```python
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
import statsmodels.api as sm
from scipy.stats import pearsonr
from scipy import stats
from sklearn.preprocessing import MinMaxScaler

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))


# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/top-spotify-songs-2023/spotify-2023.csv
```

What Does It Take to Hit the Charts

Table of Contents

1. Introduction
2. Data Cleaning
3. Data Analysis & Visualization
4. How to hit the charts on Spotify

** Key findings**

- Mode, Key and BPM play a bigger role than the rest.
- "Mode: Major / Minor" plays an important role for the success of a song.

BPM matters, and not slowing down too much is the key. It has been found:

- Best range is between 110 and 90.
- Overall average is 122.
- Optimal range: 78 to 45

Introduction The dataset contains lists of the most streamed songs from the music streaming app Spotify for the year 2023. This dataset contains BPM, musical keys, mode, and other variables.

bpm: Beats per Minute, a measure of song tempo key: key of the song mode: mode of the song (Major or Minor) danceability_%: percentage indicating how suitable the song is for dancing valence_%: positivity of the song's musical content energy_%: perceived energy level of the song acousticness_%: amount of acoustic sound in the song instrumentalness_%: amount of instrumental content in the song liveness_%: presence of live performance elements speechiness_%: amount of spoken words in the song

Data Cleaning Object & Scope

- Collect, clean & analyze Spotify dataset
- Identify variable of interest
- Share findings and insights

Methodology

- Collect data using "top-spotify-songs-2023"
- Wrangle data using preprocessing, cleaning, transforming and organizing data for further analysis including predictive.
- Explore data using available techniques
- Visualize data to uncover insightful discoveries

```
df = pd.read_csv('/kaggle/input/top-spotify-songs-2023/spotify-2023.csv',encoding = 'ISO8859-1')
df.tail()
```

```
                  track_name      artist(s)_name  artist_count  \
948             My Mind & Me        Selena Gomez             1
949  Bigger Than The Whole Sky        Taylor Swift             1
950      A Veces (feat. Feid)  Feid, Paulo Londra             2
951            En La De Ella  Feid, Sech, Jhayco             3
952                    Alone           Burna Boy             1

     released_year  released_month  released_day  in_spotify_playlists
\
948           2022              11             3                   953

949           2022              10            21                  1180

950           2022              11             3                   573

951           2022              10            20                  1320

952           2022              11             4                   782
```

```
     in_spotify_charts    streams  in_apple_playlists  ...  bpm key
mode  \
948                  0   91473363                  61  ...  144   A
Major
949                  0  121871870                   4  ...  166  F#
Major
950                  0   73513683                   2  ...   92  C#
Major
951                  0  133895612                  29  ...   97  C#
Major
952                  2   96007391                  27  ...   90   E
Minor

    danceability_%  valence_% energy_% acousticness_%
instrumentalness_%  \
948             60         24       39             57
0
949             42          7       24             83
1
950             80         81       67              4
0
951             82         67       77              8
0
952             61         32       67             15
0

     liveness_%  speechiness_%
948           8              3
949          12              6
950           8              6
951          12              5
952          11              5

[5 rows x 24 columns]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 953 entries, 0 to 952
Data columns (total 24 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   track_name            953 non-null    object
 1   artist(s)_name        953 non-null    object
 2   artist_count          953 non-null    int64
 3   released_year         953 non-null    int64
 4   released_month        953 non-null    int64
 5   released_day          953 non-null    int64
```

```
 6   in_spotify_playlists  953 non-null    int64
 7   in_spotify_charts     953 non-null    int64
 8   streams               953 non-null    object
 9   in_apple_playlists    953 non-null    int64
 10  in_apple_charts       953 non-null    int64
 11  in_deezer_playlists   953 non-null    object
 12  in_deezer_charts      953 non-null    int64
 13  in_shazam_charts      903 non-null    object
 14  bpm                   953 non-null    int64
 15  key                   858 non-null    object
 16  mode                  953 non-null    object
 17  danceability_%        953 non-null    int64
 18  valence_%             953 non-null    int64
 19  energy_%              953 non-null    int64
 20  acousticness_%        953 non-null    int64
 21  instrumentalness_%    953 non-null    int64
 22  liveness_%            953 non-null    int64
 23  speechiness_%         953 non-null    int64
dtypes: int64(17), object(7)
memory usage: 178.8+ KB
```

```
df.columns
```

```
Index(['track_name', 'artist(s)_name', 'artist_count',
'released_year',
       'released_month', 'released_day', 'in_spotify_playlists',
       'in_spotify_charts', 'streams', 'in_apple_playlists',
'in_apple_charts',
       'in_deezer_playlists', 'in_deezer_charts', 'in_shazam_charts',
'bpm',
       'key', 'mode', 'danceability_%', 'valence_%', 'energy_%',
       'acousticness_%', 'instrumentalness_%', 'liveness_%',
'speechiness_%'],
      dtype='object')
```

```
df.describe()
```

```
       artist_count  released_year  released_month  released_day  \
count    953.000000     953.000000      953.000000    953.000000
mean       1.556139    2018.238195        6.033578     13.930745
std        0.893044      11.116218        3.566435      9.201949
min        1.000000    1930.000000        1.000000      1.000000
25%        1.000000    2020.000000        3.000000      6.000000
50%        1.000000    2022.000000        6.000000     13.000000
75%        2.000000    2022.000000        9.000000     22.000000
max        8.000000    2023.000000       12.000000     31.000000

       in_spotify_playlists  in_spotify_charts  in_apple_playlists  \
count            953.000000         953.000000          953.000000
mean            5200.124869          12.009444           67.812172
```

```
std         7897.608990         19.575992         86.441493
min           31.000000          0.000000          0.000000
25%          875.000000          0.000000         13.000000
50%         2224.000000          3.000000         34.000000
75%         5542.000000         16.000000         88.000000
max        52898.000000        147.000000        672.000000

       in_apple_charts  in_deezer_charts         bpm  danceability_
%  \
count       953.000000        953.000000  953.000000        953.00000

mean         51.908709          2.666317  122.540399         66.96957

std          50.630241          6.035599   28.057802         14.63061

min           0.000000          0.000000   65.000000         23.00000

25%           7.000000          0.000000  100.000000         57.00000

50%          38.000000          0.000000  121.000000         69.00000

75%          87.000000          2.000000  140.000000         78.00000

max         275.000000         58.000000  206.000000         96.00000


        valence_%    energy_%  acousticness_%  instrumentalness_%
liveness_%  \
count  953.000000  953.000000      953.000000          953.000000
953.000000
mean    51.431270   64.279119       27.057712            1.581322
18.213012
std     23.480632   16.550526       25.996077            8.409800
13.711223
min      4.000000    9.000000        0.000000            0.000000
3.000000
25%     32.000000   53.000000        6.000000            0.000000
10.000000
50%     51.000000   66.000000       18.000000            0.000000
12.000000
75%     70.000000   77.000000       43.000000            0.000000
24.000000
max     97.000000   97.000000       97.000000           91.000000
97.000000

       speechiness_%
count     953.000000
mean       10.131165
std         9.912888
min         2.000000
```

```
25%           4.000000
50%           6.000000
75%          11.000000
max          64.000000
```

Data Cleaning

```
#This is to check if there are any missing values in the columns?
nan_values = df.isna()
any_missing_values = nan_values.any().any()

any_missing_values_in_column = nan_values.any()

missing_value_count = df.isnull().sum()

for column, has_missing in any_missing_values_in_column.items():
    if has_missing:
        count = missing_value_count[column]
        print(f"-----> Column '{column}' has {count} missing values.")

print("\nMissing Values in the Entire DataFrame?")
print(any_missing_values)

print("\nMissing Values in Each Column?")
print(any_missing_values_in_column)

print("\nMissing Value Counts in Each Column:")
print(missing_value_count)
```

```
-----> Column 'in_shazam_charts' has 50 missing values.
-----> Column 'key' has 95 missing values.

Missing Values in the Entire DataFrame?
True

Missing Values in Each Column?
track_name              False
artist(s)_name          False
artist_count            False
released_year           False
released_month          False
released_day            False
in_spotify_playlists    False
in_spotify_charts       False
streams                 False
in_apple_playlists      False
in_apple_charts         False
in_deezer_playlists     False
```

```
in_deezer_charts        False
in_shazam_charts         True
bpm                     False
key                      True
mode                    False
danceability_%          False
valence_%               False
energy_%                False
acousticness_%          False
instrumentalness_%      False
liveness_%              False
speechiness_%           False
dtype: bool

Missing Value Counts in Each Column:
track_name               0
artist(s)_name           0
artist_count             0
released_year            0
released_month           0
released_day             0
in_spotify_playlists     0
in_spotify_charts        0
streams                  0
in_apple_playlists       0
in_apple_charts          0
in_deezer_playlists      0
in_deezer_charts         0
in_shazam_charts        50
bpm                      0
key                     95
mode                     0
danceability_%           0
valence_%                0
energy_%                 0
acousticness_%           0
instrumentalness_%       0
liveness_%               0
speechiness_%            0
dtype: int64

# based on the above results, there are many missing values in the 2
columns:
# 'keys' and 'in_shazams_charts'. These missing values could have
happenned in the
# data collecting process. In this case, we choose to fill all the
missig values with
# 'Invalid'
df['key'] = df['key'].fillna('Invalid')
df['in_shazam_charts'] = df['in_shazam_charts'].fillna('Invalid')
```

```
# After running these, all the missing value has been replaced with
"Invalid"

# Let's look at the dataset after cleaning (removing missing values)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 953 entries, 0 to 952
Data columns (total 24 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   track_name            953 non-null    object
 1   artist(s)_name        953 non-null    object
 2   artist_count          953 non-null    int64 
 3   released_year         953 non-null    int64 
 4   released_month        953 non-null    int64 
 5   released_day          953 non-null    int64 
 6   in_spotify_playlists  953 non-null    int64 
 7   in_spotify_charts     953 non-null    int64 
 8   streams               953 non-null    object
 9   in_apple_playlists    953 non-null    int64 
 10  in_apple_charts       953 non-null    int64 
 11  in_deezer_playlists   953 non-null    object
 12  in_deezer_charts      953 non-null    int64 
 13  in_shazam_charts      953 non-null    object
 14  bpm                   953 non-null    int64 
 15  key                   953 non-null    object
 16  mode                  953 non-null    object
 17  danceability_%        953 non-null    int64 
 18  valence_%             953 non-null    int64 
 19  energy_%              953 non-null    int64 
 20  acousticness_%        953 non-null    int64 
 21  instrumentalness_%    953 non-null    int64 
 22  liveness_%            953 non-null    int64 
 23  speechiness_%         953 non-null    int64 
dtypes: int64(17), object(7)
memory usage: 178.8+ KB
```

```
# here, there are 2 problems that the collected in 2 columns
#'streams' and 'in_deezer_playlists' turned out not to be inputted
with the right data type
# They should have been inputted as a number and the datatype of
columns should be either
# int64 or float64. To fix this, let's convert them into numeric data
types. In that process
# replace any non-numeric data with NA
df['streams'] = pd.to_numeric(df['streams'], errors='coerce')
df['in_deezer_playlists'] = pd.to_numeric(df['in_deezer_playlists'],
errors='coerce')
```

```python
# Let's check the info of the dataset again
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 953 entries, 0 to 952
Data columns (total 24 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   track_name            953 non-null    object 
 1   artist(s)_name        953 non-null    object 
 2   artist_count          953 non-null    int64  
 3   released_year         953 non-null    int64  
 4   released_month        953 non-null    int64  
 5   released_day          953 non-null    int64  
 6   in_spotify_playlists  953 non-null    int64  
 7   in_spotify_charts     953 non-null    int64  
 8   streams               952 non-null    float64
 9   in_apple_playlists    953 non-null    int64  
 10  in_apple_charts       953 non-null    int64  
 11  in_deezer_playlists   874 non-null    float64
 12  in_deezer_charts      953 non-null    int64  
 13  in_shazam_charts      953 non-null    object 
 14  bpm                   953 non-null    int64  
 15  key                   953 non-null    object 
 16  mode                  953 non-null    object 
 17  danceability_%        953 non-null    int64  
 18  valence_%             953 non-null    int64  
 19  energy_%              953 non-null    int64  
 20  acousticness_%        953 non-null    int64  
 21  instrumentalness_%    953 non-null    int64  
 22  liveness_%            953 non-null    int64  
 23  speechiness_%         953 non-null    int64  
dtypes: float64(2), int64(17), object(5)
memory usage: 178.8+ KB
```

```python
# The 2 columns now are back to the right data type.
# Last but not least, Let's check if there is any duplicate in the
data set
df.duplicated().sum()
# If there is no duplicate the output should be zero
```

```
0
```

Data Analysis & Visualization

```python
# First, Take a look at the statistics of the data set
df.describe()
```

```
       artist_count  released_year  released_month  released_day  \
count    953.000000     953.000000      953.000000    953.000000   
mean       1.556139    2018.238195        6.033578     13.930745   
std        0.893044      11.116218        3.566435      9.201949   
min        1.000000    1930.000000        1.000000      1.000000   
25%        1.000000    2020.000000        3.000000      6.000000   
50%        1.000000    2022.000000        6.000000     13.000000   
75%        2.000000    2022.000000        9.000000     22.000000   
max        8.000000    2023.000000       12.000000     31.000000   

       in_spotify_playlists  in_spotify_charts       streams  \
count            953.000000         953.000000  9.520000e+02   
mean            5200.124869          12.009444  5.141374e+08   
std             7897.608990          19.575992  5.668569e+08   
min               31.000000           0.000000  2.762000e+03   
25%              875.000000           0.000000  1.416362e+08   
50%             2224.000000           3.000000  2.905309e+08   
75%             5542.000000          16.000000  6.738690e+08   
max            52898.000000         147.000000  3.703895e+09   

       in_apple_playlists  in_apple_charts  in_deezer_playlists  \
count          953.000000       953.000000           874.000000   
mean            67.812172        51.908709           109.740275   
std             86.441493        50.630241           178.811406   
min              0.000000         0.000000             0.000000   
25%             13.000000         7.000000            12.000000   
50%             34.000000        38.000000            36.500000   
75%             88.000000        87.000000           110.000000   
max            672.000000       275.000000           974.000000   

       in_deezer_charts         bpm  danceability_%   valence_%  
energy_%  \
count        953.000000  953.000000       953.00000  953.000000  
953.000000   
mean           2.666317  122.540399        66.96957   51.431270  
64.279119   
std            6.035599   28.057802        14.63061   23.480632  
16.550526   
min            0.000000   65.000000        23.00000    4.000000  
9.000000   
25%            0.000000  100.000000        57.00000   32.000000  
53.000000   
50%            0.000000  121.000000        69.00000   51.000000  
66.000000   
75%            2.000000  140.000000        78.00000   70.000000  
77.000000   
max           58.000000  206.000000        96.00000   97.000000  
97.000000   

       acousticness_%  instrumentalness_%  liveness_%  speechiness_%  
```

```
count      953.000000          953.000000  953.000000     953.000000
mean        27.057712            1.581322   18.213012      10.131165
std         25.996077            8.409800   13.711223       9.912888
min          0.000000            0.000000    3.000000       2.000000
25%          6.000000            0.000000   10.000000       4.000000
50%         18.000000            0.000000   12.000000       6.000000
75%         43.000000            0.000000   24.000000      11.000000
max         97.000000           91.000000   97.000000      64.000000
```

Release Date

- The year of release of the latest song in the dataset is (of course) 2023, with the oldest dating back to 1930.

Popularity

- Some of the most streamed songs never landed in the charts.
- The most number of streams is 3,703,895,000, while the least number of streams is 2,762.
- The mean number of streams is 514,137,400 with a standard deviation of 566,856,900, which indicates that the number of streams varies by a huge value.

Song Properties / Characteristics

- The highest and lowest BPM is 206 and 65, respectively, with a mean value of 122.55.
- The highest and lowest danceability is 96% and 23%, respectively, with a mean value of 66.98%.
- The highest and lowest valence is 97% and 4%, respectively, with a mean value of 51.41%.
- The highest and lowest energy is 97% and 9%, respectively, with a mean value of 64.27%.
- The highest and lowest acousticness is 97% and 0%, respectively, with a mean value of 27.08%.
- The highest and lowest instrumentalness is 91% and 0%, respectively, with a mean value of 1.58%.
- The highest and lowest liveness is 97% and 3%, respectively, with a mean value of 18.21%.
- The highest and lowest speechiness is 64% and 2%, respectively, with a mean value of 10.14%.

How to hit the charts on Spotify

```
## There are multilpe relatable factors on the table that can be used
to analyze and find out what help a song hit the charts.
# We need to set the targetted dataframe apart from the original
database.
# Possible factors can be list as: 'bpm','key','mode', 'danceability_
```

```
%', 'valence_%', 'energy_%','acousticness_%', 'instrumentalness_%',
'liveness_%', 'speechiness_%'
# So we need to create a sub dataframe with just these columns to
examine the numbers and data
# First, we may want to change the unneccessarily long track names
into only tracks ID
track_name_to_id = {name: idx for idx, name in
enumerate(df['track_name'].unique())}
df['track_id'] = df['track_name'].map(track_name_to_id) # map the
track names to the track IDs
df.drop(columns=['track_name'], inplace=True) # replace the track
names with the track IDs

# create a list storing all the names of new columns in the sub
dataframe
voi_col = ['track_id', 'streams','bpm', 'key', 'mode', 'danceability_
%', 'valence_%', 'energy_%', 'acousticness_%', 'instrumentalness_%',
'liveness_%', 'speechiness_%']
# create a new sub dataframe with only columns matched with above
lists
df_voi = df[voi_col]
df_voi
```

```
     track_id      streams  bpm key   mode  danceability_%  valence_%
\
0           0  141381703.0  125   B  Major              80         89

1           1  133716286.0   92  C#  Major              71         61

2           2  140003974.0  138   F  Major              51         32

3           3  800840817.0  170   A  Major              55         58

4           4  303236322.0  144   A  Minor              65         23

..        ...          ...  ... ..    ...             ...        ...

948       938   91473363.0  144   A  Major              60         24

949       939  121871870.0  166  F#  Major              42          7

950       940   73513683.0   92  C#  Major              80         81

951       941  133895612.0   97  C#  Major              82         67

952       942   96007391.0   90   E  Minor              61         32


     energy_%  acousticness_%  instrumentalness_%  liveness_%
speechiness_%
0          83              31                   0           8
```

```
4
1          74              7                 0         10
4
2          53             17                 0         31
6
3          72             11                 0         11
15
4          80             14                63         11
6
..        ...            ...               ...        ...
...
948        39             57                 0          8
3
949        24             83                 1         12
6
950        67              4                 0          8
6
951        77              8                 0         12
5
952        67             15                 0         11
5

[953 rows x 12 columns]
```

```
# Since key and mode contain non-numeric data so we need to modify 
these columns
# for analyzing. One of the ways to solve this is to change these into 
binary numbers
# by utilizing the get_dummies function.

#label encoding or one hot encoding?
df_features = df_voi
df_features = pd.get_dummies(df_voi, columns=['key', 'mode'], 
prefix=['key', 'mode'])

#rename columns of interests
re_col = {
    'danceability_%': 'danceability',
    'valence_%': 'valence',
    'energy_%': 'energy',
    'acousticness_%': 'acousticness',
    'instrumentalness_%': 'instrumentalness',
    'liveness_%': 'liveness',
    'speechiness_%': 'speechiness'
}

df_features.rename(columns=re_col, inplace=True)
df_features.head()
```

```
   track_id      streams  bpm  danceability  valence  energy  
acousticness  \
0         0  141381703.0  125            80       89      83  
31  
1         1  133716286.0   92            71       61      74  
7  
2         2  140003974.0  138            51       32      53  
17  
3         3  800840817.0  170            55       58      72  
11  
4         4  303236322.0  144            65       23      80  
14  

   instrumentalness  liveness  speechiness  ...  key_D  key_D#  key_E  
key_F  \
0                 0         8            4  ...  False   False  False  
False  
1                 0        10            4  ...  False   False  False  
False  
2                 0        31            6  ...  False   False  False  
True  
3                 0        11           15  ...  False   False  False  
False  
4                63        11            6  ...  False   False  False  
False  

   key_F#  key_G  key_G#  key_Invalid  mode_Major  mode_Minor  
0   False  False   False        False        True       False  
1   False  False   False        False        True       False  
2   False  False   False        False        True       False  
3   False  False   False        False        True       False  
4   False  False   False        False       False        True  

[5 rows x 24 columns]
```

```
# Here since the track_id column is not very useful, so we decided to 
drop it.
df_da = df_features.drop(columns=['track_id'])
df_da.head()
```

```
       streams  bpm  danceability  valence  energy  acousticness  \
0  141381703.0  125            80       89      83            31   
1  133716286.0   92            71       61      74             7   
2  140003974.0  138            51       32      53            17   
3  800840817.0  170            55       58      72            11   
4  303236322.0  144            65       23      80            14   

   instrumentalness  liveness  speechiness  key_A  ...  key_D  key_D#  
key_E  \
0                 0         8            4  False  ...  False   False  
```

```
False
1                    0        10              4  False  ...  False   False
False
2                    0        31              6  False  ...  False   False
False
3                    0        11             15   True  ...  False   False
False
4                   63        11              6   True  ...  False   False
False

   key_F  key_F#  key_G  key_G#  key_Invalid  mode_Major  mode_Minor
0  False   False  False   False        False        True       False
1  False   False  False   False        False        True       False
2   True   False  False   False        False        True       False
3  False   False  False   False        False        True       False
4  False   False  False   False        False       False        True

[5 rows x 23 columns]

# Binary columns contain all the keys and one of the 2 mode (either
major or minor)
# used in composing those songs on the charts. Since, we aim to
discover what to get a hit on top chart
# Figuring out facts about keys and modes may help. Now, with the
get_dummies function,
# all the data in these columns is either True or False. Let's convert
them into binary data,
# which contain either 1 or 0 value.

binary_columns = ['key_A', 'key_A#', 'key_B', 'key_C#', 'key_D',
'key_D#', 'key_E', 'key_F', 'key_F#', 'key_G', 'key_G#', 'mode_Major',
'mode_Minor']

for column in binary_columns:
    df_da[column] = df_da[column].astype(int)
print(df_da.head())

       streams  bpm  danceability  valence  energy  acousticness  \
0  141381703.0  125            80       89      83            31
1  133716286.0   92            71       61      74             7
2  140003974.0  138            51       32      53            17
3  800840817.0  170            55       58      72            11
4  303236322.0  144            65       23      80            14

   instrumentalness  liveness  speechiness  key_A  ...  key_D  key_D#
key_E  \
0                 0         8            4      0  ...      0       0
0
1                 0        10            4      0  ...      0       0
0
```

```
2                  0         31             6       0  ...      0       0
0
3                  0         11            15       1  ...      0       0
0
4                 63         11             6       1  ...      0       0
0

   key_F  key_F#  key_G  key_G#  key_Invalid  mode_Major  mode_Minor
0      0       0      0       0        False           1           0
1      0       0      0       0        False           1           0
2      1       0      0       0        False           1           0
3      0       0      0       0        False           1           0
4      0       0      0       0        False           0           1

[5 rows x 23 columns]

# now most of the columns in the dataframe is in numeric type with
numbers except Invalid or missing keys
# from the original dataframe. We can ignore that for now.
# Once again change all the data into float type to make sure
everything is in numeric before jumping in
# analyzing the data
df_da = df_da.astype(float)
df_da.head()

       streams    bpm  danceability  valence  energy  acousticness  \
0  141381703.0  125.0          80.0     89.0    83.0          31.0
1  133716286.0   92.0          71.0     61.0    74.0           7.0
2  140003974.0  138.0          51.0     32.0    53.0          17.0
3  800840817.0  170.0          55.0     58.0    72.0          11.0
4  303236322.0  144.0          65.0     23.0    80.0          14.0

   instrumentalness  liveness  speechiness  key_A  ...  key_D  key_D#
key_E  \
0               0.0       8.0          4.0    0.0  ...    0.0     0.0
0.0
1               0.0      10.0          4.0    0.0  ...    0.0     0.0
0.0
2               0.0      31.0          6.0    0.0  ...    0.0     0.0
0.0
3               0.0      11.0         15.0    1.0  ...    0.0     0.0
0.0
4              63.0      11.0          6.0    1.0  ...    0.0     0.0
0.0

   key_F  key_F#  key_G  key_G#  key_Invalid  mode_Major  mode_Minor
0    0.0     0.0    0.0     0.0          0.0         1.0         0.0
1    0.0     0.0    0.0     0.0          0.0         1.0         0.0
2    1.0     0.0    0.0     0.0          0.0         1.0         0.0
3    0.0     0.0    0.0     0.0          0.0         1.0         0.0
```

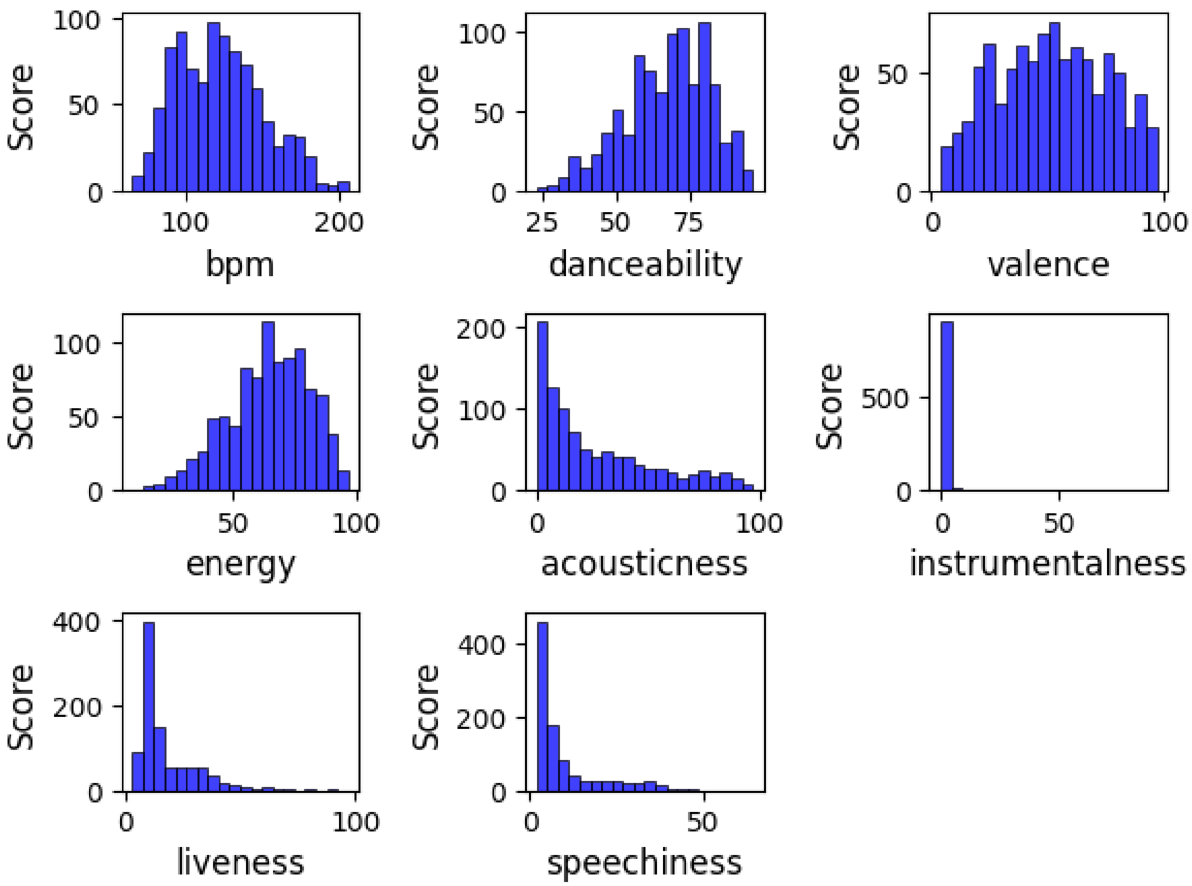
```
4    0.0     0.0    0.0     0.0            0.0         0.0         1.0

[5 rows x 23 columns]
```

```python
# By using the MixMaxScaler in SK learning library, we can normalize
the input features

scaler = MinMaxScaler()

df_da['streams'] = scaler.fit_transform(df_da[['streams']])
print(df_da.head())
```

```
    streams    bpm  danceability  valence  energy  acousticness  \
0  0.038170  125.0          80.0     89.0    83.0          31.0
1  0.036101   92.0          71.0     61.0    74.0           7.0
2  0.037798  138.0          51.0     32.0    53.0          17.0
3  0.216215  170.0          55.0     58.0    72.0          11.0
4  0.081869  144.0          65.0     23.0    80.0          14.0

   instrumentalness  liveness  speechiness  key_A  ...  key_D  key_D#
key_E  \
0               0.0       8.0          4.0    0.0  ...    0.0     0.0
0.0
1               0.0      10.0          4.0    0.0  ...    0.0     0.0
0.0
2               0.0      31.0          6.0    0.0  ...    0.0     0.0
0.0
3               0.0      11.0         15.0    1.0  ...    0.0     0.0
0.0
4              63.0      11.0          6.0    1.0  ...    0.0     0.0
0.0

   key_F  key_F#  key_G  key_G#  key_Invalid  mode_Major  mode_Minor
0    0.0     0.0    0.0     0.0          0.0         1.0         0.0
1    0.0     0.0    0.0     0.0          0.0         1.0         0.0
2    1.0     0.0    0.0     0.0          0.0         1.0         0.0
3    0.0     0.0    0.0     0.0          0.0         1.0         0.0
4    0.0     0.0    0.0     0.0          0.0         0.0         1.0

[5 rows x 23 columns]
```

```python
columns_to_plot = ['bpm', 'danceability', 'valence', 'energy',
'acousticness', 'instrumentalness', 'liveness', 'speechiness']

for i, column in enumerate(columns_to_plot, 1):
    plt.subplot(3, 3, i)
    sns.histplot(data=df_da, x=column, bins=20, color='blue')
    plt.xlabel(column, fontsize=12)
    plt.ylabel("Score", fontsize=12)
```

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



The chart illustrates the distribution of various music variables among the top 1000 most popular songs on Spotify in 2023. It comprises a set of frequency charts, with the x-axis representing music variables and the y-axis representing the frequency of songs with those values.

The most popular songs tend to have high danceability. This might be due to their upbeat and enjoyable nature, making them more appealing to listeners.

Additionally, the most popular songs often exhibit moderate energy levels, which could contribute to their listenability and appeal.

Danceability

The distribution of danceability resembles a bell curve, indicating that most songs have a moderate level of danceability around 70%. Some songs, however, have very high danceability exceeding 90%, typically falling within the dance or pop genres.

Energy

Similar to danceability, the distribution of energy also forms a bell curve, signifying that the majority of songs exhibit a moderate level of energy, approximately 60%. Some songs have higher energy levels surpassing 80%, often belonging to the rock or pop genres.

Valence

The valence distribution also resembles a bell curve, suggesting that most songs have a moderate valence around 60%. There are songs with higher valence exceeding 80%, commonly found in pop or hip-hop genres.

Acousticness

The distribution of acousticness skews towards the right, indicating a higher presence of songs with higher acousticness compared to lower acousticness. On average, songs have an acousticness level of around 30%. Some songs surpass 60% acousticness, typically belonging to folk or indie genres.

Instrumentalness

The distribution of instrumentalness forms a bell curve, showcasing that most songs have a moderate instrumentalness level around 50%. There are songs with higher instrumentalness surpassing 70%, often comprising instrumental or classical music.

Liveness

The liveness distribution also resembles a bell curve, suggesting that most songs have a moderate liveness level around 50%. Some songs exhibit higher liveness exceeding 70%, usually associated with rock or pop genres.

Speechiness

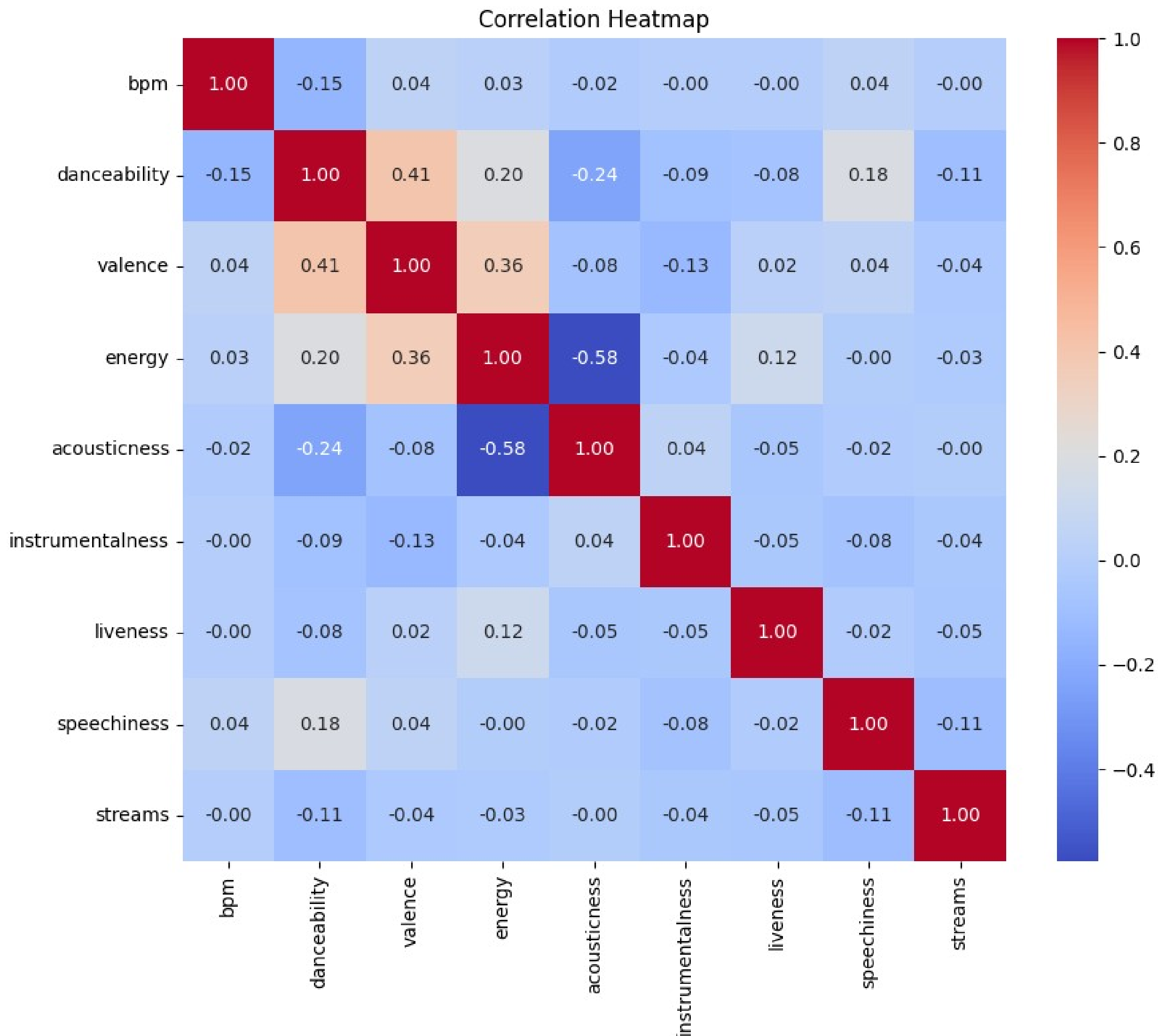
The speechiness distribution skews towards the left, indicating a higher presence of songs with lower speechiness compared to higher speechiness. On average, songs have a speechiness level of about 20%. Some songs exceed 40% speechiness, commonly found in rap or hip-hop genres.

Overall, the chart indicates that the most popular songs on Spotify tend to have high danceability and moderate energy levels. These songs also typically exhibit moderate valence, higher acousticness, moderate instrumentalness, moderate liveness, and lower speechiness.

```
# One of the best way to figure out the correlation and relationship between the features
# in the dataframe. We use corr() function to create the correlation matrix between all the columns
columns_to_correlate = ['bpm', 'danceability', 'valence', 'energy', 'acousticness', 'instrumentalness', 'liveness', 'speechiness', 'streams']

correlation_matrix = df_da[columns_to_correlate].corr()
print(correlation_matrix)
```

```
                        bpm  danceability   valence    energy
acousticness  \
```

```
bpm                  1.000000     -0.147095  0.041195  0.025794      -
0.017694
danceability        -0.147095      1.000000  0.408451  0.198095      -
0.236165
valence              0.041195      0.408451  1.000000  0.357612      -
0.081907
energy               0.025794      0.198095  0.357612  1.000000      -
0.577344
acousticness        -0.017694     -0.236165 -0.081907 -0.577344
1.000000
instrumentalness -0.001195     -0.089138 -0.132890 -0.038547
0.042796
liveness            -0.000761     -0.077538  0.021278  0.117302      -
0.050142
speechiness          0.039260      0.184977  0.041081 -0.004846      -
0.022501
streams             -0.002438     -0.105457 -0.040831 -0.026051      -
0.004485

                  instrumentalness  liveness  speechiness   streams
bpm                      -0.001195 -0.000761     0.039260 -0.002438
danceability             -0.089138 -0.077538     0.184977 -0.105457
valence                  -0.132890  0.021278     0.041081 -0.040831
energy                   -0.038547  0.117302    -0.004846 -0.026051
acousticness              0.042796 -0.050142    -0.022501 -0.004485
instrumentalness          1.000000 -0.045967    -0.083396 -0.044902
liveness                 -0.045967  1.000000    -0.022525 -0.048337
speechiness              -0.083396 -0.022525     1.000000 -0.112333
streams                  -0.044902 -0.048337    -0.112333  1.000000

# Next, take the correlation matrix between all the feature to draw a
heat map

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f", square=True)

plt.title("Correlation Heatmap")
plt.show()
```



The correlation heatmap illustrates the relationship between various music variables among the top 1000 most popular songs on Spotify in 2023. The heatmap employs a color scale to represent the strength of correlation, where shades of blue indicate a positive correlation and shades of red denote a negative correlation.

The most strongly correlated music variables are danceability and valence. This might be due to songs with high danceability often possessing high valence as well. For instance, dance songs commonly exhibit both of these characteristics.

Conversely, the weakest correlation exists between Acousticness and Energy. This could be because high Acousticness doesn't necessarily imply high energy. For example, acoustic songs might have low energy.

Danceability and Valence

The correlation between danceability and valence is very strong, with a correlation coefficient of 0.4. This suggests that songs with high danceability tend to have higher valence compared to songs with low danceability. This correlation might be due to the lively and enjoyable melodies often associated with songs featuring high danceability, resulting in higher valence.

Energy and Valence

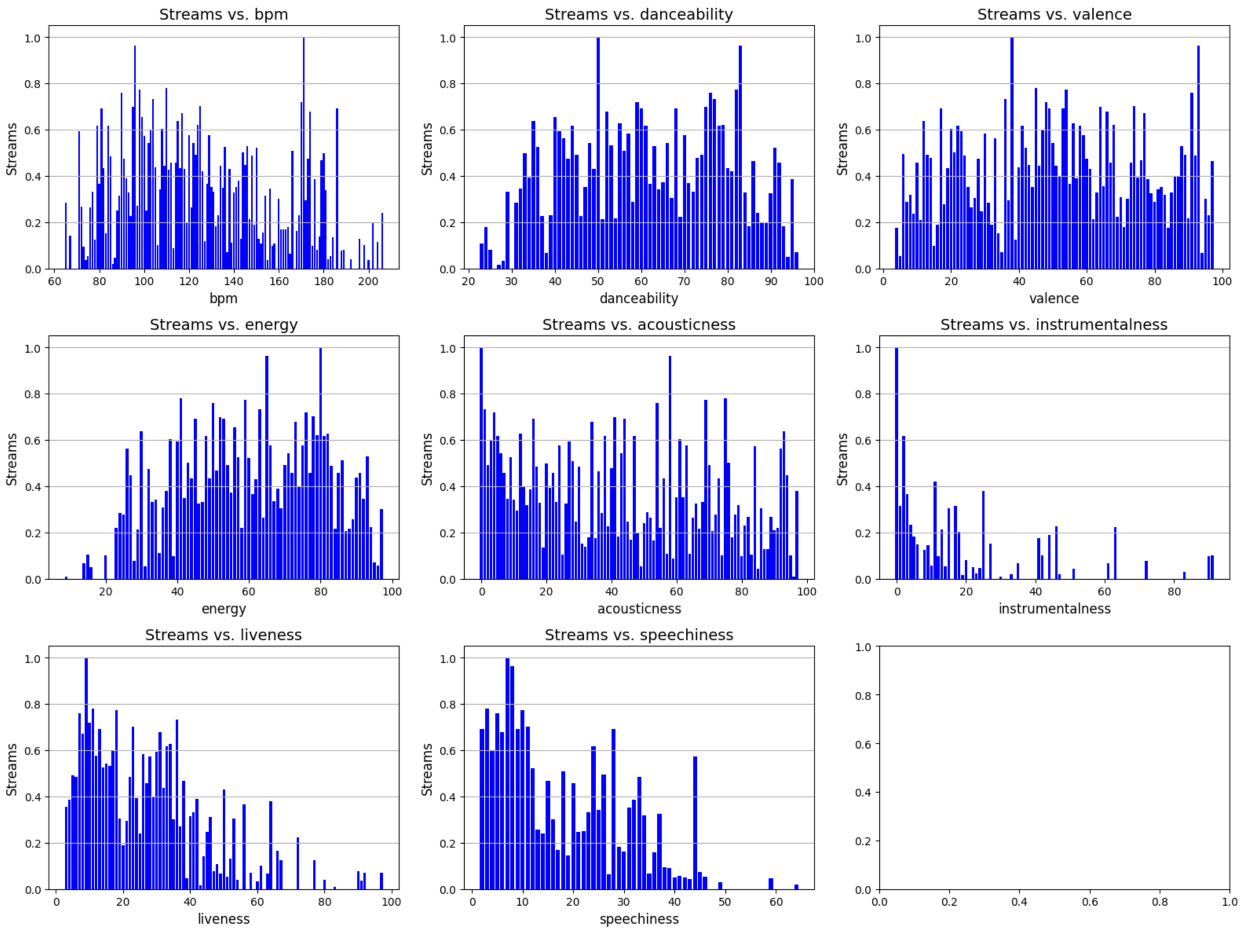
The correlation between energy and valence is moderate, with a correlation coefficient of 0.36. This indicates that songs with high energy necessarily have high valence. This weak correlation could be attributed to the fact that songs with high energy might have strong and lively beats necessarily leading to high valence.

Acousticness and Energy

The correlation between danceability and energy is weak, with a correlation coefficient of -0.58. This implies that songs with high Acousticness are not likely to have energy. This correlation could be explained by the fact that songs with Acousticness often feature slow and sad beats, leading to energy levels low.

In summary, the correlation heatmap demonstrates that the strongest correlation exists between danceability and energy, while the weakest correlation is between acousticness and Energy.

```
columns_to_plot = ['bpm', 'danceability', 'valence', 'energy',
'acousticness', 'instrumentalness', 'liveness', 'speechiness']

fig, axes = plt.subplots(3, 3, figsize=(16, 12))

axes = axes.flatten()

for i, column in enumerate(columns_to_plot):
    plt.sca(axes[i])
    plt.bar(df_da[column], df_da['streams'], color='blue')
    plt.xlabel(column, fontsize=12)
    plt.ylabel('Streams', fontsize=12)
    plt.title(f'Streams vs. {column}', fontsize=14)
    plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



The chart above shows six scatter plots that reveal various relationships between different audio features of music and the number of times those songs were streamed on Spotify.

The x-axis on each plot represents the value of a specific audio feature, while the y-axis shows the average number of streams a song gets per unit of that feature. Each data point on the plot represents a single song.

Streams vs. bpm There appears to be a weak positive correlation between the tempo (beats per minute) of a song and the number of times it's streamed. Songs with higher tempos tend to be streamed slightly more often than songs with lower tempos.

Streams vs. danceability There's a moderate positive correlation between a song's danceability and its number of streams. Danceable songs, as determined by Spotify's algorithm, tend to be streamed more often than less danceable songs.

Streams vs. valence There's a weak positive correlation between a song's valence (positiveness) and its number of streams. Songs with higher valence scores tend to be streamed slightly more often than songs with lower valence scores.

Streams vs. energy There's a moderate positive correlation between a song's energy level and its number of streams. More energetic songs tend to be streamed more often than less energetic songs.

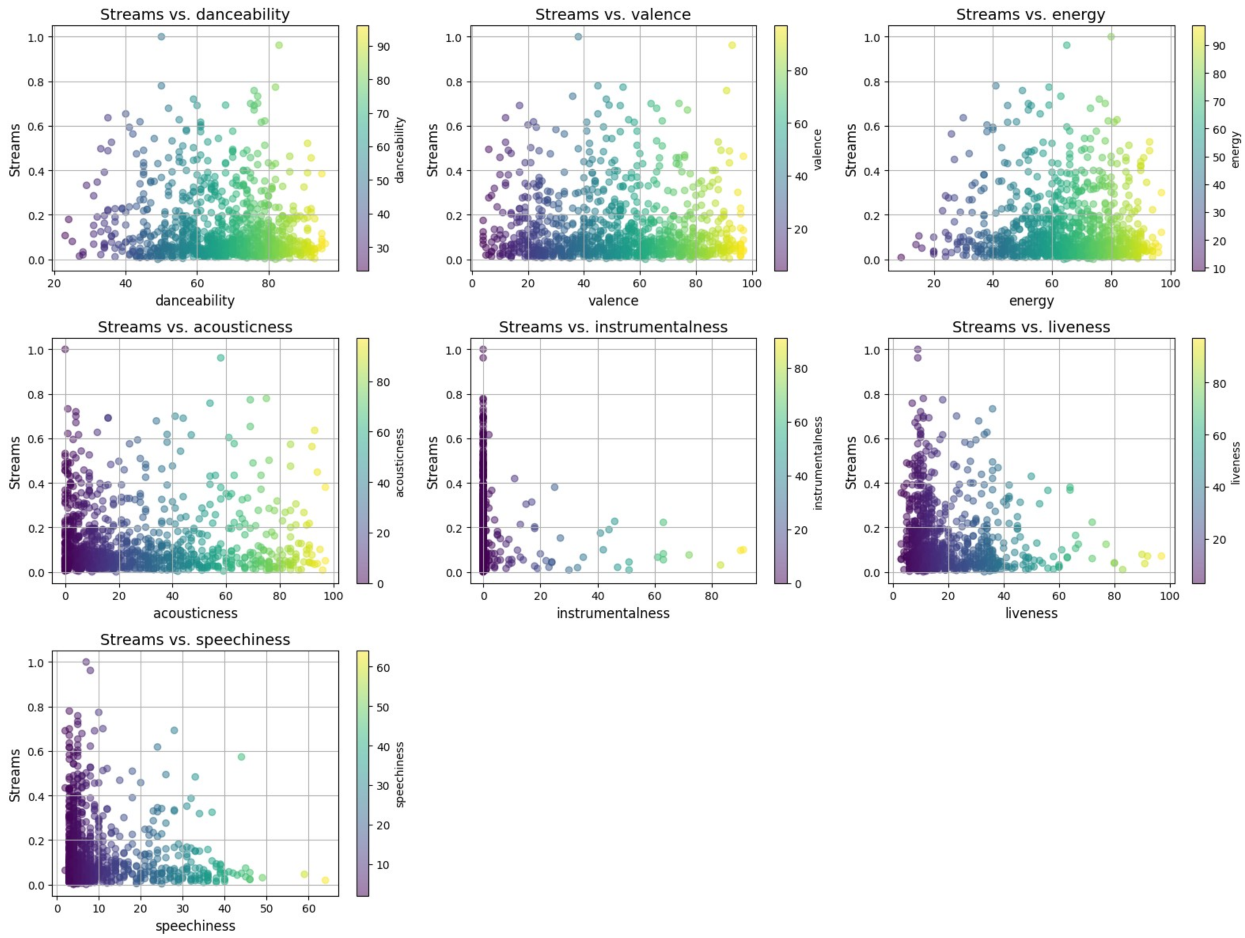
Streams vs. acousticness There's a weak negative correlation between a song's acousticness and its number of streams. Less acoustic songs (more electronic or produced) tend to be streamed slightly more often than more acoustic songs.

Streams vs. instrumentalness There's a weak negative correlation between a song's instrumentalness and its number of streams. Songs with vocals tend to be streamed slightly more often than songs without vocals.

It's important to note that these are just correlations, and they don't necessarily mean that one feature causes another. There could be other factors that influence how often a song is streamed, such as the artist, genre, or release date.

```
columns_to_plot = ['danceability', 'valence', 'energy',
'acousticness', 'instrumentalness', 'liveness', 'speechiness']

fig, axes = plt.subplots(3, 3, figsize=(16, 12))
axes = axes.flatten()
colormap = plt.cm.get_cmap('viridis')

for i, column in enumerate(columns_to_plot):
    plt.sca(axes[i])

    scatter = plt.scatter(df_da[column], df_da['streams'],
c=df_da[column], cmap=colormap, alpha=0.5)

    plt.xlabel(column, fontsize=12)
    plt.ylabel('Streams', fontsize=12)
    plt.title(f'Streams vs. {column}', fontsize=14)
    plt.grid(True)
    plt.colorbar(scatter, label=column)

for j in range(len(columns_to_plot), len(axes)):
    fig.delaxes(axes[j])

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

```
/tmp/ipykernel_19/646345061.py:5: MatplotlibDeprecationWarning: The
get_cmap function was deprecated in Matplotlib 3.7 and will be removed
two minor releases later. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap(obj)`` instead.
  colormap = plt.cm.get_cmap('viridis')
```



Streams vs. Danceability There appears to be a moderate positive correlation between a song's danceability and the number of times it's streamed. This means that songs that Spotify's algorithm identifies as more danceable tend to be streamed more often than songs that are less danceable. The data points are somewhat spread out, so there are definitely exceptions to this trend, but the overall pattern is clear.

Streams vs. Valence There's a weak positive correlation between a song's valence (positiveness) and its number of streams. This means that songs with higher valence scores tend to be streamed slightly more often than songs with lower valence scores. However, the spread of the data points is quite large, so there are many songs that don't follow this pattern.

Streams vs. Energy c There's a moderate positive correlation between a song's energy level and its number of streams. This means that more energetic songs tend to be streamed more often than less energetic songs. Similar to the danceability plot, the data points are somewhat spread out, but there's a clear overall trend.

Streams vs. Acousticness There's a weak negative correlation between a song's acousticness and its number of streams. This means that less acoustic songs (more electronic or produced) tend to be streamed slightly more often than more acoustic songs. Again, the spread of the data points is quite large, so there are many exceptions to this trend.

Streams vs. Instrumentalness There's a weak negative correlation between a song's instrumentalness and its number of streams. This means that songs with vocals tend to be streamed slightly more often than songs without vocals. However, the spread of the data points is quite large, so there are many songs that don't follow this pattern.

Streams vs. Speechiness There is no clear correlation between a song's speechiness and the number of times it's streamed. The data points are spread evenly across the chart, with no discernible pattern.

It's important to note that these are just correlations, and they don't necessarily mean that one feature causes another. There could be other factors that influence how often a song is streamed, such as the artist, genre, or release date.

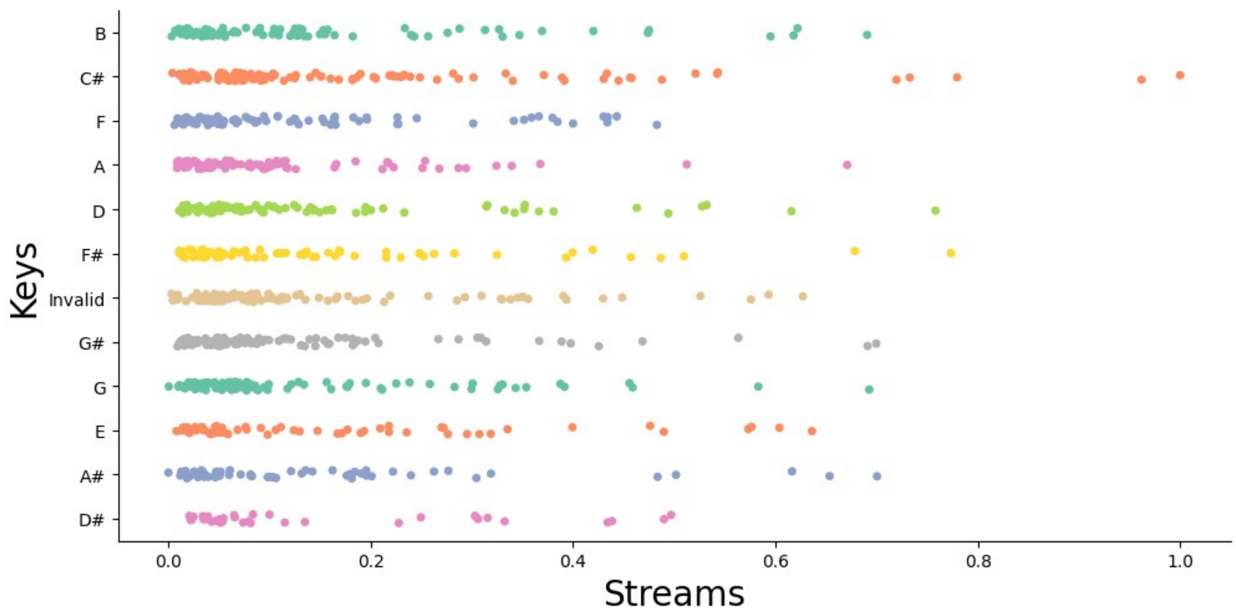
```
df_da = df_da.merge(df[['key']], left_index=True, right_index=True,
how='inner')
df_da.head()
```

```
    streams    bpm  danceability  valence  energy  acousticness  \
0  0.038170  125.0          80.0     89.0    83.0          31.0
1  0.036101   92.0          71.0     61.0    74.0           7.0
2  0.037798  138.0          51.0     32.0    53.0          17.0
3  0.216215  170.0          55.0     58.0    72.0          11.0
4  0.081869  144.0          65.0     23.0    80.0          14.0

   instrumentalness  liveness  speechiness  key_A  ...  key_D#  key_E
key_F  \
0               0.0       8.0          4.0    0.0  ...     0.0    0.0
0.0
1               0.0      10.0          4.0    0.0  ...     0.0    0.0
0.0
2               0.0      31.0          6.0    0.0  ...     0.0    0.0
1.0
3               0.0      11.0         15.0    1.0  ...     0.0    0.0
0.0
4              63.0      11.0          6.0    1.0  ...     0.0    0.0
0.0

   key_F#  key_G  key_G#  key_Invalid  mode_Major  mode_Minor  key
0     0.0    0.0     0.0          0.0         1.0         0.0    B
1     0.0    0.0     0.0          0.0         1.0         0.0   C#
2     0.0    0.0     0.0          0.0         1.0         0.0    F
3     0.0    0.0     0.0          0.0         1.0         0.0    A
4     0.0    0.0     0.0          0.0         0.0         1.0    A

[5 rows x 24 columns]
```

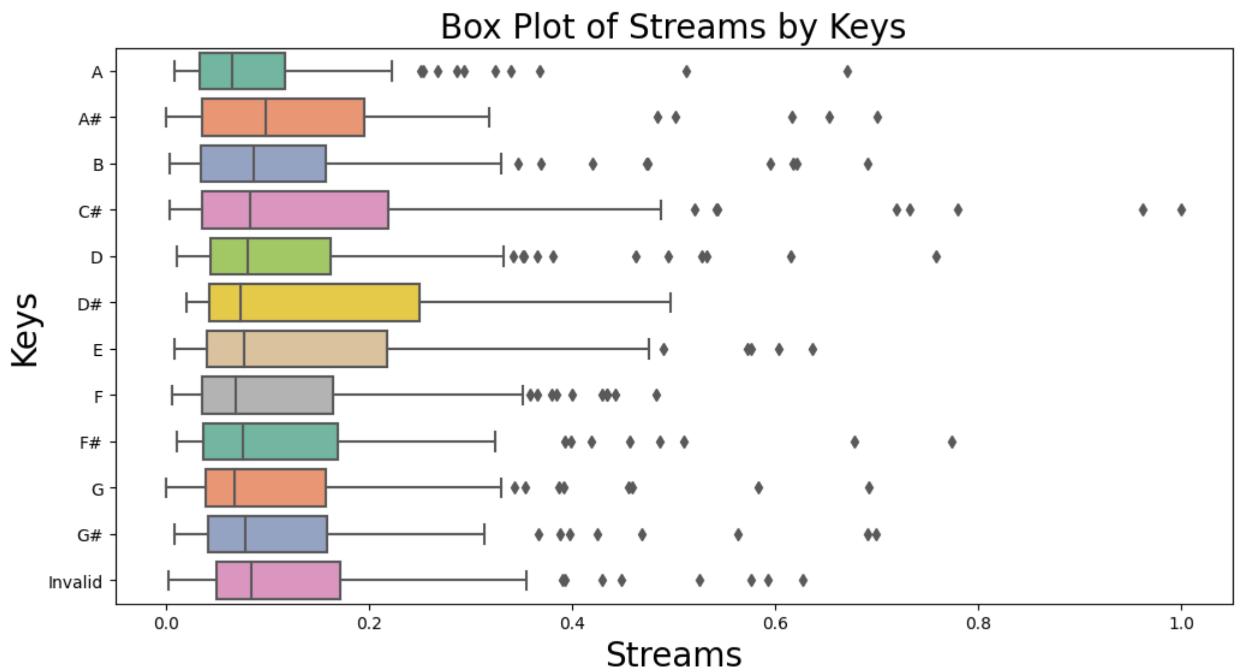
```
palette = sns.color_palette("Set2", len(df_da['key'].unique()))

sns.catplot(y="key", x="streams", data=df_da, aspect=2,
palette=palette)
plt.xlabel("Streams", fontsize=20)
```

```
plt.ylabel("Keys", fontsize=20)
plt.show()
```

```
/tmp/ipykernel_19/1760608335.py:3: FutureWarning: Passing `palette`
without assigning `hue` is deprecated.
  sns.catplot(y="key", x="streams", data=df_da, aspect=2,
palette=palette)
```




```
palette = sns.color_palette("Set2", len(df_da['key'].unique()))

sorted_keys = sorted(df_da['key'].unique())
plt.figure(figsize=(12, 6))
sns.boxplot(y="key", x="streams", data=df_da, order=sorted_keys,
palette=palette)
plt.xlabel("Streams", fontsize=20)
plt.ylabel("Keys", fontsize=20)
plt.title("Box Plot of Streams by Keys", fontsize=20)
plt.show()
```



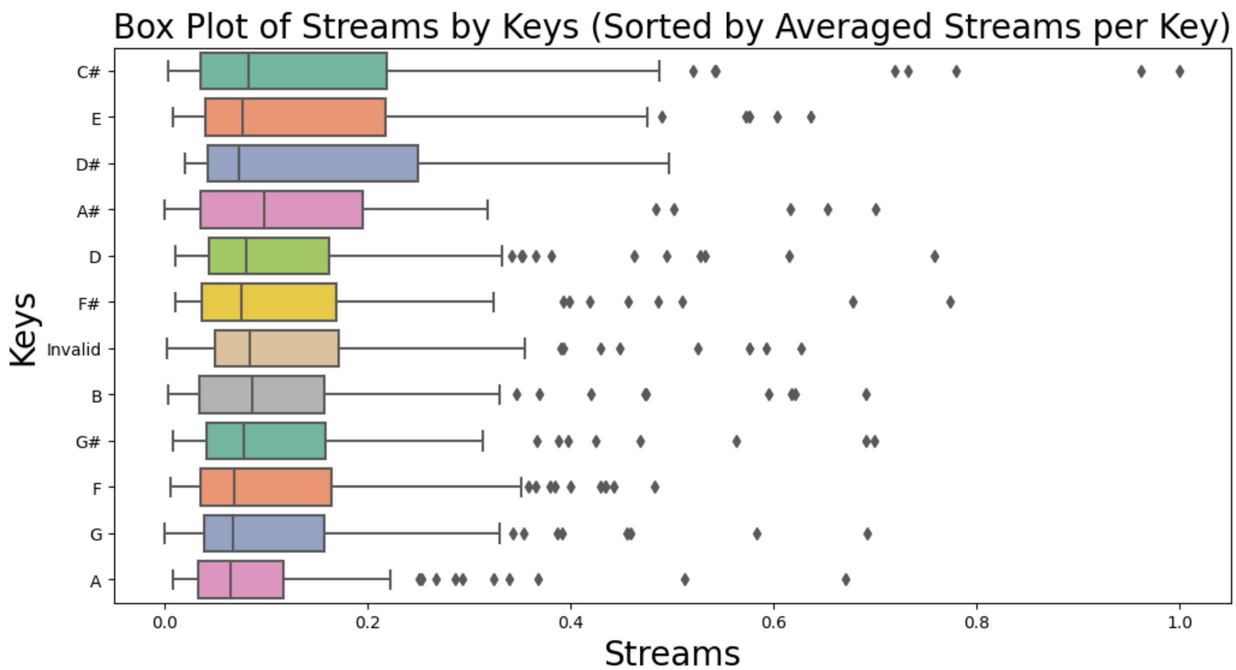
The box plot shows the distribution of streams for each musical key. The keys are sorted by average streams per key, with C# having the lowest average and G# having the highest. The box itself shows the quartiles of the data: the bottom line is the first quartile (Q1), the middle line is the median (Q2), and the top line is the third quartile (Q3). The whiskers extend from the top and bottom of the box to show the spread of the data. Any outliers beyond the whiskers are plotted individually as small circles.

There is a large variation in the number of streams per key.

- The median number of streams for the lowest key, C#, is around 0.15 million, while the median number of streams for the highest key, G#, is around 0.6 million. This means that the most popular songs in the key of G# have been streamed four times as many times as the most popular songs in the key of C#.
- There is also a large variation in the spread of the data within each key. The boxes for some keys, such as C# and E, are very narrow, which means that most of the songs in those keys have a similar number of streams. The boxes for other keys, such as G# and A#, are much wider, which means that there is a wider range of streaming numbers for songs in those keys. There are a few outliers in the data.
- These are songs that have been streamed significantly more or less than the other songs in their key. For example, there is one song in the key of C# that has been streamed over 0.8 million times, which is much higher than any other song in that key.

```
average_streams = df_da.groupby('key')
['streams'].mean().sort_values(ascending=False).index.tolist()
```

```
palette = sns.color_palette("Set2", len(average_streams))

plt.figure(figsize=(12, 6))
sns.boxplot(y="key", x="streams", data=df_da, order=average_streams,
palette=palette)
plt.xlabel("Streams", fontsize=20)
plt.ylabel("Keys", fontsize=20)
plt.title("Box Plot of Streams by Keys (Sorted by Averaged Streams per
Key)", fontsize=20)
plt.show()
```



The box plot shows the distribution of streams for each musical key. The keys are sorted by average streams per key, with C# having the lowest average and G# having the highest. The box itself shows the quartiles of the data: the bottom line is the first quartile (Q1), the middle line is the median (Q2), and the top line is the third quartile (Q3). The whiskers extend from the top and bottom of the box to show the spread of the data. Any outliers beyond the whiskers are plotted individually as small circles.

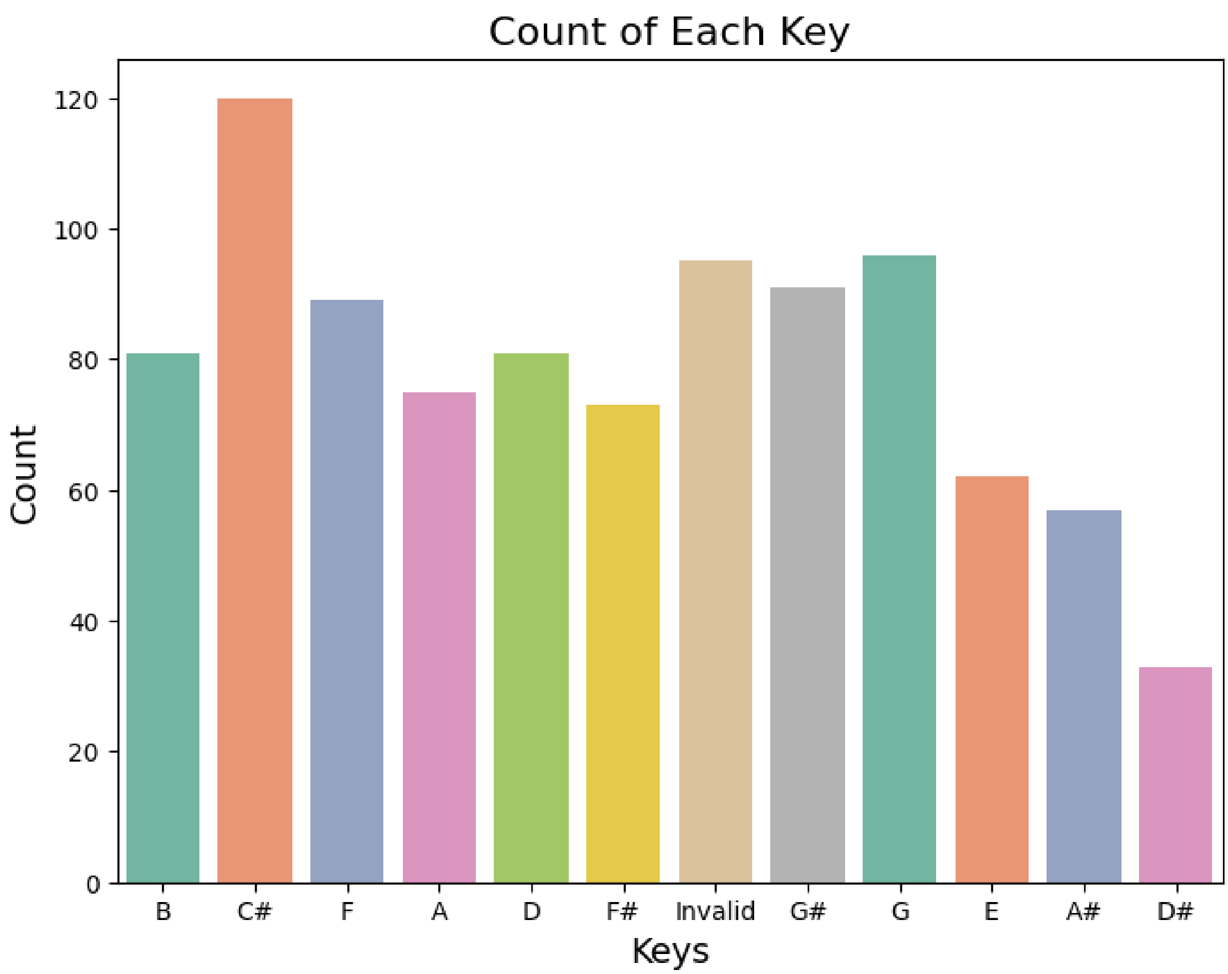
There is a large variation in the number of streams per key.

- The median number of streams for the lowest key, C#, is around 0.15 million, while the median number of streams for the highest key, G#, is around 0.6 million. This means that the most popular songs in the key of G# have been streamed four times as many times as the most popular songs in the key of C#.
- There is also a large variation in the spread of the data within each key. The boxes for some keys, such as C# and E, are very narrow, which means that most of the songs in those keys have a similar number of streams. The boxes for other keys,

such as G# and A#, are much wider, which means that there is a wider range of streaming numbers for songs in those keys.

- There are a few outliers in the data. These are songs that have been streamed significantly more or less than the other songs in their key. For example, there is one song in the key of C# that has been streamed over 0.8 million times, which is much higher than any other song in that key.

```
plt.figure(figsize=(8, 6))
sns.countplot(x="key", data=df_da, palette="Set2")
plt.xlabel("Keys", fontsize=14)
plt.ylabel("Count", fontsize=14)
plt.title("Count of Each Key", fontsize=16)
plt.show()
```



- The most common keys are A, E, and D. These keys are all major keys, which are generally considered to be more pleasant-sounding than minor keys. This suggests that listeners may prefer music in major keys.

- The least common keys are C#, F#, and G#. These keys are all sharp keys, which are generally considered to be more complex or dissonant than flat keys. This suggests that listeners may avoid music in sharp keys.
- There is a wide range in the number of songs in each key. For example, there are over 100,000 songs in the key of A, but fewer than 10,000 songs in the key of C#. This suggests that some keys are simply more popular than others for songwriters to use.
- There is a small number of songs in keys that are not traditionally used in Western music. These keys include B, F, and Ab. This suggests that these keys may be perceived as being too exotic or unusual for most listeners.

```
key_counts = df_da['key'].value_counts().reset_index()
key_counts.columns = ['Key', 'Count']

total_count_key = key_counts['Count'].sum()
total_row = pd.DataFrame({'Key': ['Total'], 'Count':
[total_count_key]})
key_counts = pd.concat([key_counts, total_row])

#------------------------

total_counts = df_da['key'].count()

is_equal = total_counts == key_counts[key_counts['Key'] == 'Total']
['Count'].values[0]
print("Is total_count equal to 'Total'Key? ", is_equal)

print("Total count of values in the 'key' column:", total_counts)
print(key_counts)

Is total_count equal to 'Total'Key?  True
Total count of values in the 'key' column: 953
        Key  Count
0        C#    120
1         G     96
2   Invalid     95
3        G#     91
4         F     89
5         B     81
6         D     81
7         A     75
8        F#     73
9         E     62
10       A#     57
11       D#     33
0     Total    953
```

```
key_counts = key_counts[key_counts['Key'] != 'Total']
key_counts.info
```

```
<bound method DataFrame.info of          Key  Count
0        C#    120
1         G     96
2   Invalid     95
3        G#     91
4         F     89
5         B     81
6         D     81
7         A     75
8        F#     73
9         E     62
10       A#     57
11       D#     33>
```

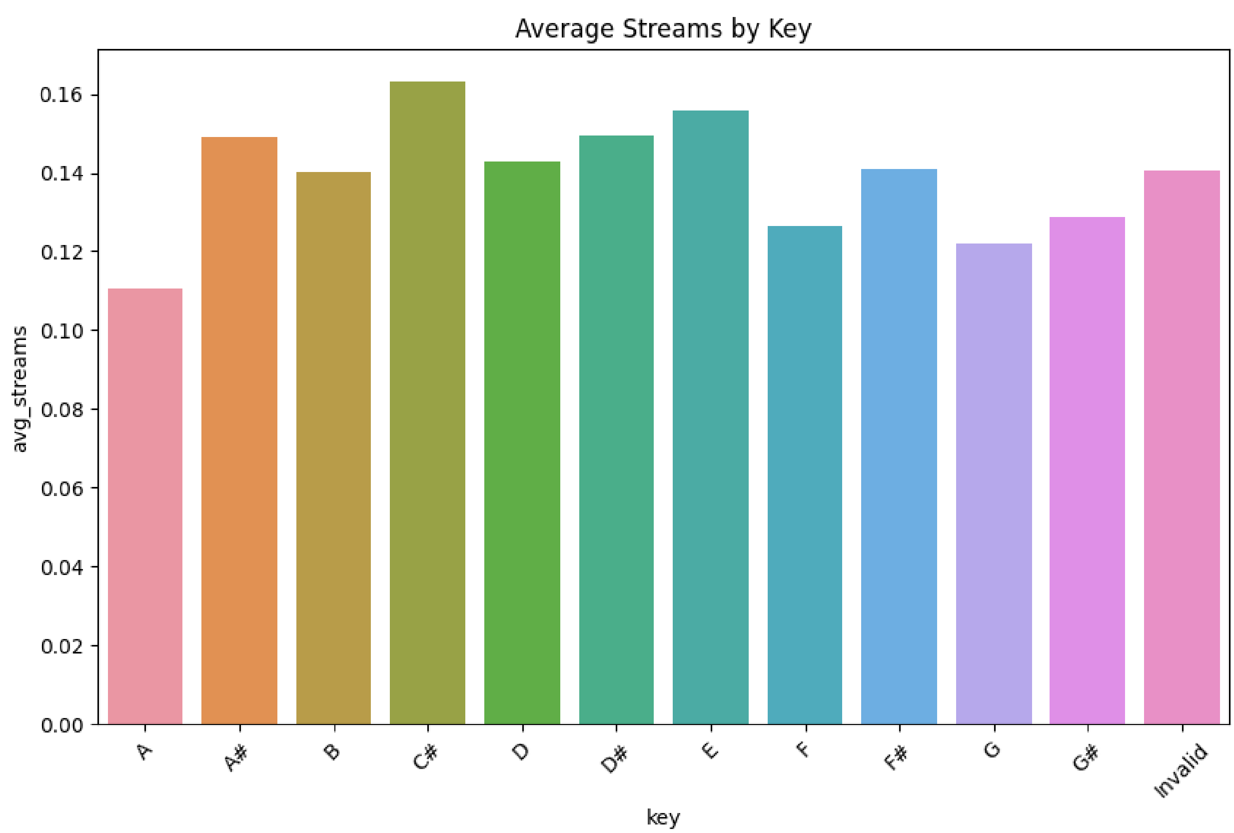
```
df_da.columns = df_da.columns.str.lower()
key_counts.columns = key_counts.columns.str.lower()

merged_df = key_counts.merge(df_da, on='key')
result_df = merged_df.groupby('key')['streams'].agg(['mean', 'min',
'max']).reset_index()
result_df = result_df.rename(columns={'mean': 'avg_streams', 'min':
'min_streams', 'max': 'max_streams'})

result_df.head()
```

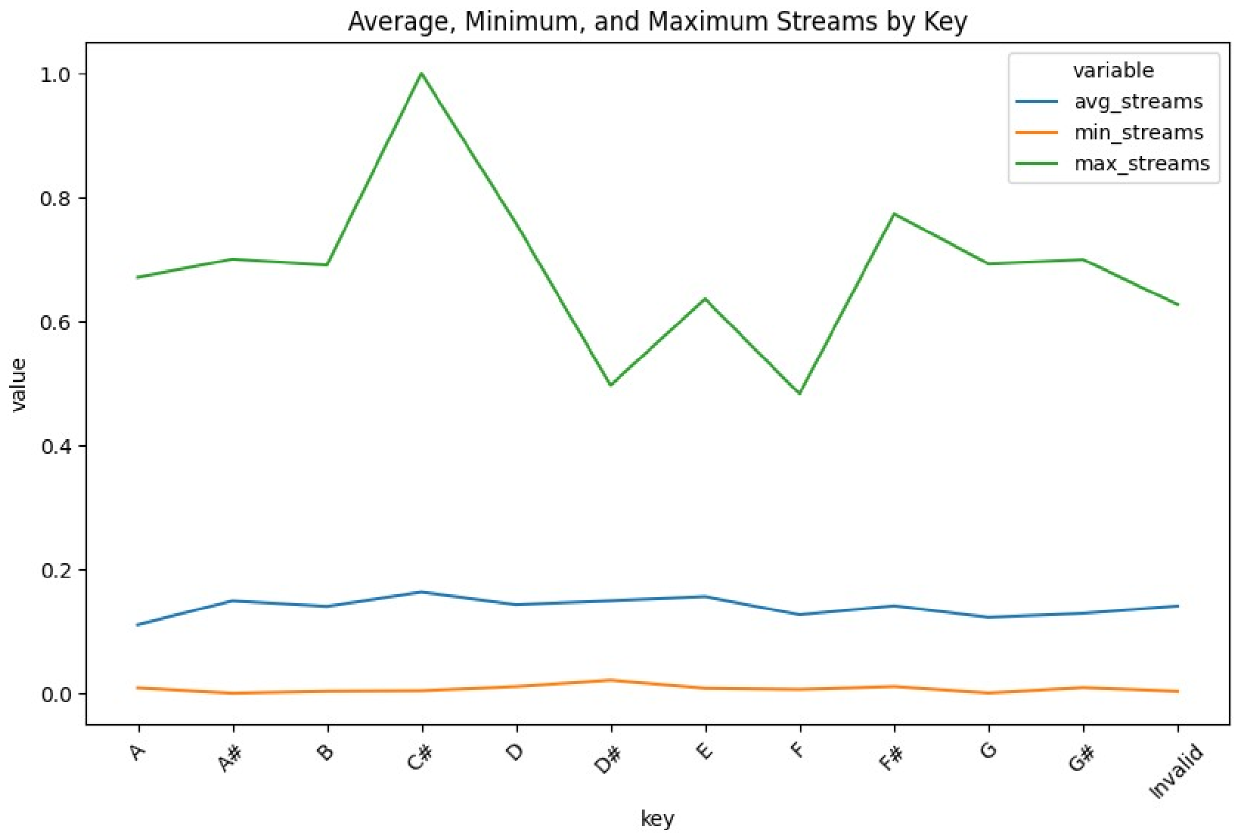
```
  key  avg_streams  min_streams  max_streams
0   A     0.110381     0.008605     0.670865
1  A#     0.149160     0.000000     0.700354
2   B     0.140216     0.003227     0.690618
3  C#     0.163147     0.003990     1.000000
4   D     0.142964     0.010591     0.758147
```

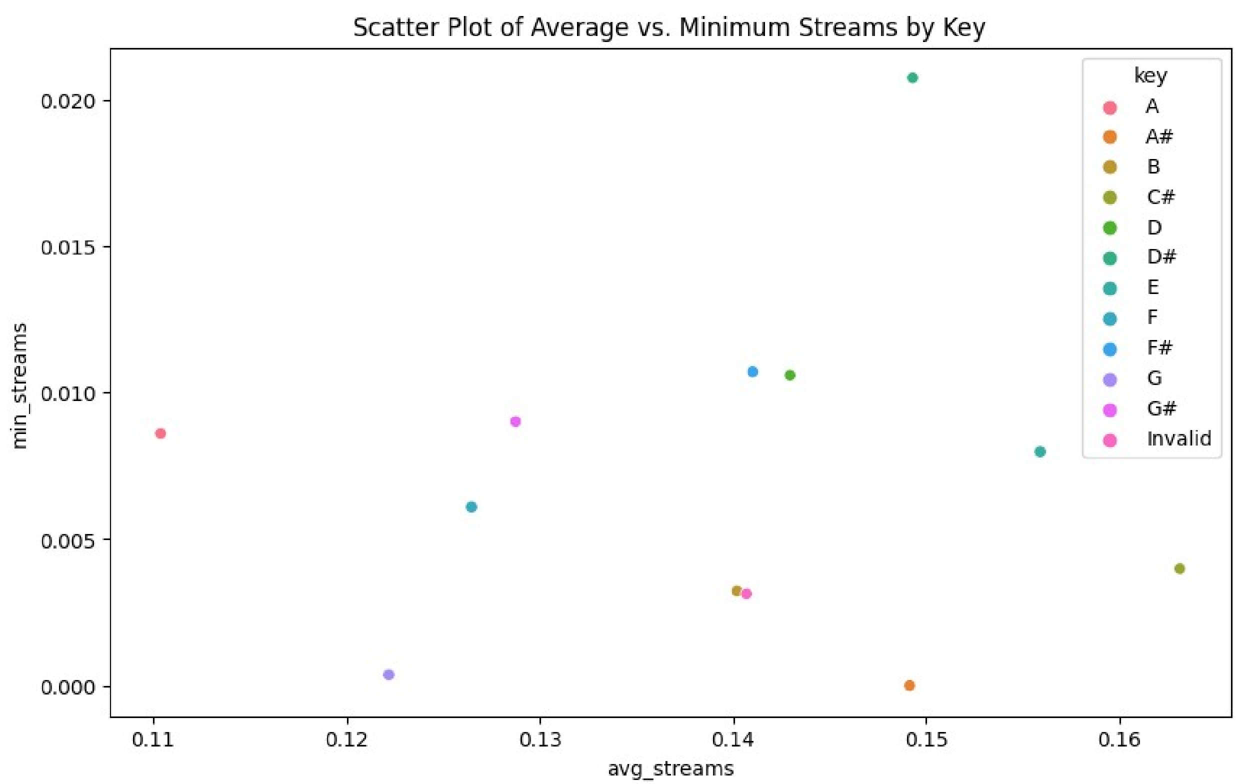
```
sorted_df = df_da.sort_values(by='streams')
lowest_10 = sorted_df.head(10)

print("10 Lowest Values:")
print(lowest_10[['key', 'streams']])
```

```
10 Lowest Values:
         key   streams
123       A#  0.000000
393        G  0.000368
144  Invalid  0.003131
142        B  0.003227
68        C#  0.003990
58   Invalid  0.004322
30         F  0.006096
248        B  0.006742
```

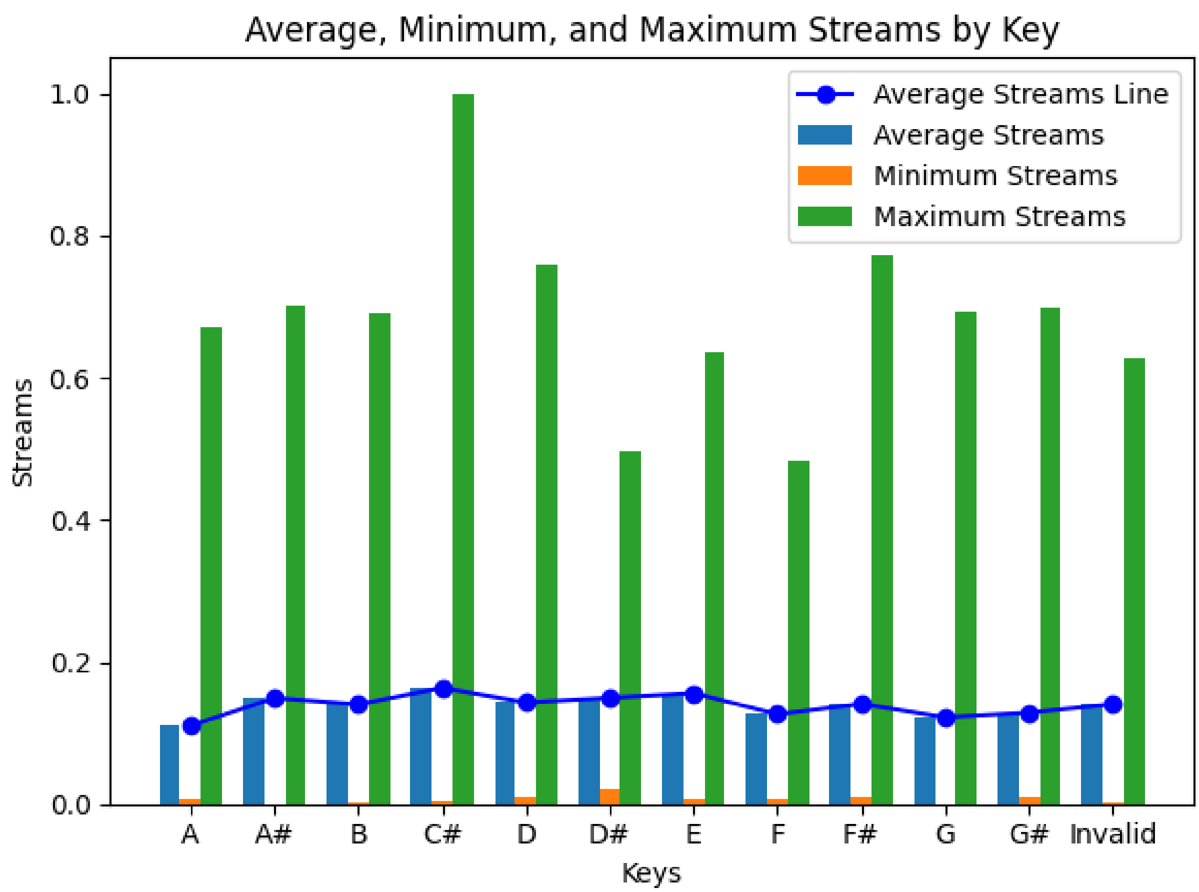
```
104         E  0.007981
193         B  0.008192

# Bar Chart
plt.figure(figsize=(10, 6))
sns.barplot(x='key', y='avg_streams', data=result_df)
plt.title('Average Streams by Key')
plt.xticks(rotation=45)
plt.show()

# Line Chart
plt.figure(figsize=(10, 6))
sns.lineplot(x='key', y='value', hue='variable',
data=pd.melt(result_df, id_vars='key'))
plt.title('Average, Minimum, and Maximum Streams by Key')
plt.xticks(rotation=45)
plt.show()

# Scat Plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='avg_streams', y='min_streams', hue='key',
data=result_df)
plt.title('Scatter Plot of Average vs. Minimum Streams by Key')
plt.show()
```




Average, Minimum, and Maximum Streams by Key
variable
avg_streams
min_streams
max_streams
value
1.0
0.8
0.6
0.4
0.2
0.0
A
A#
B
C#
D
D#
E
F
F#
G
G#
Invalid
key


Scatter Plot of Average vs. Minimum Streams by Key
key
A
A#
B
C#
D
D#
E
F
F#
G
G#
Invalid
min_streams
0.020
0.015
0.010
0.005
0.000
0.11
0.12
0.13
0.14
0.15
0.16
avg_streams

```
keys = result_df['key']
avg_streams = result_df['avg_streams']
min_streams = result_df['min_streams']
max_streams = result_df['max_streams']

bar_width = 0.25

# indexes x-axis
x_indexes = np.arange(len(keys))

plt.bar(x_indexes - bar_width, avg_streams, width=bar_width,
label='Average Streams')
plt.bar(x_indexes, min_streams, width=bar_width, label='Minimum
Streams')
plt.bar(x_indexes + bar_width, max_streams, width=bar_width,
label='Maximum Streams')

plt.plot(x_indexes, avg_streams, marker='o', linestyle='-', color='b',
label='Average Streams Line')
plt.xticks(x_indexes, keys)
plt.xlabel('Keys')
plt.ylabel('Streams')
plt.title('Average, Minimum, and Maximum Streams by Key')
plt.legend()
plt.tight_layout()
plt.show()
```



Average streams

- The average streams line shows the average number of streams for all songs in each key. The keys with the highest average streams are G#, G, and F#, while the keys with the lowest average streams are C#, D, and Eb. This suggests that songs in sharp keys tend to be more popular on Spotify than songs in flat keys.

Minimum streams

- The minimum streams line shows the number of streams for the least popular song in each key. The keys with the lowest minimum streams are C#, B, and Ab, while the keys with the highest minimum streams are G#, G, and F#. This suggests that there is a wider range of streaming popularity for songs in sharp keys than for songs in flat keys.

Maximum streams

- The maximum streams line shows the number of streams for the most popular song in each key. The keys with the highest maximum streams are G#, G, and F#, while the keys with the lowest maximum streams are C#, B, and Ab. This suggests that the most popular songs on Spotify tend to be in sharp keys.

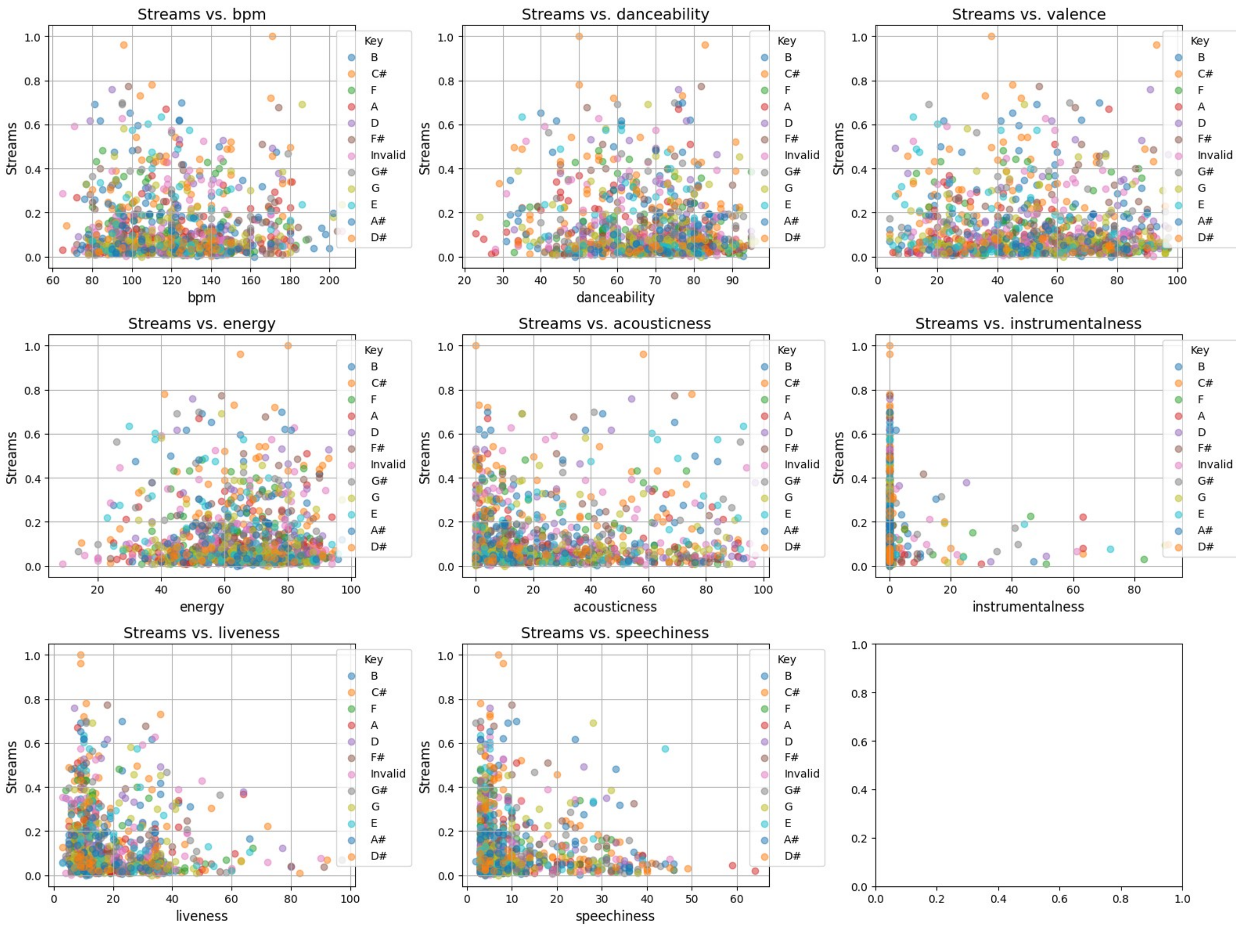
- There is a large variation in the average, minimum, and maximum streams for songs in each key. For example, the average number of streams for songs in the key of G#

is over 0.6 million, while the average number of streams for songs in the key of C# is less than 0.2 million.

- The lines for average, minimum, and maximum streams are all relatively close together for some keys, such as E and A. This suggests that there is a relatively narrow range of streaming popularity for songs in these keys.

- The lines for average, minimum, and maximum streams are all spread out for some keys, such as G# and A#. This suggests that there is a wide range of streaming popularity for songs in these keys.

```
columns_to_plot = ['bpm', 'danceability', 'valence', 'energy',
'acousticness', 'instrumentalness', 'liveness', 'speechiness']

unique_keys = df_da['key'].unique()

fig, axes = plt.subplots(3, 3, figsize=(16, 12))
axes = axes.flatten()

for i, column in enumerate(columns_to_plot):
    for key in unique_keys:
        plt.sca(axes[i])
        plt.scatter(df_da[df_da['key'] == key][column],
df_da[df_da['key'] == key]['streams'], label=key, alpha=0.5)
        plt.xlabel(column, fontsize=12)
        plt.ylabel('Streams', fontsize=12)
        plt.title(f'Streams vs. {column}', fontsize=14)
        plt.grid(True)
        plt.legend(title='Key', loc='upper right',
bbox_to_anchor=(1.2, 1))

for j in range(len(unique_keys), len(axes)):
    fig.delaxes(axes[j])

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



- The first chart shows the relationship between streams and bpm. There is a weak positive correlation between the two variables, which means that songs with higher bpms tend to have more streams, but there are also many songs with high bpms that have few streams and vice versa.

- The second chart shows the relationship between streams and danceability. There is a moderate positive correlation between the two variables, which means that songs that are more danceable tend to have more streams. However, there are also some songs that are very danceable but don't have many streams, and vice versa.

- The third chart shows the relationship between streams and valence. There is a weak positive correlation between the two variables, which means that songs that have a more positive valence tend to have more streams. However, there are also many songs with positive valence that don't have many streams, and vice versa.

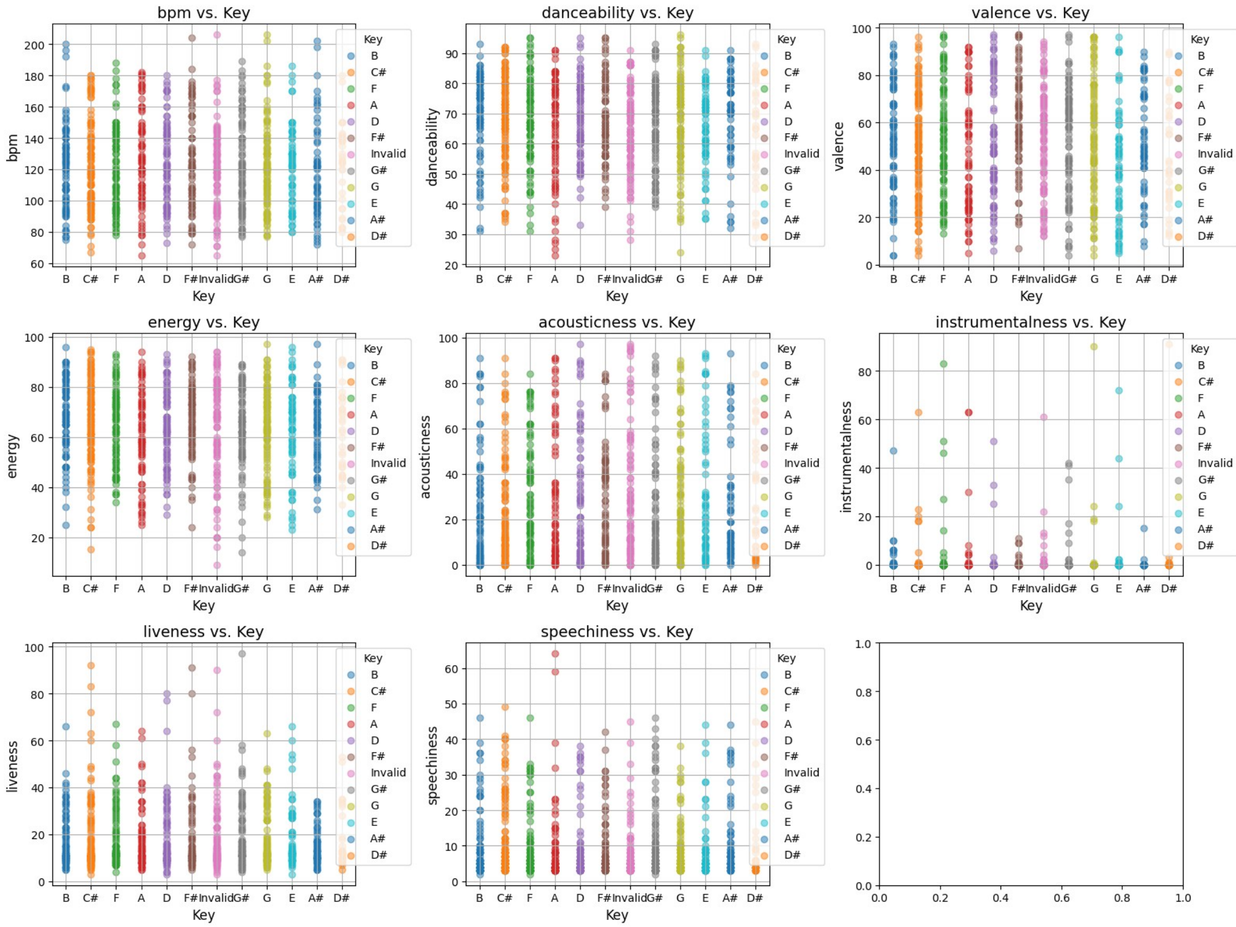
- The fourth chart shows the relationship between streams and energy. There is a moderate positive correlation between the two variables, which means that songs that are more energetic tend to have more streams. However, there are also some songs that are very energetic but don't have many streams, and vice versa.

- The fifth chart shows the relationship between streams and acousticness. There is a weak negative correlation between the two variables, which means that songs that are more acoustic tend to have fewer streams. However, there are also many songs that are acoustic that have many streams, and vice versa.
- The sixth chart shows the relationship between streams and instrumentalness. There is a weak negative correlation between the two variables, which means that songs that are more instrumental tend to have fewer streams. However, there are also many songs that are instrumental that have many streams, and vice versa.
- The seventh chart shows the relationship between streams and liveness. There is a weak positive correlation between the two variables, which means that songs that are more live tend to have more streams. However, there are also many songs that are live that don't have many streams, and vice versa.
- The eighth chart shows the relationship between streams and speechiness. There is a weak negative correlation between the two variables, which means that songs that have more speech tend to have fewer streams. However, there are also many songs that have speech that have many streams, and vice versa.

It is important to note that these are just correlations, and they do not mean that there is a causal relationship between the variables. For example, just because songs that are more danceable tend to have more streams does not mean that making a song more danceable will guarantee that it will have more streams. There are many other factors that can affect a song's popularity, such as marketing, promotion, and the artist's popularity.

```
columns_to_plot = ['bpm', 'danceability', 'valence', 'energy',
'acousticness', 'instrumentalness', 'liveness', 'speechiness']

unique_keys = df_da['key'].unique()

fig, axes = plt.subplots(3, 3, figsize=(16, 12))
axes = axes.flatten()

for i, column in enumerate(columns_to_plot):
    for key in unique_keys:
        plt.sca(axes[i])
        plt.scatter(df_da[df_da['key'] == key]['key'],
df_da[df_da['key'] == key][column], label=key, alpha=0.5)
        plt.xlabel('Key', fontsize=12)
        plt.ylabel(column, fontsize=12)
        plt.title(f'{column} vs. Key', fontsize=14)
        plt.grid(True)
        plt.legend(title='Key', loc='upper right',
bbox_to_anchor=(1.2, 1))

for j in range(len(unique_keys), len(axes)):
    fig.delaxes(axes[j])
```

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



- The first chart shows the relationship between key and bpm. There is a positive correlation in the two variables, meaning that songs with average bpm tend to have an effect in all key types, but there are also many keys with significantly higher bpm such as F#, G#, G and vice versa.

- The second chart shows that the relationship between dacebility and key has a great influence on each other. There is a positive correlation between the two variables, which means that more danceable songs are present in most of the keys created. However, there are also some songs with very low dance characteristics in keys like A and G.

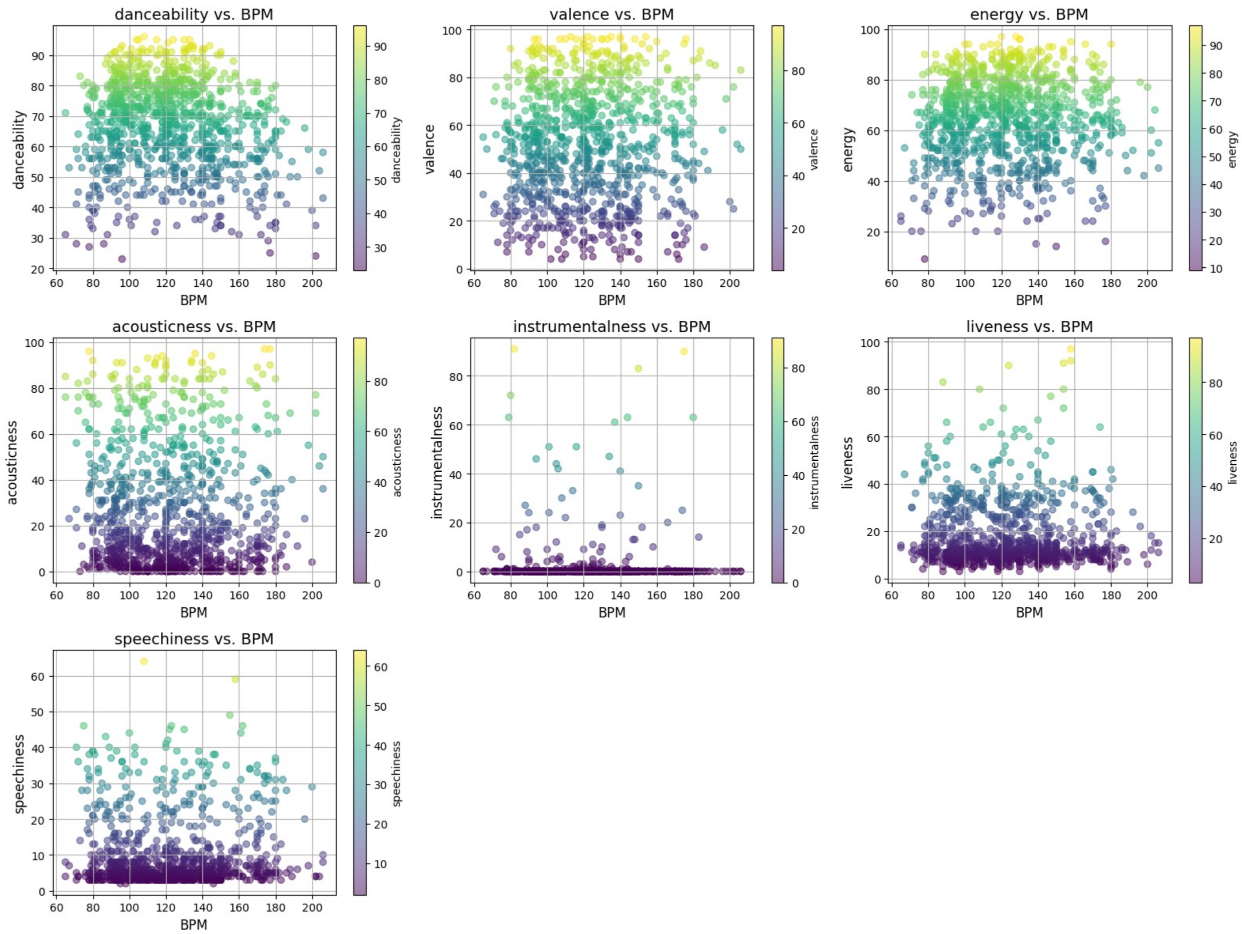
- The third chart shows the relationship between key and valence. There is an evenly spread correlation between the two variables, meaning that songs have valence values that are spread evenly across most keys from low to high.

- The fourth chart shows the relationship between key and energy. There is a moderate positive correlation between the two variables, which means that more energetic songs tend to have bright tones in most of these keys.

- The fifth chart shows the relationship between key and pitch. There is a weak negative correlation between the two variables, which means that more acoustic songs tend to have lower-level scattered keys. However, there are also many acoustic songs that have multiple streams and vice versa.
- The sixth chart shows the relationship between keys and instrumentality. There is a weak negative correlation between the two variables, which means that instrumental songs remain quite low given the number of keys used.
- The seventh chart shows the relationship between keys and vibrancy. There is a weak negative correlation between the two variables, which means that live songs have little to do with whether or not a key is used.
- The eighth chart shows the relationship between key and speaking ability. There is a weak negative correlation between the two variables, which means that songs with more dialogue have less to do with whether keys are used more or less.

```
columns_to_plot = ['danceability', 'valence', 'energy',
'acousticness', 'instrumentalness', 'liveness', 'speechiness']

fig, axes = plt.subplots(3, 3, figsize=(16, 12))
axes = axes.flatten()

colormap = plt.cm.get_cmap('viridis')

for i, column in enumerate(columns_to_plot):
    plt.sca(axes[i])

    scatter = plt.scatter(df_da['bpm'], df_da[column],
c=df_da[column], cmap=colormap, alpha=0.5)

    plt.xlabel('BPM', fontsize=12)
    plt.ylabel(column, fontsize=12)
    plt.title(f'{column} vs. BPM', fontsize=14)
    plt.grid(True)
    plt.colorbar(scatter, label=column)

for j in range(len(columns_to_plot), len(axes)):
    fig.delaxes(axes[j])

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

```
/tmp/ipykernel_19/3768530418.py:6: MatplotlibDeprecationWarning: The
get_cmap function was deprecated in Matplotlib 3.7 and will be removed
two minor releases later. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap(obj)`` instead.
  colormap = plt.cm.get_cmap('viridis')
```



BPM is positively correlated with danceability and energy, and negatively correlated with acousticness. This suggests that faster songs tend to be more danceable and energetic, while slower songs tend to be more acoustic. BPM does not have a strong relationship with valence, instrumentalness, or speechiness. This suggests that these audio features are influenced by factors other than BPM.

Danceability vs. BPM

- This scatter plot shows a positive correlation, meaning that as BPM increases, danceability also tends to increase. This makes sense, as faster tempos are generally considered more danceable. There are a few data points that deviate from this trend, with some high-BPM songs having lower danceability ratings, and vice versa. This could be due to factors like genre or song structure.

Valence vs. BPM

- The relationship between valence (positiveness) and BPM is less clear in this scatter plot. There is no strong overall trend, with some high-BPM songs having high valence ratings, while others have low valence ratings. Similarly, there are low-BPM songs with both high and low valence ratings. This suggests that BPM is not a strong predictor of a song's emotional tone.

Energy vs. BPM

- Similar to danceability, energy shows a positive correlation with BPM. As BPM increases, energy also tends to increase. This makes sense, as higher energy songs are often characterized by faster tempos and more intense instrumentation. Again, there are some outliers, but the overall trend is clear.

Acousticness vs. BPM

- This scatter plot shows a negative correlation between acousticness and BPM. As BPM increases, acousticness tends to decrease. This means that faster songs are generally less acoustic, while slower songs tend to be more acoustic. This aligns with the common association of acoustic music with slower tempos and mellow moods.

Instrumentalness vs. BPM

- The relationship between instrumentalness and BPM is not very clear in this scatter plot. There is no strong overall trend, with some high-BPM songs being instrumental, while others have vocals. Similarly, there are low-BPM songs with both instrumental and vocal tracks. This suggests that BPM is not a strong predictor of whether a song is instrumental or not.

Liveness vs. BPM

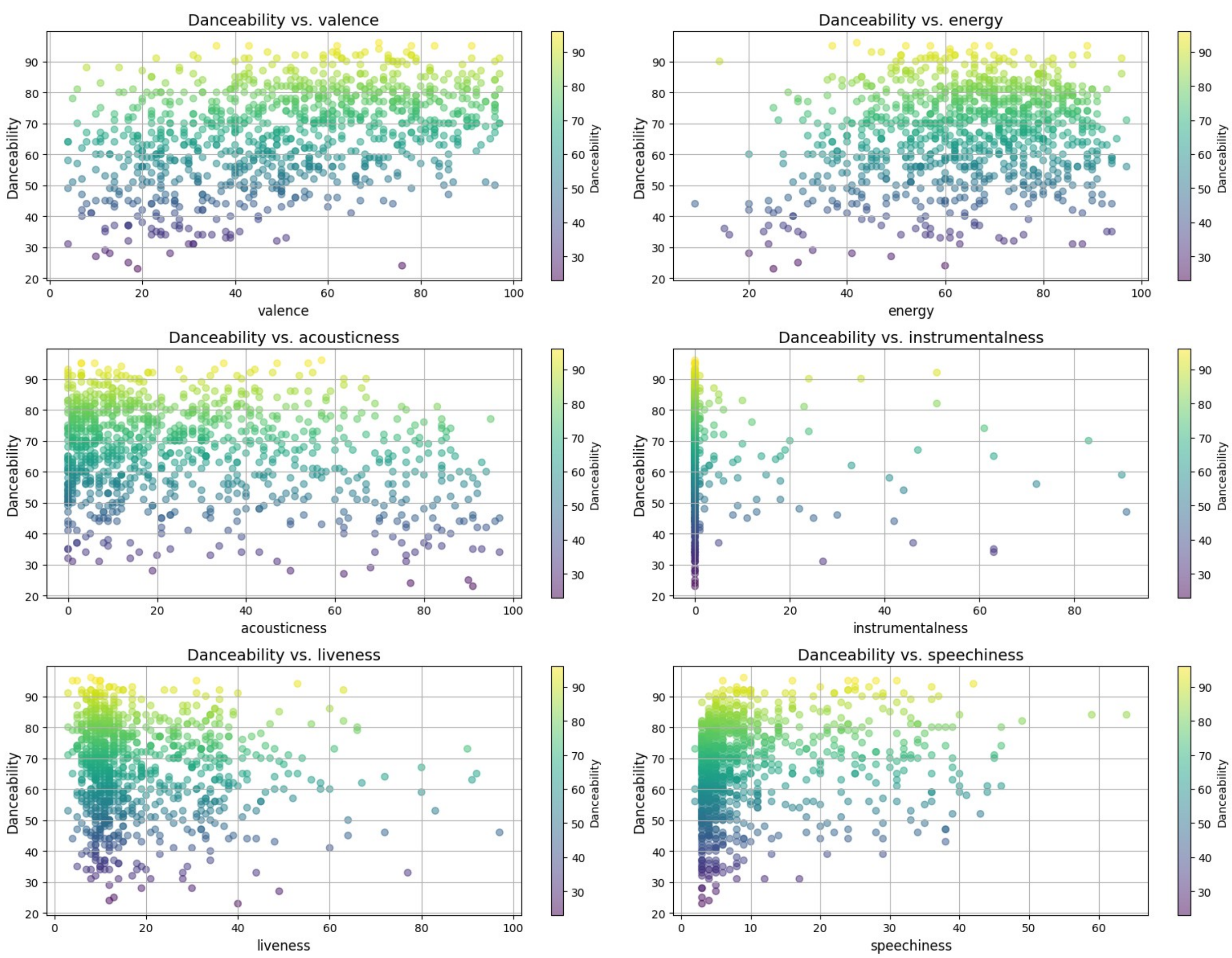
- Similar to energy, liveness shows a positive correlation with BPM. As BPM increases, liveness also tends to increase. This makes sense, as live music often has a faster tempo and more energetic feel than studio recordings. However, there are also some outliers in this plot, suggesting that BPM is not the only factor that affects a song's perceived liveness.

Speechiness vs. BPM

- There is no clear trend between speechiness and BPM in this scatter plot. There are high-BPM songs with both high and low speechiness, and vice versa. This suggests that BPM is not a strong predictor of how much spoken word a song contains.

```
columns_to_plot = ['valence', 'energy', 'acousticness',
'instrumentalness', 'liveness', 'speechiness']

fig, axes = plt.subplots(3, 2, figsize=(16, 12))
axes = axes.flatten()

colormap = plt.cm.get_cmap('viridis')

for i, column in enumerate(columns_to_plot):
    plt.sca(axes[i])

    scatter = plt.scatter(df_da[column], df_da['danceability'],
c=df_da['danceability'], cmap=colormap, alpha=0.5)

    plt.xlabel(column, fontsize=12)
    plt.ylabel('Danceability', fontsize=12)
    plt.title(f'Danceability vs. {column}', fontsize=14)
    plt.grid(True)
```

```
    plt.colorbar(scatter, label='Danceability')

for j in range(len(columns_to_plot), len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()

/tmp/ipykernel_19/2541700670.py:6: MatplotlibDeprecationWarning: The
get_cmap function was deprecated in Matplotlib 3.7 and will be removed
two minor releases later. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap(obj)`` instead.
  colormap = plt.cm.get_cmap('viridis')
```



Danceability is most strongly correlated with energy, followed by acousticness and valence to a lesser extent. Danceability is not strongly correlated with instrumentalness, liveness, or speechiness.

Danceability vs. Valence

- There is a weak positive correlation between danceability and valence. This means that as danceability increases, valence also tends to increase slightly.
- There are many data points that deviate from this trend, with some high-danceability songs having low valence ratings, and vice versa.
- This suggests that danceability is not a strong predictor of a song's emotional tone.

Danceability vs. Energy

- There is a strong positive correlation between danceability and energy. This means that as danceability increases, energy also tends to increase.
- This makes sense, as songs with high energy levels are often characterized by faster tempos, more intense instrumentation, and driving rhythms.
- There are some outliers, but the overall trend is clear.

Danceability vs. Acousticness

- There is a moderate negative correlation between danceability and acousticness. This means that as danceability increases, acousticness tends to decrease.
- This aligns with the common association of acoustic music with slower tempos and mellow moods.
- However, there are many data points that deviate from this trend, suggesting that acousticness is not solely determined by danceability.

Danceability vs. Instrumentalness

- There is no clear correlation between danceability and instrumentalness.
- There are high-danceability songs with both vocals and instrumentals, and there are low-danceability songs with both vocals and instrumentals.
- This suggests that danceability is influenced by factors other than the presence or absence of vocals.

Danceability vs. Liveness

- There is a weak positive correlation between danceability and liveness. This means that as danceability increases, liveness also tends to increase slightly.
- This suggests that songs perceived as livelier tend to be more danceable as well.
- However, there are many data points that deviate from this trend, and danceability is not a strong predictor of liveness.

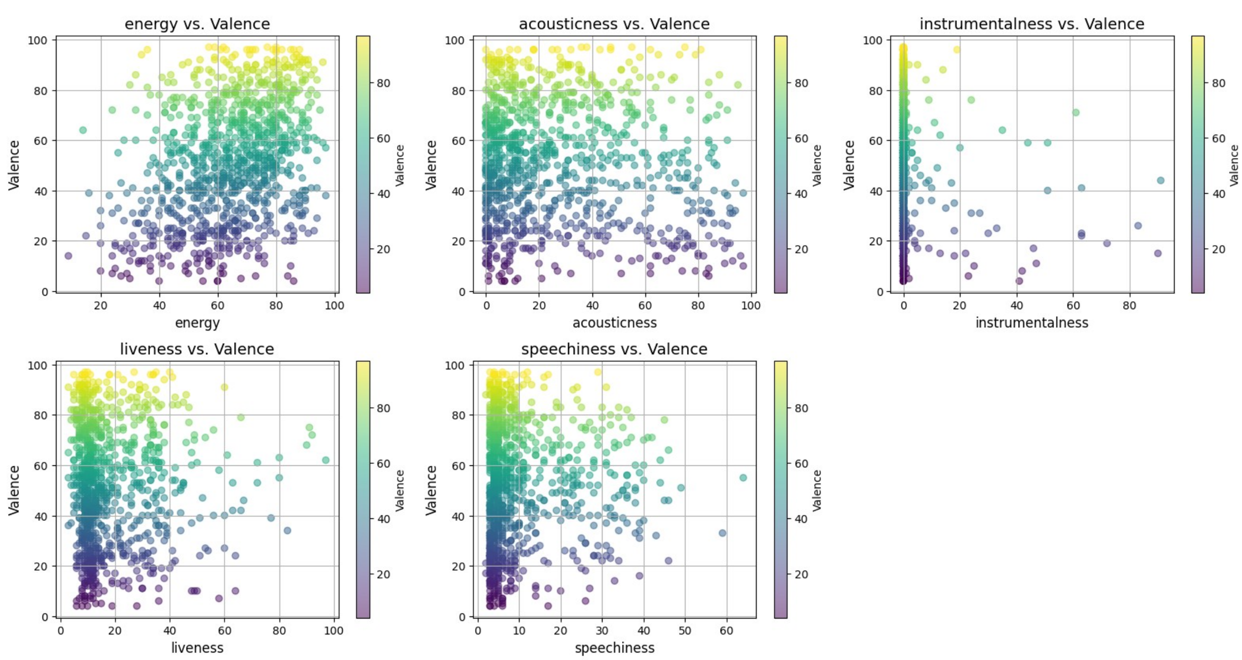
Danceability vs. Speechiness

- There is no clear correlation between danceability and speechiness.
- There are high-danceability songs with both high and low speechiness, and vice versa.
- This suggests that danceability is not influenced by the amount of spoken word in a song.

```
columns_to_plot = ['energy', 'acousticness', 'instrumentalness',
'liveness', 'speechiness']

fig, axes = plt.subplots(3, 3, figsize=(16, 12))
axes = axes.flatten()
```

```
colormap = plt.cm.get_cmap('viridis')

for i, column in enumerate(columns_to_plot):
    plt.sca(axes[i])

    scatter = plt.scatter(df_da[column], df_da['valence'],
c=df_da['valence'], cmap=colormap, alpha=0.5)

    plt.xlabel(column, fontsize=12)
    plt.ylabel('Valence', fontsize=12)
    plt.title(f'{column} vs. Valence', fontsize=14)
    plt.grid(True)
    plt.colorbar(scatter, label='Valence')

for j in range(len(columns_to_plot), len(axes)):
    fig.delaxes(axes[j])

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

```
/tmp/ipykernel_19/2977257625.py:6: MatplotlibDeprecationWarning: The
get_cmap function was deprecated in Matplotlib 3.7 and will be removed
two minor releases later. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap(obj)`` instead.
  colormap = plt.cm.get_cmap('viridis')
```



Energy vs Valence

- There is a negative correlation between energy and valence. This means that as the energy of a song increases, the valence (positiveness) tends to decrease. This makes sense, as high-energy songs are often associated with excitement or intensity, which can be positive or negative depending on the context. For example, a happy dance song would have high energy and valence, while an angry metal song would have high energy and low valence.

Acousticness vs Valence

- There is a positive correlation between acousticness and valence. This means that as the acousticness of a song increases, the valence (positiveness) tends to increase as well. This makes sense, as acoustic songs are often seen as more mellow and relaxing, which are typically associated with positive emotions.

Instrumentalness vs Valence

- There is a weak positive correlation between instrumentalness and valence. This means that as the instrumentalness of a song increases, the valence (positiveness) tends to increase slightly as well. This is likely because instrumental music is often used in relaxing or sentimental contexts, which can evoke positive emotions.

Liveness vs Valence

- There is a weak negative correlation between liveness and valence. This means that as the liveness of a song increases, the valence (positiveness) tends to decrease slightly as well. This is likely because live music can be more unpredictable and raw than studio recordings, which can make it seem less positive.

Speechiness vs Valence

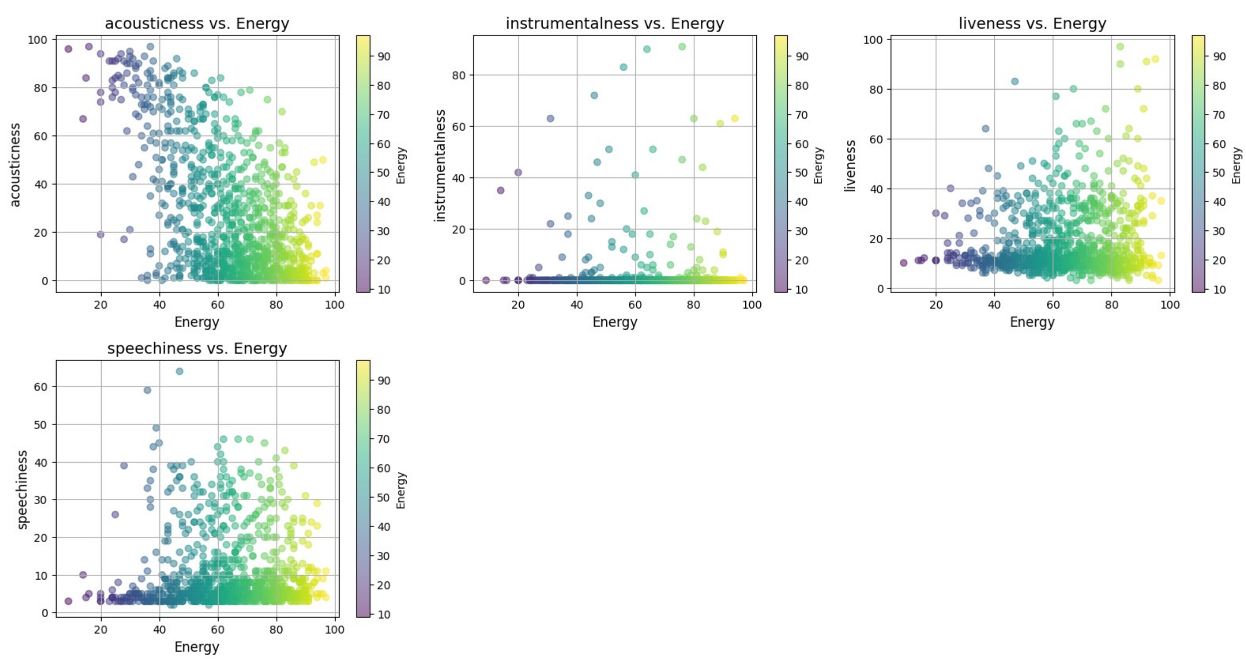
- There is a weak negative correlation between speechiness and valence. This means that as the speechiness of a song increases, the valence (positiveness) tends to decrease slightly as well. This is likely because spoken word can be used to express a wider range of emotions, including negative ones, than singing.

These scatter plots provide some interesting insights into the relationship between different audio features and the perceived emotion of a song. It is important to note that these are just correlations, and there are always exceptions. For example, there are plenty of high-energy songs that are happy and positive, and there are also plenty of acoustic songs that are sad or angry.

```
columns_to_plot = ['acousticness', 'instrumentalness', 'liveness',
'speechiness']

fig, axes = plt.subplots(3, 3, figsize=(16, 12))
axes = axes.flatten()

colormap = plt.cm.get_cmap('viridis')

for i, column in enumerate(columns_to_plot):
    plt.sca(axes[i])

    scatter = plt.scatter(df_da['energy'], df_da[column],
c=df_da['energy'], cmap=colormap, alpha=0.5)
```

```
    plt.xlabel('Energy', fontsize=12)
    plt.ylabel(column, fontsize=12)
    plt.title(f'{column} vs. Energy', fontsize=14)
    plt.grid(True)

    plt.colorbar(scatter, label='Energy')

for j in range(len(columns_to_plot), len(axes)):
    fig.delaxes(axes[j])

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

```
/tmp/ipykernel_19/1342802974.py:6: MatplotlibDeprecationWarning: The
get_cmap function was deprecated in Matplotlib 3.7 and will be removed
two minor releases later. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap(obj)`` instead.
  colormap = plt.cm.get_cmap('viridis')
```



Energy is most strongly correlated with Acousticness followed by Liveness to a lesser extent. Energy is not strongly correlated with Instrumentalness, or Speechiness.

Energy vs. Acousticness

- There is a moderate negative correlation between Energy and Acousticness. This means that as Energy increases, Acousticness tends to decrease, and vice versa. This aligns with the common association of acoustic music with slower tempos and mellow moods, which tend to have lower Energy levels. However, there are many data points that deviate from this trend, suggesting that Acousticness is not solely determined by Energy.

Energy vs. Instrumentalness

- There is no clear correlation between Energy and Instrumentalness. There are high-Energy songs with both vocals and instrumentals, and there are low-Energy songs with both vocals and instrumentals. This suggests that Energy is influenced by factors other than the presence or absence of vocals.

Energy vs. Liveness

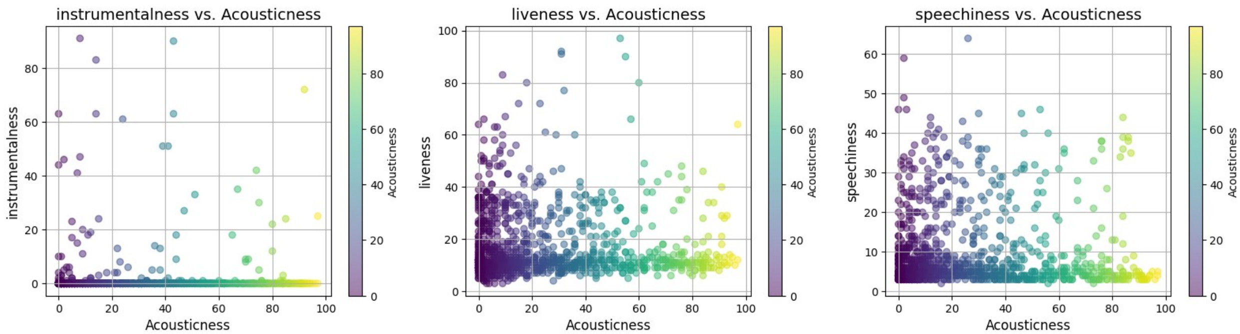
- There is a moderate positive correlation between Energy and Liveness. This means that as Energy increases, Liveness also tends to increase, and vice versa. This suggests that songs perceived as livelier tend to have higher Energy levels as well. However, there are many data points that deviate from this trend, and Energy is not a strong predictor of Liveness.

Energy vs. Speechiness

- There is no clear correlation between Energy and Speechiness. There are high-Energy songs with both high and low Speechiness, and vice versa. This suggests that Energy is not influenced by the amount of spoken word in a song.

```
columns_to_plot = ['instrumentalness', 'liveness', 'speechiness']

fig, axes = plt.subplots(3, 3, figsize=(16, 12))
axes = axes.flatten()

colormap = plt.cm.get_cmap('viridis')

for i, column in enumerate(columns_to_plot):
    plt.sca(axes[i])

    scatter = plt.scatter(df_da['acousticness'], df_da[column], 
c=df_da['acousticness'], cmap=colormap, alpha=0.5)

    plt.xlabel('Acousticness', fontsize=12)
    plt.ylabel(column, fontsize=12)
    plt.title(f'{column} vs. Acousticness', fontsize=14)
    plt.grid(True)
    plt.colorbar(scatter, label='Acousticness')

for j in range(len(columns_to_plot), len(axes)):
    fig.delaxes(axes[j])

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

```
/tmp/ipykernel_19/421492842.py:6: MatplotlibDeprecationWarning: The 
get_cmap function was deprecated in Matplotlib 3.7 and will be removed 
two minor releases later. Use ``matplotlib.colormaps[name]`` or 
``matplotlib.colormaps.get_cmap(obj)`` instead.
  colormap = plt.cm.get_cmap('viridis')
```



Acousticness vs. Instrumentalness

- This scatter plot shows a weak positive correlation between acousticness and instrumentalness. This means that as the acousticness of a song increases, the instrumentalness also tends to increase slightly. This makes sense, as acoustic music often relies heavily on instrumental arrangements, while more electronic or pop music may have more vocals or synthesized elements. There are a few outliers in the top right corner of the plot, which represent songs that are both highly acoustic and highly instrumental. These could be songs with orchestral arrangements or solo instrumental pieces.

Acousticness vs. Liveness

- This scatter plot shows a weak negative correlation between acousticness and liveness. This means that as the acousticness of a song increases, the liveness tends to decrease slightly. This makes sense, as acoustic music is often recorded in a studio setting, while live music is often more raw and unpolished. There are a few outliers in the bottom left corner of the plot, which represent songs that are both highly acoustic and highly live. These could be live acoustic recordings or songs that were recorded with a live band.

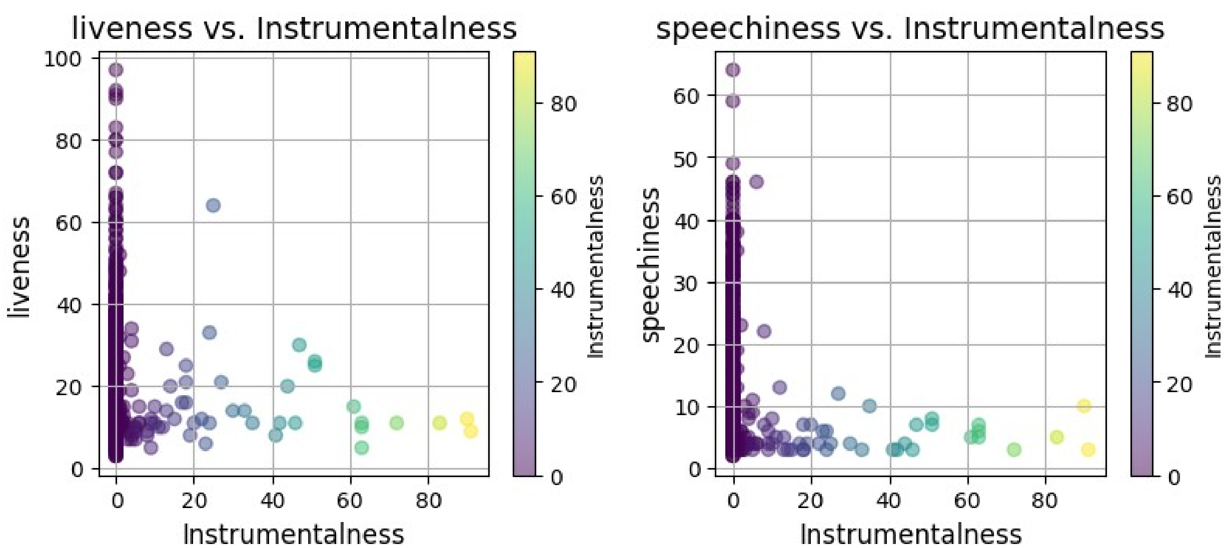
Acousticness vs. Speechiness

- This scatter plot shows a weak negative correlation between acousticness and speechiness. This means that as the acousticness of a song increases, the speechiness tends to decrease slightly. This makes sense, as acoustic music typically focuses on singing or instrumental melodies, while spoken word is more common in other genres. There are a few outliers in the top right corner of the plot, which represent songs that are both highly acoustic and highly speechy. These could be spoken word pieces set to music or songs with a lot of spoken word outros or outros.

```
columns_to_plot = ['liveness', 'speechiness']

fig, axes = plt.subplots(3, 3, figsize=(12, 10))
axes = axes.flatten()

colormap = plt.cm.get_cmap('viridis')

for i, column in enumerate(columns_to_plot):
    plt.sca(axes[i])
```

```
    scatter = plt.scatter(df_da['instrumentalness'], df_da[column],
c=df_da['instrumentalness'], cmap=colormap, alpha=0.5)

    plt.xlabel('Instrumentalness', fontsize=12)
    plt.ylabel(column, fontsize=12)
    plt.title(f'{column} vs. Instrumentalness', fontsize=14)
    plt.grid(True)
    plt.colorbar(scatter, label='Instrumentalness')

for j in range(len(columns_to_plot), len(axes)):
    fig.delaxes(axes[j])

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

```
/tmp/ipykernel_19/3196358449.py:6: MatplotlibDeprecationWarning: The
get_cmap function was deprecated in Matplotlib 3.7 and will be removed
two minor releases later. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap(obj)`` instead.
  colormap = plt.cm.get_cmap('viridis')
```



Speechiness vs. Instrumentalness

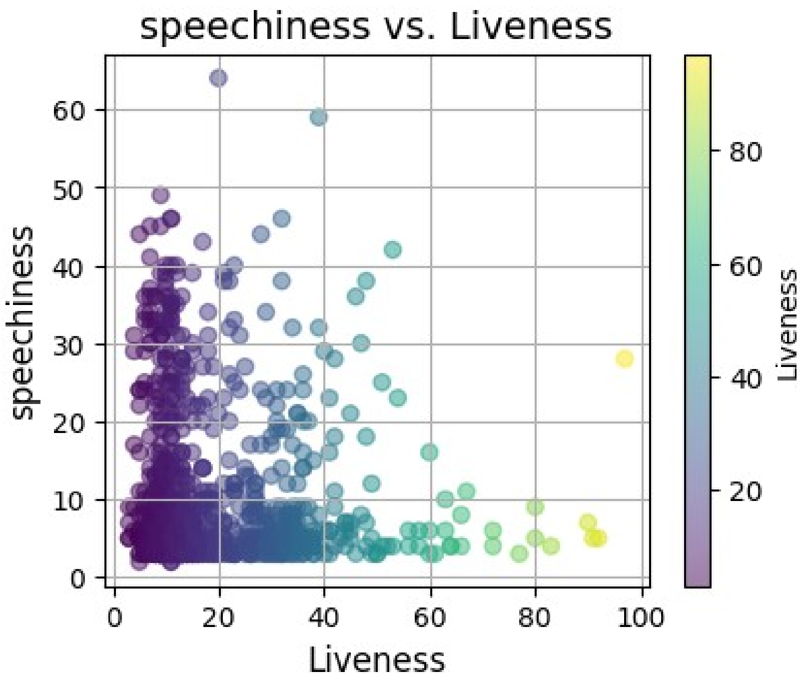
- This scatter plot reveals a weak negative correlation between speechiness and instrumentalness. In simpler terms, as the speechiness of a song increases, its instrumentalness tends to decrease. This makes sense since songs heavy on spoken word poetry or rap vocals usually have fewer or simpler instrumental parts.
- On the other hand, songs that are purely instrumental, or have minimal vocals, tend to have higher instrumentalness scores.
- The data points are quite dense in the center of the plot, suggesting that most songs have a moderate level of both speechiness and instrumentalness. The smattering of data points towards the bottom right and top left corners represent outliers.

Liveness vs. Instrumentalness

- This scatter plot displays a slightly positive correlation between liveness and instrumentalness. As the liveness of a song increases, its instrumentalness tends to increase slightly as well. This might be because live music often features extended instrumental solos or improvisations, compared to the more controlled studio recordings.
- Data points are again concentrated in the middle of the plot, with sparser outliers in the opposite corners. This suggests that most songs on Spotify have a moderate level of both liveness and instrumentalness.

```
columns_to_plot = ['speechiness']

fig, axes = plt.subplots(3, 3, figsize=(12, 10))
axes = axes.flatten()

colormap = plt.cm.get_cmap('viridis')

for i, column in enumerate(columns_to_plot):
    plt.sca(axes[i])
    scatter = plt.scatter(df_da['liveness'], df_da[column], 
c=df_da['liveness'], cmap=colormap, alpha=0.5)

    plt.xlabel('Liveness', fontsize=12)
    plt.ylabel(column, fontsize=12)
    plt.title(f'{column} vs. Liveness', fontsize=14)
    plt.grid(True)
    plt.colorbar(scatter, label='Liveness')

for j in range(len(columns_to_plot), len(axes)):
    fig.delaxes(axes[j])

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

```
/tmp/ipykernel_19/1913927620.py:6: MatplotlibDeprecationWarning: The 
get_cmap function was deprecated in Matplotlib 3.7 and will be removed 
two minor releases later. Use ``matplotlib.colormaps[name]`` or 
``matplotlib.colormaps.get_cmap(obj)`` instead.
  colormap = plt.cm.get_cmap('viridis')
```



Based on the scatter plot, there appears to be a weak negative correlation between speechiness and liveness. This means that as the speechiness of a song increases, the liveness tends to decrease slightly. In other words, songs with a lot of spoken word or rap vocals are less likely to be live recordings.

- Live music often prioritizes singing and instrumentation over spoken word. Singers and musicians might want to avoid talking too much between songs to keep the energy of the performance high.
- Spoken word can be difficult to hear in a live setting. Depending on the venue and sound system, it can be challenging to hear spoken word clearly over the background noise of a live audience.
- Genres with high speechiness are less likely to be played live. Genres like hip-hop and rap, which often feature spoken word vocals, are less common in live music settings than genres like rock or pop, which typically focus on singing.

```
bpm = df_da['bpm']
danceability = df_da['danceability']
streams = df_da['streams']

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(bpm, danceability, streams, c=streams, cmap='viridis',
marker='o')
ax.set_xlabel('BPM')
ax.set_ylabel('Danceability')
ax.set_zlabel('Streams')

cbar = fig.colorbar(ax.scatter(bpm, danceability, streams, c=streams,
```

```
cmap='viridis', marker='o'), ax=ax)
cbar.set_label('Streams', rotation=90)
plt.title('3D Scatter Plot: BPM vs. Danceability vs. Streams')
plt.show()
```