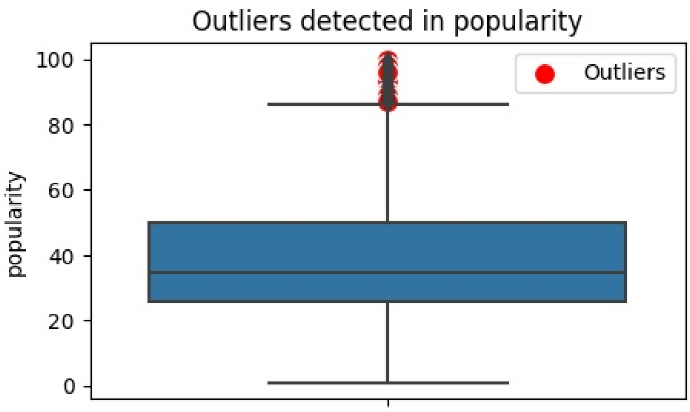
In [18]:
```
def aplicar_IQR(df, columnas):
    # Dictionary to store the identified outliers
    outliers = {}

    for column in columnas:
        Q1 = df[column].quantile(0.25)  # First quartile (Q1)
        Q3 = df[column].quantile(0.75)  # Third quartile (Q3)
        IQR = Q3 - Q1  # Interquartile range (IQR)

        lower_bound = Q1 - 1.5 * IQR  # Lower limit
        upper_bound = Q3 + 1.5 * IQR  # Upper limit

        # Filter outliers
        outliers[column] = df[(df[column] < lower_bound) | (df[column] > upper_bound)][column]

    return outliers

# Variables with asymmetric distributions based on histograms
columnas_asimetricas = ['popularity', 'duration_ms', 'speechiness', 'acousticness', 'instrumentalness', 'livene

# Apply the IQR function to the identified columns
outliers_detectados = aplicar_IQR(spotify, columnas_asimetricas)

# Show detected outliers by column
for columna, outliers in outliers_detectados.items():
    print(f"Outliers en {columna}:")
    print(outliers)
    print("\n")
```

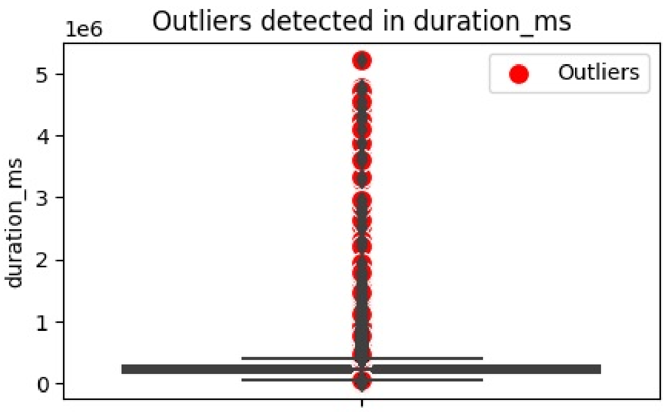
```
Outliers en popularity:
2000      87.0
2003      93.0
3000      87.0
3003      93.0
3300      87.0
          ... 
102018    90.0
103008    87.0
103154    88.0
104050    89.0
107001    87.0
Name: popularity, Length: 218, dtype: float64


Outliers en duration_ms:
253       447306.0
650       406103.0
752       445533.0
851       578064.0
896       403911.0
            ...   
113932    464398.0
113945    396646.0
113959    456981.0
113969    493293.0
113988    462397.0
Name: duration_ms, Length: 5616, dtype: float64


Outliers en speechiness:
370       0.236
692       0.204
713       0.197
768       0.403
815       0.189
          ...  
113549    0.409
113818    0.187
113923    0.160
113940    0.352
113981    0.162
Name: speechiness, Length: 13211, dtype: float64


Outliers en acousticness:
Series([], Name: acousticness, dtype: float64)


Outliers en instrumentalness:
56        0.168
62        0.833
72        0.215
77        0.266
116       0.183
          ...  
113979    0.958
113986    0.949
113990    0.924
113995    0.928
113996    0.976
Name: instrumentalness, Length: 25246, dtype: float64


Outliers en liveness:
51        0.669
73        0.660
343       0.799
518       0.940
535       0.610
          ...  
113967    0.588
113973    0.729
113979    0.696
113983    0.706
113989    0.662
Name: liveness, Length: 8642, dtype: float64
```

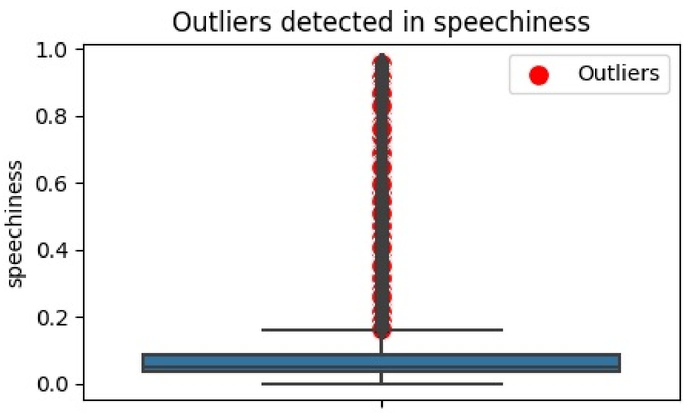
IQR Chart

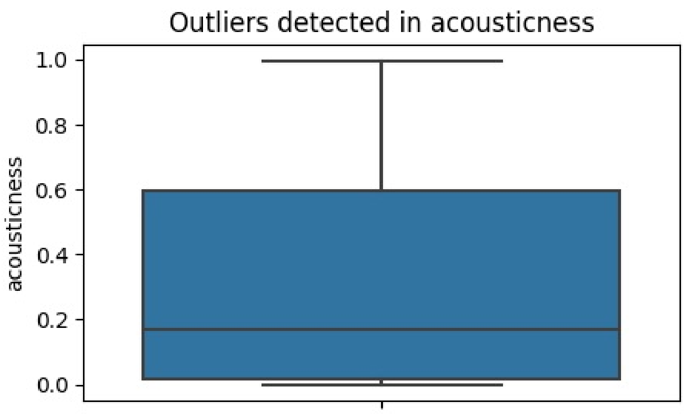
In [19]:

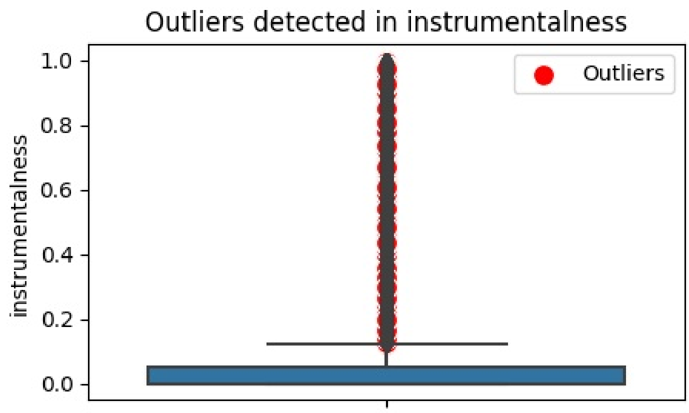
```python
# Iterate over each column with outliers
for column, outliers in outliers_detectados.items():
    plt.figure(figsize=(5, 3))

# Create a boxplot for the current column
    sns.boxplot(y=spotify[column])

# Plot the detected outliers in red
    sns.scatterplot(y=outliers, x=[0]*len(outliers), color='red', label='Outliers', s=100, marker='o')

    plt.title(f'Outliers detected in {column}')
    plt.show()
```

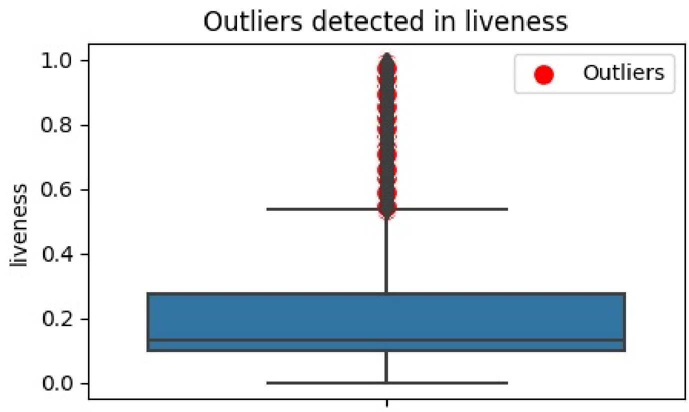












Conclusion IQR

1. popularity

- **Action**: Outliers will be kept.
- **Reason**: High popularity values are representative and useful for further analysis.

2. duration_ms

- **Action**: Logarithmic transformation.
- **Reason**: Values are very skewed within our analysis.

3. speechiness

- **Action**: Outliers will be kept.
- **Reason**: Values are important to differentiate between genres and styles of music.

4. acousticness

- **Action**: Keep values as they are.
- **Reason**: Distribution is relatively uniform and does not present a significant number of outliers that affect the analysis.

5. instrumentalness

- **Action**: Outliers will be kept.
- **Reason**: These values are representative for genres or styles of instrumental music, such as classical music or soundtracks.

6. liveness

- **Action**: Outliers will be retained.
- **Reason**: The presence of live sound is important in certain genres and performances, so it is relevant to maintain these values in the analysis.

Log transformation of popularity

In [20]:
```
fig = px.box(spotify, x='popularity')
fig.show()
```

Transformation

In [21]:
```
spotify['log_popularity'] = np.log1p(spotify['popularity'])

fig1 = px.box(spotify, x='log_popularity')
fig1.show()
```

Log transformation of duration_ms

In [22]:
```
fig = px.box(spotify, x='duration_ms')
fig.show()
```

Outliers duration_ms

For this, the main concepts provided by the diagram will be explained:

1. Median: The value that divides the data into 2 halves, representing the midpoint of the songs' duration which would be '212.906' which translated into minutes and seconds would be '3 minutes and 33 seconds'.

2. Quartiles:

- a) The lower border (Q1) shows that the duration of the songs in 25% of the dataset translated into minutes and seconds would be '2 minutes and 54 seconds'..

- b) The upper border (Q3) shows that the duration of the songs in 75% of the dataset translated into minutes and seconds would be '4 minutes and 21 seconds'.

3. Outliers: Would represent atypical values that are much longer than the average duration.

Transformation

In [23]:

```
spotify['log_duration_ms'] = np.log1p(spotify['duration_ms'])

fig1 = px.box(spotify, x='log_duration_ms')
fig1.show()
```

Z-score

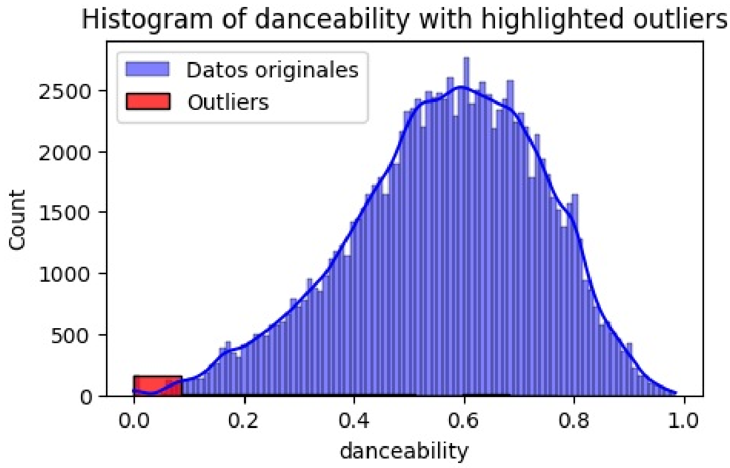
In [24]:
```
# Select variables with approximately normal distributions
variables_normales = ['danceability', 'energy', 'tempo']

# Calculate the Z-score for the selected variables
z_scores = np.abs(stats.zscore(spotify[variables_normales]))

# Filter rows that have at least one outlier (Z-score > 3)
outliers_zscore = spotify[(z_scores > 3).any(axis=1)]

# Show data containing outliers detected with Z-score
print("Outliers detected with Z-score:")
print(outliers_zscore[variables_normales])
```

```
Outliers detected with Z-score:
        danceability   energy    tempo
1087          0.4690  0.82400  220.081
1136          0.4690  0.82400  220.081
1144          0.4690  0.82400  220.081
2877          0.1660  0.96900  215.513
4097          0.0713  0.00696  213.848
...              ...      ...      ...
106161        0.3020  0.48100  216.334
111908        0.2410  0.28100  222.605
113428        0.0000  0.18800    0.000
113688        0.0000  0.00002    0.000
113856        0.0000  0.22400    0.000

[201 rows x 3 columns]
```

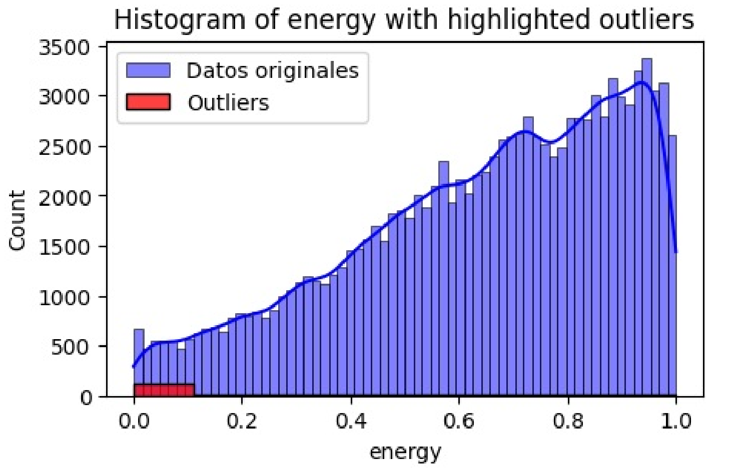
z-score graph

In [25]:
```
# Visualize the variables with histograms and highlight the outliers
for column in variables_normales:
    plt.figure(figsize=(5, 3))

    # Create the histogram of the original column
    sns.histplot(spotify[column], kde=True, color='blue', label='Datos originales')

    # Highlight outliers in red
    sns.histplot(outliers_zscore[column], kde=False, color='red', label='Outliers')

    plt.title(f"Histogram of {column} with highlighted outliers")
    plt.legend()
    plt.show()
```

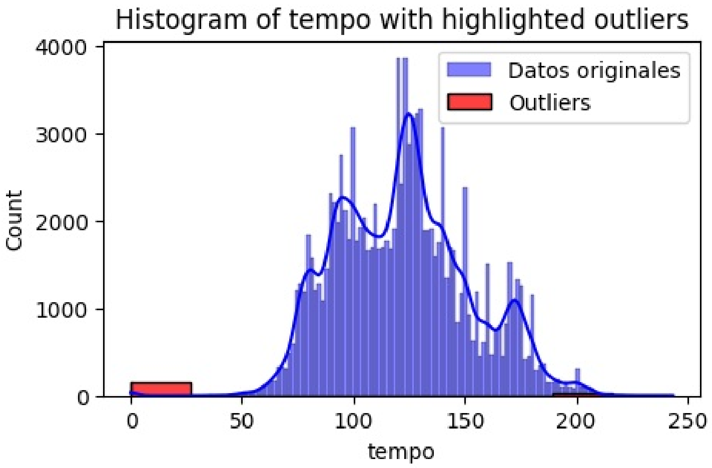
```
C:\Users\diego\OneDrive\Imágenes\Escritorio\Spotify-Recomendation-Machine-Learning\.venv\Lib\site-packages\seabo
rn\_oldcore.py:1119: FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before ope
rating instead.

C:\Users\diego\OneDrive\Imágenes\Escritorio\Spotify-Recomendation-Machine-Learning\.venv\Lib\site-packages\seabo
rn\_oldcore.py:1119: FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before ope
rating instead.
```




```
C:\Users\diego\OneDrive\Imágenes\Escritorio\Spotify-Recomendation-Machine-Learning\.venv\Lib\site-packages\seabo
rn\_oldcore.py:1119: FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before ope
rating instead.

C:\Users\diego\OneDrive\Imágenes\Escritorio\Spotify-Recomendation-Machine-Learning\.venv\Lib\site-packages\seabo
rn\_oldcore.py:1119: FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before ope
rating instead.
```




```
C:\Users\diego\OneDrive\Imágenes\Escritorio\Spotify-Recomendation-Machine-Learning\.venv\Lib\site-packages\seabo
rn\_oldcore.py:1119: FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before ope
rating instead.

C:\Users\diego\OneDrive\Imágenes\Escritorio\Spotify-Recomendation-Machine-Learning\.venv\Lib\site-packages\seabo
rn\_oldcore.py:1119: FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before ope
rating instead.
```



z-score Conclusión

1. danceability

- **Action**: Outliers will be kept.
- **Reason**: Values close to 0 could represent music genres or specific songs that are not danceable. Since there are not many, they are kept.

2. energy

- **Action**: Outliers will be kept.
- **Reason**: Low energy values could be associated with softer or slower songs, which are important in certain genres.

3. tempo

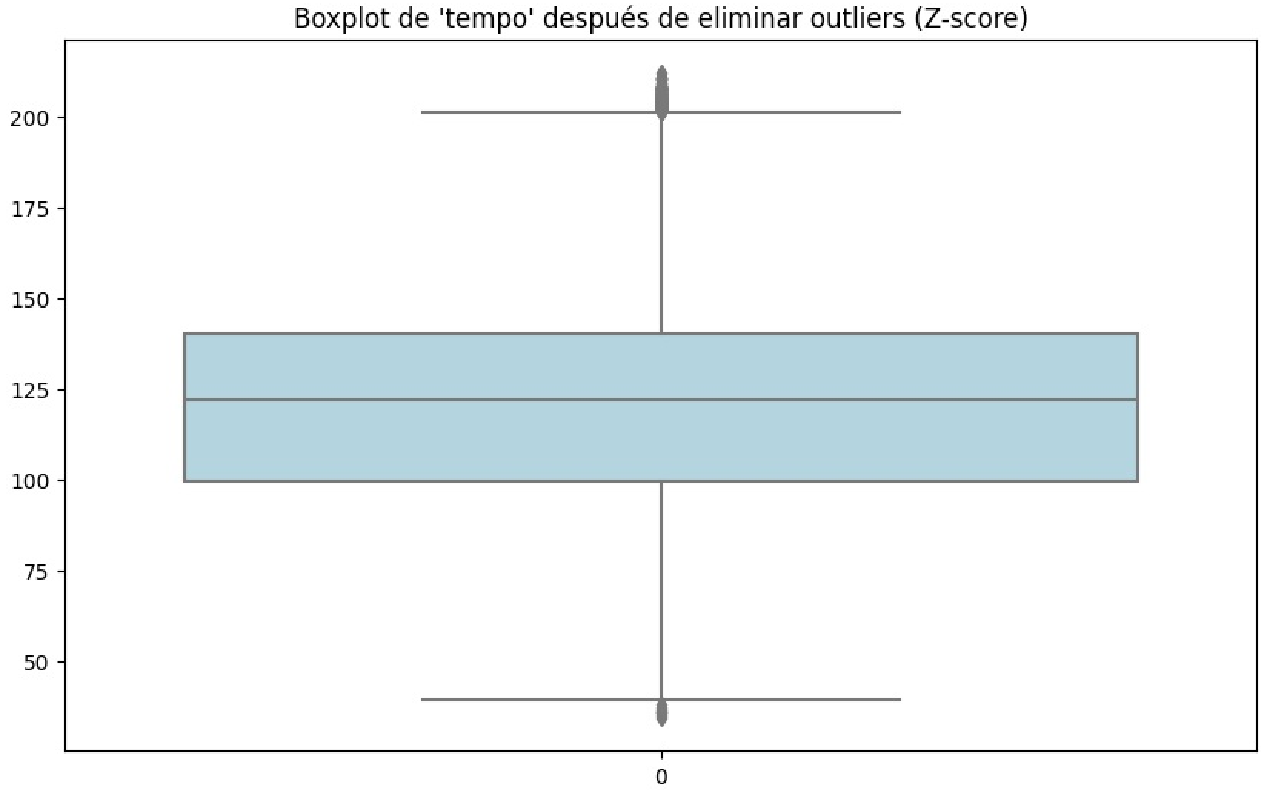
- **Action**: Remove extreme outliers.
- **Reason**: Values very close to 0 seem to be anomalous. These may represent erroneous data or songs that are out of the norm in terms of tempo.

Elimination of outliers tempo

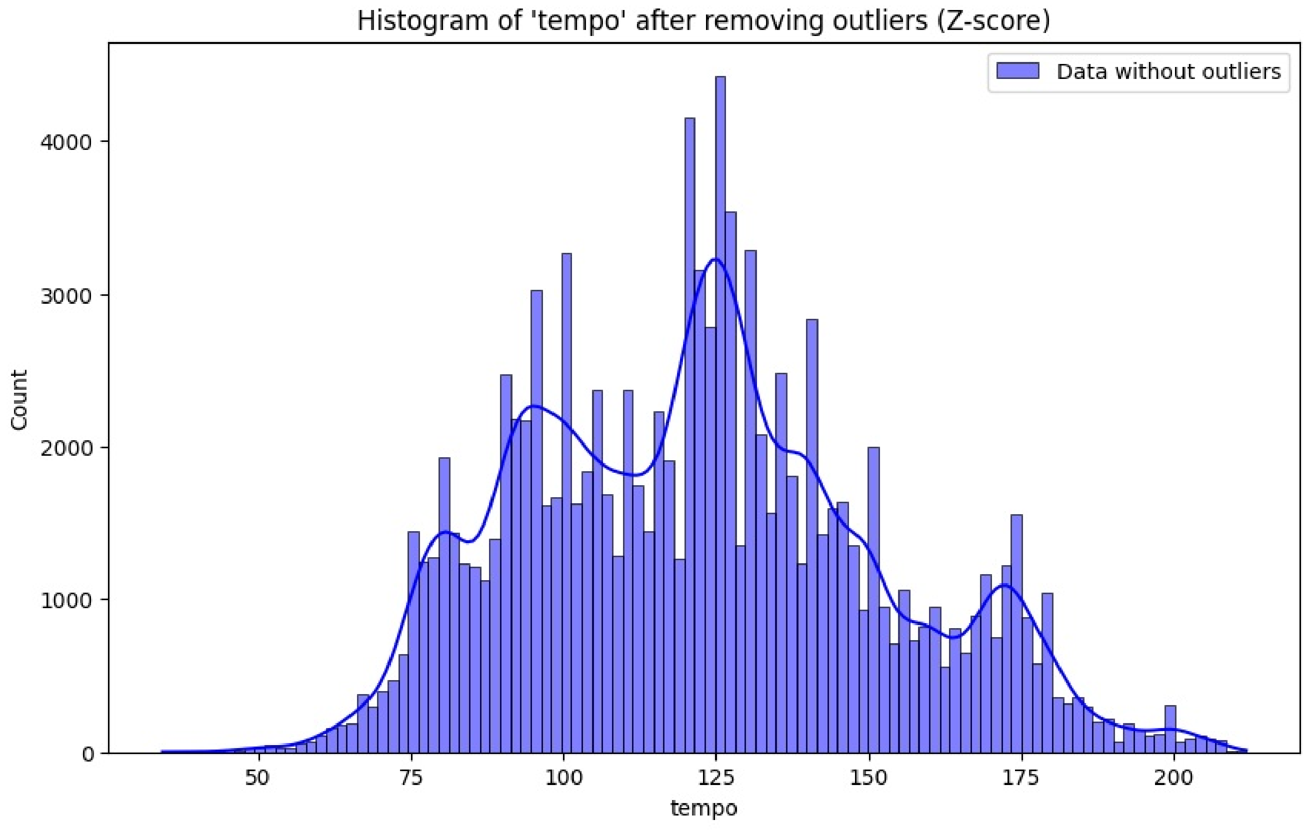
In [26]:

```
# Calculate Z-score for 'tempo' only
z_scores_tempo = np.abs(stats.zscore(spotify['tempo']))

# Filter rows that have outliers in 'tempo' (Z-score > 3)
spotify_sin_outliers_tempo = spotify[z_scores_tempo < 3]

# Show how many rows were deleted
print(f"Original data: {spotify.shape[0]}")
print(f"Data after removing outliers from 'tempo': {spotify_sin_outliers_tempo.shape[0]}")

# Visualize the data with a boxplot after removing the 'tempo' outliers
plt.figure(figsize=(10, 6))
sns.boxplot(data=spotify_sin_outliers_tempo['tempo'], color="lightblue")
plt.title("Boxplot de 'tempo' después de eliminar outliers (Z-score)")
plt.show()

# Display the tempo histogram without outliers
plt.figure(figsize=(10, 6))
sns.histplot(spotify_sin_outliers_tempo['tempo'], kde=True, color='blue', label='Data without outliers')
plt.title("Histogram of 'tempo' after removing outliers (Z-score)")
plt.legend()
plt.show()
```

```
Original data: 114000
Data after removing outliers from 'tempo': 113799
```

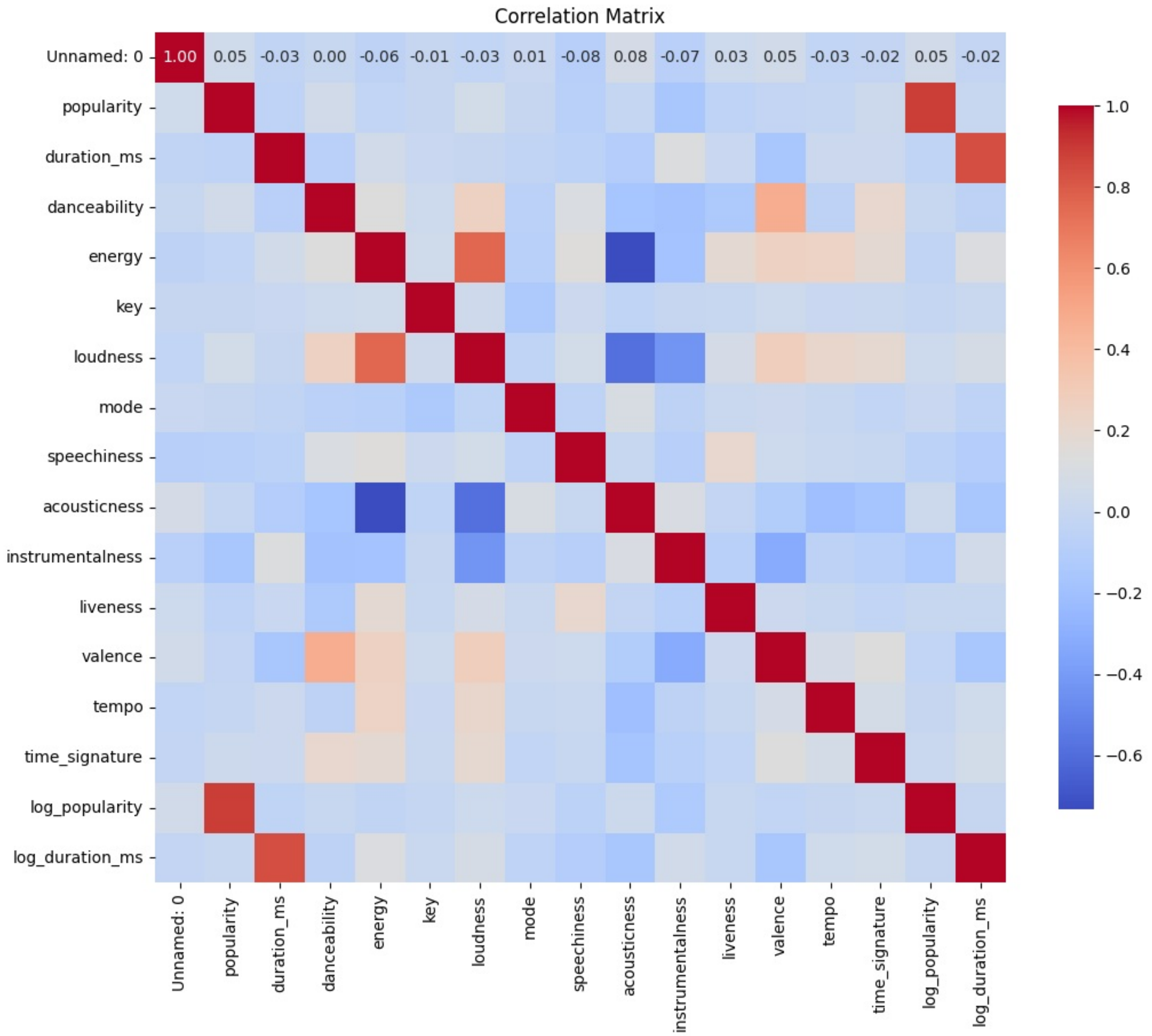



```
C:\Users\diego\OneDrive\Imágenes\Escritorio\Spotify-Recomendation-Machine-Learning\.venv\Lib\site-packages\seabo
rn\_oldcore.py:1119: FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before ope
rating instead.
```



Correlation Matrix

In [27]:
```
datosNumericos = spotify.select_dtypes(include=[np.number])

matriz_correlacion = datosNumericos.corr()

plt.figure(figsize=(12, 10))
sns.heatmap(matriz_correlacion, annot=True, fmt=".2f", cmap="coolwarm", square=True,
            cbar_kws={'shrink': .8}, annot_kws={"size": 10})
plt.title('Correlation Matrix')
plt.show()
```



Result

The correlation shown serves to demonstrate that some fields in the dataset are positively correlated (1) or negatively correlated (-1), and if they have a value of 0, there is no linear correlation between the variables:

- For this purpose, only the fields with numerical values were used, where the red cells indicate that there is a positive correlation and the blue cells the opposite.

1. The 'Danceability' field has a strong positive correlation with a valence of 0.48, suggesting that the 'valence' field is implicitly related.
2. The 'energy' field has a strong negative correlation with a valence of -0.73, suggesting that the 'acousticness' field is directly related.

Business Questions

1) ¿Cuántas canciones hay por género de música?

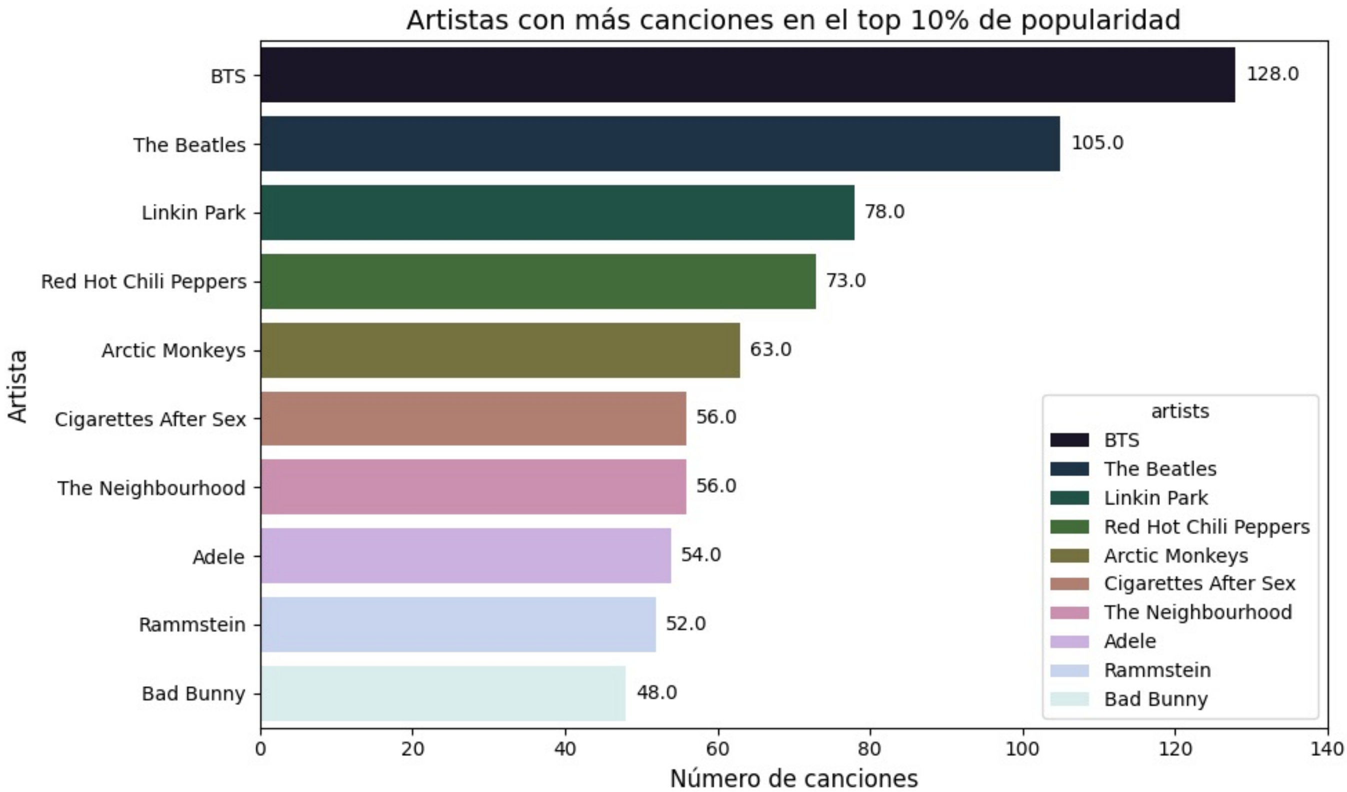
In [28]:
```
print("A continuación se mostraran el nombre de las columnas y el total de sus canciones:\n")
```

```
conteo_por_genero = spotify.groupby('track_genre')['track_id'].count().reset_index()

conteo_por_genero.columns = ['Género', 'Número de canciones']

print(conteo_por_genero)

print("Cada genero de música tiene un total de 1000 canciones")
```

```
A continuación se mostraran el nombre de las columnas y el total de sus canciones:

          Género  Número de canciones
0        acoustic                 1000
1        afrobeat                 1000
2        alt-rock                 1000
3     alternative                 1000
4         ambient                 1000
..            ...                  ...
109        techno                 1000
110        trance                 1000
111      trip-hop                 1000
112       turkish                 1000
113   world-music                 1000

[114 rows x 2 columns]
Cada genero de música tiene un total de 1000 canciones
```

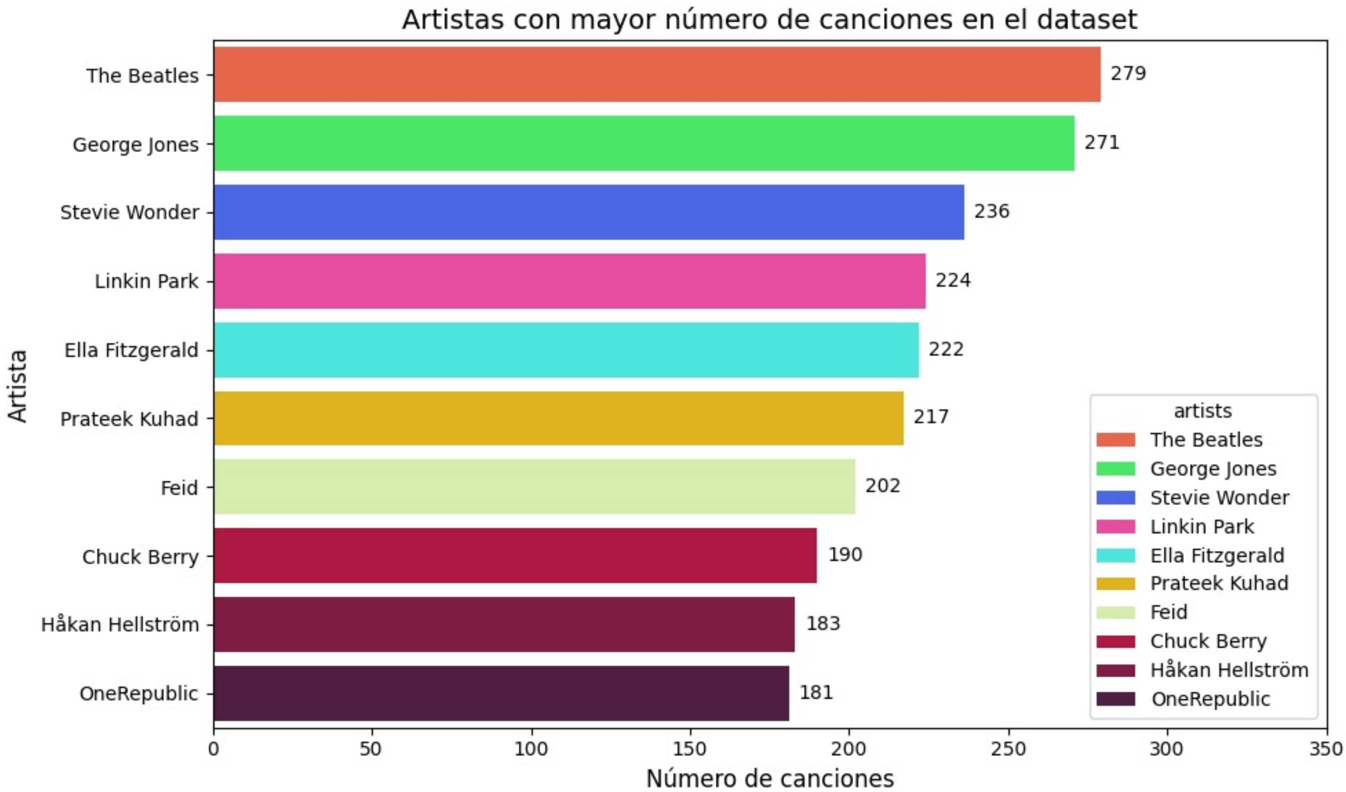
2) ¿Qué artista tiene más canciones en el top 10% de popularidad?

In [29]:
```
top_10_percent = spotify['popularity'].quantile(0.9)
top_10_songs = spotify[spotify['popularity'] >= top_10_percent]
top_artists = top_10_songs['artists'].value_counts().head(10)

plt.figure(figsize=(10, 6))
bars = sns.barplot(x=top_artists.values, y=top_artists.index, hue=top_artists.index, dodge=False, palette="cubel

plt.title('Artistas con más canciones en el top 10% de popularidad', fontsize=14)
plt.xlabel('Número de canciones', fontsize=12)
plt.ylabel('Artista', fontsize=12)

plt.xlim(0, 140)

for bar in bars.patches:
    bars.annotate(format(bar.get_width(), '.1f'),
                  (bar.get_width(), bar.get_y() + bar.get_height() / 2),
                  ha='left', va='center', size=10, xytext=(5, 0),
                  textcoords='offset points')

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



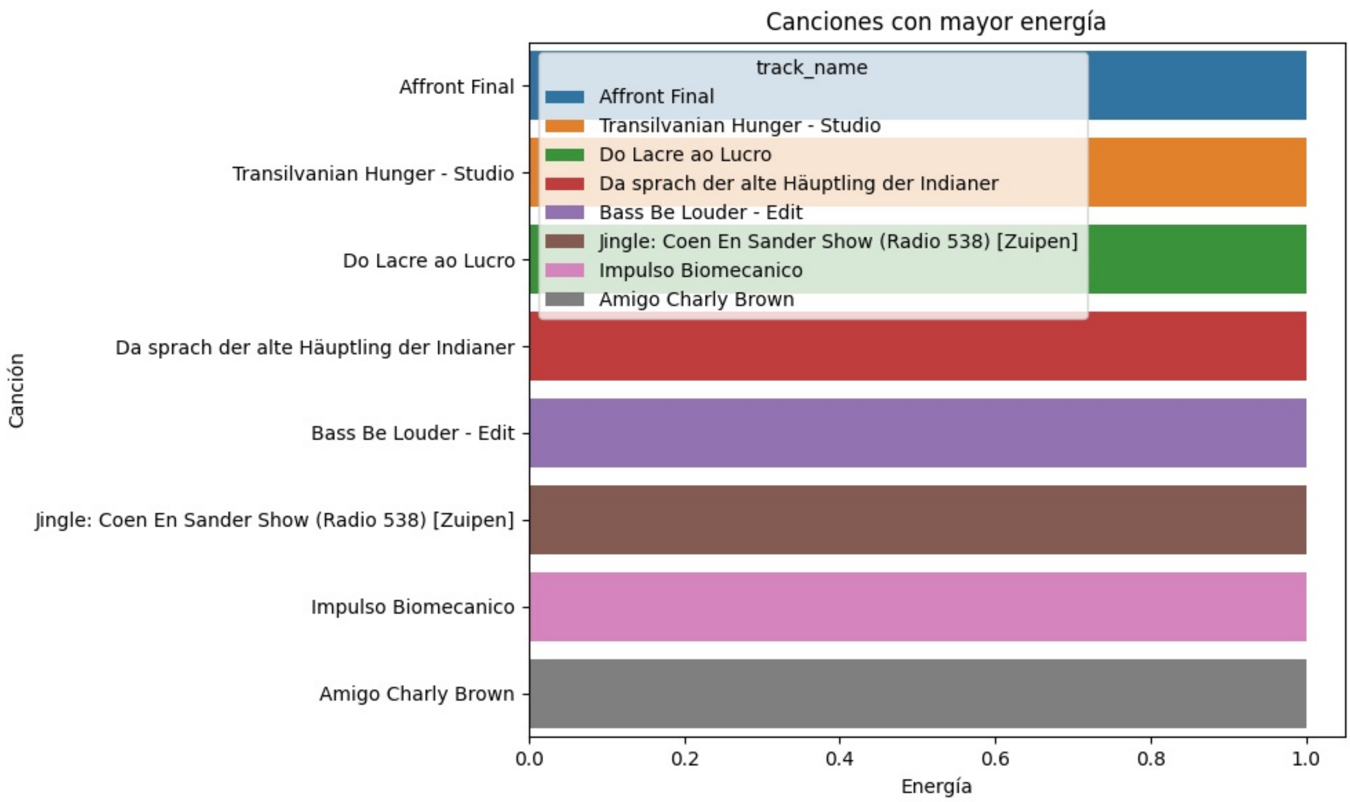
3) ¿Qué género tiene la mayor cantidad de canciones explícitas?(hacer que se muestre el numero exacto del resultado)

In [30]:
```
artist_count = spotify['artists'].value_counts().head(10)

colors = ['#FF5733', '#33FF57', '#3357FF', '#FF33A1', '#33FFF2', '#FFC300', '#DAF7A6', '#C70039', '#900C3F', '#!

plt.figure(figsize=(10, 6))
bars = sns.barplot(x=artist_count.values, y=artist_count.index, hue=artist_count.index, dodge=False, palette=co

plt.xlim(0, 350)

plt.title('Artistas con mayor número de canciones en el dataset', fontsize=14)
plt.xlabel('Número de canciones', fontsize=12)
plt.ylabel('Artista', fontsize=12)

for bar in bars.patches:
    bars.annotate(format(bar.get_width(), '.0f'),
                  (bar.get_width(), bar.get_y() + bar.get_height() / 2),
                  ha='left', va='center', size=10, xytext=(5, 0),
                  textcoords='offset points')

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



4) ¿Cuáles son las canciones con la mayor energía en el dataset?

In [31]:
```
top_energy_songs = spotify.nlargest(10, 'energy')

plt.figure(figsize=(10, 6))
sns.barplot(x=top_energy_songs['energy'], y=top_energy_songs['track_name'], hue=top_energy_songs['track_name'],

plt.title('Canciones con mayor energía')
plt.xlabel('Energía')
plt.ylabel('Canción')
plt.tight_layout()
plt.show()
```



5) ¿Qué artista tiene el mayor número de canciones en el dataset?

In [ ]:
```
# Contar los artistas con mayor número de canciones
artist_count = spotify['artists'].value_counts().head(10)

# Crear el gráfico asignando el color a cada barra
plt.figure(figsize=(10, 6))
bars = sns.barplot(x=artist_count.index, y=artist_count.values, hue=None, dodge=False, palette="coolwarm")

# Establecer título y etiquetas de los ejes
plt.title('Artistas con mayor número de canciones en el dataset', fontsize=14)
plt.xlabel('Artista', fontsize=12)
plt.ylabel('Número de canciones', fontsize=12)

# Ajustar el límite del eje y a 290
plt.ylim(0, 300)

# Rotar etiquetas del eje x
plt.xticks(rotation=45, ha='right')

# Añadir valor a cada barra
for bar in bars.patches:
    bars.annotate(format(bar.get_height(), '.1f'),
                  (bar.get_x() + bar.get_width() / 2, bar.get_height()),
                  ha='center', va='bottom', size=10, xytext=(0, 5),
                  textcoords='offset points')

# Ajustar el layout
plt.tight_layout()

# Mostrar el gráfico
plt.show()
```

6) ¿Cuál es el tiempo total de escucha de todas las canciones en un género específico?

In [ ]:
```
import plotly.express as px

# Agrupación y cálculo de la duración total por género
genre_duration = spotify.groupby('track_genre')['duration_ms'].sum().sort_values(ascending=False).head(10)

# Convertir la duración a minutos
genre_duration_minutes = genre_duration / (1000 * 60)

# Crear gráfico interactivo
fig = px.bar(
    x=genre_duration_minutes.values,
    y=genre_duration_minutes.index,
    labels={'x': 'Tiempo total de escucha (minutos)', 'y': 'Género'},
    title='Tiempo total de escucha por género (en minutos)',
```

```
    color=genre_duration_minutes.index,
    color_continuous_scale='Viridis',
    orientation='h'
)

# Ajustar límites del eje x
fig.update_layout(xaxis_range=[0, 7000])

# Mostrar gráfico interactivo
fig.show()
```

7): ¿Cuál es el género con la duración promedio más corta de las canciones?

In [ ]:
```
import plotly.express as px

# Datos para el gráfico
genre_duration = spotify.groupby('track_genre')['duration_ms'].sum().sort_values(ascending=False).head(10)

# Definir una paleta de colores personalizada (similar a la usada por defecto en Seaborn)
colors = px.colors.qualitative.Plotly  # Puedes cambiar esta lista con los colores que prefieras

# Crear gráfico interactivo con Plotly
fig = px.bar(
    x=genre_duration.values / (1000 * 60),
    y=genre_duration.index,
    labels={'x': 'Tiempo total de escucha (minutos)', 'y': 'Género'},
    title='Tiempo total de escucha por género (en minutos)',
    orientation='h',
    color=genre_duration.index,  # Colorear por género
    color_discrete_sequence=colors  # Aplicar la paleta de colores
)

# Ajustes de layout
fig.update_layout(
    xaxis_range=[0, 7000],
    xaxis_title="Tiempo total de escucha (minutos)",
    yaxis_title="Género",
    title_font_size=14,
    xaxis_title_font_size=12,
    yaxis_title_font_size=12,
    margin=dict(l=50, r=50, t=50, b=50),
    showlegend=False  # Quitar la leyenda si no es necesaria
)

# Mostrar gráfico interactivo
fig.show()
```

8) ¿Qué artista tiene la canción más larga en términos de duración?

In [ ]:
```
longest_songs = spotify[['track_name', 'artists', 'duration_ms']].sort_values(by='duration_ms', ascending=False

fig = px.bar(longest_songs,
             x=longest_songs['duration_ms'] / 1000,
             y=longest_songs['track_name'],
             color='track_name',
             labels={'duration_ms': 'Duración (segundos)', 'track_name': 'Canción'},
             title='Canciones más largas (en segundos)',
             orientation='h',
             color_discrete_sequence=px.colors.sequential.Magma)

fig.update_layout(xaxis_title='Duración (segundos)',
                  yaxis_title='Canción',
                  yaxis={'categoryorder':'total ascending'},
                  xaxis_range=[0, 6000],
                  template='plotly_white')

fig.show()
```

Sound_features

Se creo una

In [ ]:
```
sound_features = ['acousticness', 'danceability', 'energy', 'instrumentalness', 'liveness', 'valence']

sound_features
```

Exploration tops

Top Genre

```
In [ ]: # Group by gender and calculate the average popularity of each
top_genres = spotify.groupby('track_genre')['popularity'].mean().sort_values(ascending=False).head(10)

# Filter the dataset to include only these genres
top_genres_data = spotify[spotify['track_genre'].isin(top_genres.index)]

print(top_genres)
```

```
In [ ]: fig = px.bar(top_genres_data,
             x=top_genres.index,
             y=top_genres.values,
             color=top_genres.values,
             color_continuous_scale='RdBu',
             labels={'x': ' Musical Genre', 'y': 'Popularity'},
             title='Top 10 most popular genres')

fig.show()
```

```
In [ ]: import plotly.express as px

fig = px.pie(top_genres_data, names=top_genres.index, values=top_genres.values,color=top_genres.values,title='T

fig.show()
```

Top genre + sound_features Scatter

Chart to find a pattern that determines why x gender is better.

```
In [ ]: mean_features_by_genre = top_genres_data.groupby('track_genre')[sound_features].mean().reset_index()

fig = px.histogram(mean_features_by_genre,
             x="track_genre", y=sound_features,
             barmode='group',
             labels={'track_genre': 'Musical Genre','value': 'Average Features'},
             title='Average Characteristics by Gender')

fig.show()
```

No pattern was found to determine which is the best genre, it all depends on people's musical taste.

Top Artists

```
In [ ]: # Now let's calculate the average popularity of each artist in ascending order
artistas_populares_promedio_asc = spotify.groupby('artists')['popularity'].mean().sort_values(ascending=False).

# Creating the graph with the average popularity in ascending order
fig = px.bar(
    artistas_populares_promedio_asc.reset_index(),
    x='popularity',
    y='artists',
    orientation='h',
    title='Top 10 Artists with Highest Average Popularity (Ascending Order)',
    labels={'popularity': 'Average Popularity', 'artists': 'Artists'},
    color='popularity',
    color_continuous_scale='Viridis'
)

fig.show()
```

Top Album

```
In [ ]: top_albums = spotify.groupby('album_name')['popularity'].mean().sort_values(ascending=False).head(10)

# Creating the pie chart with plotly express
fig = px.pie(
    top_albums.reset_index(),
    names='album_name',
    values='popularity',
    title='Top 10 Albums with the Highest Average Popularity',
    labels={'album_name': 'Álbum', 'popularity': 'Average Popularity'},
    color_discrete_sequence=px.colors.sequential.Viridis
)

# Show the graph
```

```
fig.show()
```

DataSet Save

Data With Outliers

In [ ]:
```
rute_cvs_save = os.path.join('..','data','02_intermediate','2.spotify.csv')

spotify.to_csv (rute_cvs_save, index=False)
```

DataSet Without Outliers

In [ ]:
```
rute_cvs_save = os.path.join('..','data','02_intermediate','2.spotifySinOutlier.csv')

spotify_sin_outliers_tempo.to_csv (rute_cvs_save, index=False)
```