In [1]:
```
import pandas as pd
import numpy as np

import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objects as go


from pyclustertend import hopkins
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA

from collections import Counter
```

In [2]:
```
SEED = 2021
DATA_PATH = '../data/data.csv'
```

Data exploration

In [3]:
```
df = pd.read_csv(DATA_PATH, index_col='Unnamed: 0')
df = df.drop('Unnamed: 1', axis=1)
df.head()
```

Out[3]:

	name	artist	popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms
All Out 50s	Don't Be Cruel	Elvis Presley	59	0.697	0.550	2	-11.496	1	0.1790	0.856	0.000034	0.0907	0.844	84.802	122893
All Out 50s	I've Got You Under My Skin - Remastered 1998	Frank Sinatra	66	0.585	0.247	1	-12.612	1	0.0400	0.452	0.000009	0.1070	0.591	127.150	223760
All Out 50s	Smoke Gets In Your Eyes	The Platters	0	0.290	0.227	3	-13.060	1	0.0311	0.944	0.000079	0.6170	0.224	114.278	157293
All Out 50s	What'd I Say, Pt. 1 & 2	Ray Charles	62	0.540	0.681	4	-5.440	1	0.0508	0.808	0.000000	0.1620	0.794	88.385	307053
All Out 50s	Dream A Little Dream Of Me	Ella Fitzgerald	0	0.455	0.167	0	-13.613	1	0.0739	0.918	0.000000	0.1730	0.404	76.118	185067

In [4]:
```
df.describe()
```

Out[4]:

	popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms
count	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000
mean	53.521429	0.609131	0.626458	4.864286	-8.230987	0.771429	0.055982	0.316753	0.012008	0.170348	0.646758	120.096399	217595.778571
std	27.710249	0.143148	0.201856	3.539701	3.489530	0.420213	0.052818	0.285192	0.068739	0.139569	0.230293	27.711337	54122.588371
min	0.000000	0.138000	0.021000	0.000000	-24.385000	0.000000	0.022800	0.000065	0.000000	0.023400	0.065000	62.658000	109960.000000
25%	49.000000	0.526000	0.486250	2.000000	-10.483250	1.000000	0.032100	0.062175	0.000000	0.084750	0.484750	100.003000	178906.500000
50%	65.000000	0.623000	0.651500	5.000000	-7.514000	1.000000	0.038700	0.225000	0.000002	0.119000	0.682000	118.366000	215226.500000
75%	72.000000	0.707000	0.786000	8.000000	-5.638500	1.000000	0.056000	0.544000	0.000149	0.203000	0.831000	132.950500	247076.750000
max	85.000000	0.967000	0.993000	11.000000	1.085000	1.000000	0.463000	0.982000	0.954000	0.882000	0.985000	207.356000	522307.000000

popularity

The popularity of the track. The value will be between 0 and 100, with 100 being the most popular. The popularity of a track is a value between 0 and 100, with 100 being the most popular. The popularity is calculated by algorithm and is based, in the most part, on the total number of plays the track has had and how recent those plays are.

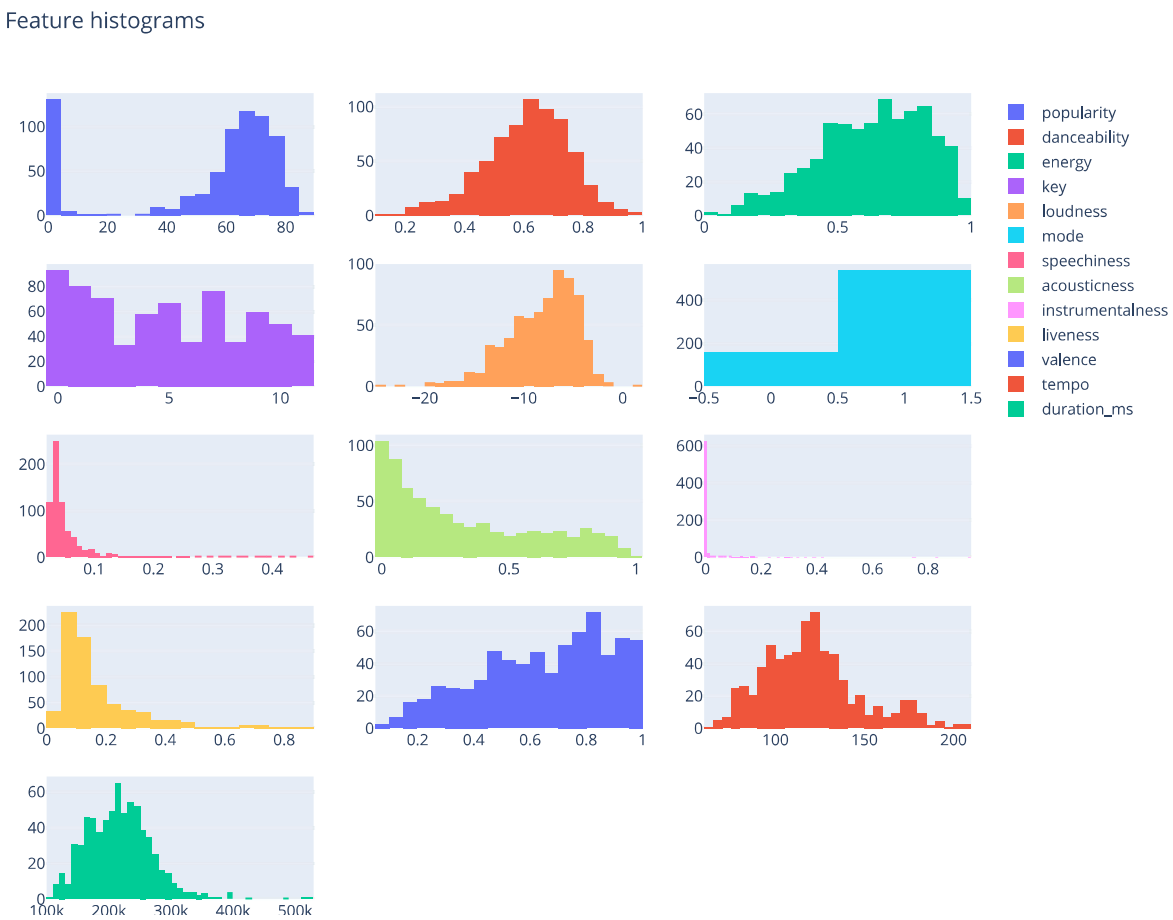
Generally speaking, songs that are being played a lot now will have a higher popularity than songs that were played a lot in the past. Duplicate tracks (e.g. the same track from a single and an album) are rated independently. Artist and album popularity is derived mathematically from track popularity. Note that the popularity value may lag actual popularity by a few days: the value is not updated in real time.

In [5]:
```
len(df[df['popularity'] == 0])
```

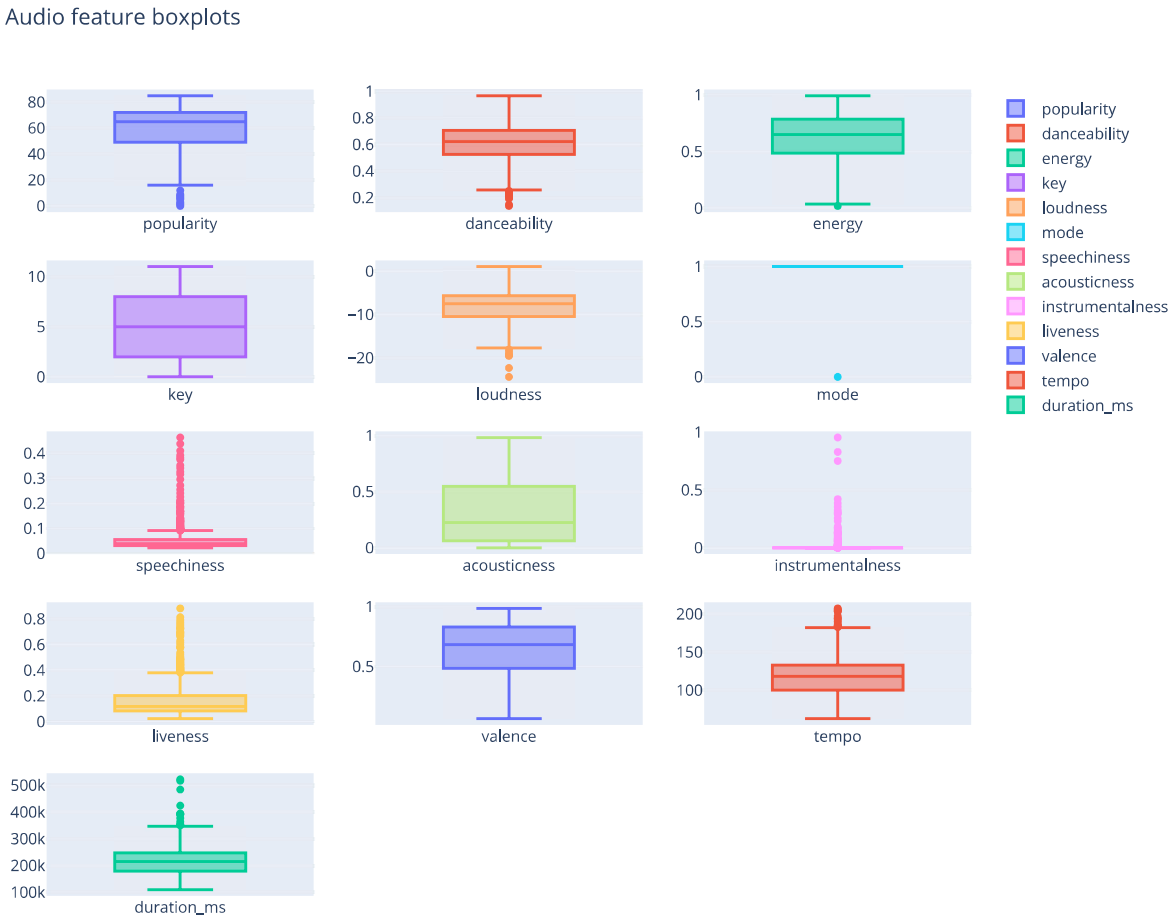
Out[5]: 113

In [6]:
```
feature_cols = df.drop(['name', 'artist'], axis=1).columns
```

In [7]:
```
fig = make_subplots(rows=5, cols=3)

for i, col in enumerate(feature_cols):
    row_index = i // 3
    col_index = i % 3
    fig.add_trace(go.Histogram(x=df[col], name=col), row=row_index + 1, col=col_index + 1)
fig.update_layout(title_text="Feature histograms", width=960, height=800, )
fig.show()
```

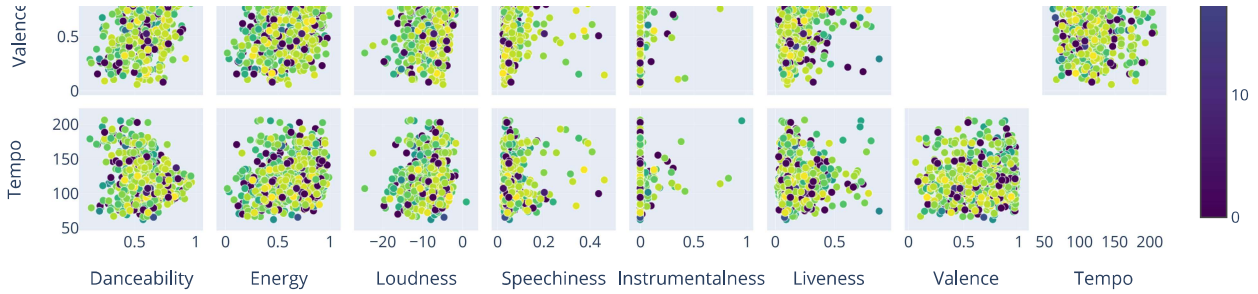


In [8]:
```
fig = make_subplots(rows=5, cols=3)

for i, col in enumerate(feature_cols):
    row_index = i // 3
    col_index = i % 3
    fig.add_trace(go.Box(y=df[col], name=col), row=row_index + 1, col=col_index + 1)
fig.update_layout(title_text="Audio feature boxplots", width=960, height=800, )
fig.show()
```



In [9]:
```
fig = go.Figure(data=go.Splom(
```

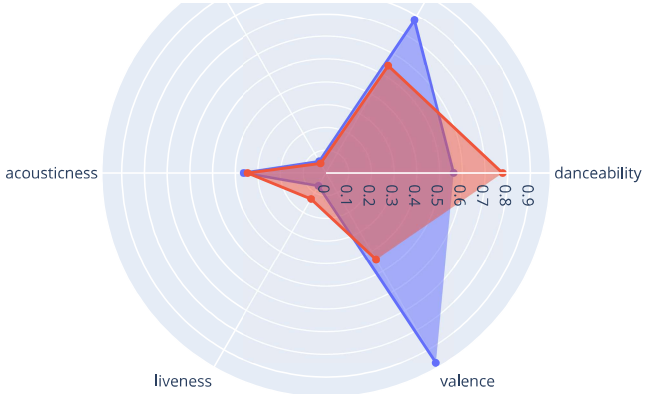
```
                    dimensions=[dict(label='Danceability', values=df['danceability']),
                                dict(label='Energy', values=df['energy']),
                                dict(label='Loudness', values=df['loudness']),
                                dict(label='Speechiness', values=df['speechiness']),
                                dict(label='Instrumentalness', values=df['instrumentalness']),
                                dict(label='Liveness', values=df['liveness']),
                                dict(label='Valence', values=df['valence']),
                                dict(label='Tempo', values=df['tempo'])],
                    marker=dict(color=df['popularity'],
                                colorscale='Viridis',
                                line_color='white',
                                line_width=0.5,
                                colorbar=dict(thickness=20)),
                    diagonal=dict(visible=False)))

fig.update_layout(title="Audio features scatterplot matrix colored by popularity",
                  dragmode='select',
                  width=1000,
                  height=1000)

fig.show()
```



Let's compare 2 random samples from the dataset

In [10]:
```
cols = ['danceability', 'energy',  'speechiness', 'acousticness', 'liveness', 'valence']

def get_sample(seed=2021):
    sample = df[cols + ['name']].sample(random_state=seed)
    name = sample['name']
    sample = sample.drop('name', axis=1).T
    sample = sample.values.squeeze(1)
    #sample = (sample - np.mean(sample)) / np.std(sample)
    return sample, name.values[0]
```

In [11]:
```
sample_1, name_1 = get_sample(42)
sample_2, name_2 = get_sample(2021)

fig = go.Figure()

fig.add_trace(go.Scatterpolar(
      r=sample_1,
      theta=cols,
      fill='toself',
      name=name_1
    ))

fig.add_trace(go.Scatterpolar(
      r=sample_2,
      theta=cols,
      fill='toself',
      name=name_2
))

fig.update_layout(
  polar=dict(
    radialaxis=dict(
      visible=True
    ),
  )
)

fig.show()
```



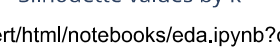
Clustering

In [12]:
```
def check_clustering_tendency(data):
    return hopkins(data, data.shape[0])


def silhouette(data, max_k):
    silhouette_values = []

    for k in range(2, max_k):
        kmeans = KMeans(n_clusters=k, random_state=42).fit(data)
        silhouette_value = silhouette_score(data, kmeans.labels_)
        silhouette_values.append(silhouette_value)
    return silhouette_values


def get_labels(data, k):
    kmeans = KMeans(n_clusters=k, random_state=SEED).fit(data)
    return kmeans.labels_
```

In [13]:
```
check_clustering_tendency(df[feature_cols])
```

Out[13]: 0.08410674950914157

In [14]:
```
silhouette_results = silhouette(df[feature_cols], 10)

fig = px.line(silhouette_results, markers=True, x=[i for i in range(2, 10)], y=silhouette_results, title='Silhouette values by k')
fig.add_annotation(x=3, y=0.5490264,
            text="Maximum silhouette value indicates that 3 is best k",
            showarrow=True,
            arrowhead=5)
fig.update_xaxes(title_text='k')
fig.update_yaxes(title_text='silhouette')

fig.show()
```



Choosing the optimal value of k

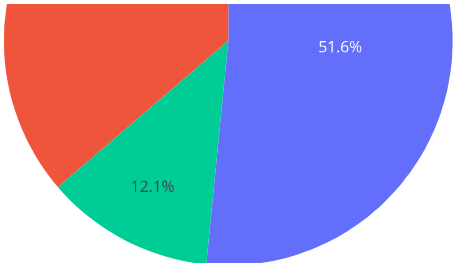
In [15]:
```
max_metric_value = max(silhouette_results)
silhouette_results.index(max_metric_value) + 2
```

Out[15]: 3

In [16]:
```
classes = get_labels(df[feature_cols], k=3)
```

Visualization

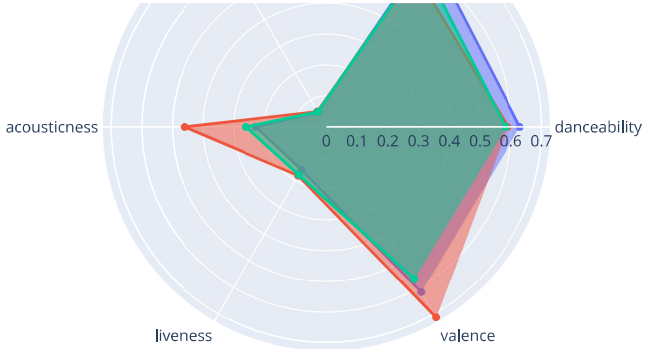
In [17]:
```
labels = [0, 1, 2]
values = list(Counter(classes).values())

fig = go.Figure(data=[go.Pie(labels=labels, values=values)])

fig.update_layout(title='Class distribution in percent')
fig.show()
```



Polar plot for mean values of audio features by classes

In [18]:
```
df['label'] = classes
```

In [19]:
```
fig = go.Figure()

fig.add_trace(go.Scatterpolar(
      r=df[df['label'] == 0][cols].mean(),
      theta=cols,
      fill='toself',
      name='class 0'
    ))

fig.add_trace(go.Scatterpolar(
      r=df[df['label'] == 1][cols].mean(),
      theta=cols,
      fill='toself',
      name='class 1'
))

fig.add_trace(go.Scatterpolar(
      r=df[df['label'] == 2][cols].mean(),
      theta=cols,
      fill='toself',
      name='class 2'
))

fig.update_layout(
  polar=dict(
    radialaxis=dict(
      visible=True
    ),
  )
)

fig.show()
```

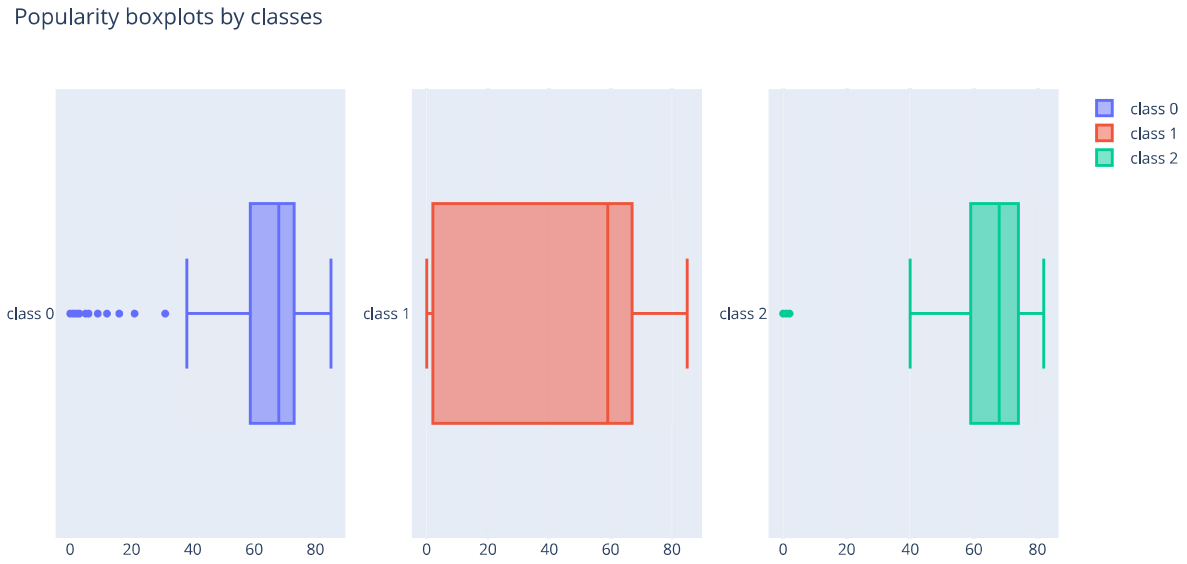


Classes interpretation:

- 0 - songs with high energy, danceability, medium valence, and low acousticness — Dancing
- 1 - songs with medium energy/danceability, high valence, and acousticness — Happy acoustic
- 2 - songs with low valence/danceability, medium energy, and acousticness — Soft acoustic

In [20]:

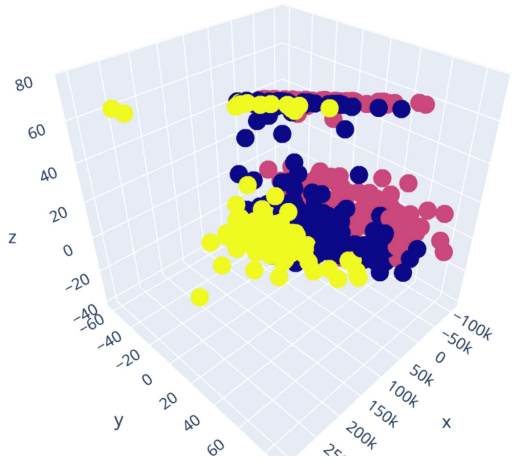
```
fig = make_subplots(rows=1, cols=3)

for i in range(0, 4):
    col_index = i % 3
    fig.add_trace(go.Box(x=df[df['label'] == i]['popularity'], name=f'class {i}'), row=1, col=col_index + 1)
fig.update_layout(title_text="Popularity boxplots by classes", width=960)
fig.show()
```



In [21]:
```
pca = PCA(n_components=3)
pca.fit(df[feature_cols])
result = pca.transform(df[feature_cols])
```

After PCA transformation points are well separable in 3-dimensional space

In [23]:
```
fig = px.scatter_3d(result, x=result[:, 0], y=result[:, 1], z=result[:, 2],
                    color=df['label'])
fig.show()
```



In [24]:
```
n_components = 3

most_important = [np.abs(pca.components_[i]).argmax() for i in range(n_components)]
feature_names = df[feature_cols].columns
most_important_names = [feature_names[most_important[i]] for i in range(n_components)]
most_important_names
```

Out[24]:
```
['duration_ms', 'tempo', 'popularity']
```