Caio de Lima Saigg, 254677

In [111…

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dri
ve.mount("/content/drive", force_remount=True).
```

In [112…

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from pandas.api.types import is_numeric_dtype
```

In [113…

```
CATEGORICAL_FEATURES = ['track_name','artist(s)_name','key','mode']

# Some numerical columns are written as string with numbers bigger than 999 writ
def convert_to_numeric(val):
    try:
        if isinstance(val, str):
            val = val.replace(',','')
        return pd.to_numeric(val)
    except ValueError:
        return np.nan

df = pd.read_csv("/content/drive/MyDrive/IMLDS/spotify-2023.csv", encoding='lati
NUMERICAL_FEATURES = []
for key in df.keys():
    if key not in CATEGORICAL_FEATURES:
        NUMERICAL_FEATURES.append(key)
        df[key] = df[key].map(convert_to_numeric)
df = df.dropna()
```

In [114…

```
print("MEAN VALUE FOR EACH NUMERIC FEATURE\n")
print(df.mean(numeric_only=True))
```

```
MEAN VALUE FOR EACH NUMERIC FEATURE

artist_count            1.568627e+00
released_year           2.018517e+03
released_month          6.024510e+00
released_day            1.371201e+01
in_spotify_playlists    4.852316e+03
in_spotify_charts       1.173652e+01
streams                 4.689858e+08
in_apple_playlists      6.021569e+01
in_apple_charts         4.953431e+01
in_deezer_playlists     3.720539e+02
in_deezer_charts        2.454657e+00
in_shazam_charts        5.762255e+01
bpm                     1.225809e+02
danceability_%          6.740931e+01
valence_%               5.117279e+01
energy_%                6.435662e+01
acousticness_%          2.633333e+01
instrumentalness_%      1.678922e+00
liveness_%              1.817034e+01
speechiness_%           1.053554e+01
dtype: float64
```

In [115…

```
print("MEDIAN FOR EACH NUMERIC FEATURE\n")
print(df.median(numeric_only=True))
```

```
MEDIAN FOR EACH NUMERIC FEATURE

artist_count                  1.0
released_year              2022.0
released_month                5.0
released_day                 13.0
in_spotify_playlists       2037.5
in_spotify_charts             3.0
streams               263836779.5
in_apple_playlists           32.0
in_apple_charts              34.5
in_deezer_playlists          39.0
in_deezer_charts              0.0
in_shazam_charts              3.0
bpm                         120.0
danceability_%               70.0
valence_%                    51.0
energy_%                     66.0
acousticness_%               17.0
instrumentalness_%            0.0
liveness_%                   12.0
speechiness_%                 6.0
dtype: float64
```

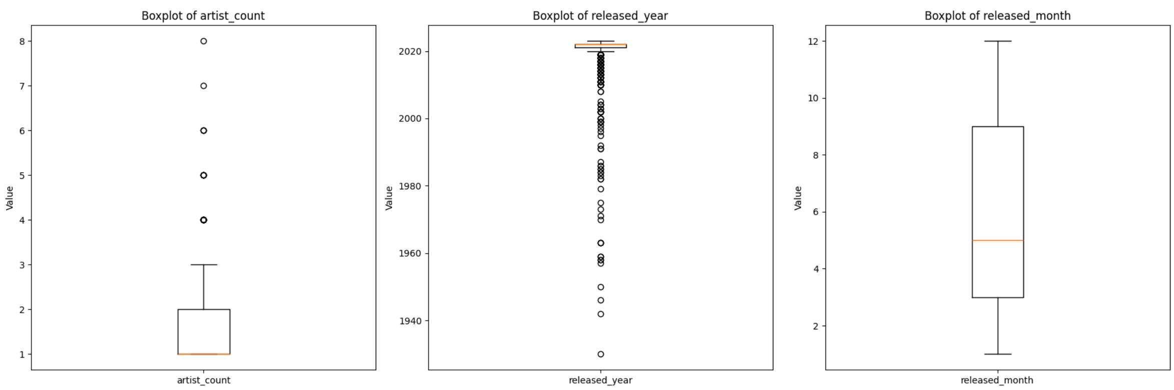
In [116…

```
print("VARIANCE FOR EACH NUMERIC FEATURE\n")
print(df.var(numeric_only=True))
```

```
VARIANCE FOR EACH NUMERIC FEATURE

artist_count            7.682906e-01
released_year           1.145322e+02
released_month          1.274786e+01
released_day            8.639181e+01
in_spotify_playlists    5.999378e+07
in_spotify_charts       3.468741e+02
streams                 2.736616e+17
in_apple_playlists      5.618037e+03
in_apple_charts         2.457238e+03
in_deezer_playlists     1.340406e+06
in_deezer_charts        2.915622e+01
in_shazam_charts        2.359797e+04
bpm                     7.945996e+02
danceability_%          2.157611e+02
valence_%               5.579394e+02
energy_%                2.597463e+02
acousticness_%          6.491084e+02
instrumentalness_%      7.695690e+01
liveness_%              1.836090e+02
speechiness_%           1.045067e+02
dtype: float64
```

In [117…

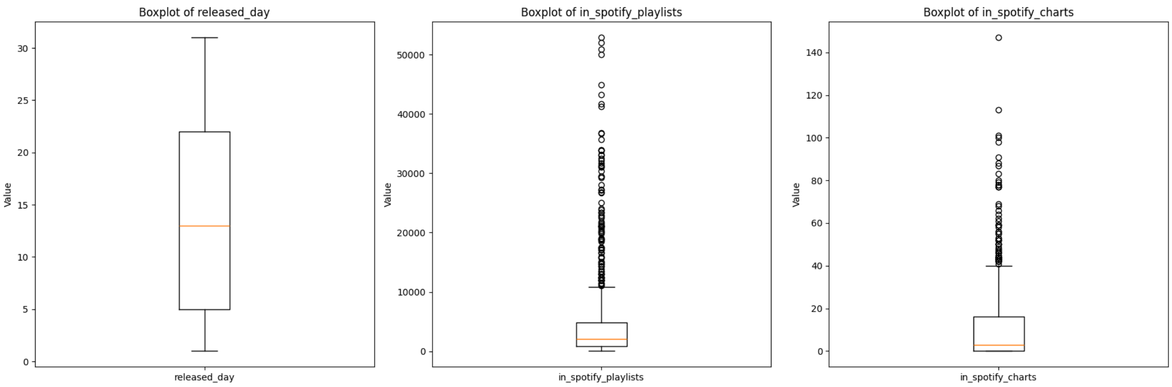
```
i = 0
batch_size = 3
batch = NUMERICAL_FEATURES[i:i + batch_size]

fig, axes = plt.subplots(1, batch_size, figsize=(18, 6))  # 1x3 layout

for j, feature in enumerate(batch):
    ax = axes[j]  # Get the subplot axis
    ax.boxplot(df[feature].dropna(), labels=[feature])  # Plot boxplot
    ax.set_title(f"Boxplot of {feature}")
    ax.set_ylabel("Value")


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



In [118…

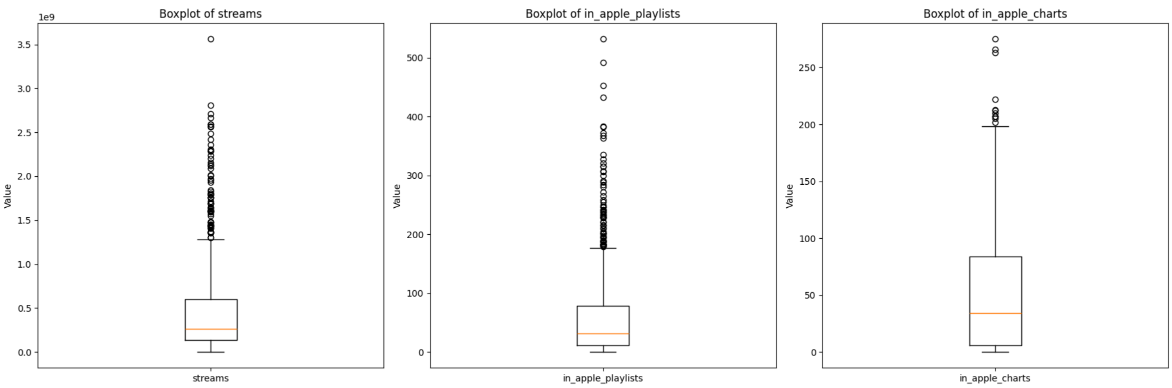
```
i = i + batch_size
batch_size = 3
batch = NUMERICAL_FEATURES[i:i + batch_size]

fig, axes = plt.subplots(1, batch_size, figsize=(18, 6))  # 1x3 layout

for j, feature in enumerate(batch):
    ax = axes[j]  # Get the subplot axis
```

```python
    ax.boxplot(df[feature].dropna(), labels=[feature])  # Plot boxplot
    ax.set_title(f"Boxplot of {feature}")
    ax.set_ylabel("Value")


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



In [119…

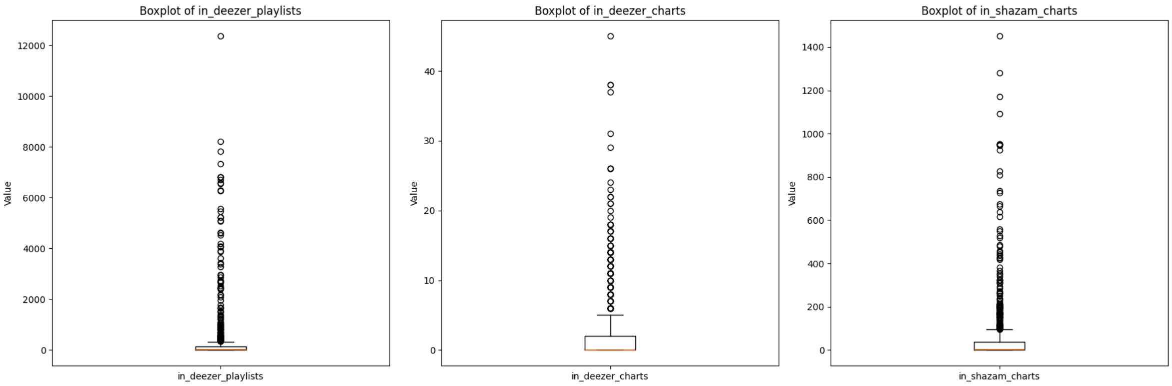
```python
i = i + batch_size
batch_size = 3
batch = NUMERICAL_FEATURES[i:i + batch_size]

fig, axes = plt.subplots(1, batch_size, figsize=(18, 6))  # 1x3 layout

for j, feature in enumerate(batch):
    ax = axes[j]  # Get the subplot axis
    ax.boxplot(df[feature].dropna(), labels=[feature])  # Plot boxplot
    ax.set_title(f"Boxplot of {feature}")
    ax.set_ylabel("Value")


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



In [120…

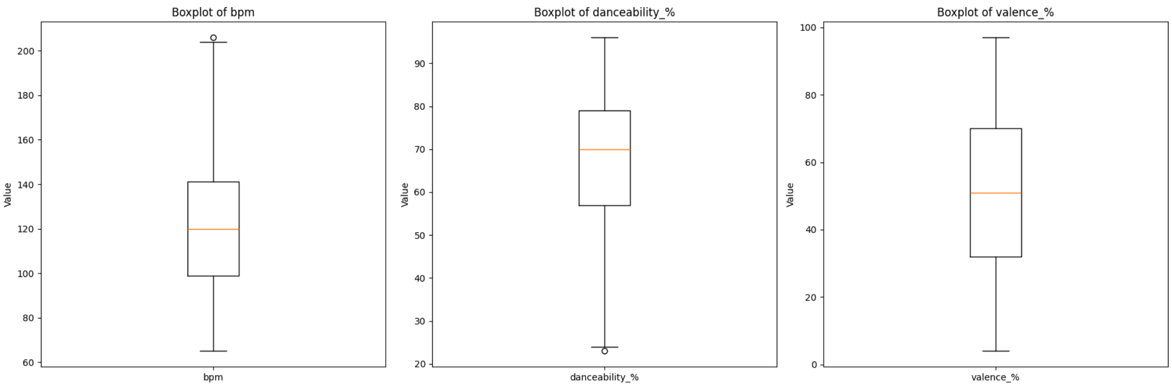
```python
i = i + batch_size
batch_size = 3
batch = NUMERICAL_FEATURES[i:i + batch_size]

fig, axes = plt.subplots(1, batch_size, figsize=(18, 6))  # 1x3 layout

for j, feature in enumerate(batch):
    ax = axes[j]  # Get the subplot axis
    ax.boxplot(df[feature].dropna(), labels=[feature])  # Plot boxplot
    ax.set_title(f"Boxplot of {feature}")
    ax.set_ylabel("Value")
```

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



In [121…

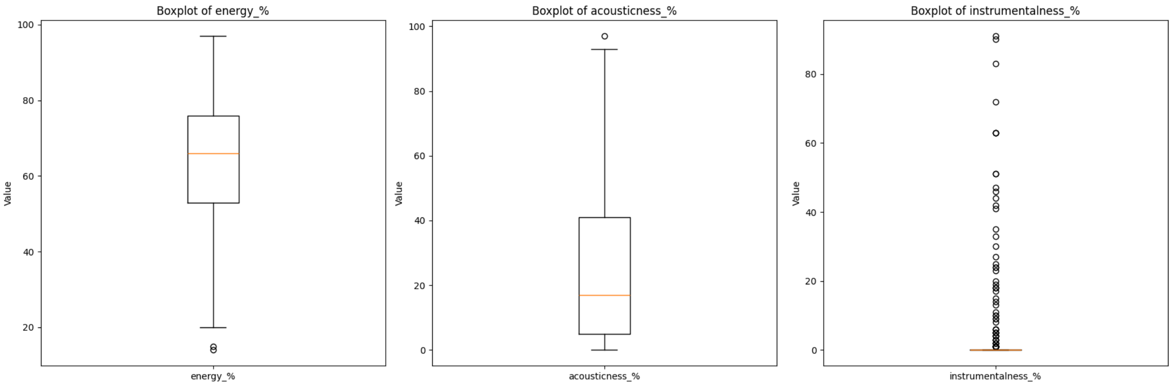
```
i = i + batch_size
batch_size = 3
batch = NUMERICAL_FEATURES[i:i + batch_size]

fig, axes = plt.subplots(1, batch_size, figsize=(18, 6))  # 1x3 layout

for j, feature in enumerate(batch):
    ax = axes[j]  # Get the subplot axis
    ax.boxplot(df[feature].dropna(), labels=[feature])  # Plot boxplot
    ax.set_title(f"Boxplot of {feature}")
    ax.set_ylabel("Value")


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



In [122…

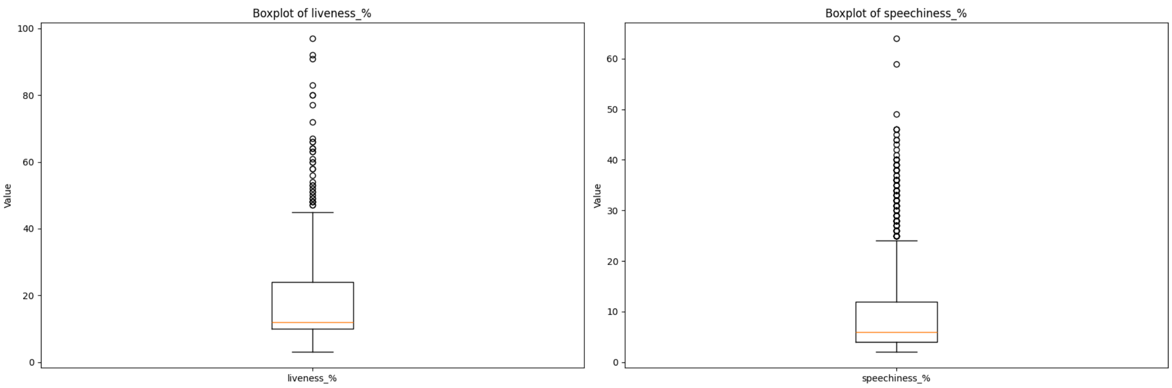
```
i = i + batch_size
batch_size = 3
batch = NUMERICAL_FEATURES[i:i + batch_size]

fig, axes = plt.subplots(1, batch_size, figsize=(18, 6))  # 1x3 layout

for j, feature in enumerate(batch):
    ax = axes[j]  # Get the subplot axis
    ax.boxplot(df[feature].dropna(), labels=[feature])  # Plot boxplot
    ax.set_title(f"Boxplot of {feature}")
    ax.set_ylabel("Value")
```

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



In [123…

```
i = i + batch_size
batch_size = 2
batch = NUMERICAL_FEATURES[i:i + batch_size]

fig, axes = plt.subplots(1, batch_size, figsize=(18, 6))  # 1x3 layout

for j, feature in enumerate(batch):
    ax = axes[j]  # Get the subplot axis
    ax.boxplot(df[feature].dropna(), labels=[feature])  # Plot boxplot
    ax.set_title(f"Boxplot of {feature}")
    ax.set_ylabel("Value")


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



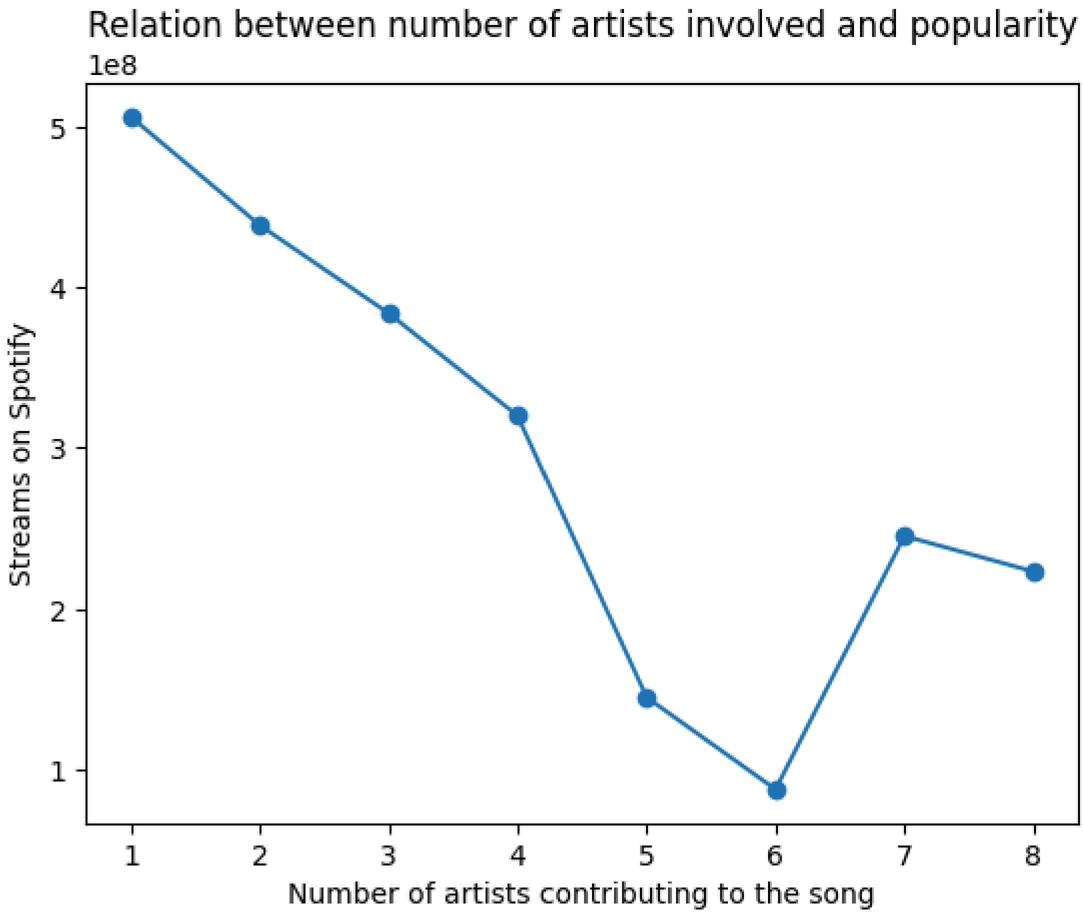
The boxplots reveal that many features contain a significant number of outliers. Features with a higher quantity of outliers also have greater variance and a greater difference between the mean and median, as the mean is more sensitive to the influence of outliers compared to the median.

Additionally, the range of values for each feature is quite wide, and the boxplots shows notable variations in their distributions. This suggests that the data spans diverse scales and patterns across the features.

In [124…

```
counts = df['artist_count'].value_counts()
df['weight'] = df['artist_count'].map(1 / counts)

weighted_mean = df.groupby('artist_count').apply(lambda x: np.average(x['streams
plt.plot(weighted_mean.index, weighted_mean.values, marker='o')
plt.ylabel('Streams on Spotify')
plt.xlabel('Number of artists contributing to the song')
plt.title('Relation between number of artists involved and popularity')
plt.show()
df = df.drop('weight', axis=1)
```

```
<ipython-input-124-f639d1a274c1>:4: DeprecationWarning: DataFrameGroupBy.apply op
erated on the grouping columns. This behavior is deprecated, and in a future vers
ion of pandas the grouping columns will be excluded from the operation. Either pa
ss `include_groups=False` to exclude the groupings or explicitly select the group
ing columns after groupby to silence this warning.
  weighted_mean = df.groupby('artist_count').apply(lambda x: np.average(x['stream
s'], weights=x['weight']))
```

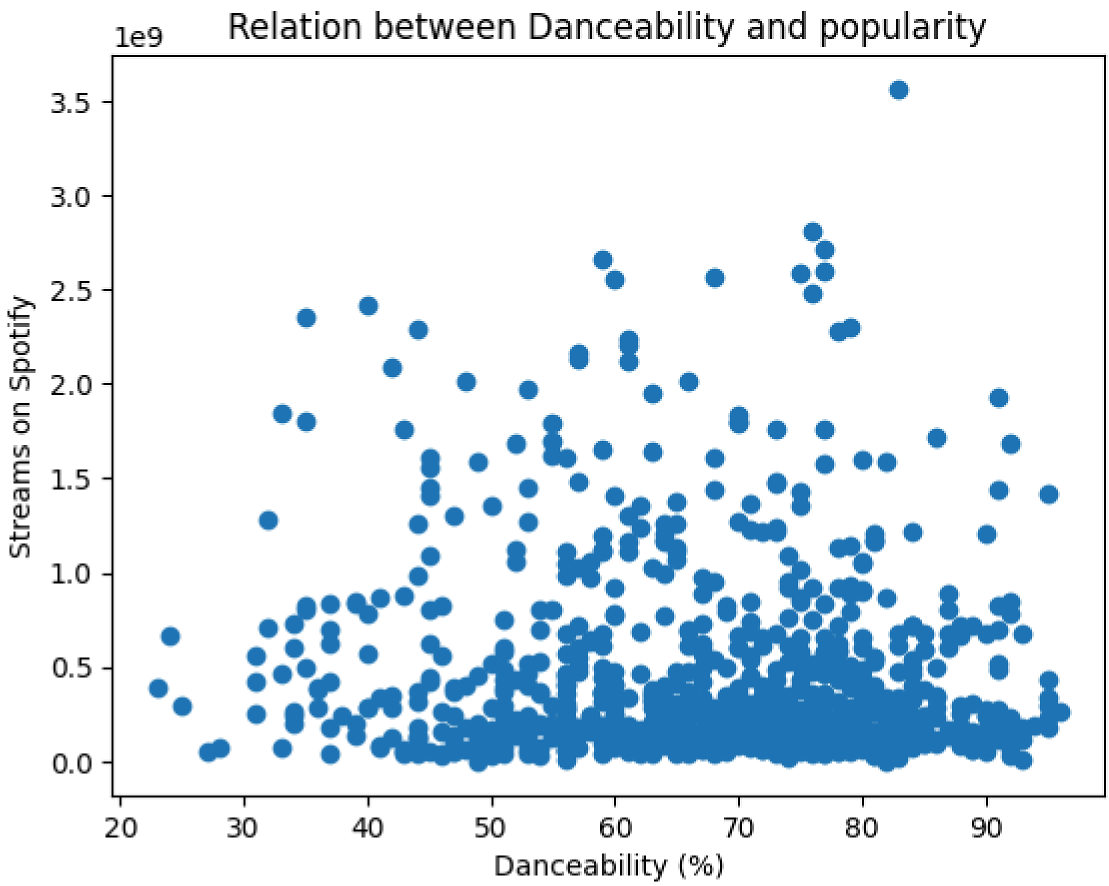


Given that songs with a single artist are far more common than those with multiple artists, we applied a weighted average to the calculation of streams per artist_count. This ensures that less frequent cases are properly accounted for. However, the resulting graph indicates that a song's popularity, as measured by streams, does not increase proportionally with the number of contributing artists.

The plots indicate no relation between beats per minute and the danceability, energy, liveness, etc... of the song

In [125…

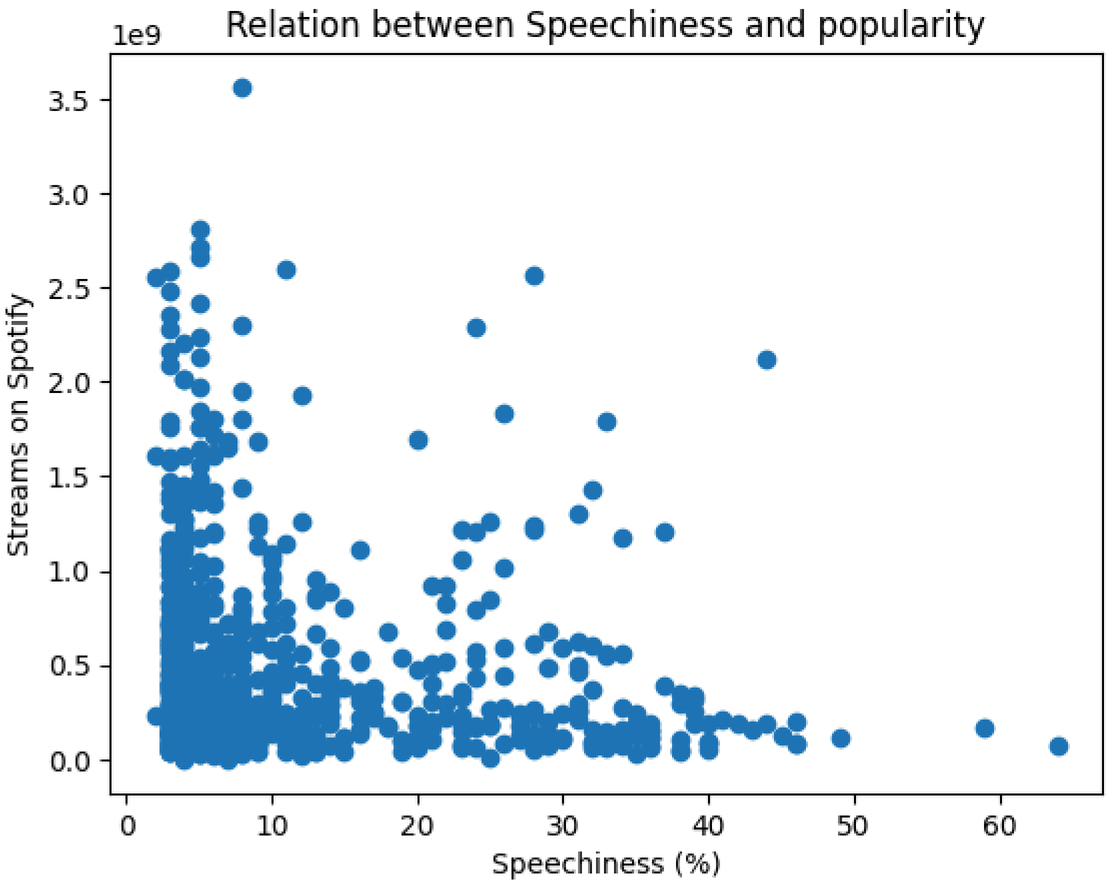
```
plt.scatter(df['danceability_%'] , df['streams'])

plt.ylabel('Streams on Spotify')
plt.xlabel('Danceability (%)')
plt.title('Relation between Danceability and popularity')
plt.show()
```



The scatter plot shows no clear influence of danceability on popularity

In [126…

```
plt.scatter(df['speechiness_%'] , df['streams'])

plt.ylabel('Streams on Spotify')
plt.xlabel('Speechiness (%)')
plt.title('Relation between Speechiness and popularity')
plt.show()
```

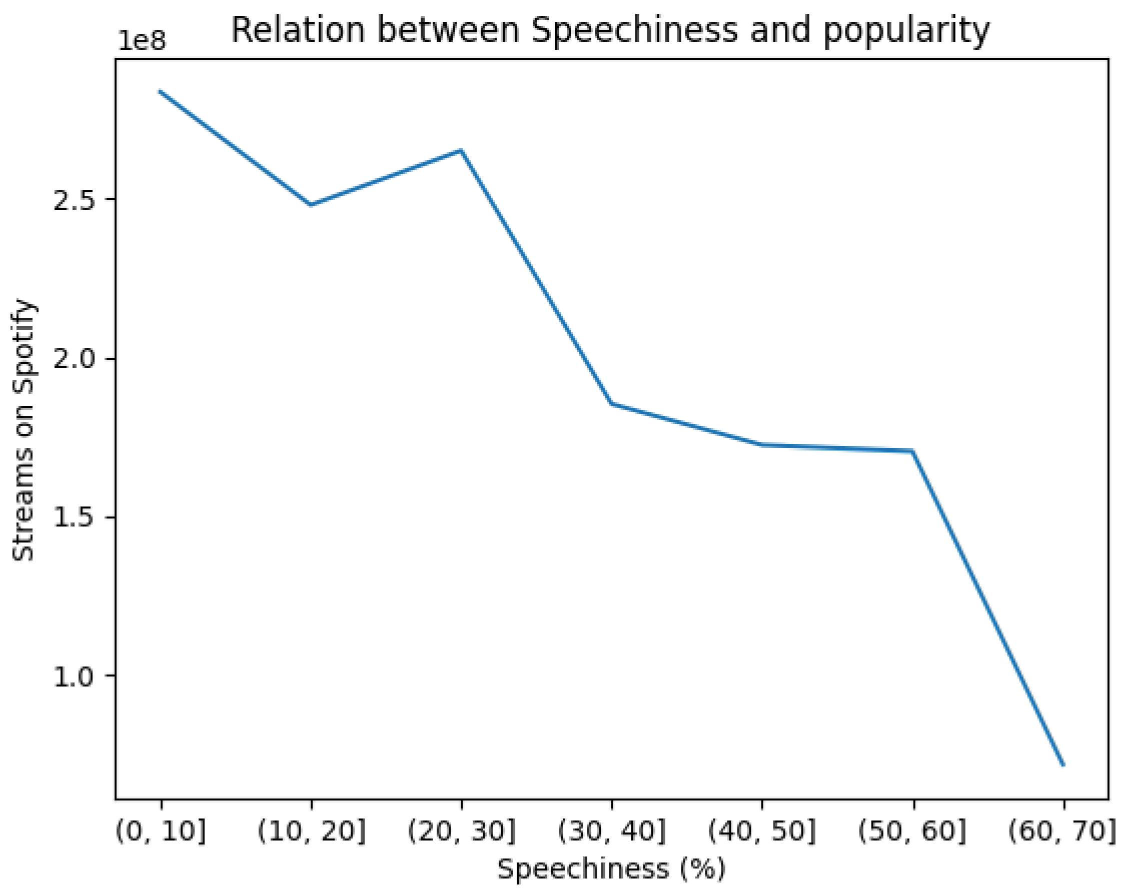


We can see that the graph is very consistent with the exception of some outliers. In this case we are going to take into account then the median value instead of the mean to measure the average popularity of a song.

In [127…

```
bins = pd.cut(df['speechiness_%'], bins=[0, 10, 20, 30, 40, 50, 60, 70, 80, 100]
speechMedian = df.groupby(bins)['streams'].median()
speechMedian.index = speechMedian.index.astype(str)
plt.plot(speechMedian.index,speechMedian.values)
plt.ylabel('Streams on Spotify')
plt.xlabel('Speechiness (%)')
plt.title('Relation between Speechiness and popularity')
plt.show()
```

```
<ipython-input-127-fc00e49a77fb>:2: FutureWarning: The default of observed=False
is deprecated and will be changed to True in a future version of pandas. Pass obs
erved=False to retain current behavior or observed=True to adopt the future defau
lt and silence this warning.
  speechMedian = df.groupby(bins)['streams'].median()
```

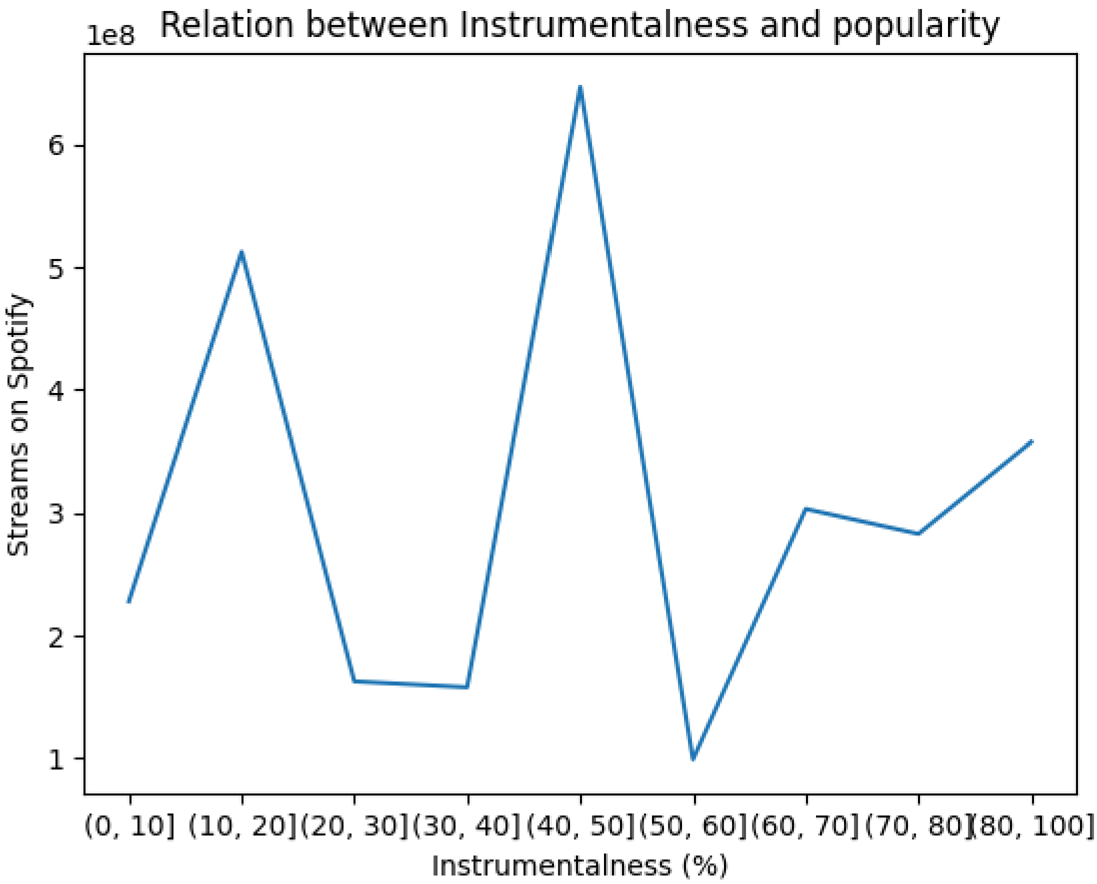


From the graph we can see that songs with less spoken words tend to have a higher quantity of streams

In [128…

```
bins = pd.cut(df['instrumentalness_%'], bins=[0, 10, 20, 30, 40, 50, 60, 70, 80,
instrumentalMedian = df.groupby(bins)['streams'].median()
instrumentalMedian.index = instrumentalMedian.index.astype(str)
plt.plot(instrumentalMedian.index,instrumentalMedian.values)
plt.ylabel('Streams on Spotify')
plt.xlabel('Instrumentalness (%)')
plt.title('Relation between Instrumentalness and popularity')
plt.show()
```

```
<ipython-input-128-759fa6f5ec3d>:2: FutureWarning: The default of observed=False
is deprecated and will be changed to True in a future version of pandas. Pass obs
erved=False to retain current behavior or observed=True to adopt the future defau
lt and silence this warning.
  instrumentalMedian = df.groupby(bins)['streams'].median()
```

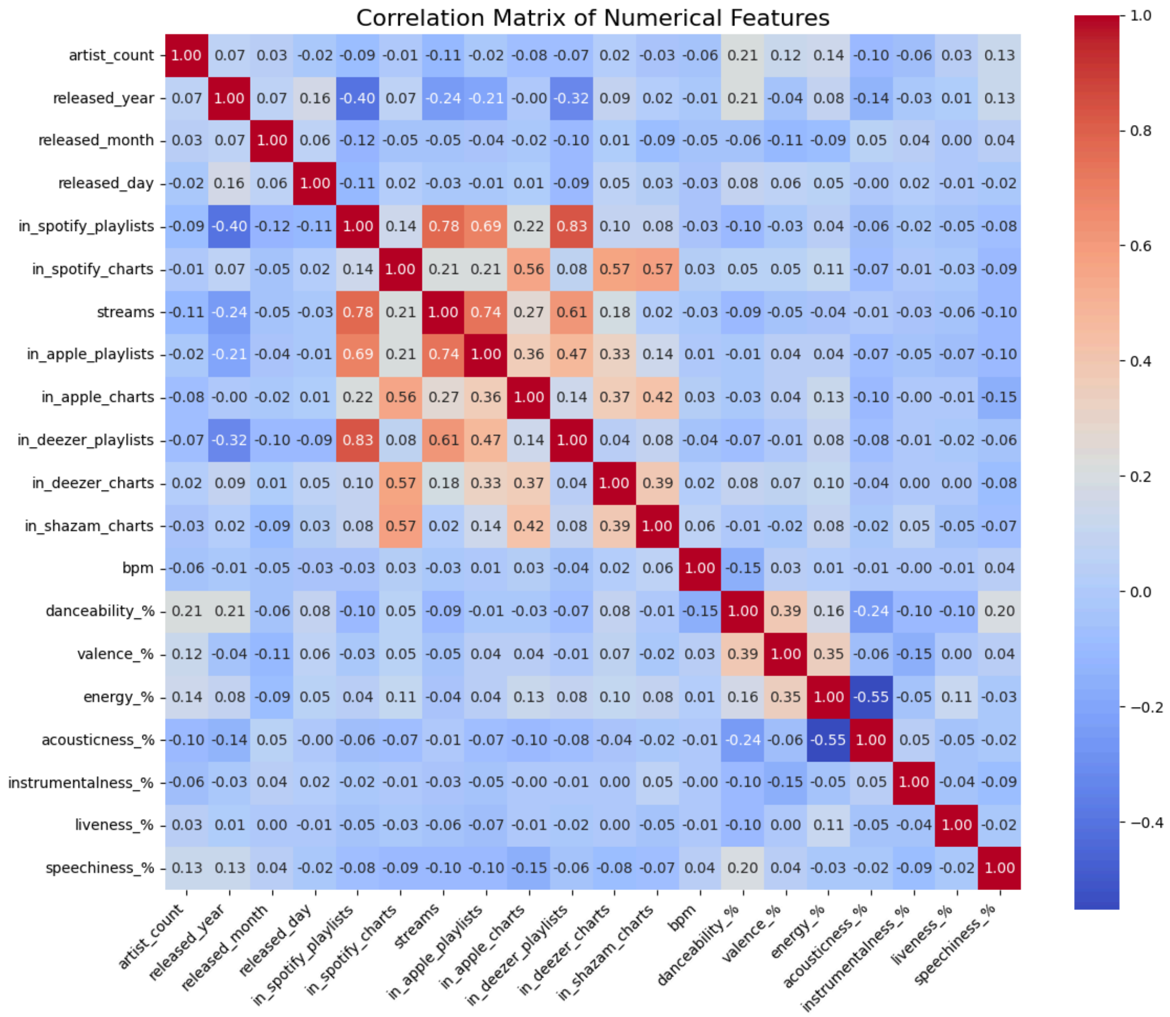


Doing the same analysis with the amount of instrumental content shows no direct relation

Correlations

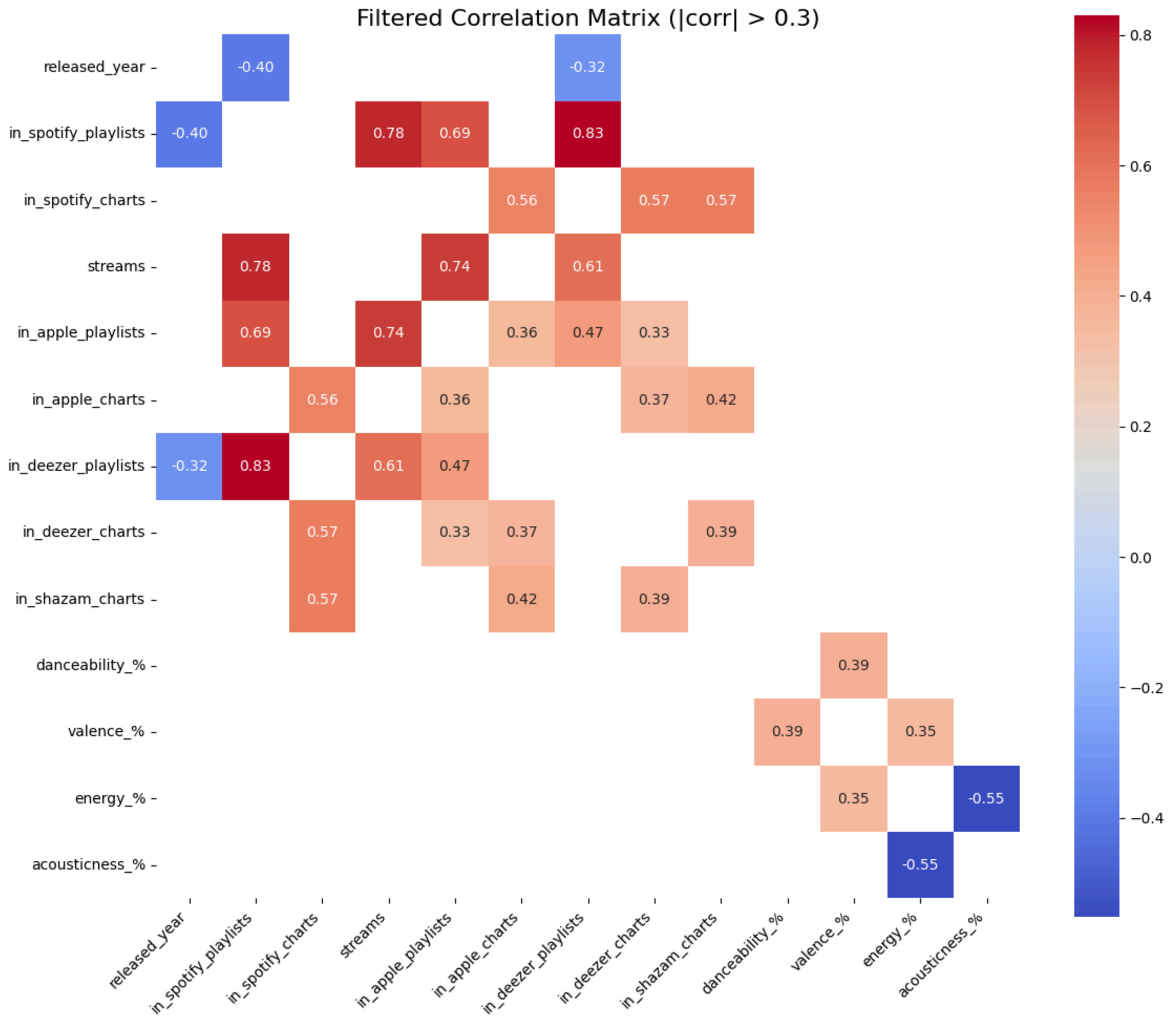
In [129…

```
correlation_matrix = df[[x for x in df.keys() if x not in CATEGORICAL_FEATURES]]

plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", cbar=Tru
plt.title("Correlation Matrix of Numerical Features", fontsize=16)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



In [130…

```
filtered_corr = correlation_matrix[((correlation_matrix > 0.3) | (correlation_ma
import numpy as np
np.fill_diagonal(filtered_corr.values, np.nan)  # Replace diagonal with NaN to i
filtered_corr = filtered_corr.dropna(how='all').dropna(axis=1, how='all')  # Rem

# Plot heatmap for filtered correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(filtered_corr, annot=True, fmt=".2f", cmap="coolwarm", cbar=True, sq
plt.title("Filtered Correlation Matrix (|corr| > 0.3)", fontsize=16)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



The filtered correlation matrix reveals a clear relationship between the number of streams and the song's presence on playlists across various music streaming platforms.

The charts between platforms also appear to have some correlation between them.

Newer songs tend to be in fewer playlists.

There is an inverse relation between energy and acousticness.

Shapiro-Wilk

In [131…

```
print('Numerical features that come from a normal distribution (Shapiro-Wilk tes
for key in [x for x in df.keys() if x not in CATEGORICAL_FEATURES]:
    s,p = stats.shapiro(df[key])
    if p > 0.05:
        print(key)
```

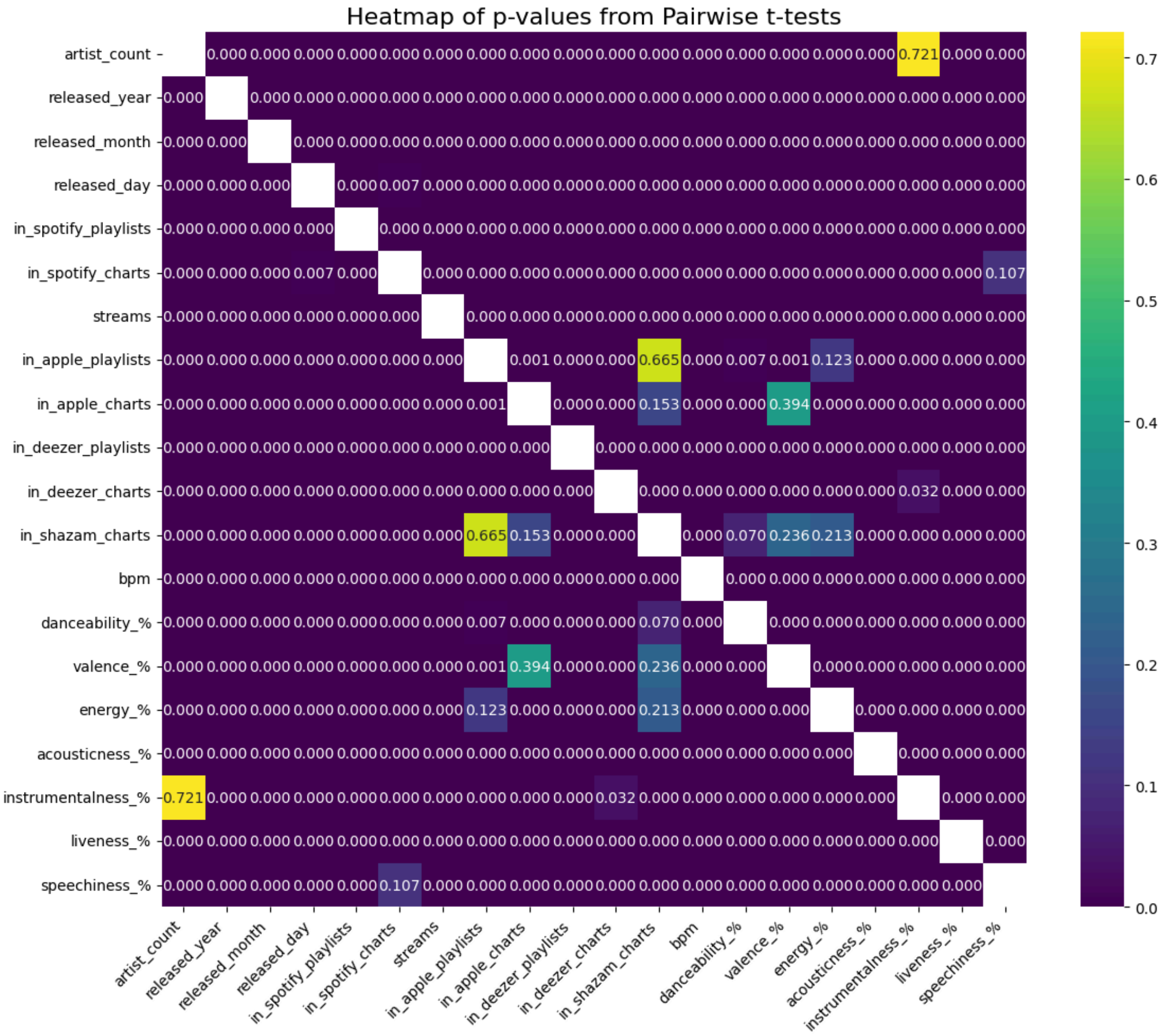
```
Numerical features that come from a normal distribution (Shapiro-Wilk test):
```

The Shapiro-Wilk test concluded that none of the features come from a normal distribution.

T-test

In [132…

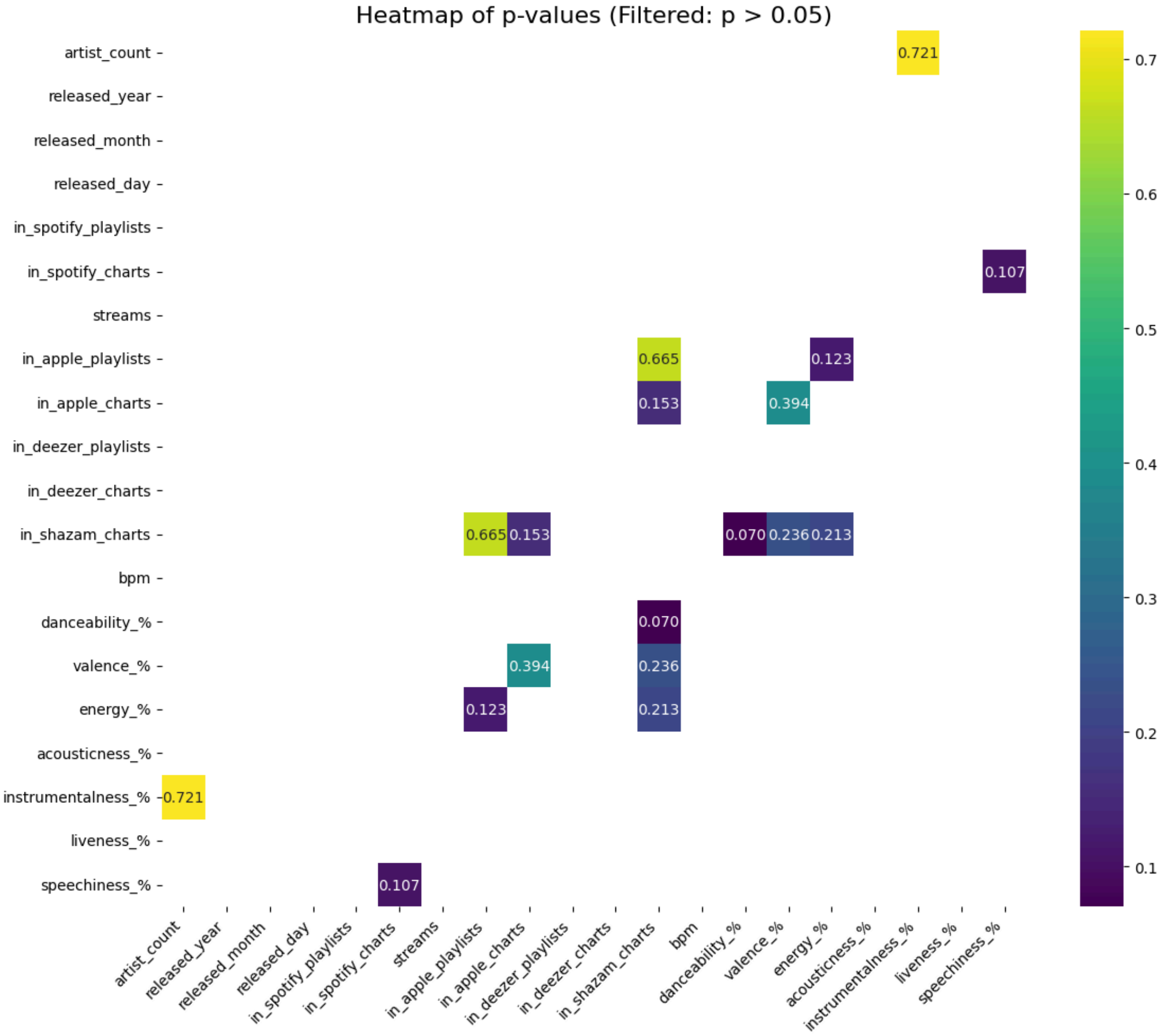
```
import itertools

p_value_matrix = pd.DataFrame(np.nan, index=NUMERICAL_FEATURES, columns=NUMERICA

for pair in itertools.combinations(NUMERICAL_FEATURES, 2):
    _, p_value = stats.ttest_ind(df[pair[0]].dropna(), df[pair[1]].dropna())
    p_value_matrix.loc[pair[0], pair[1]] = p_value
    p_value_matrix.loc[pair[1], pair[0]] = p_value

plt.figure(figsize=(12, 10))
sns.heatmap(p_value_matrix, annot=True, fmt=".3f", cmap="viridis", cbar=True)
plt.title("Heatmap of p-values from Pairwise t-tests", fontsize=16)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



In [133…

```
mask = p_value_matrix <= 0.05
```

```
# Heatmap visualization with filtering
plt.figure(figsize=(12, 10))
sns.heatmap(p_value_matrix, annot=True, fmt=".3f", cmap="viridis", cbar=True, ma
plt.title("Heatmap of p-values (Filtered: p > 0.05)", fontsize=16)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



From the filtered heatmap, we can see the pairs of features where their means are not significantly different (p-value > 0.05). However, this result does not provide meaningful insights, as we've already observed that our features do not follow a normal distribution. Additionally, many of the feature pairs with p-values greater than 0.05 measure entirely different aspects, making the comparison irrelevant in these cases.

The exception to this is the pair 'in_apple_charts' and 'in_shazam_charts,' as they measure similar aspects. In this case, we can infer a correlation between the two.

Chi-square

In [134…

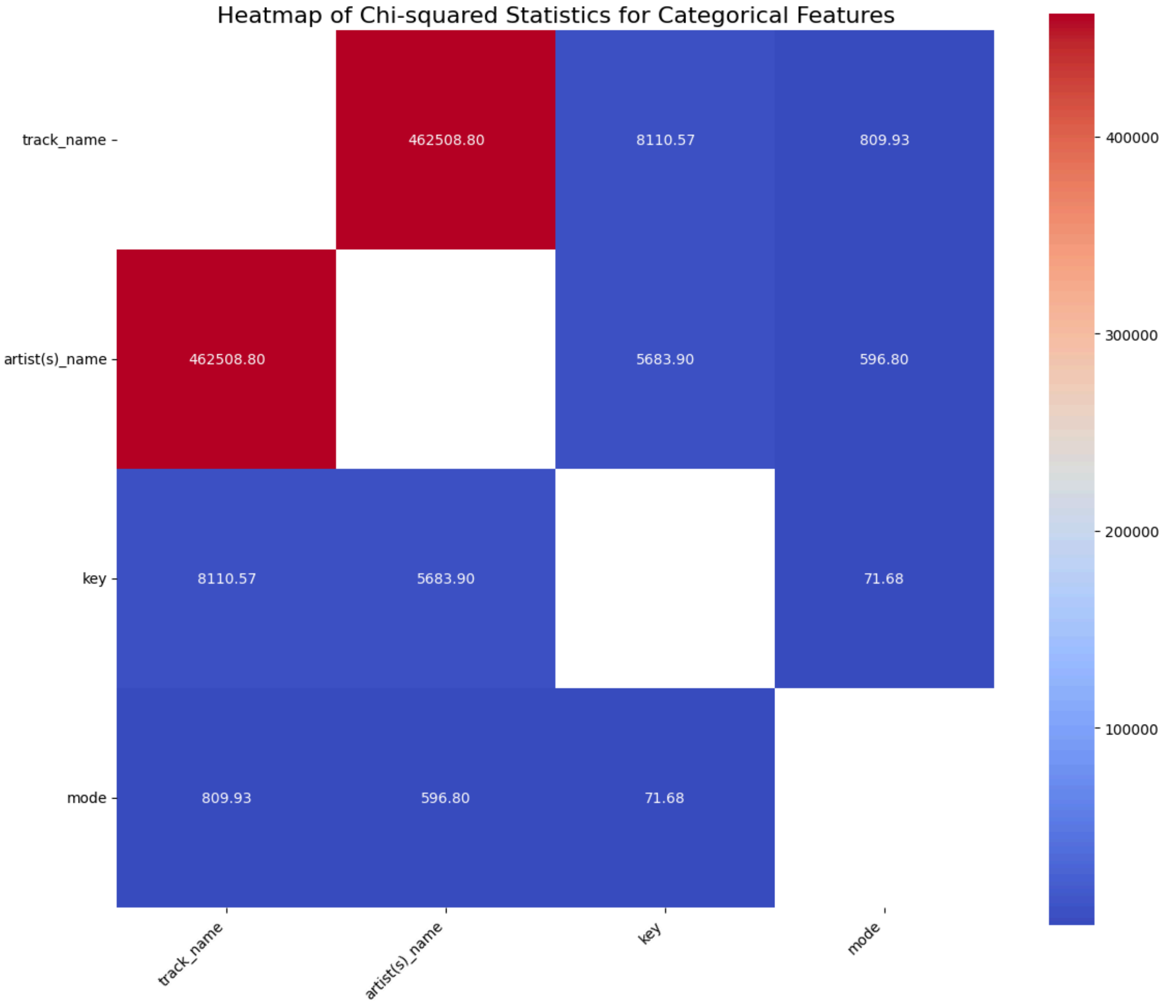
```
chi2_matrix = pd.DataFrame(np.nan, index=CATEGORICAL_FEATURES, columns=CATEGORIC

for pair in itertools.combinations(CATEGORICAL_FEATURES, 2):
    contingency_table = pd.crosstab(df[pair[0]], df[pair[1]])

    chi2, p, dof, expected = stats.chi2_contingency(contingency_table)

    # Store the chi-squared statistic in the matrix
    chi2_matrix.loc[pair[0], pair[1]] = chi2
    chi2_matrix.loc[pair[1], pair[0]] = chi2  # Symmetric matrix

    print(f'Features analyzed: {pair[0]}, {pair[1]}')
    print(f"Chi-squared statistic: {chi2}")
    print(f"P-value: {p}")
    print(f"Degrees of freedom: {dof}")
    print("Expected frequencies:")
    print(expected)
```

```
Features analyzed: track_name, artist(s)_name
Chi-squared statistic: 462508.8000000003
P-value: 0.01138607054588001
Degrees of freedom: 460321
Expected frequencies:
[[0.00245098 0.00122549 0.00122549 ... 0.00122549 0.00122549 0.00122549]
 [0.00245098 0.00122549 0.00122549 ... 0.00122549 0.00122549 0.00122549]
 [0.00245098 0.00122549 0.00122549 ... 0.00122549 0.00122549 0.00122549]
 ...
 [0.00245098 0.00122549 0.00122549 ... 0.00122549 0.00122549 0.00122549]
 [0.00245098 0.00122549 0.00122549 ... 0.00122549 0.00122549 0.00122549]
 [0.00245098 0.00122549 0.00122549 ... 0.00122549 0.00122549 0.00122549]]
Features analyzed: track_name, key
Chi-squared statistic: 8110.565955783348
P-value: 0.4337698366423736
Degrees of freedom: 8090
Expected frequencies:
[[0.08578431 0.06740196 0.09436275 ... 0.08455882 0.11151961 0.10416667]
 [0.08578431 0.06740196 0.09436275 ... 0.08455882 0.11151961 0.10416667]
 [0.08578431 0.06740196 0.09436275 ... 0.08455882 0.11151961 0.10416667]
 ...
 [0.08578431 0.06740196 0.09436275 ... 0.08455882 0.11151961 0.10416667]
 [0.08578431 0.06740196 0.09436275 ... 0.08455882 0.11151961 0.10416667]
 [0.08578431 0.06740196 0.09436275 ... 0.08455882 0.11151961 0.10416667]]
Features analyzed: track_name, mode
Chi-squared statistic: 809.9326063845944
P-value: 0.4841464987542653
Degrees of freedom: 809
Expected frequencies:
[[0.55269608 0.44730392]
 [0.55269608 0.44730392]
 [0.55269608 0.44730392]
 ...
 [0.55269608 0.44730392]
 [0.55269608 0.44730392]
 [0.55269608 0.44730392]]
Features analyzed: artist(s)_name, key
Chi-squared statistic: 5683.896970022894
P-value: 0.5203295840842688
Degrees of freedom: 5690
Expected frequencies:
[[0.17156863 0.13480392 0.18872549 ... 0.16911765 0.22303922 0.20833333]
 [0.08578431 0.06740196 0.09436275 ... 0.08455882 0.11151961 0.10416667]
 [0.08578431 0.06740196 0.09436275 ... 0.08455882 0.11151961 0.10416667]
 ...
 [0.08578431 0.06740196 0.09436275 ... 0.08455882 0.11151961 0.10416667]
 [0.08578431 0.06740196 0.09436275 ... 0.08455882 0.11151961 0.10416667]
 [0.08578431 0.06740196 0.09436275 ... 0.08455882 0.11151961 0.10416667]]
Features analyzed: artist(s)_name, mode
Chi-squared statistic: 596.7978126959621
P-value: 0.20308313408815176
Degrees of freedom: 569
Expected frequencies:
[[1.10539216 0.89460784]
 [0.55269608 0.44730392]
 [0.55269608 0.44730392]
 ...
 [0.55269608 0.44730392]
 [0.55269608 0.44730392]
 [0.55269608 0.44730392]]
```

```
Features analyzed: key, mode
Chi-squared statistic: 71.67721221043163
P-value: 2.1012601936671123e-11
Degrees of freedom: 10
Expected frequencies:
[[38.68872549 31.31127451]
 [30.39828431 24.60171569]
 [42.55759804 34.44240196]
 [63.56004902 51.43995098]
 [43.11029412 34.88970588]
 [16.58088235 13.41911765]
 [32.60906863 26.39093137]
 [48.08455882 38.91544118]
 [38.13602941 30.86397059]
 [50.29534314 40.70465686]
 [46.97916667 38.02083333]]
```

In [135…

```python
# Heatmap visualization for Chi-squared statistics
plt.figure(figsize=(12, 10))
sns.heatmap(chi2_matrix, annot=True, fmt=".2f", cmap="coolwarm", cbar=True, squa
plt.title("Heatmap of Chi-squared Statistics for Categorical Features", fontsize
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



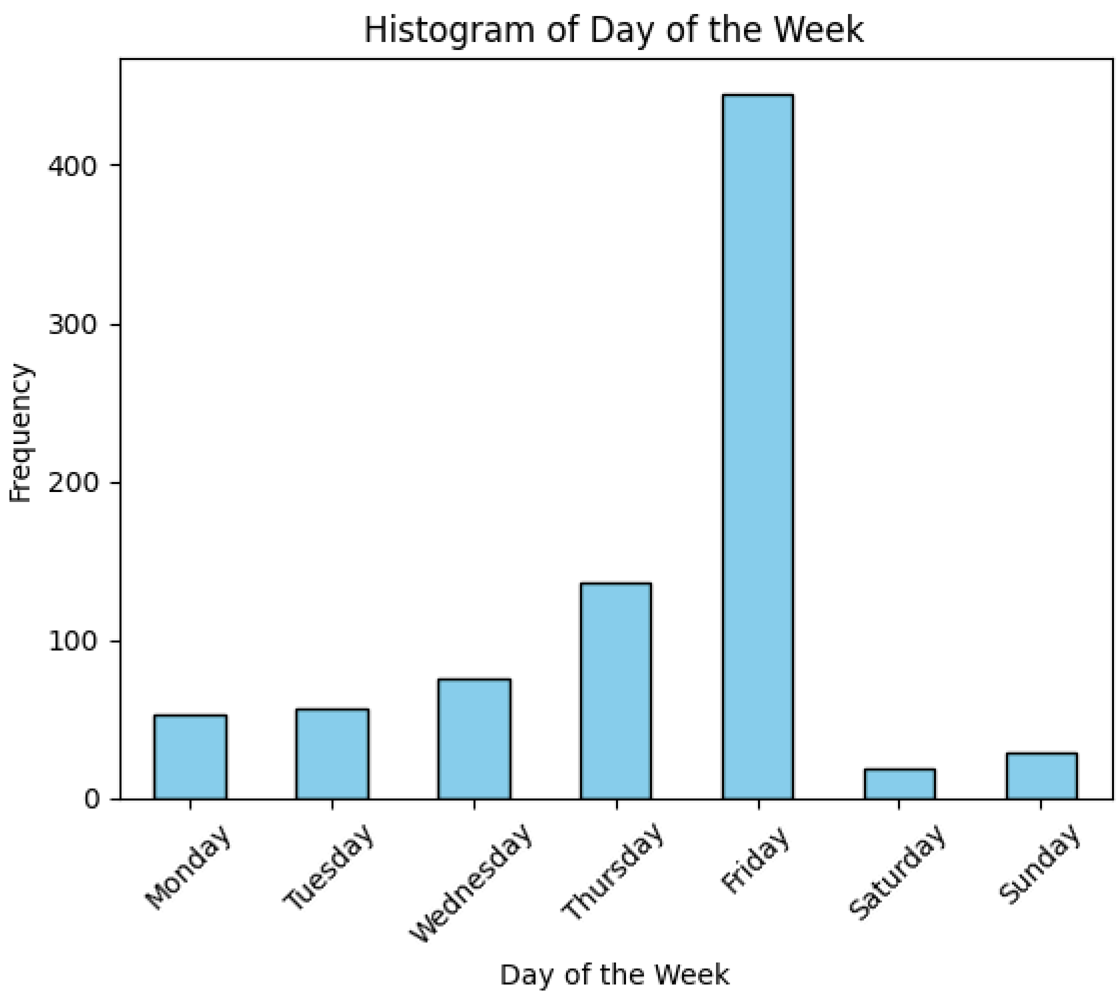
The track name appears to be related with the key and the mode of the song. The same is valid for the artist name.

New features

In [136…
```
import datetime
def get_day_of_week(year, month, day):
    try:
        date = datetime.date(year, month, day)
        return date.strftime("%A")
    except ValueError:
        return None
df['day_of_week'] = df.apply(lambda row: get_day_of_week(row['released_year'], r
print(df['day_of_week'])
```

```
0        Friday
1      Thursday
2        Friday
3        Friday
4      Thursday
         ...
948    Thursday
949      Friday
950    Thursday
951    Thursday
952      Friday
Name: day_of_week, Length: 816, dtype: object
```

In [137…
```
ordered_days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturda

df['day_of_week'] = pd.Categorical(df['day_of_week'], categories=ordered_days, c

df['day_of_week'].value_counts().sort_index().plot(kind='bar', color='skyblue',

# Customize the plot
plt.xlabel('Day of the Week')
plt.ylabel('Frequency')
plt.title('Histogram of Day of the Week')
plt.xticks(rotation=45)
plt.show()
```



A new Categorical feature was created based on which day the week each song was released. This feature can be considered ordinal. We can see that a high number of songs where released on fridays.

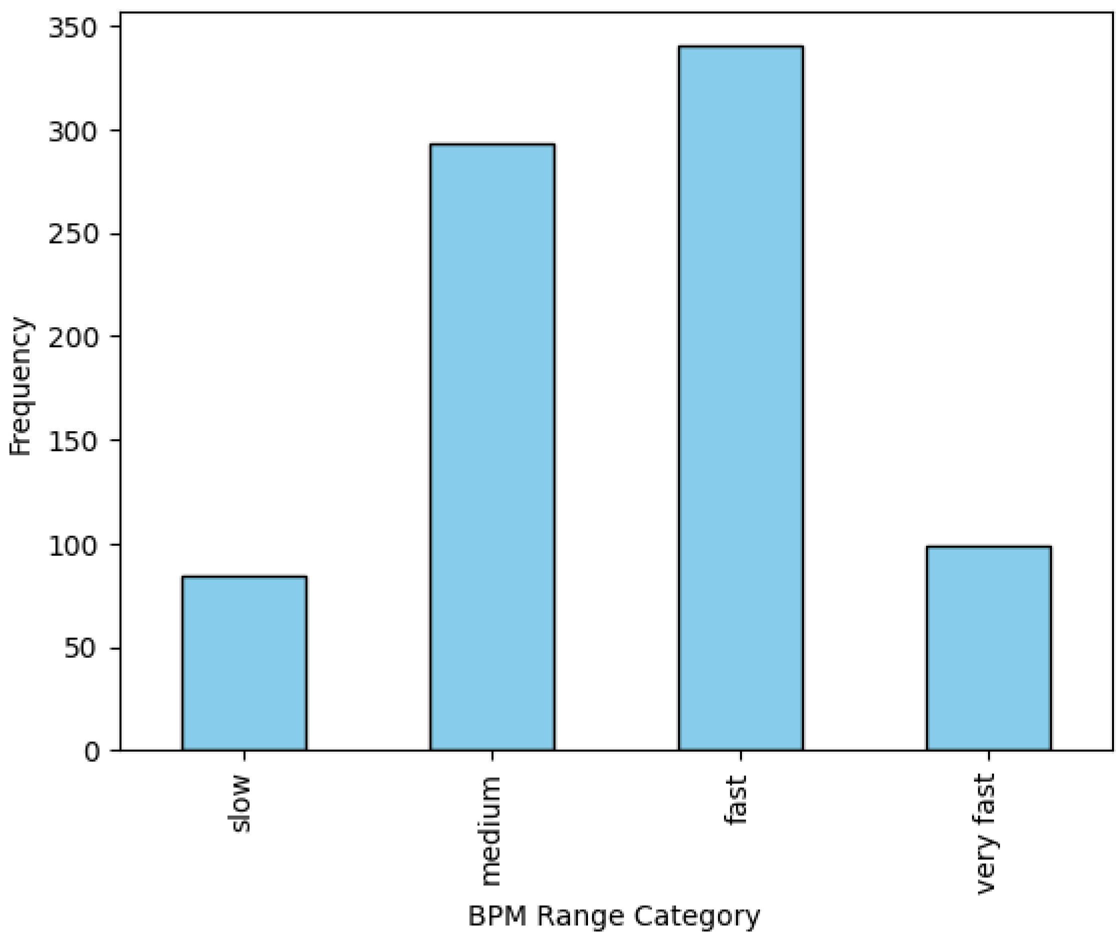
In [138…

```
def categorize_bpm(bpm):
    if bpm < 90:
        return 'slow'
    elif bpm < 120:
        return 'medium'
    elif bpm < 160:
        return 'fast'
    else:
        return 'very fast'

df['BPM Range Category'] = df['bpm'].apply(lambda x: categorize_bpm(x))
print(df['BPM Range Category'])
```

```
0           fast
1         medium
2           fast
3      very fast
4           fast
         ...
948         fast
949    very fast
950       medium
951       medium
952       medium
Name: BPM Range Category, Length: 816, dtype: object
```

In [139…

```
ordered_bpm = ['slow','medium','fast','very fast']

df['BPM Range Category'] = pd.Categorical(df['BPM Range Category'], categories=c

df['BPM Range Category'].value_counts().sort_index().plot(kind='bar', color='sky

# Customize the plot
plt.xlabel('BPM Range Category')
plt.ylabel('Frequency')
plt.show()
```



A new categorical feature was created classifying each song based on their beats per minute. This new feature can be considered ordinal.