Music Recommendation System

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import regex as re
import datetime

import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
df= pd.read_csv("data.csv")
df_genre = pd.read_csv('data_by_genres.csv')
df_year = pd.read_csv('data_by_year.csv')
df_artist = pd.read_csv('data_by_artist.csv')
df.columns
```

Out[ ]:
```
Index(['valence', 'year', 'acousticness', 'artists', 'danceability',
       'duration_ms', 'energy', 'explicit', 'id', 'instrumentalness', 'key',
       'liveness', 'loudness', 'mode', 'name', 'popularity', 'release_date',
       'speechiness', 'tempo'],
      dtype='object')
```

In [ ]:
```
dataset=[df,df_genre,df_year,df_artist]
for info in dataset:
    print(info.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23 entries, 0 to 22
Data columns (total 19 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   valence           23 non-null     float64
 1   year              23 non-null     int64
 2   acousticness      23 non-null     float64
 3   artists           23 non-null     object
 4   danceability      23 non-null     float64
 5   duration_ms       23 non-null     int64
 6   energy            23 non-null     float64
 7   explicit          23 non-null     int64
 8   id                23 non-null     object
 9   instrumentalness  23 non-null     float64
 10  key               23 non-null     int64
 11  liveness          23 non-null     float64
 12  loudness          23 non-null     float64
 13  mode              23 non-null     int64
 14  name              23 non-null     object
 15  popularity        23 non-null     int64
 16  release_date      23 non-null     object
 17  speechiness       23 non-null     float64
 18  tempo             23 non-null     float64
dtypes: float64(9), int64(6), object(4)
memory usage: 3.5+ KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2973 entries, 0 to 2972
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   mode              2973 non-null   int64
 1   genres            2973 non-null   object
 2   acousticness      2973 non-null   float64
 3   danceability      2973 non-null   float64
 4   duration_ms       2973 non-null   float64
 5   energy            2973 non-null   float64
 6   instrumentalness  2973 non-null   float64
 7   liveness          2973 non-null   float64
 8   loudness          2973 non-null   float64
 9   speechiness       2973 non-null   float64
 10  tempo             2973 non-null   float64
 11  valence           2973 non-null   float64
 12  popularity        2973 non-null   float64
 13  key               2973 non-null   int64
dtypes: float64(11), int64(2), object(1)
memory usage: 325.3+ KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   mode              100 non-null    int64
 1   year              100 non-null    int64
 2   acousticness      100 non-null    float64
 3   danceability      100 non-null    float64
 4   duration_ms       100 non-null    float64
 5   energy            100 non-null    float64
```

```
 6   instrumentalness  100 non-null    float64
 7   liveness          100 non-null    float64
 8   loudness          100 non-null    float64
 9   speechiness       100 non-null    float64
 10  tempo             100 non-null    float64
 11  valence           100 non-null    float64
 12  popularity        100 non-null    float64
 13  key               100 non-null    int64
dtypes: float64(11), int64(3)
memory usage: 11.1 KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28680 entries, 0 to 28679
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   mode              28680 non-null  int64
 1   count             28680 non-null  int64
 2   acousticness      28680 non-null  float64
 3   artists           28680 non-null  object
 4   danceability      28680 non-null  float64
 5   duration_ms       28680 non-null  float64
 6   energy            28680 non-null  float64
 7   instrumentalness  28680 non-null  float64
 8   liveness          28680 non-null  float64
 9   loudness          28680 non-null  float64
 10  speechiness       28680 non-null  float64
 11  tempo             28680 non-null  float64
 12  valence           28680 non-null  float64
 13  popularity        28680 non-null  float64
 14  key               28680 non-null  int64
dtypes: float64(11), int64(3), object(1)
memory usage: 3.3+ MB
None
```

In [ ]:

```
# duplicacy in df,df_genre,df_year,df_artist
for duplicate in dataset:
    print(duplicate.duplicated(keep=False).sum())
```

```
0
0
0
0
```

In [ ]:

```
# unique values in df,df_genre,df_year,df_artist
for unique in dataset:
    print("-"*26)
    print(df.nunique())
```

```
-------------------------
valence             23
year                 3
acousticness        23
artists             23
danceability        23
duration_ms         23
energy              23
explicit             2
id                  23
instrumentalness    19
key                 10
liveness            22
loudness            23
mode                 2
name                23
popularity          17
release_date        18
speechiness         23
tempo               23
dtype: int64
-------------------------
valence             23
year                 3
acousticness        23
artists             23
danceability        23
duration_ms         23
energy              23
explicit             2
id                  23
instrumentalness    19
key                 10
liveness            22
loudness            23
mode                 2
name                23
popularity          17
release_date        18
speechiness         23
tempo               23
dtype: int64
-------------------------
valence             23
year                 3
acousticness        23
artists             23
danceability        23
duration_ms         23
energy              23
explicit             2
id                  23
instrumentalness    19
key                 10
liveness            22
loudness            23
mode                 2
name                23
popularity          17
release_date        18
```

```
speechiness         23
tempo               23
dtype: int64
--------------------------
valence             23
year                 3
acousticness        23
artists             23
danceability        23
duration_ms         23
energy              23
explicit             2
id                  23
instrumentalness    19
key                 10
liveness            22
loudness            23
mode                 2
name                23
popularity          17
release_date        18
speechiness         23
tempo               23
dtype: int64
```

In [ ]:

```
# unique values in df,df_genre,df_year,df_artist
for describe in dataset:
    print("-"*26)
    print(describe.describe())
```

```
--------------------------
         valence         year  acousticness  danceability   duration_ms  \
count  23.000000    23.000000     23.000000     23.000000      23.00000   
mean    0.485548  1993.304348      0.428909      0.564261  289450.00000   
std     0.285851    39.678750      0.398579      0.209433  183878.97182   
min     0.039400  1921.000000      0.000412      0.175000  133500.00000   
25%     0.217500  2003.000000      0.085400      0.431500  185020.00000   
50%     0.549000  2003.000000      0.206000      0.598000  216600.00000   
75%     0.695500  2020.000000      0.886000      0.741500  289837.00000   
max     0.963000  2020.000000      0.994000      0.917000  831667.00000   

          energy   explicit  instrumentalness        key   liveness  \
count  23.000000  23.000000         23.000000  23.000000  23.000000   
mean    0.528995   0.173913          0.175944   5.434783   0.271487   
std     0.284759   0.387553          0.353559   2.873485   0.264998   
min     0.007590   0.000000          0.000000   0.000000   0.077400   
25%     0.325000   0.000000          0.000005   3.500000   0.104500   
50%     0.569000   0.000000          0.000052   6.000000   0.164000   
75%     0.737000   0.000000          0.040200   7.000000   0.293500   
max     0.979000   1.000000          0.959000  10.000000   0.995000   

        loudness       mode  popularity  speechiness       tempo  
count  23.000000  23.000000   23.000000    23.000000   23.000000  
mean  -10.458565   0.782609   47.130435     0.109539  109.919696  
std     8.256127   0.421741   26.237212     0.106287   27.707176  
min   -35.072000   0.000000    2.000000     0.024900   60.936000  
25%   -10.723500   1.000000   39.500000     0.036000   93.339500  
50%    -8.480000   1.000000   48.000000     0.063100  103.054000  
75%    -5.750500   1.000000   69.000000     0.125500  132.401500  
max    -2.226000   1.000000   76.000000     0.415000  170.853000  
--------------------------
              mode  acousticness  danceability   duration_ms       energy  \
count  2973.000000   2973.000000   2973.000000  2.973000e+03  2973.000000   
mean      0.833165      0.401241      0.537187  2.517209e+05     0.561143   
std       0.372891      0.319760      0.150668  9.465686e+04     0.234486   
min       0.000000      0.000003      0.056900  3.094600e+04     0.001002   
25%       1.000000      0.119050      0.441202  2.063788e+05     0.395058   
50%       1.000000      0.321745      0.546496  2.375453e+05     0.601195   
75%       1.000000      0.673991      0.647500  2.772720e+05     0.730127   
max       1.000000      0.996000      0.929000  2.382587e+06     0.994667   

       instrumentalness     liveness     loudness  speechiness        tempo  \
count       2973.000000  2973.000000  2973.000000  2973.000000  2973.000000   
mean           0.211366     0.192800   -10.509848     0.083588   119.018723   
std            0.267329     0.092356     5.369202     0.080483    17.469188   
min            0.000000     0.022200   -41.825000     0.023800    47.135722   
25%            0.004835     0.137687   -12.427656     0.044900   109.198143   
50%            0.080700     0.178764    -9.221817     0.059457   119.194167   
75%            0.343333     0.220856    -6.920125     0.091000   127.508750   
max            0.992000     0.960000     0.060000     0.946219   204.212000   

           valence   popularity          key  
count  2973.000000  2973.000000  2973.000000  
mean      0.492748    39.919185     5.938782  
std       0.201820    16.748723     3.368110  
min       0.003353     0.000000     0.000000  
25%       0.348578    32.491279     3.000000  
50%       0.500048    43.056569     7.000000  
75%       0.640257    51.138889     9.000000  
max       0.980000    80.666667    11.000000  
```

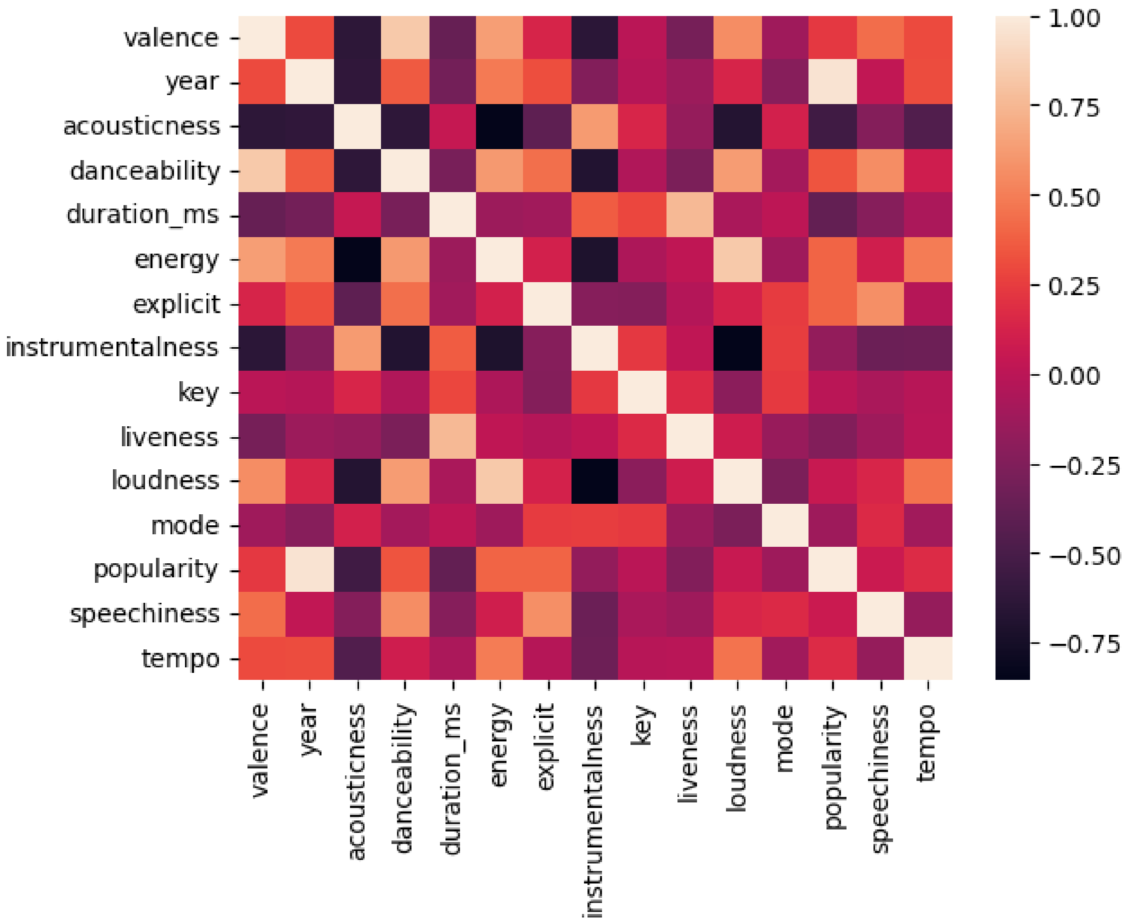
```
--------------------------
       mode         year  acousticness  danceability    duration_ms  \
count  100.0   100.000000    100.000000    100.000000     100.000000
mean     1.0  1970.500000      0.556317      0.536783  227296.752234
std      0.0    29.011492      0.275358      0.052356   25630.048065
min      1.0  1921.000000      0.219931      0.414445  156881.657475
25%      1.0  1945.750000      0.289516      0.500800  210889.193536
50%      1.0  1970.500000      0.459190      0.540976  235520.850833
75%      1.0  1995.250000      0.856711      0.570948  247702.738058
max      1.0  2020.000000      0.962607      0.692904  267677.823086

           energy  instrumentalness    liveness    loudness  speechiness  \
count  100.000000        100.000000  100.000000  100.000000   100.000000
mean     0.452705          0.193582    0.208224  -11.969054     0.105861
std      0.161738          0.122488    0.017903    3.105610     0.082128
min      0.207948          0.016376    0.168450  -19.275282     0.049098
25%      0.280733          0.103323    0.197509  -14.189232     0.064244
50%      0.495997          0.127644    0.206074  -11.773061     0.085763
75%      0.598008          0.276707    0.218493   -9.950542     0.104438
max      0.681778          0.581701    0.264335   -6.595067     0.490001

            tempo     valence  popularity       key
count  100.000000  100.000000  100.000000  100.0000
mean   116.015674    0.532120   27.376065    3.7900
std      5.669645    0.057809   20.703197    3.5627
min    100.884521    0.379327    0.140845    0.0000
25%    111.718626    0.497174    3.298200    0.0000
50%    117.455548    0.541503   33.619250    2.0000
75%    120.606644    0.570080   44.943375    7.0000
max    124.283129    0.663725   65.256542   10.0000
--------------------------
               mode         count  acousticness  danceability   duration_ms  \
count  28680.000000  28680.000000  28680.000000  28680.000000  2.868000e+04
mean       0.759170     13.847211      0.498373      0.546490  2.388780e+05
std        0.427595     53.372544      0.370614      0.176474  1.211318e+05
min        0.000000      1.000000      0.000000      0.000000  1.879550e+04
25%        1.000000      2.000000      0.122296      0.431000  1.823304e+05
50%        1.000000      3.000000      0.478458      0.557000  2.186400e+05
75%        1.000000      8.000000      0.896000      0.675000  2.684670e+05
max        1.000000   3169.000000      0.996000      0.986000  5.403500e+06

             energy  instrumentalness      liveness      loudness  \
count  28680.000000      28680.000000  28680.000000  28680.000000
mean       0.497488          0.174756      0.202441    -11.140498
std        0.254885          0.298406      0.140884      5.771749
min        0.000000          0.000000      0.000000    -60.000000
25%        0.283568          0.000004      0.110362    -13.972292
50%        0.504000          0.001880      0.161000    -10.088938
75%        0.702783          0.215291      0.247000     -6.889000
max        1.000000          1.000000      0.991000      1.342000

        speechiness         tempo       valence    popularity           key
count  28680.000000  28680.000000  28680.000000  28680.000000  28680.000000
mean       0.094014    115.844830      0.512723     34.060945      5.412901
std        0.111986     25.003834      0.244421     22.376438      3.480552
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.039200     99.366500      0.329000     12.000000      2.000000
50%        0.052200    115.357400      0.523243     39.000000      6.000000
75%        0.095300    129.848750      0.703000     51.000000      8.000000
max        0.964000    217.743000      0.991000     93.000000     11.000000
```

In [ ]:
```
df.corr()
```

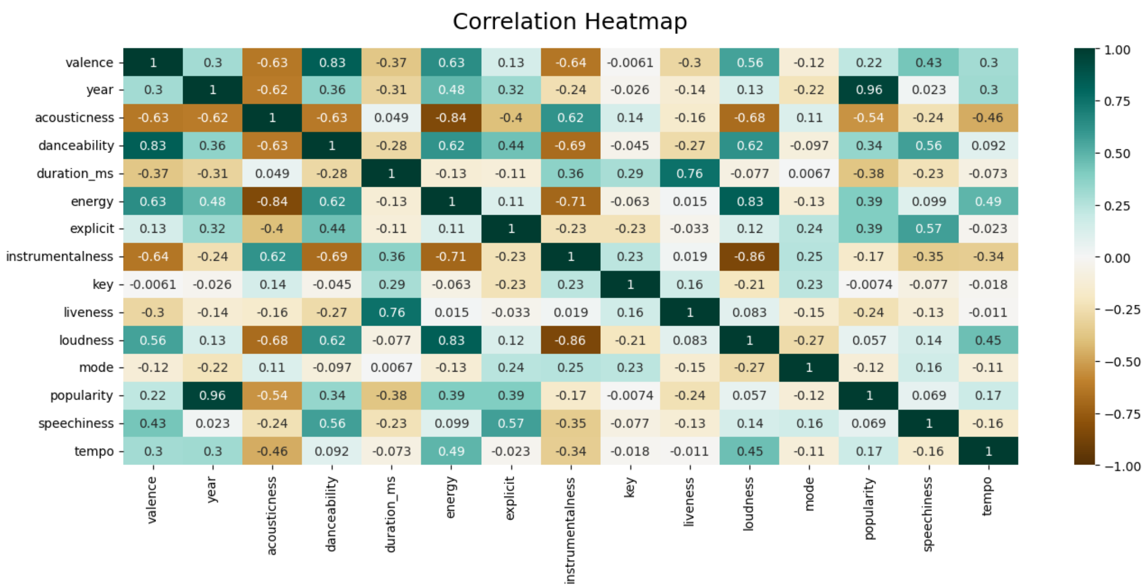
Out[ ]:

	valence	year	acousticness	danceability	duration_ms	energy	ex
valence	1.000000	0.302162	-0.634777	0.830801	-0.368543	0.634503	0.13
year	0.302162	1.000000	-0.621355	0.357510	-0.308038	0.479710	0.31
acousticness	-0.634777	-0.621355	1.000000	-0.630119	0.049085	-0.841872	-0.40
danceability	0.830801	0.357510	-0.630119	1.000000	-0.284727	0.615223	0.43
duration_ms	-0.368543	-0.308038	0.049085	-0.284727	1.000000	-0.131567	-0.10
energy	0.634503	0.479710	-0.841872	0.615223	-0.131567	1.000000	0.11
explicit	0.132860	0.315637	-0.400234	0.439589	-0.109845	0.110803	1.00
instrumentalness	-0.641412	-0.243435	0.616613	-0.686280	0.364454	-0.709671	-0.23
key	-0.006069	-0.025532	0.140536	-0.044685	0.287071	-0.063132	-0.23
liveness	-0.296086	-0.138276	-0.163948	-0.273193	0.755077	0.015204	-0.03
loudness	0.560510	0.131692	-0.680622	0.624367	-0.076733	0.828849	0.11
mode	-0.121506	-0.224033	0.113394	-0.096592	0.006711	-0.128317	0.24
popularity	0.224667	0.957201	-0.542146	0.336395	-0.378562	0.393905	0.39
speechiness	0.427230	0.022908	-0.238654	0.564448	-0.226630	0.099193	0.56
tempo	0.298032	0.304688	-0.464544	0.091939	-0.072998	0.486373	-0.02

In [ ]:
```
sns.heatmap(df.corr())
plt.show()
```

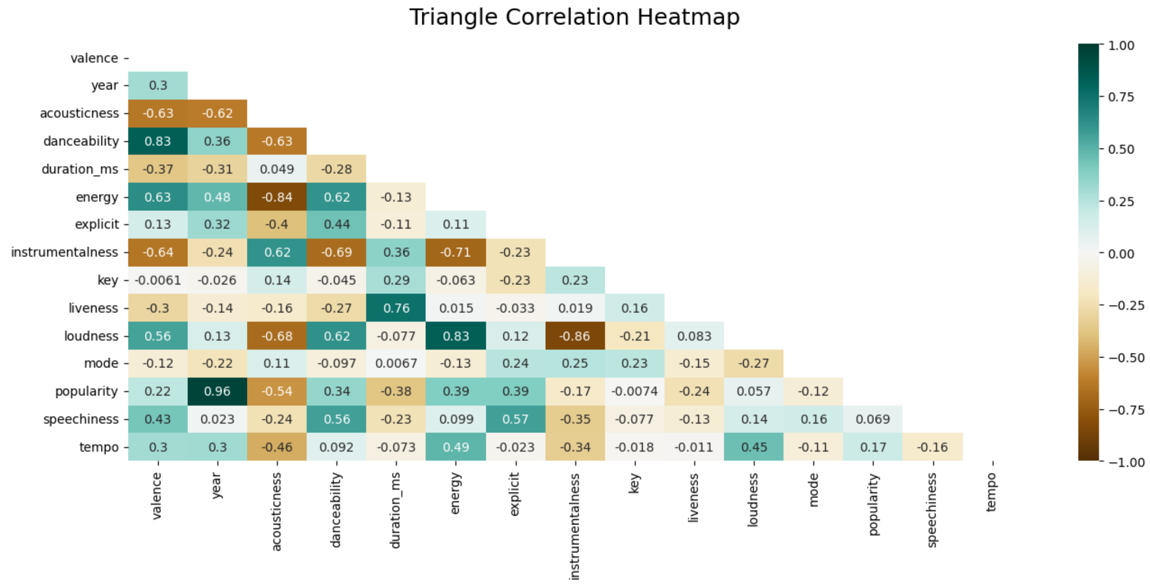


In [ ]:
```
plt.figure(figsize=(16, 6))
heatmap = sns.heatmap(df.corr(), vmin=-1, vmax=1, annot=True, cmap='BrBG')
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':18}, pad=16)
plt.savefig('heatmap.png', dpi=300, bbox_inches='tight')
```



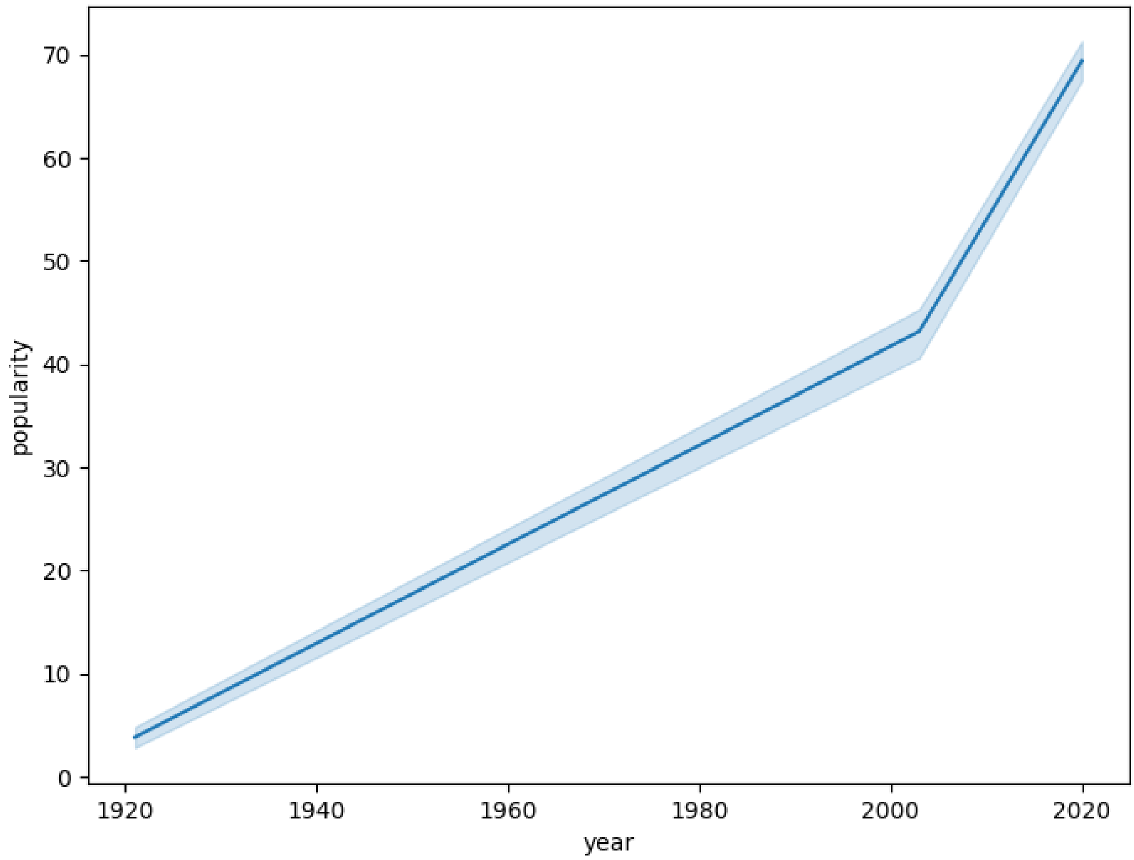
In [ ]:
```
mask = np.triu(np.ones_like(df.corr(), dtype=np.bool))
plt.figure(figsize=(16, 6))
heatmap = sns.heatmap(df.corr(), mask=mask, vmin=-1, vmax=1, annot=True, cmap='B
heatmap.set_title('Triangle Correlation Heatmap', fontdict={'fontsize':18}, pad=
```

Out[ ]:
```
Text(0.5, 1.0, 'Triangle Correlation Heatmap')
```



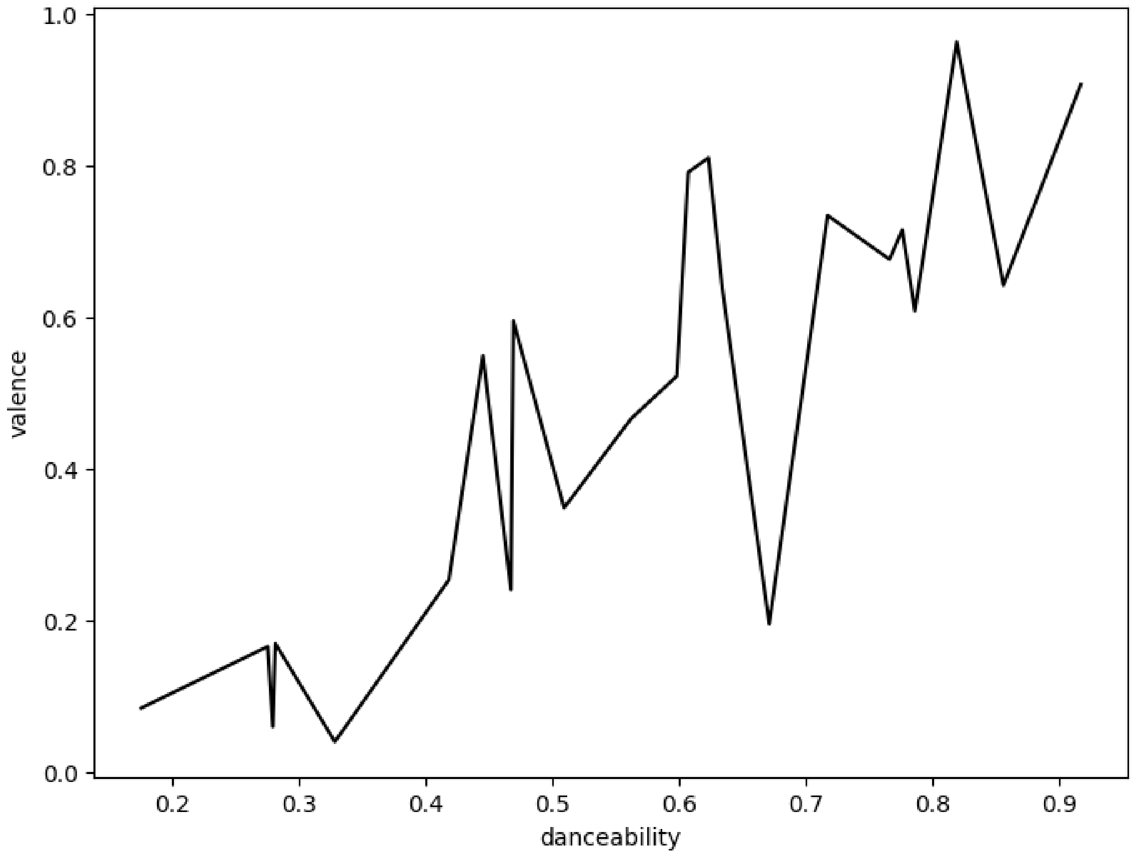
In [ ]:
```
plt.figure(figsize=(8, 6))
sns.lineplot(x=df['year'], y=df['popularity'])
# Upwards trend in danceability from 1920 until a downward trend from approximat
# This graph shows an upward trend in danceability from about 1945-1950 onwards.
```

Out[ ]: `<AxesSubplot: xlabel='year', ylabel='popularity'>`



In [ ]:
```
plt.figure(figsize=(8, 6))
sns.lineplot(x=df['danceability'], y=df['valence'], color= "black")
# Funny looking graph...upwards trend in popularity vs danceability until I'm gu
# "happy".
# Perhaps like Billie Eilish type songs?
```

Out[ ]: `<AxesSubplot: xlabel='danceability', ylabel='valence'>`



Data Preprocessing

In [ ]:
```
df= df.drop(columns=['id','release_date'])
```

Text Preprocessing

In [ ]:
```
# %%capture

def remove_special_characters(text):
    pattern = r'[^a-zA-Z0-9\s]'
    cleaned_text = re.sub(pattern, '', text)
    return cleaned_text

df['name'] = df['name'].apply(remove_special_characters)
df['artists'] = df['artists'].apply(remove_special_characters)

# print(df['name'])
# print(df['artists'])
```

Accesing information about any song

In [ ]:
```
df.iloc[5] #enter the serial no. of the particular song from data.csv to know pr
df.iloc[5].artists
```

Out[ ]:
```
'Wanda Jackson The Cramps'
```

Content Based Filtering

- Recommends items based on the previous items the same consumer selected in the past.
- Best used when the focus is on one user.
- attributes of the items are the most crucial factors
- recommends items by looking at their characteristics, like genres or descriptions, & matches them with what users have liked before.

2 step process
Firstly, extract features out of the content of the song descriptions to create an object representation.
Second, define a similarity function among these object representations which mimics what human understands as an item-item similarity

- It begins by identifying the keywords to understand the context of the content.
- In this step, it avoids unnecessary words such as stop words.
- Then it finds the same kind of context in other content to find the similarities.
- To determine the similarities between two or more contents, the content-based method uses cosine similarities.
- It finds similarities by analyzing the correlation between two or more users.
- Then finally it generates recommendations by calculating the weighted average of all user ratings for active users.

In [ ]:
```
def user(artist_name,song_name,df):
    desired_song = df[(df['artists'].str.contains(artist_name)) & (df['name'] ==
    desired_song=desired_song.drop(columns=['artists','name'])
    return (np.array(desired_song)).flatten()

# user('Anuel AA Daddy Yankee KAROL G Ozuna J Balvin','China',df)
whole_data= [user(row['artists'], row['name'],df)for index, row in df.iterrows()
whole_data[:3]
```

Out[ ]:
```
[array([ 5.94000e-02,  1.92100e+03,  9.82000e-01,  2.79000e-01,
         8.31667e+05,  2.11000e-01,  0.00000e+00,  8.78000e-01,
         1.00000e+01,  6.65000e-01, -2.00960e+01,  1.00000e+00,
         4.00000e+00,  3.66000e-02,  8.09540e+01]),
 array([ 9.63000e-01,  1.92100e+03,  7.32000e-01,  8.19000e-01,
         1.80533e+05,  3.41000e-01,  0.00000e+00,  0.00000e+00,
         7.00000e+00,  1.60000e-01, -1.24410e+01,  1.00000e+00,
         5.00000e+00,  4.15000e-01,  6.09360e+01]),
 array([ 3.94000e-02,  1.92100e+03,  9.61000e-01,  3.28000e-01,
         5.00062e+05,  1.66000e-01,  0.00000e+00,  9.13000e-01,
         3.00000e+00,  1.01000e-01, -1.48500e+01,  1.00000e+00,
         5.00000e+00,  3.39000e-02,  1.10339e+02])]
```

In [ ]:
```
# content_based_filtering syntax

# from sklearn.metrics.pairwise import cosine_similarity
# similarity = cosine_similarity([user('Hector Berlioz Arturo Toscanini','Rkczy
# top_similar_songs= np.sort(similarity).flatten()[::-1]
# print("Top 5 songs on Cosine Similarity to Rakcozy March:", top_similar_songs[
```

In [ ]:
```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
```

```
def content_based_filtering(song_features_list, target_song_index):
    similarities = {}
    target_song_features = song_features_list[target_song_index].reshape(1, -1)

    # Iterate over each song in the list
    for i, features in enumerate(song_features_list):
        # Skip the target song itself
        if i == target_song_index:
            continue

        # Compute cosine similarity between target song features and current son
        similarity = cosine_similarity(target_song_features, features.reshape(1,

        # Store the similarity score for each song
        similarities[i] = similarity

    # Filter recommendations based on a similarity threshold (e.g., >= 0.5)
    recommendations = {df.iloc[i]['name']: similarity for i, similarity in simil
    return recommendations

# Assuming the index of the target song is 0 (for 'Rkczy March')
target_song_index = 11

# Get recommendations based on content-based filtering
recommendations = content_based_filtering(whole_data, target_song_index)

# Print the top recommended songs
print("Top recommended songs similar to 'Rakcozy March':")
for song_index, similarity in sorted(recommendations.items(), key=lambda x: x[1]
    # target_song_name = df.iloc[song_index]['name']
    print(f"Similarity Score: {similarity} of {song_index}: ")
```

```
Top recommended songs similar to 'Rakcozy March':
Similarity Score: 0.9999999896667197 of Covered in Rain  Live at the Oak Mounta
in Amphitheater Birmingham AL  September 2002:
Similarity Score: 0.9999997075784949 of Piano Concerto No 3 in D Minor Op 30 II
I Finale Alla breve:
Similarity Score: 0.9999996990880994 of Gati Bali:
Similarity Score: 0.9999957165053953 of Darkness:
Similarity Score: 0.9999933934491422 of China:
```

Collaborative filtering

- Collaborative filtering is based on the idea that users who have similar tastes or behaviors will like similar item.
- It does not require any information about the items themselves, such as their genres, features, or descriptions.
- Ratings, reviews, (thumbs up, stars, ratings) or implicit (views, clicks, time spent, purchases) or actions of many users can be used to predict other user behaviour

Two subtypes of collaborative filtering: User-based Item-based

we have chosen Item-based filtering on based on pooularity

In [ ]:
```
# # Collaborative filtering syntax
# def collaborative_filtering(user_preferences, user):
```

```
#     user_ratings = user_preferences[user]
#     recommendations = {song: rating for song, rating in user_ratings.items() i
#     return recommendations
```

In [ ]:

```
df.iloc[11]
```

Out[ ]:

```
valence                                                    0.348
year                                                        2003
acousticness                                               0.173
artists                                        Dave Matthews Band
danceability                                               0.509
duration_ms                                               652707
energy                                                     0.632
explicit                                                       0
instrumentalness                                         0.00613
key                                                            4
liveness                                                   0.995
loudness                                                  -7.144
mode                                                           0
name                Cortez the Killer  Live at Central Park New Yo...
popularity                                                    37
speechiness                                               0.0341
tempo                                                     113.91
Name: 11, dtype: object
```

In [ ]:

```
def collaborative_filtering(song_popularity, target_song_index, df, target_decad
    if target_decade % 10 != 0:
        raise ValueError("Target decade must be a multiple of 10.")

    if target_song_index==-1:
        print('new user')
    elif target_song_index < 0 or target_song_index >= len(song_popularity):
        raise ValueError("Invalid target song index.")
    # Check if the target song's year matches the target decade

    if target_song_index !=-1:
        target_song_year = df.iloc[target_song_index]['year']
        if target_song_year // 10 != target_decade // 10:
            raise ValueError("The target song does not belong to the specified d

    recommendations = {}
    for i, popularity in enumerate(song_popularity):
        if i != target_song_index and popularity >= 4:  # Exclude the target son
            # Check if the song belongs to the target decade
            song_year = df.iloc[i]['year']
            if song_year // 10 == target_decade // 10:
                recommendations[df.iloc[i]['name']] = popularity

    # Sort recommendations by popularity in descending order
    sorted_recommendations = dict(sorted(recommendations.items(), key=lambda ite

    return sorted_recommendations

song_popularity = df['popularity'].to_list()


target_song_index =-1  # Index of 'Rkczy March' in the DataFrame
target_decade=2000
if target_song_index==-1:
    target_decade = datetime.datetime.now().year//10 *10 # Target decade (e.g.,
```

```
else:
    target_decade

try:
    # Get recommendations based on collaborative filtering for the specified dec
    recommendations = collaborative_filtering(song_popularity, target_song_index
    target_song_name = df.iloc[target_song_index]['name']

    # Print the recommended songs
    print(f"Recommended songs similar to ---{target_song_name}--- song from the
    for song, popularity in recommendations.items():
        print(f"{popularity}: {song}")

except ValueError as e:
    print(f"Error: {e}")
```

```
new user
Recommended songs similar to ---Billetes Azules with J Balvin--- song from the
2020s based on popularity:
76: AYA
74: Billetes Azules with J Balvin
72: China
70: We Contain Multitudes from home
70: Med slutna gon
70: Darkness
68: Halloweenie III Seven Days
66: Soda feat Take A Daytrip
66: Sunblind
66: NASTY GIRL  ON CAMERA
65: Timeless Interlude
```

Hybrid Filtering

- You can also mix both approaches (hybrid) and get the best of both.
- In a hybrid approach, we combine the outputs of content-based and collaborative filtering methods to generate more accurate and diverse recommendations. By integrating both approaches, we can leverage the advantages of each method while mitigating their weaknesses.
- For example, content-based filtering can handle the cold-start problem by recommending items based on their features, while collaborative filtering can capture user preferences in the absence of item metadata.

In [ ]:
```
# Hybrid filtering syntax
# def hybrid_filtering(user_preferences, song_features, user):
#     collaborative_results = collaborative_filtering(user_preferences, user)
#     hybrid_recommendations = {}
#     for song, _ in collaborative_results.items():
#         content_based_results = content_based_filtering(song_features, song)
#         hybrid_recommendations.update(content_based_results)
#     return hybrid_recommendations

# # Example usage
# user = "user1"
# hybrid_recommendations = hybrid_filtering(user_preferences, song_features, use
# print("Hybrid Recommendations for User 1:", hybrid_recommendations)
```

In [ ]:
```
def hybrid_recommendation(song_features_list, song_popularity, target_song_index
    # Content-based filtering
    content_based_recommendations = content_based_filtering(song_features_list,t

    # Collaborative filtering
    collaborative_recommendations=collaborative_filtering(song_popularity, targe

    # Combine recommendations from both methods
    hybrid_recommendations = {}
    for song, popularity in collaborative_recommendations.items():
        if song in content_based_recommendations:
            # Combine similarity score and popularity
            hybrid_score = content_based_recommendations[song] * popularity
            hybrid_recommendations[song] = hybrid_score
        else:
            hybrid_recommendations[song] = popularity

    # Sort recommendations by hybrid score in descending order
    sorted_hybrid_recommendations = dict(sorted(hybrid_recommendations.items(),

    return sorted_hybrid_recommendations

# Example usage
target_song_index =-1  # Index of 'Rkczy March' in the DataFrame
target_decade=2000
if target_song_index==-1:
    target_decade = datetime.datetime.now().year//10 *10 # Target decade (e.g.,
else:
    target_decade  # Target decade (e.g., 1920s)

try:
    # Get hybrid recommendations
    hybrid_recommendations = hybrid_recommendation(whole_data, song_popularity,

    # Get the name of the target song
    target_song_name = df.iloc[target_song_index]['name']

    # Print the recommended songs
    print(f"Hybrid recommended songs similar to '{target_song_name}' from the {t
    for song, score in hybrid_recommendations.items():
        print(f"{score}: {song}")

except ValueError as e:
    print(f"Error: {e}")
```

```
new user
Hybrid recommended songs similar to 'Billetes Azules with J Balvin' from the 20
20s based on both content and popularity:
75.9999539260729: AYA
74.0: Billetes Azules with J Balvin
71.99943267572519: China
69.9999991648008: We Contain Multitudes from home
69.99929769444624: Med slutna gon
69.99923359998682: Darkness
67.99973717066482: Halloweenie III Seven Days
65.9998907025499: Soda feat Take A Daytrip
65.9998805257099: NASTY GIRL  ON CAMERA
65.99975961577024: Sunblind
64.99998422398909: Timeless Interlude
```

Product cold start

User actions are incredibly important since these determine the future of both product-to-product and personalized, user-history-based recommendations.

Visitor cold start

The user or visitor cold start simply means that a recommendation engine meets a new visitor for the first time. because there is no user history about her, the system doesn’t know the personal preferences of the user.

then we will show her the recommendations of the current decade or viral songs by marking her as new user->target_song_index=-1 if visitor is old_age, recommend her from the login-age factor if visitor location is different, recommend location popular songs of current target_decade

Conclusion

In conclusion, hybrid filtering offers a powerful and flexible approach for building recommendation systems that can deliver highly relevant and personalized recommendations to users. By combining the strengths of content-based and collaborative filtering methods, hybrid filtering enables recommendation systems to overcome limitations and achieve superior performance in real-world applications.