In [29]:

```
pip install --upgrade pip
```

```
Requirement already satisfied: pip in /Users/sravya/anaconda3/lib/python3.10/site-packages (23.2.
1)
Note: you may need to restart the kernel to use updated packages.
```

In [30]:

```
pip install wordcloud
```

```
Collecting wordcloud
  Obtaining dependency information for wordcloud from https://files.pythonhosted.org/packages/7d/
5c/7d2db8c698af6ec81c9038631ff54febed221be2450db5016856d2181c66/wordcloud-1.9.2-cp310-cp310-macos
x_10_9_x86_64.whl.metadata (https://files.pythonhosted.org/packages/7d/5c/7d2db8c698af6ec81c90386
31ff54febed221be2450db5016856d2181c66/wordcloud-1.9.2-cp310-cp310-macosx_10_9_x86_64.whl.metadat
a)
  Downloading wordcloud-1.9.2-cp310-cp310-macosx_10_9_x86_64.whl.metadata (3.3 kB)
Requirement already satisfied: numpy>=1.6.1 in /Users/sravya/anaconda3/lib/python3.10/site-packag
es (from wordcloud) (1.23.5)
Requirement already satisfied: pillow in /Users/sravya/anaconda3/lib/python3.10/site-packages (fr
om wordcloud) (9.4.0)
Requirement already satisfied: matplotlib in /Users/sravya/anaconda3/lib/python3.10/site-packages
(from wordcloud) (3.7.0)
Requirement already satisfied: contourpy>=1.0.1 in /Users/sravya/anaconda3/lib/python3.10/site-pa
ckages (from matplotlib->wordcloud) (1.0.5)
Requirement already satisfied: cycler>=0.10 in /Users/sravya/anaconda3/lib/python3.10/site-packag
es (from matplotlib->wordcloud) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /Users/sravya/anaconda3/lib/python3.10/site-p
ackages (from matplotlib->wordcloud) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/sravya/anaconda3/lib/python3.10/site-p
ackages (from matplotlib->wordcloud) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /Users/sravya/anaconda3/lib/python3.10/site-pac
kages (from matplotlib->wordcloud) (23.0)
Requirement already satisfied: pyparsing>=2.3.1 in /Users/sravya/anaconda3/lib/python3.10/site-pa
ckages (from matplotlib->wordcloud) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /Users/sravya/anaconda3/lib/python3.10/sit
e-packages (from matplotlib->wordcloud) (2.8.2)
Requirement already satisfied: six>=1.5 in /Users/sravya/anaconda3/lib/python3.10/site-packages
(from python-dateutil>=2.7->matplotlib->wordcloud) (1.16.0)
Downloading wordcloud-1.9.2-cp310-cp310-macosx_10_9_x86_64.whl (160 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 160.5/160.5 kB 1.2 MB/s eta 0:00:00a 0:00:01
Installing collected packages: wordcloud
Successfully installed wordcloud-1.9.2
Note: you may need to restart the kernel to use updated packages.
```

In [32]:

```
import numpy as np
import pandas as pd

import seaborn as sns
import plotly.express as px
import plotly.graph_objs as go
import matplotlib.pyplot as plt
from wordcloud import WordCloud

from collections import defaultdict
from scipy.spatial.distance import cdist
from sklearn.preprocessing import MinMaxScaler, StandardScaler

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

In [33]:

```
data = pd.read_csv("data.csv")
genre_data = pd.read_csv('data_by_genres.csv')
year_data = pd.read_csv('data_by_year.csv')
artist_data = pd.read_csv('data_by_artist.csv')
```

In [34]:

```
data.head()
```

Out[34]:

	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	id	instrumentaln
0	0.0594	1921	0.982	['Sergei Rachmaninoff', 'James Levine', 'Berli...	0.279	831667	0.211	0	4BJqT0PrAfrxzMOxytFOIz	0.878(
1	0.9630	1921	0.732	['Dennis Day']	0.819	180533	0.341	0	7xPhfUan2yNtyFG0cUWkt8	0.000(
2	0.0394	1921	0.961	['KHP Kridhamardawa Karaton Ngayogyakarta Hadi...	0.328	500062	0.166	0	1o6I8BglA6ylDMrIELygv1	0.913(
3	0.1650	1921	0.967	['Frank Parker']	0.275	210000	0.309	0	3ftBPsC5vPBKxYSee08FDH	0.000(
4	0.2530	1921	0.957	['Phil Regan']	0.418	166693	0.193	0	4d6HGyGT8e121BsdKmw9v6	0.000(

In [35]:

```
genre_data.head(5)
```

Out[35]:

	mode	genres	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tem
0	1	21st century classical	0.979333	0.162883	1.602977e+05	0.071317	0.606834	0.361600	-31.514333	0.040567	75.3365
1	1	432hz	0.494780	0.299333	1.048887e+06	0.450678	0.477762	0.131000	-16.854000	0.076817	120.2856
2	1	8-bit	0.762000	0.712000	1.151770e+05	0.818000	0.876000	0.126000	-9.180000	0.047000	133.4440
3	1	[]	0.651417	0.529093	2.328809e+05	0.419146	0.205309	0.218696	-12.288965	0.107872	112.8573
4	1	a cappella	0.676557	0.538961	1.906285e+05	0.316434	0.003003	0.172254	-12.479387	0.082851	112.1103

In [36]:

```
year_data.head(5)
```

Out[36]:

	mode	year	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo
0	1	1921	0.886896	0.418597	260537.166667	0.231815	0.344878	0.205710	-17.048667	0.073662	101.531493
1	1	1922	0.938592	0.482042	165469.746479	0.237815	0.434195	0.240720	-19.275282	0.116655	100.884521
2	1	1923	0.957247	0.577341	177942.362162	0.262406	0.371733	0.227462	-14.129211	0.093949	114.010730
3	1	1924	0.940200	0.549894	191046.707627	0.344347	0.581701	0.235219	-14.231343	0.092089	120.689572
4	1	1925	0.962607	0.573863	184986.924460	0.278594	0.418297	0.237668	-14.146414	0.111918	115.521921

In [37]:

```
artist_data.head(5)
```

Out[37]:

	mode	count	acousticness	artists	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechine
0	1	9	0.590111	"Cats" 1981 Original London Cast	0.467222	250318.555556	0.394003	0.011400	0.290833	-14.448000	0.210
1	1	26	0.862538	"Cats" 1983 Broadway Cast	0.441731	287280.000000	0.406808	0.081158	0.315215	-10.690000	0.176
2	1	7	0.856571	"Fiddler On The Roof" Motion Picture Chorus	0.348286	328920.000000	0.286571	0.024593	0.325786	-15.230714	0.118
3	1	27	0.884926	"Fiddler On The Roof" Motion Picture Orchestra	0.425074	262890.962963	0.245770	0.073587	0.275481	-15.639370	0.123
4	1	7	0.510714	"Joseph And The Amazing Technicolor Dreamcoat"...	0.467143	270436.142857	0.488286	0.009400	0.195000	-10.236714	0.098

In [38]:

```
datasets = [("data", data), ("genre_data", genre_data), ("year_data", year_data), ("artist_data", artist_data)
```

In [39]:

```
data['year'] = pd.to_datetime(data['year'], format='%Y')
data['release_date'] = pd.to_datetime(data['release_date'])
year_data['year'] = pd.to_datetime(year_data['year'], format='%Y')
```

In [41]:

```
for name, df in datasets:
    print(f"Info about the dataset: {name}")
    print("-"*30)
    print(df.info())
    print()
```

```
Info about the dataset: data
------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 170653 entries, 0 to 170652
Data columns (total 19 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   valence           170653 non-null  float64
 1   year              170653 non-null  datetime64[ns]
 2   acousticness      170653 non-null  float64
 3   artists           170653 non-null  object
 4   danceability      170653 non-null  float64
 5   duration_ms       170653 non-null  int64
 6   energy            170653 non-null  float64
 7   explicit          170653 non-null  int64
 8   id                170653 non-null  object
 9   instrumentalness  170653 non-null  float64
 10  key               170653 non-null  int64
 11  liveness          170653 non-null  float64
```

In [42]:

```
for name, df in datasets:
    print(f"Duplicates in the dataset: {name}")
    print("-"*30)
    print(df.duplicated(keep=False).sum())
    print()
```

```
Duplicates in the dataset: data
------------------------------
0

Duplicates in the dataset: genre_data
------------------------------
0

Duplicates in the dataset: year_data
------------------------------
0

Duplicates in the dataset: artist_data
------------------------------
0
```

In [43]:

```
for name, df in datasets:
    print(f"Unique Values in: {name}")
    print("-"*30)
    print(df.nunique())
    print()
```

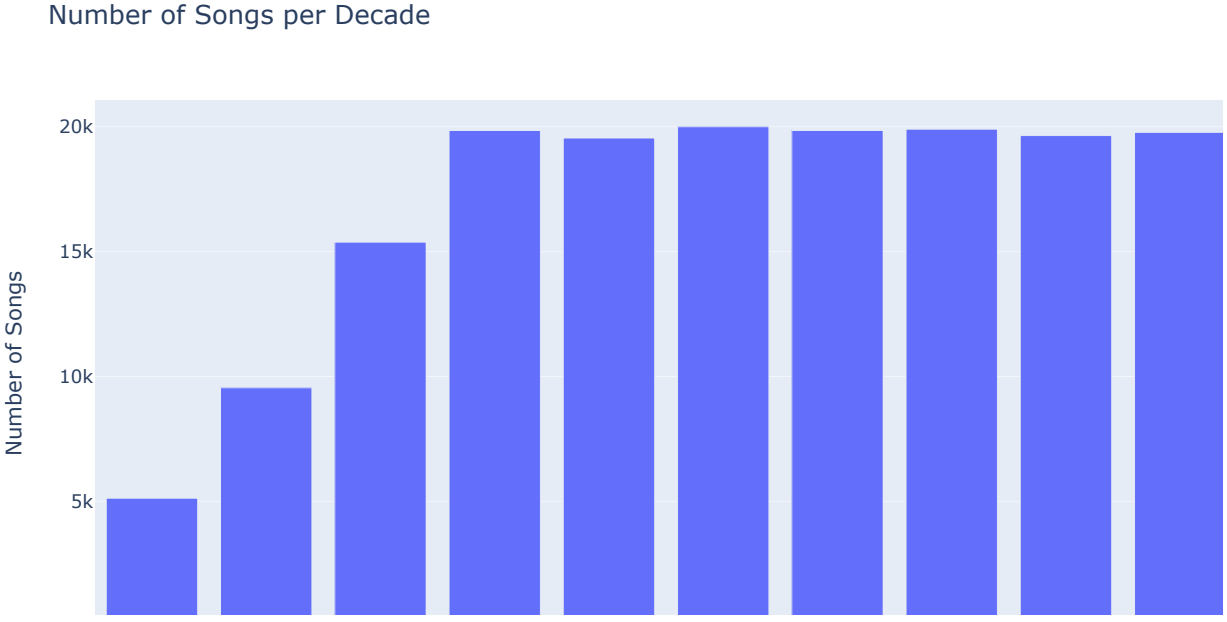
```
Unique Values in: data
------------------------------
valence               1733
year                   100
acousticness          4689
artists              34088
danceability          1240
duration_ms          51755
energy                2332
explicit                 2
id                  170653
instrumentalness      5401
key                     12
liveness              1740
loudness             25410
mode                     2
name                133638
popularity             100
release_date         10968
speechiness           1626
tempo                84694
dtype: int64

Unique Values in: genre_data
------------------------------
mode                    2
genres               2973
acousticness         2798
danceability         2725
duration_ms          2872
energy               2778
instrumentalness     2731
liveness             2709
loudness             2873
speechiness          2707
tempo                2872
valence              2745
popularity           2188
key                    12
dtype: int64

Unique Values in: year_data
------------------------------
mode                   1
year                 100
acousticness         100
danceability         100
duration_ms          100
energy               100
instrumentalness     100
liveness             100
loudness             100
speechiness          100
tempo                100
valence              100
popularity           100
key                    7
dtype: int64

Unique Values in: artist_data
------------------------------
mode                    2
count                 379
acousticness        14127
artists             28680
danceability        10650
duration_ms         23960
energy              12126
instrumentalness    15517
liveness            12156
loudness            21862
speechiness         10950
tempo               24801
valence             11882
popularity           4663
key                    12
dtype: int64
```

In [45]:

```
fig = px.line(year_data, x='year', y='popularity', title='Popularity Trends Over Years')
fig.show()
```

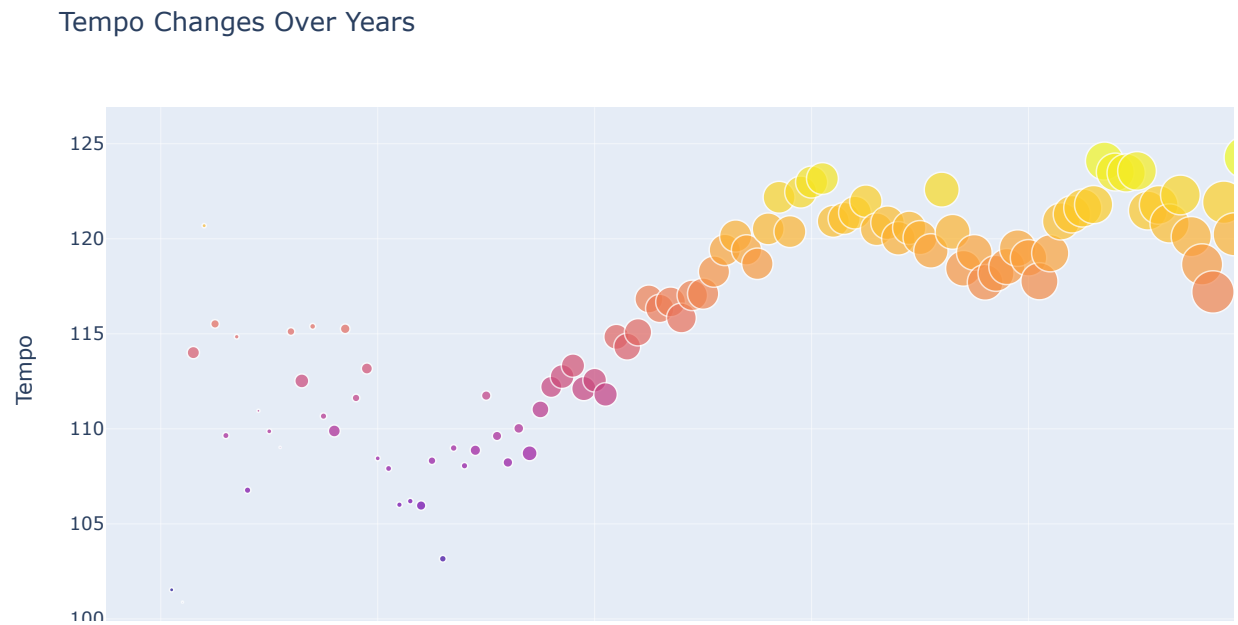
In [46]:

```
data['release_decade'] = (data['release_date'].dt.year // 10) * 10
decade_counts = data['release_decade'].value_counts().sort_index()
fig = px.bar(x=decade_counts.index, y=decade_counts.values, labels={'x': 'Decade', 'y': 'Number of Songs'},
             title='Number of Songs per Decade')
fig.update_layout(xaxis_type='category')
fig.show()
```



In [47]:

```
fig = px.scatter(year_data, x='year', y='tempo', color='tempo', size='popularity',
                 title='Tempo Changes Over Years', labels={'tempo': 'Tempo'})
fig.show()
```

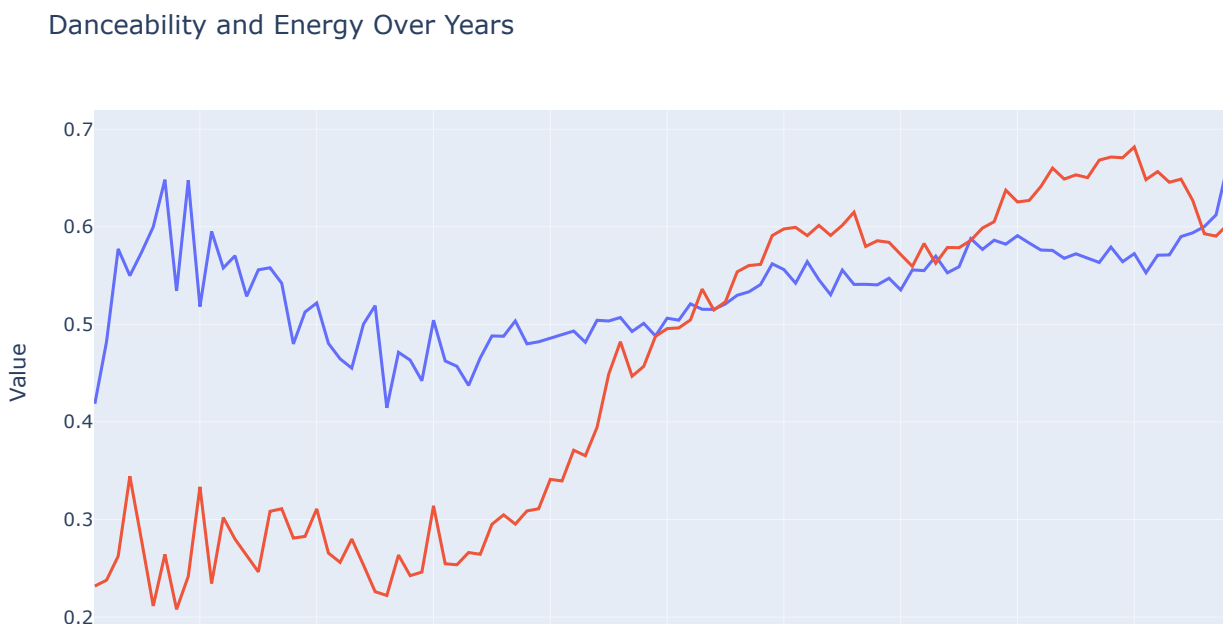


In [48]:

```
fig = px.line(year_data, x='year', y='danceability', title='Average Danceability Over Years')
fig.show()
```

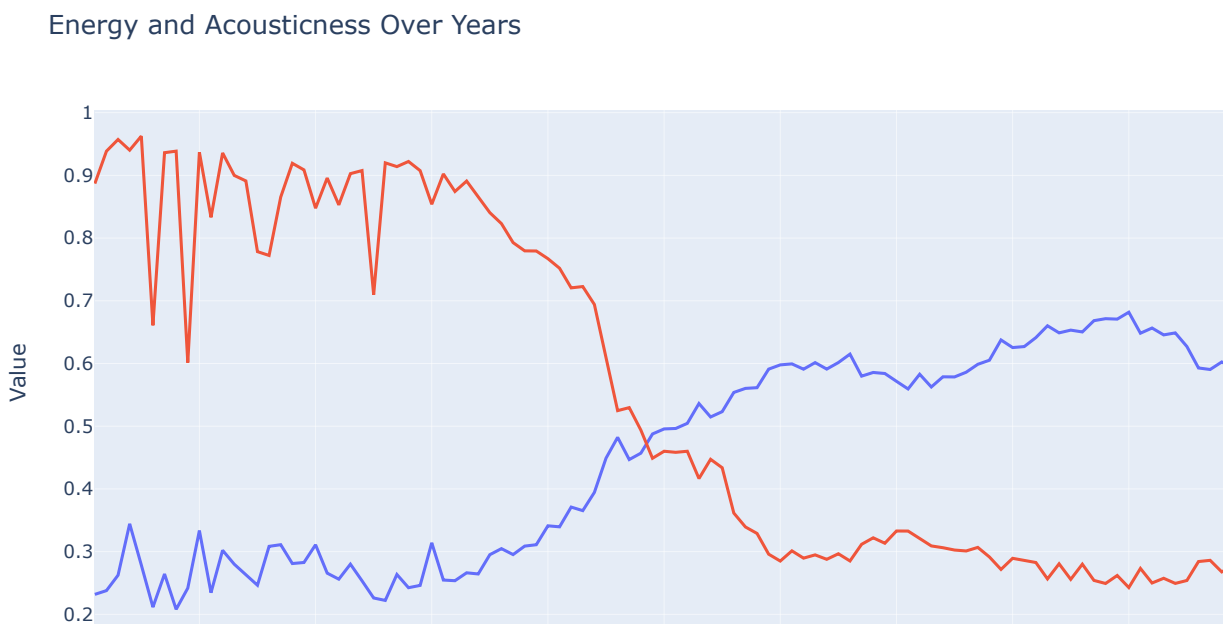
In [49]:

```
fig = go.Figure()

fig.add_trace(go.Scatter(x=year_data['year'], y=year_data['danceability'], mode='lines', name='Danceability'))
fig.add_trace(go.Scatter(x=year_data['year'], y=year_data['energy'], mode='lines', name='Energy'))

fig.update_layout(title='Danceability and Energy Over Years', xaxis_title='Year', yaxis_title='Value')
fig.show()
```



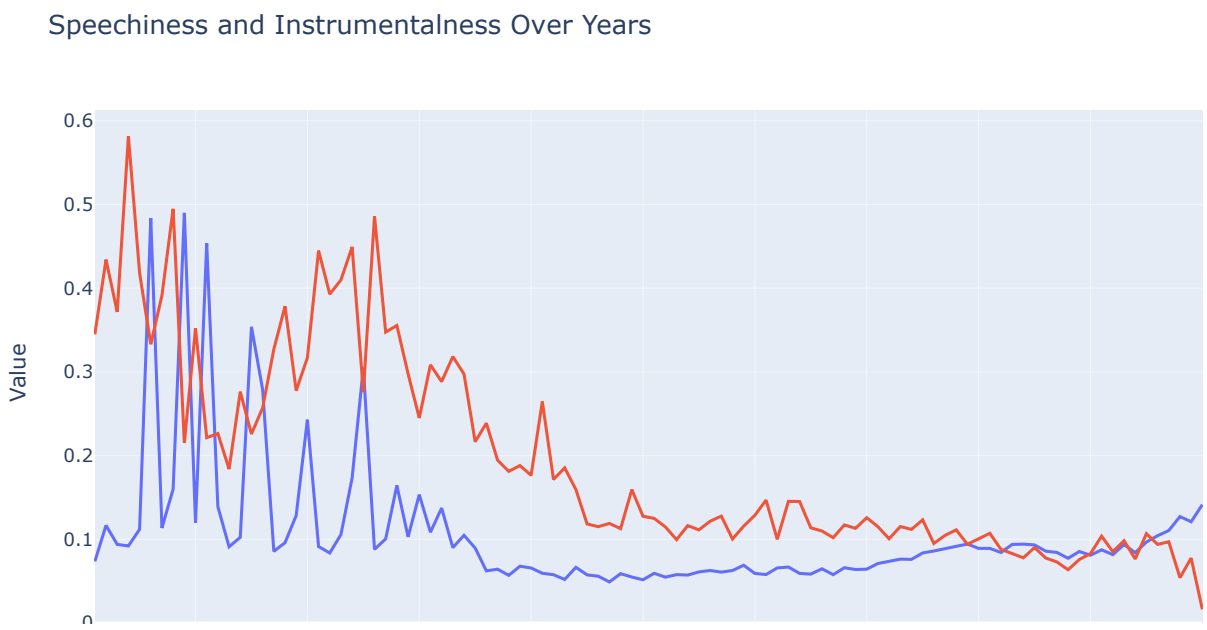
In [50]:

```
fig = go.Figure()

fig.add_trace(go.Scatter(x=year_data['year'], y=year_data['energy'], mode='lines', name='Energy'))
fig.add_trace(go.Scatter(x=year_data['year'], y=year_data['acousticness'], mode='lines', name='Acousticness'))

fig.update_layout(title='Energy and Acousticness Over Years', xaxis_title='Year', yaxis_title='Value')
fig.show()
```



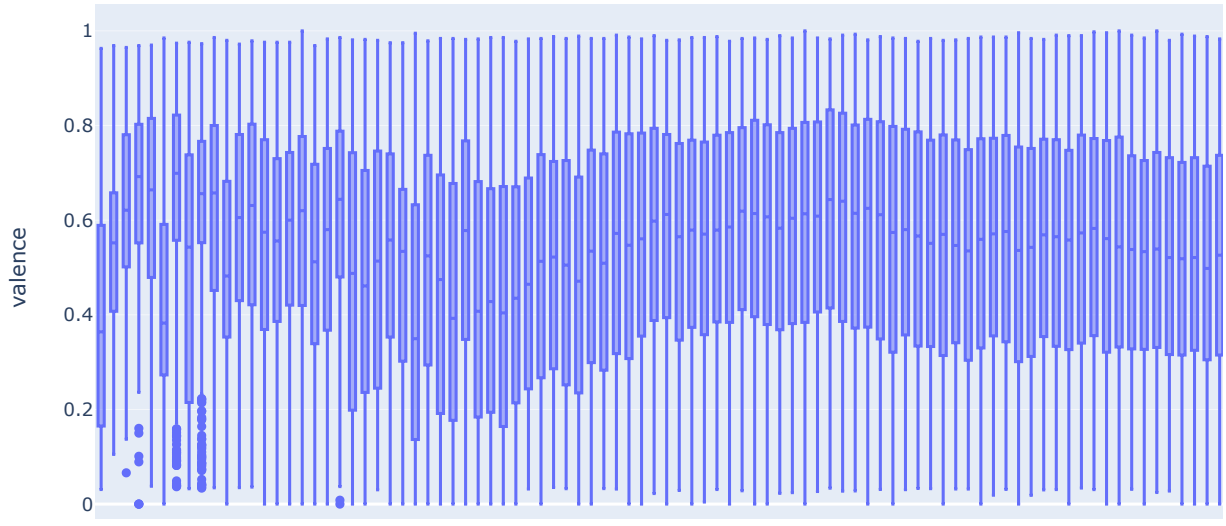
In [51]:

```
fig = go.Figure()

fig.add_trace(go.Scatter(x=year_data['year'], y=year_data['speechiness'], mode='lines', name='Speechiness'))
fig.add_trace(go.Scatter(x=year_data['year'], y=year_data['instrumentalness'], mode='lines', name='Instrumental

fig.update_layout(title='Speechiness and Instrumentalness Over Years', xaxis_title='Year', yaxis_title='Value'
fig.show()
```



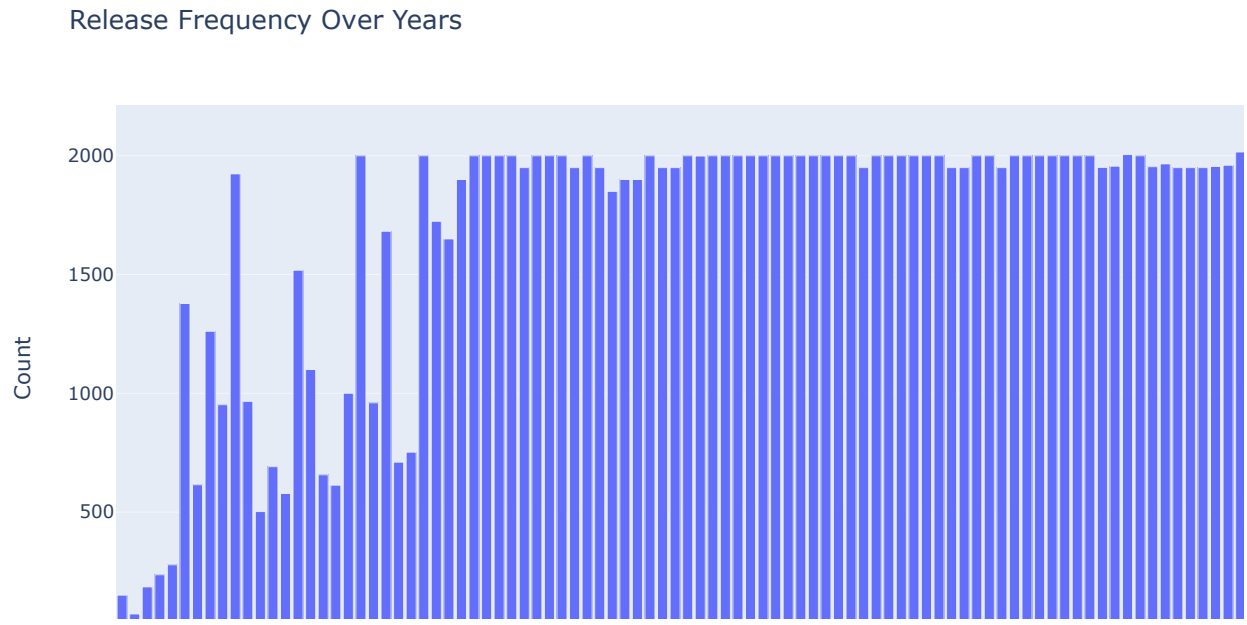
In [52]:

```
fig = px.box(data, x=data['release_date'].dt.year, y='valence', title='Valence Distribution by Release Year')
fig.show()
```



In [53]:

```
release_counts = data['release_date'].dt.year.value_counts().reset_index()
release_counts.columns = ['Year', 'Count']

fig = px.bar(release_counts, x='Year', y='Count', title='Release Frequency Over Years')
fig.show()
```

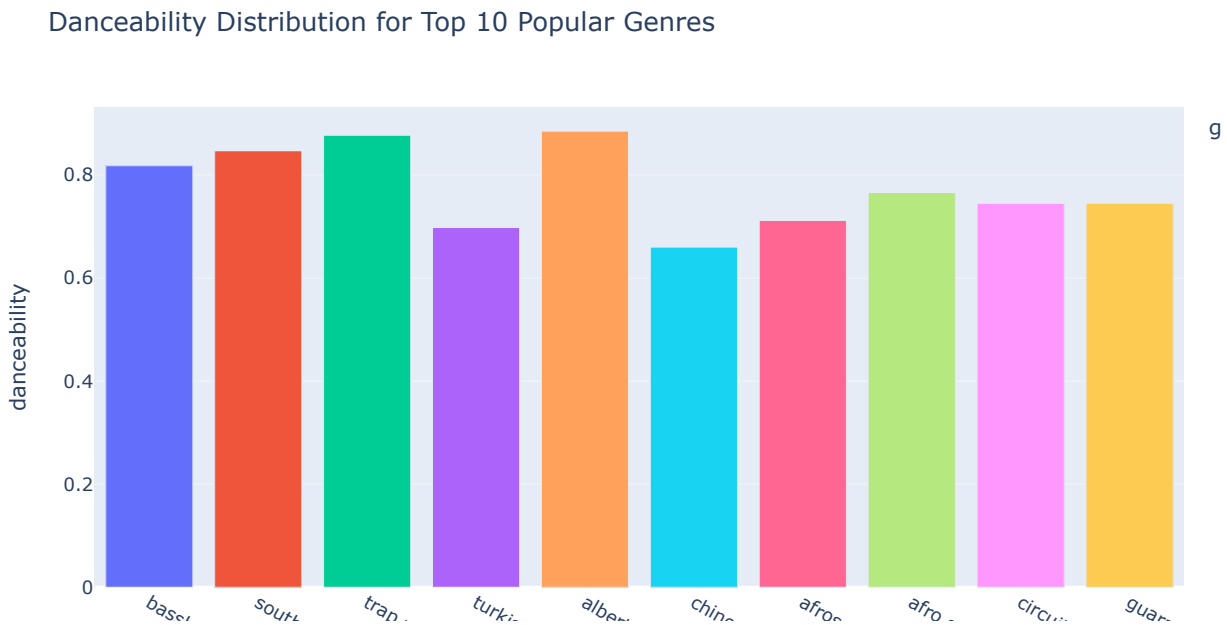


In [54]:

```
top_10_genre_data = genre_data.nlargest(10, 'popularity')

fig = px.bar(top_10_genre_data, x='popularity', y='genres', orientation='h',
             title='Top Genres by Popularity', color='genres')
fig.show()
```

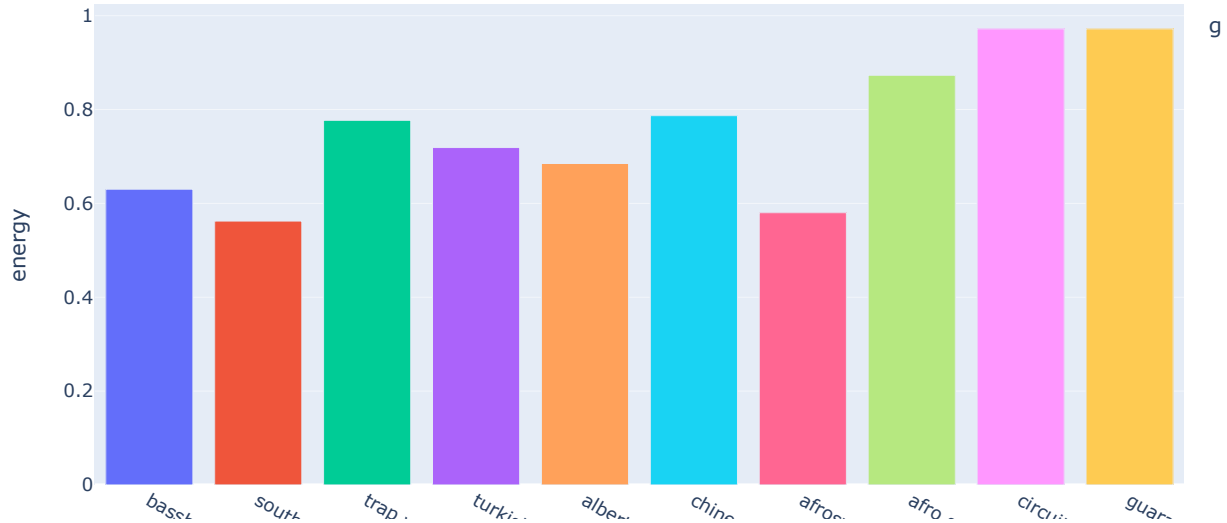
In [55]:

```
fig = px.bar(top_10_genre_data, x='genres', y='danceability', color='genres',
             title='Danceability Distribution for Top 10 Popular Genres')
fig.show()
```



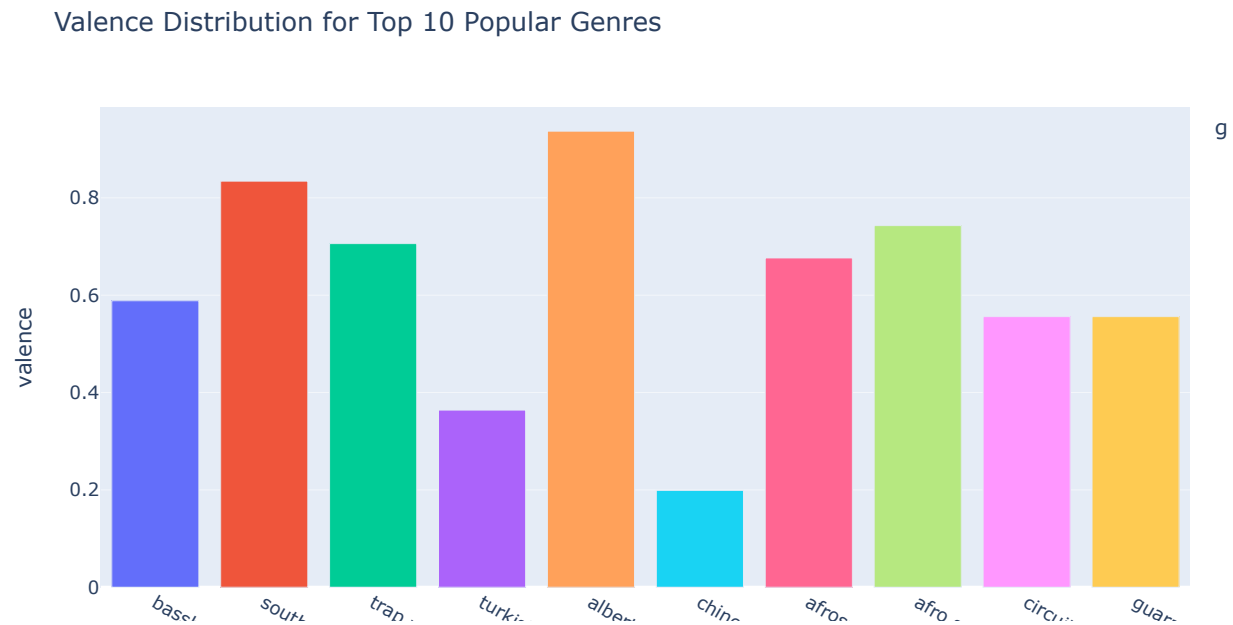
In [56]:

```
fig = px.bar(top_10_genre_data, x='genres', y='energy', color='genres',
             title='Energy Distribution for Top 10 Popular Genres')
fig.show()
```



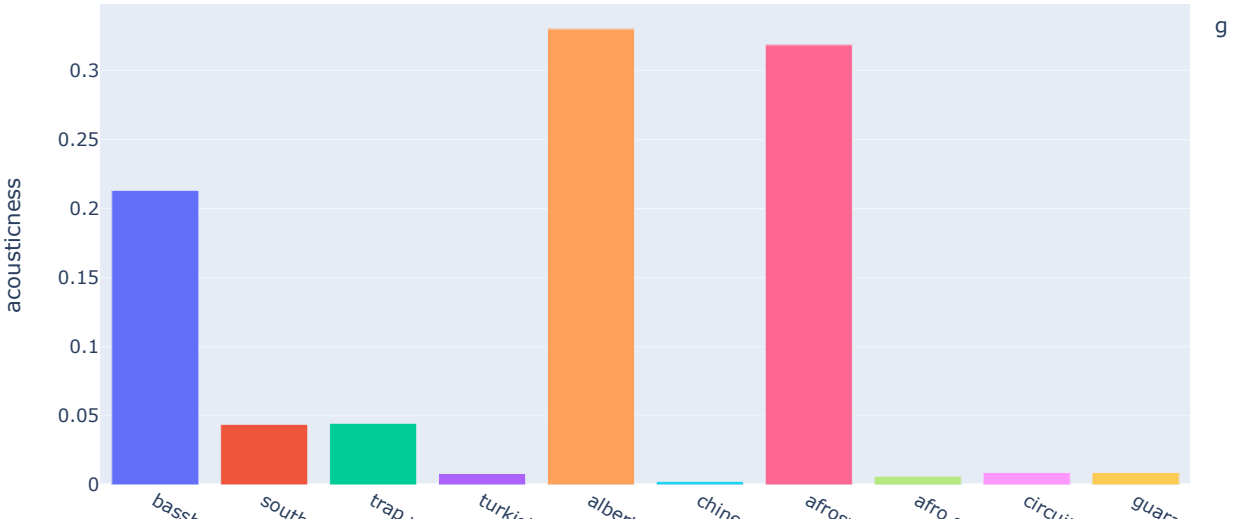
In [57]:

```
fig = px.bar(top_10_genre_data, x='genres', y='valence', color='genres',
             title='Valence Distribution for Top 10 Popular Genres')
fig.show()
```



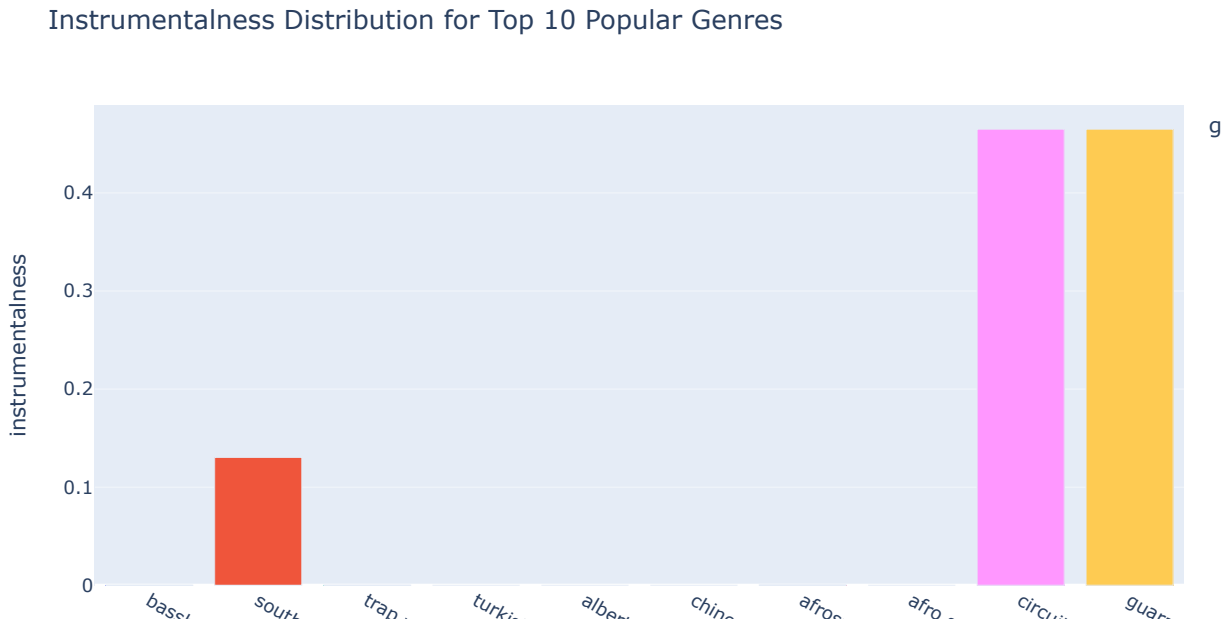
In [58]:

```
fig = px.bar(top_10_genre_data, x='genres', y='acousticness', color='genres',
             title='Acousticness Distribution for Top 10 Popular Genres')
fig.show()
```



In [59]:

```
fig = px.bar(top_10_genre_data, x='genres', y='instrumentalness', color='genres',
             title='Instrumentalness Distribution for Top 10 Popular Genres')
fig.show()
```

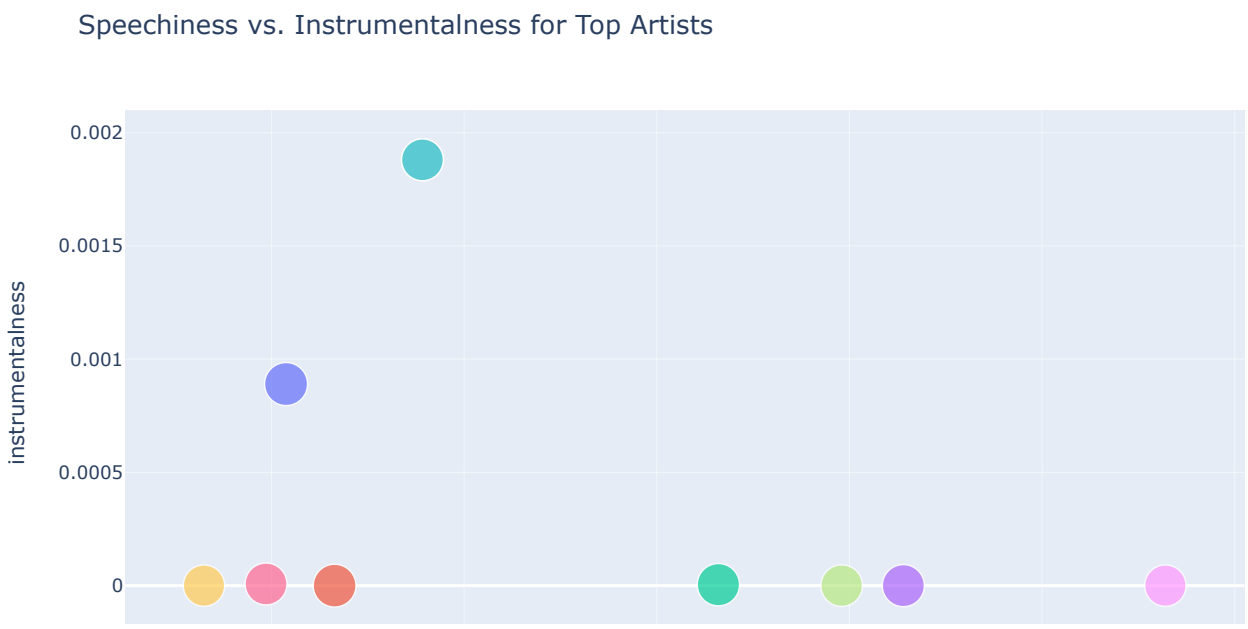


In [60]:

```
top_10_artist_data = artist_data.nlargest(10, 'popularity')

fig = px.bar(top_10_artist_data, x='popularity', y='artists', orientation='h', color='artists',
             title='Top Artists by Popularity')
fig.show()
```

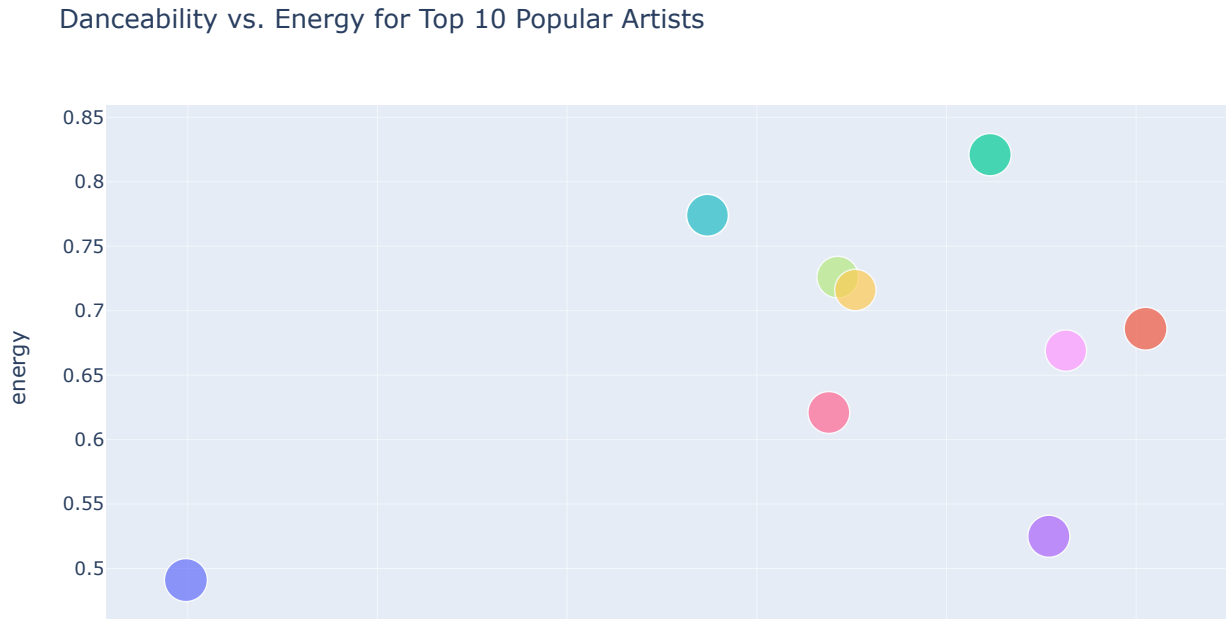
In [61]:

```
fig = px.scatter(top_10_artist_data, x='speechiness', y='instrumentalness', color='artists',
                 size='popularity', hover_name='artists',
                 title='Speechiness vs. Instrumentalness for Top Artists')
fig.show()
```



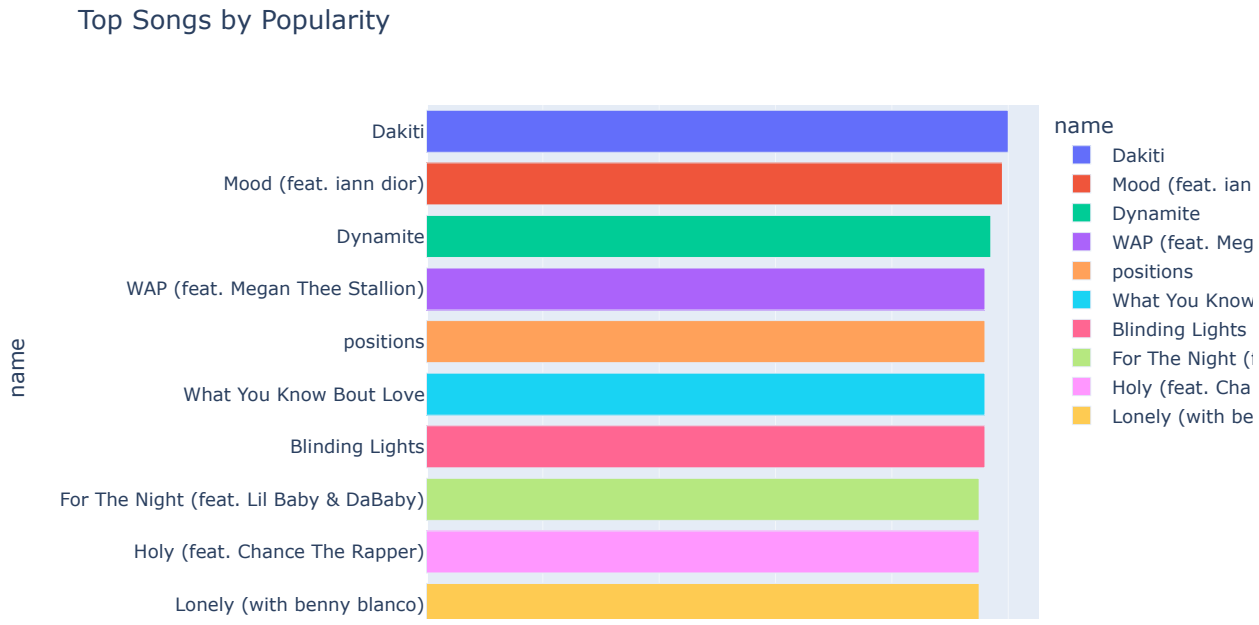
In [62]:

```
fig = px.scatter(top_10_artist_data, x='danceability', y='energy', color='artists',
                 size='popularity', hover_name='artists',
                 title='Danceability vs. Energy for Top 10 Popular Artists')
fig.show()
```



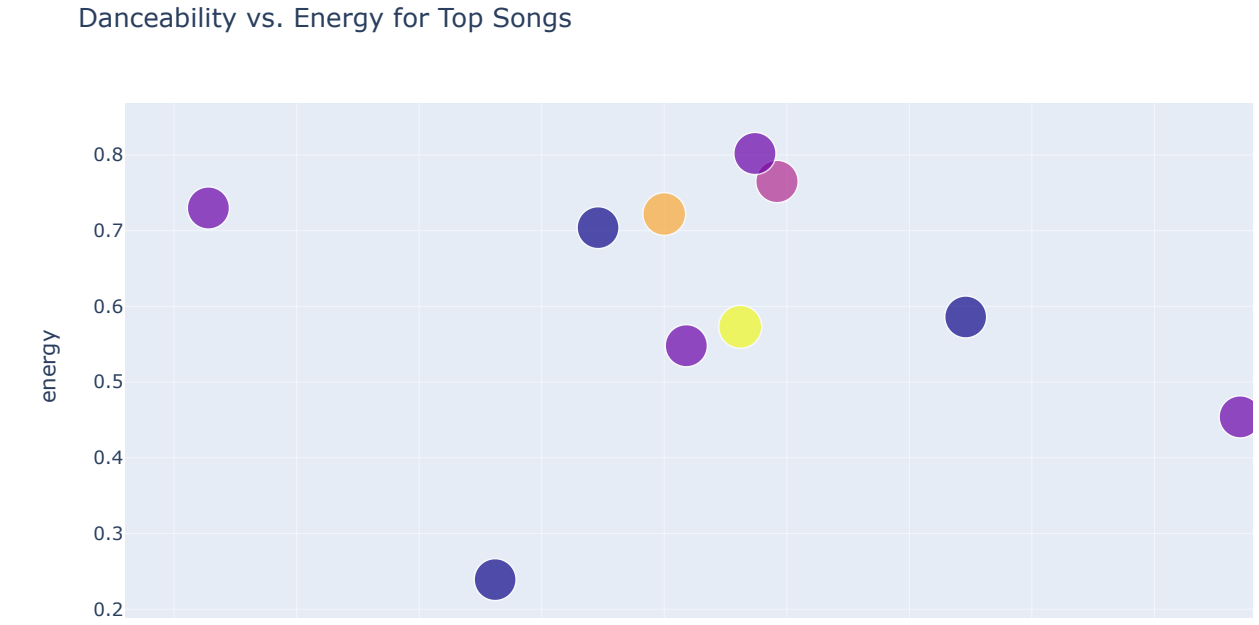
In [63]:

```
top_songs = data.nlargest(10, 'popularity')

fig = px.bar(top_songs, x='popularity', y='name', orientation='h',
             title='Top Songs by Popularity', color='name')
fig.show()
```



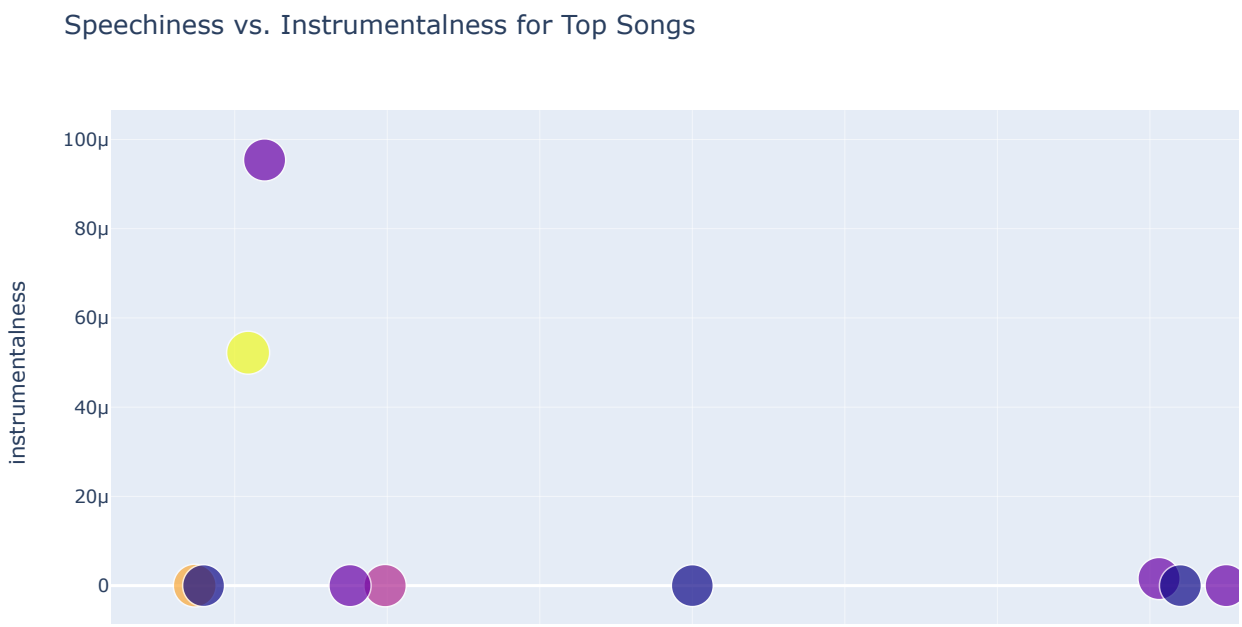
In [64]:

```
fig = px.scatter(top_songs, x='danceability', y='energy', color='popularity',
                 size='popularity', hover_name='name',
                 title='Danceability vs. Energy for Top Songs')
fig.show()
```



In [65]:

```
fig = px.scatter(top_songs, x='speechiness', y='instrumentalness', color='popularity',
                 size='popularity', hover_name='name',
                 title='Speechiness vs. Instrumentalness for Top Songs')
fig.show()
```



Recomendation System

In [71]:

```
number_cols = ['valence', 'year', 'acousticness', 'danceability', 'duration_ms', 'energy', 'explicit', 'year',
               'instrumentalness', 'key', 'liveness', 'loudness', 'mode', 'popularity', 'speechiness', 'tempo']
```

In [72]:

```
def get_song_data(name, data):
    try:
        return data[data['name'].str.lower() == name].iloc[0]
        return song_data
    except IndexError:
        return None
```

In [73]:

```
def get_mean_vector(song_list, data):
    song_vectors = []
    for song in song_list:
        song_data = get_song_data(song['name'], data)
        if song_data is None:
            print('Warning: {} does not exist in the dataset'.format(song['name']))
            return None
        song_vector = song_data[number_cols].values
        song_vectors.append(song_vector)
    song_matrix = np.array(list(song_vectors))
    return np.mean(song_matrix, axis=0)
```

In [74]:

```
def flatten_dict_list(dict_list):
    flattened_dict = defaultdict()
    for key in dict_list[0].keys():
        flattened_dict[key] = []
    for dictionary in dict_list:
        for key, value in dictionary.items():
            flattened_dict[key].append(value)
    return flattened_dict
```

In [75]:

```
min_max_scaler = MinMaxScaler()
normalized_data = min_max_scaler.fit_transform(data[number_cols])

standard_scaler = StandardScaler()
scaled_normalized_data = standard_scaler.fit_transform(normalized_data)
```

In [76]:

```
f recommend_songs(seed_songs, data, n_recommendations=10):
  metadata_cols = ['name', 'artists', 'year']
  song_center = get_mean_vector(seed_songs, data)

  if song_center is None:
      return []

  normalized_song_center = min_max_scaler.transform([song_center])

  scaled_normalized_song_center = standard_scaler.transform(normalized_song_center)

  distances = cdist(scaled_normalized_song_center, scaled_normalized_data, 'euclidean')
  index = np.argsort(distances)[0]

  rec_songs = []
  for i in index:
      song_name = data.iloc[i]['name']
      if song_name not in [song['name'] for song in seed_songs] and song_name not in [song['name'] for song in
          rec_songs.append(data.iloc[i])
          if len(rec_songs) == n_recommendations:
              break

  return pd.DataFrame(rec_songs)[metadata_cols].to_dict(orient='records')
```

In [77]:

```
seed_songs = [
    {'name': 'Paranoid'},
    {'name': 'Blinding Lights'},
]
seed_songs = [{'name': name['name'].lower()} for name in seed_songs]

n_recommendations = 15

recommended_songs = recommend_songs(seed_songs, data, n_recommendations)

recommended_df = pd.DataFrame(recommended_songs)

for idx, song in enumerate(recommended_songs, start=1):
    print(f"{idx}. {song['name']} by {song['artists']} ({song['year']})")
```

```
1. Infinity by ['One Direction'] (2015)
2. Secrets by ['OneRepublic'] (2009)
3. In My Blood by ['Shawn Mendes'] (2018)
4. Head Above Water by ['Avril Lavigne'] (2019)
5. Green Light by ['Lorde'] (2017)
6. My Wish by ['Rascal Flatts'] (2006)
7. Magic Shop by ['BTS'] (2018)
8. Good Things Fall Apart (with Jon Bellion) by ['ILLENIUM', 'Jon Bellion'] (2019)
9. Inside Out (feat. Griff) by ['Zedd', 'Griff'] (2020)
10. A.M. by ['One Direction'] (2015)
11. Love You Goodbye by ['One Direction'] (2015)
12. Story of My Life by ['One Direction'] (2013)
13. Perfect by ['Simple Plan'] (2018)
14. arms by ['Christina Perri'] (2011)
15. Breezeblocks by ['alt-J'] (2012)
```

In [78]:

```
recommended_df['text'] = recommended_df.apply(lambda row: f"{row.name + 1}. {row['name']} by {row['artists']}
fig = px.bar(recommended_df, y='name', x=range(n_recommendations, 0, -1), title='Recommended Songs', orientatio
fig.update_layout(xaxis_title='Recommendation Rank', yaxis_title='Songs', showlegend=False, uniformtext_minsize
fig.update_traces(width=1)
fig.show()
```

In [ ]:

```
```