1. Introduction

In this notebook, I visualized music tracks data from the Kaggle Spotify datasets, which contain more than 170 thousand tacks from 30,000 artists over the last 100 years. This notebook has essential data exploration analysis and also tries to find out how music has evolved across the years.

As far as I am concerned, the analysis has some limitations:

- The scope of Spotify's tracks collection has more records in recent years than in the early 20s-50s.
- The popularity score only reflects Spotify users' preference (likely to be relatively young user groups.)
- The popularity is scored in the current time. It only shows how people nowadays like old music, not necessarily represent how the classic track's popularity back in the time (a good feature for this will be the record sales for the album).

The datasets has the following features and definitions:

- **duration_ms:**
  The length of the track in milliseconds (ms)
- **artists:**
  The list of artists credited for the production of the track
- **year:**
  The release year of track
- **key:**
  The primary key of the track encoded as integers between 0 and 11
- **mode:**
  The binary value representing whether the track starts with a major (1) chord progression or not (0)
- **release_date:**
  The date of release of the track in yyyy-mm-dd, yyyy-mm, or even yyyy format
- **acousticness:**
  The relative metric of the track being acoustic
- **danceability:**
  The relative measurement of the track being danceable
- **energy:**
  The energy of the track
- **instrumentalness:**
  The relative ratio of the track being instrumental
- **liveness:**
  The relative duration of the track sounding as a live performance
- **loudness:**
  Relative loudness of the track in the typical range [-60, 0] in decibel (dB)

- **speechiness:**
  The relative length of the track containing any kind of human voice
- **valence:**
  The positiveness of the track
- **tempo:**
  The tempo of the track in Beat Per Minute (BPM)
- **name:**
  The title of the track
- **popularity:**
  The popularity of the song lately, default country = US

Import libraries

In [45]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from math import pi
import warnings
warnings.filterwarnings('ignore')
from IPython.display import YouTubeVideo
from IPython.display import Audio
from sklearn.preprocessing import MinMaxScaler
from matplotlib.colors import ListedColormap
import matplotlib.animation as ani
from IPython.display import HTML
from matplotlib import animation, rc
%matplotlib inline
from matplotlib.animation import FuncAnimation, PillowWriter
import matplotlib
matplotlib.rcParams['animation.embed_limit'] = 2**128
import matplotlib.patches as mpatches
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

Import datasets

Kaggle has provided 4 datasets:

- data: the main dataset that based on each track
- data_by_artist: The average of features group by each artist
- data_by_genres: The average of features group by each genres, this is the only dataset that has genres information
- data_by_year: The average of features group by each year

In [2]:
```
data = pd.read_csv('data.csv')
data.shape
```

Out[2]:
```
(174389, 19)
```

In [3]:
```
data.head()
```

Out[3]:

	acousticness	artists	danceability	duration_ms	energy	explicit	id	inst
0	0.991000	['Mamie Smith']	0.598	168333	0.224	0	0cS0A1fUEUd1EW3FcF8AEI	
1	0.643000	["Screamin' Jay Hawkins"]	0.852	150200	0.517	0	0hbkKFIJm7Z05H8ZI9w30f	
2	0.993000	['Mamie Smith']	0.647	163827	0.186	0	11m7laMUgmOKqI3oYzuhne	
3	0.000173	['Oscar Velazquez']	0.730	422087	0.798	0	19Lc5SfJJ5O1oaxY0fpwfh	
4	0.295000	['Mixe']	0.704	165224	0.707	1	2hJjbsLCytGsnAHfdsLejp	

Clean up the format of artists column:

In [4]:
```
data.artists = data['artists'].str.replace('[','')
data.artists = data['artists'].str.replace(']','')
data.artists = data['artists'].str.replace("'",'')
data.artists = data['artists'].str.replace('"','')
data.artists
```

Out[4]:
```
0                       Mamie Smith
1                Screamin Jay Hawkins
2                       Mamie Smith
3                    Oscar Velazquez
4                              Mixe
                     ...
174384    DJ Combo, Sander-7, Tony T
174385                  Alessia Cara
174386                     Roger Fly
174387                  Taylor Swift
174388                     Roger Fly
Name: artists, Length: 174389, dtype: object
```

In [5]:
```
artist_df = pd.read_csv('data_by_artist.csv')
artist_df.shape
```

Out[5]:
```
(32539, 15)
```

In [6]:
```
artist_df.head()
```

Out[6]:

	artists	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudr
0	"Cats" 1981 Original London Cast	0.598500	0.470100	267072.000000	0.376203	0.010261	0.283050	-14.434

	artists	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudr
1	"Cats" 1983 Broadway Cast	0.862538	0.441731	287280.000000	0.406808	0.081158	0.315215	-10.690
2	"Fiddler On The Roof" Motion Picture Chorus	0.856571	0.348286	328920.000000	0.286571	0.024593	0.325786	-15.230
3	"Fiddler On The Roof" Motion Picture Orchestra	0.884926	0.425074	262890.962963	0.245770	0.073587	0.275481	-15.639
4	"Joseph And The Amazing Technicolor Dreamcoat"...	0.510714	0.467143	270436.142857	0.488286	0.009400	0.195000	-10.236

In [7]:
```
genres_df = pd.read_csv('data_by_genres.csv')
genres_df.shape
```

Out[7]: (3232, 14)

In [8]:
```
genres_df.head()
```

Out[8]:

	genres	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness
0	21st century classical	0.754600	0.284100	3.525932e+05	0.159580	0.484374	0.168580	-22.153400
1	432hz	0.485515	0.312000	1.047430e+06	0.391678	0.477250	0.265940	-18.131267
2	8-bit	0.028900	0.673000	1.334540e+05	0.950000	0.630000	0.069000	-7.899000
3	[]	0.535793	0.546937	2.495312e+05	0.485430	0.278442	0.220970	-11.624754
4	a cappella	0.694276	0.516172	2.018391e+05	0.330533	0.036080	0.222983	-12.656547

In [9]:
```
year_df = pd.read_csv('data_by_year.csv')
year_df.shape
```

Out[9]: (102, 14)

In [10]:
```
year_df.head()
```

Out[10]:

	year	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	sp
0	1920	0.631242	0.515750	238092.997135	0.418700	0.354219	0.216049	-12.654020	
1	1921	0.862105	0.432171	257891.762821	0.241136	0.337158	0.205219	-16.811660	

	year	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	sp
2	1922	0.828934	0.575620	140135.140496	0.226173	0.254776	0.256662	-20.840083	
3	1923	0.957247	0.577341	177942.362162	0.262406	0.371733	0.227462	-14.129211	
4	1924	0.940200	0.549894	191046.707627	0.344347	0.581701	0.235219	-14.231343	

In [11]:
```
g_df = pd.read_csv('data_w_genres.csv')
g_df.head()
```

Out[11]:

	artists	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudr
0	"Cats" 1981 Original London Cast	0.598500	0.470100	267072.000000	0.376203	0.010261	0.283050	-14.434
1	"Cats" 1983 Broadway Cast	0.862538	0.441731	287280.000000	0.406808	0.081158	0.315215	-10.690
2	"Fiddler On The Roof" Motion Picture Chorus	0.856571	0.348286	328920.000000	0.286571	0.024593	0.325786	-15.230
3	"Fiddler On The Roof" Motion Picture Orchestra	0.884926	0.425074	262890.962963	0.245770	0.073587	0.275481	-15.639
4	"Joseph And The Amazing Technicolor Dreamcoat"...	0.510714	0.467143	270436.142857	0.488286	0.009400	0.195000	-10.236

2. Exploratory Data Analysis (EDA)

Data Distribution

In [12]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 174389 entries, 0 to 174388
Data columns (total 19 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   acousticness      174389 non-null  float64
 1   artists           174389 non-null  object
 2   danceability      174389 non-null  float64
 3   duration_ms       174389 non-null  int64
 4   energy            174389 non-null  float64
 5   explicit          174389 non-null  int64
 6   id                174389 non-null  object
```

```
 7   instrumentalness  174389 non-null  float64
 8   key               174389 non-null  int64
 9   liveness          174389 non-null  float64
 10  loudness          174389 non-null  float64
 11  mode              174389 non-null  int64
 12  name              174389 non-null  object
 13  popularity        174389 non-null  int64
 14  release_date      174389 non-null  object
 15  speechiness       174389 non-null  float64
 16  tempo             174389 non-null  float64
 17  valence           174389 non-null  float64
 18  year              174389 non-null  int64
dtypes: float64(9), int64(6), object(4)
memory usage: 25.3+ MB
```

In [49]:
```
#Describe those numeric features
data[['acousticness','danceability','energy','liveness','loudness','mode','popularity',
```

Out[49]:

	acousticness	danceability	energy	liveness	loudness	mode	I
count	174389.000000	174389.000000	174389.000000	174389.000000	174389.000000	174389.000000	1743
mean	0.499228	0.536758	0.482721	0.211123	-11.750865	0.702384	
std	0.379936	0.176025	0.272685	0.180493	5.691591	0.457211	
min	0.000000	0.000000	0.000000	0.000000	-60.000000	0.000000	
25%	0.087700	0.414000	0.249000	0.099200	-14.908000	0.000000	
50%	0.517000	0.548000	0.465000	0.138000	-10.836000	1.000000	
75%	0.895000	0.669000	0.711000	0.270000	-7.499000	1.000000	
max	0.996000	0.988000	1.000000	1.000000	3.855000	1.000000	1

Most of them range from 0-1 but some came from their own range. Loudness can go from -60 to 3.85, Tempos are from 0 to 243

Let's hear some tracks

What are those Maximum and Minimum sounds like? It is a music dataset, let's hear them!

The loudest tracks:

In [14]:
```
df_loud = data.sort_values('loudness',ascending=False).head().reset_index()
df_loud[['artists','name','loudness']]
```

Out[14]:

	artists	name	loudness
0	Apocolothoth	Sold	3.855
1	The Stooges	Your Pretty Face Is Going to Hell - Alternate ...	3.744
2	Wolfchilde	Weight of Years	3.367
3	The Stooges	Death Trip - Iggy Pop Mix	1.963
4	DYING SPASM	drag	1.830

One of the loudest track: 'Your Pretty Face Is Going to Hell'(turn your volume down)

In [15]:
```
audio_loud = Audio(filename='Your Pretty Face Is Going to Hell.mp3')
audio_loud
```

Out[15]:

0:00 / 4:54

The most energetic tracks:

In [16]:
```
df_energy = data.sort_values(['energy','popularity'],ascending=False).head().reset_inde
df_energy[['artists','name','energy']]
```

Out[16]:

	artists	name	energy
0	Rain Recordings	Forest Rain	1.0
1	Creatress	Steady Forest Rain	1.0
2	Epic Soundscapes	Heavy Rain	1.0
3	Rain Sounds ACE	Moderate Rain	1.0
4	Darkthrone	Transilvanian Hunger - Studio	1.0

Surprisingly, the top 4 energetic tracks with the highest popularity scores are all rain sound recordings. It might be due to how Spotify's algorithm calculates the energy score.

In [17]:
```
audio_energy = Audio(filename='Forest Rain.mp3')
audio_energy
```

Out[17]:

0:00 / 2:50

The most danceable tracks:

In [18]:
```
df_dance = data.sort_values(['danceability','popularity'],ascending=False).head().reset
df_dance[['artists','name','danceability']]
```

Out[18]:

	artists	name	danceability
0	Tone-Loc	Funky Cold Medina	0.988
1	Spooner Street, Rio Dela Duna, Leonardo La Mark	Cool - Leonardo La Mark Remix	0.987
2	Pitbull, Trina, Young Bo	Go Girl	0.986
3	Tone-Loc	Funky Cold Medina - Re-Recorded	0.985
4	Nilla Pizzi	O mama mama - Remix 2014	0.985

One of the top five tracks:

In [19]:
```
audio_dance = Audio(filename='Tone-Loc Funky Cold Medina.mp3')
audio_dance
```

Out[19]:

0:00 / 4:17

The happiest (highest valence) tracks:

In [20]:
```
df_happy = data.sort_values(['valence','popularity'],ascending=False).head().reset_inde
df_happy[['artists','name','valence']]
```

Out[20]:

	artists	name	valence
0	Montez de Durango	Pasito Duranguense	1.000
1	Spongebob Squarepants	Electric Zoo	1.000
2	Raffi	Les Petites Marionettes	1.000
3	Raymond Scott	Chatter	1.000
4	8-Bit Arcade	2000 Light Years Away (8-Bit Computer Game Ver...	0.997

In [22]:
```
audio_happy = Audio(filename='Pasito Duranguense.mp3')
audio_happy
```

Out[22]:

0:00 / 3:20

The clamest (lowest valence) tracks:

In [22]:
```
df_clamest = data.sort_values(['valence','popularity'],ascending=(True,False)).head().r
df_clamest[['artists','name','valence']]
```

Out[22]:

	artists	name	valence
0	Erik Eriksson, White Noise Baby Sleep, White N...	Clean White Noise - Loopable with no fade	0.0
1	Granular	White Noise - 500 hz	0.0
2	Granular	White Noise - 145 hz	0.0
3	High Altitude Samples	Soft Brown Noise	0.0
4	Water Sound Natural White Noise	Deep Sleep Recovery Noise	0.0

Seems the most non-positive tracks are those white noise recording that help people sleep:

In [23]:
```
audio_clam = Audio(filename='Clean White Noise - Loopable With No Fade.mp3')
audio_clam
```

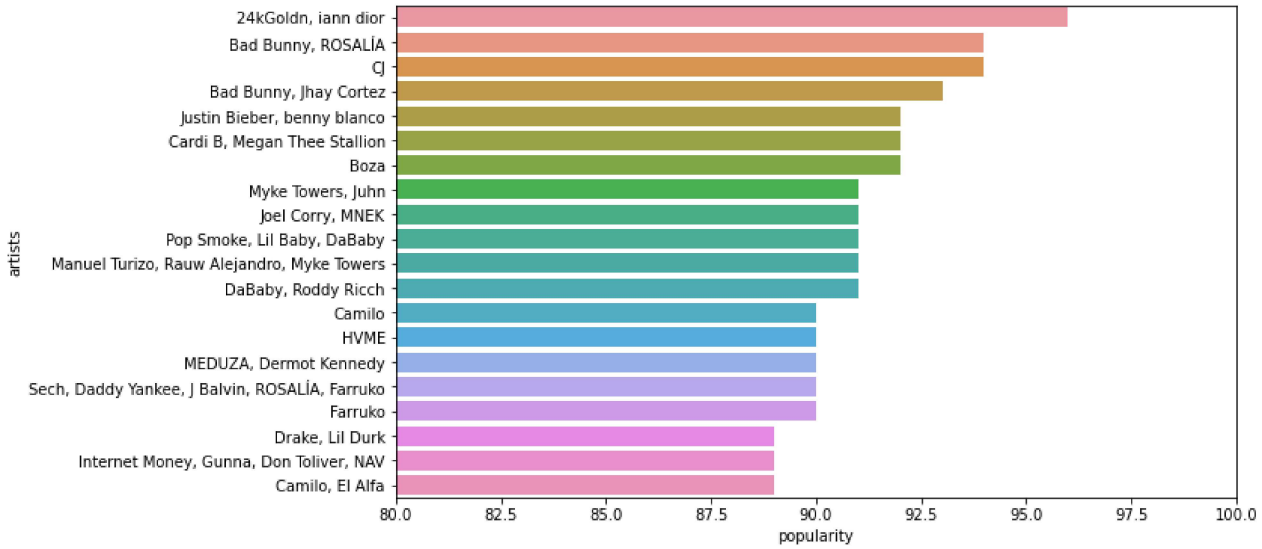
Out[23]:

0:00 / 1:32

The most popular artists:

In [24]:
```
artist_pop = pd.DataFrame(data.groupby('artists')['popularity'].mean().reset_index())
artist_pop = artist_pop.sort_values(['popularity'],ascending=(False)).head(20).reset_in
```

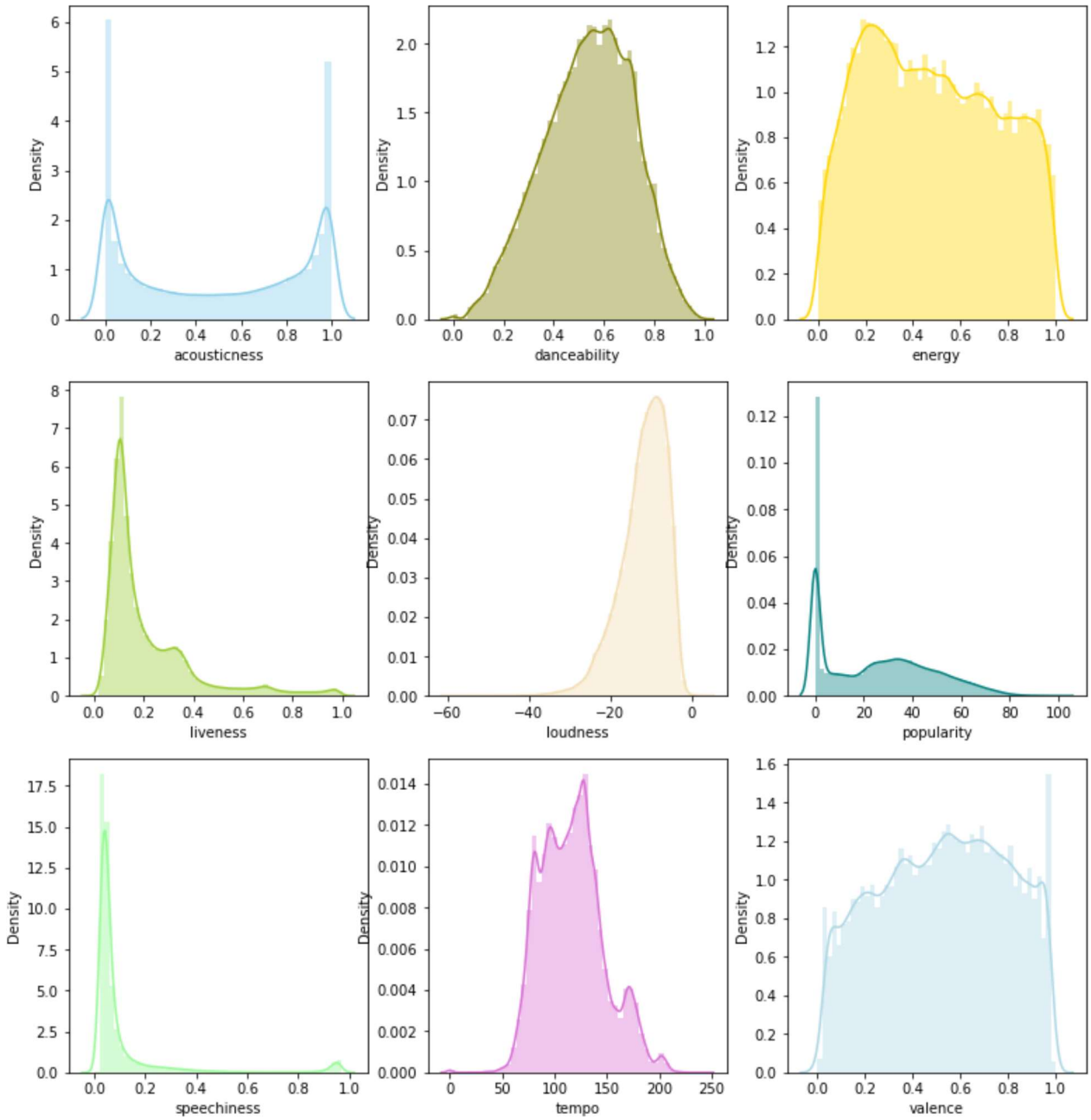
```
plt.figure(figsize=(10,6))
# make barplot
sns.barplot(x=artist_pop.popularity, y=artist_pop.artists, data=artist_pop)
plt.xlim(80,100)
plt.show()
```



Feature Distrubutions

In [25]:
```
f, axes = plt.subplots(3, 3, figsize=(13, 14), sharex=False)
sns.distplot( data["acousticness"] , color="skyblue", ax=axes[0, 0])
sns.distplot( data["danceability"] , color="olive", ax=axes[0, 1])
sns.distplot( data["energy"] , color="gold", ax=axes[0, 2])
sns.distplot( data["liveness"] , color="yellowgreen",ax=axes[1, 0])
sns.distplot( data["loudness"] , color="wheat", ax=axes[1, 1])
sns.distplot( data["popularity"] , color="teal", ax=axes[1, 2])
sns.distplot( data["speechiness"] , color="palegreen", ax=axes[2, 0])
sns.distplot( data["tempo"] , color="orchid", ax=axes[2, 1])
sns.distplot( data["valence"] , color="lightblue", ax=axes[2, 2])
```

Out[25]:
```
<AxesSubplot:xlabel='valence', ylabel='Density'>
```



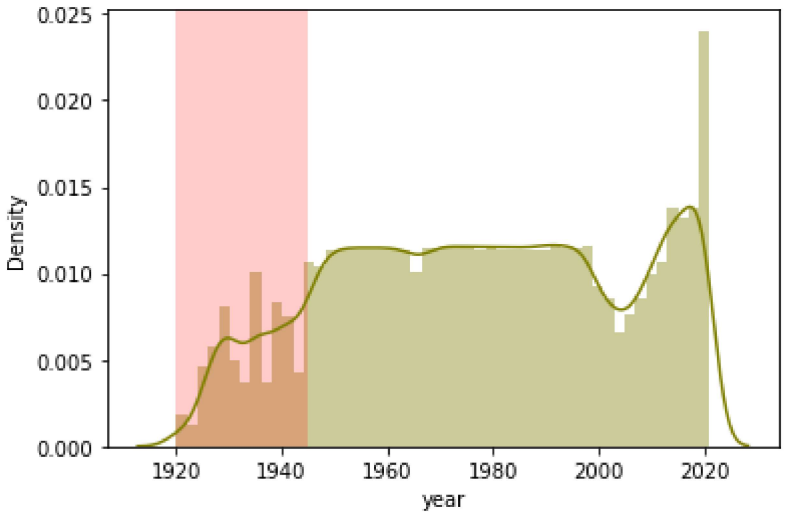
From those charts, we can see:

- Most of the tracks are not live music.
- Most of the tracks are low in 'speechness.' For example, Raps have a certain 'speech' level while Parlour music does not.
- Danceability seems like a normal shape, which suggests that it is randomly distrubuted
- Valence(sentiment) is balanced in general, but with a high number of records near 1.0 - very positive song. Generally, there are more positive songs than negative songs in this collection
- Acuousticness is polarized. Most of the tracks are either pure instrumental or vocal, which make sense.
- Many tracks have '0' popularity. Most of the popularity scores sit from 20 to 60.

Let's examine the distribution by years. The chart below shows that fewer tracks are available before 1945, and the number varies significantly among those years. On the other hand, a large amount of collection shows up from 2019 to the present.

In [26]:
```
sns.distplot( data["year"] , color="olive")
plt.axvspan(1920, 1945, facecolor='r', alpha=0.2)
```

Out[26]:
```
<matplotlib.patches.Polygon at 0x1d73b46ca88>
```



Popularity - What are those 0s score?

There are a large number of records with '0' popularity scores in the previous histogram matrix. What are those tracks that are being '0' popular? Do they have some shared characteristics? Let's take a look at those and see if there is any pattern.

In [27]:
```
#some 0 popularity tracks
mask = data.popularity==0
df0=data[mask].tail().reset_index()
df0[['artists','name','popularity']]
```

Out[27]:

	artists	name	popularity
0	Alessia Cara	A Little More	0
1	DJ Combo, Sander-7, Tony T	The One	0
2	Alessia Cara	A Little More	0
3	Roger Fly	Together	0
4	Roger Fly	Improvisations	0

In [28]:
```
#describe 0s
data[mask].describe()
```

Out[28]:

	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	
count	40905.000000	40905.000000	4.090500e+04	40905.000000	40905.000000	40905.000000	40905.0
mean	0.673725	0.534144	2.325309e+05	0.370634	0.080137	0.344230	5.2
std	0.389875	0.184625	2.024124e+05	0.262042	0.271508	0.391599	3.5
min	0.000000	0.000000	4.937000e+03	0.000000	0.000000	0.000000	0.0
25%	0.284000	0.393000	1.523080e+05	0.173000	0.000000	0.000014	2.0

	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	
50%	0.918000	0.552000	1.909470e+05	0.285000	0.000000	0.067700	5.0
75%	0.987000	0.687000	2.528240e+05	0.531000	0.000000	0.816000	8.0
max	0.996000	0.987000	5.338302e+06	1.000000	1.000000	0.999000	11.0

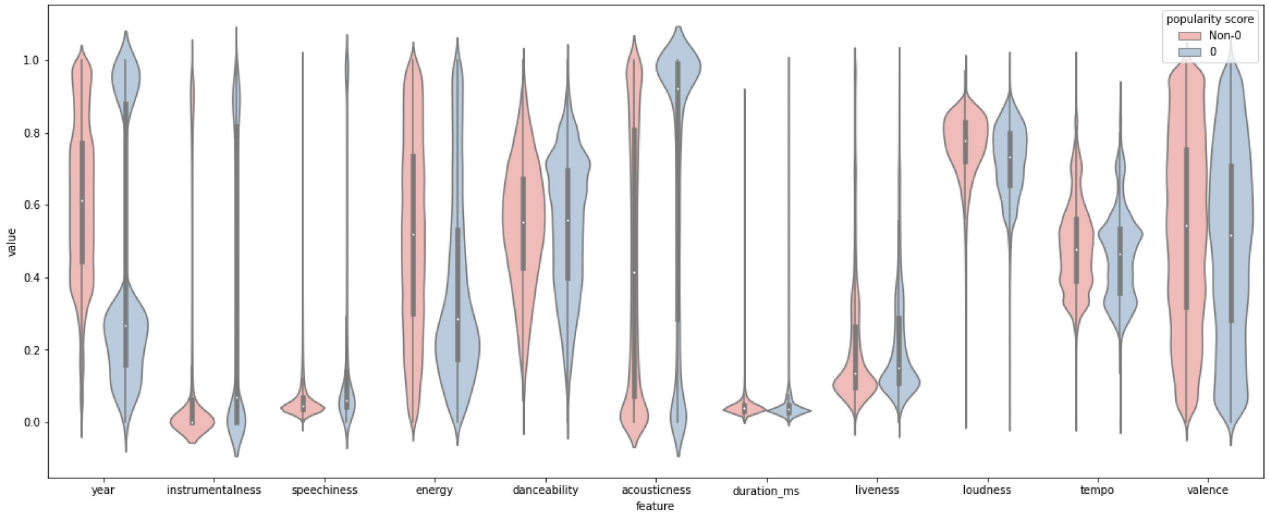
In [29]:
```
#describe non-0s
data[data.popularity>0].describe()
```

Out[29]:

	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	
count	133484.000000	133484.000000	1.334840e+05	133484.000000	133484.000000	133484.000000	13
mean	0.445756	0.537559	2.328956e+05	0.517069	0.064457	0.152212	
std	0.360302	0.173297	1.273368e+05	0.266594	0.245566	0.301002	
min	0.000000	0.000000	1.470800e+04	0.000000	0.000000	0.000000	
25%	0.070700	0.421000	1.696658e+05	0.299000	0.000000	0.000000	
50%	0.412000	0.547000	2.124000e+05	0.519000	0.000000	0.000179	
75%	0.805000	0.663000	2.679730e+05	0.737000	0.000000	0.061800	
max	0.996000	0.988000	4.892761e+06	1.000000	1.000000	1.000000	

If we compare the average of the 0s popular with the other data, the 0s one are generally older(year 1960 vs. 1982), less energetic(0.37 vs. 0.51), more instrumental(0.34 vs. 0.15), and more speeches involved(0.19 vs. 0.08.)

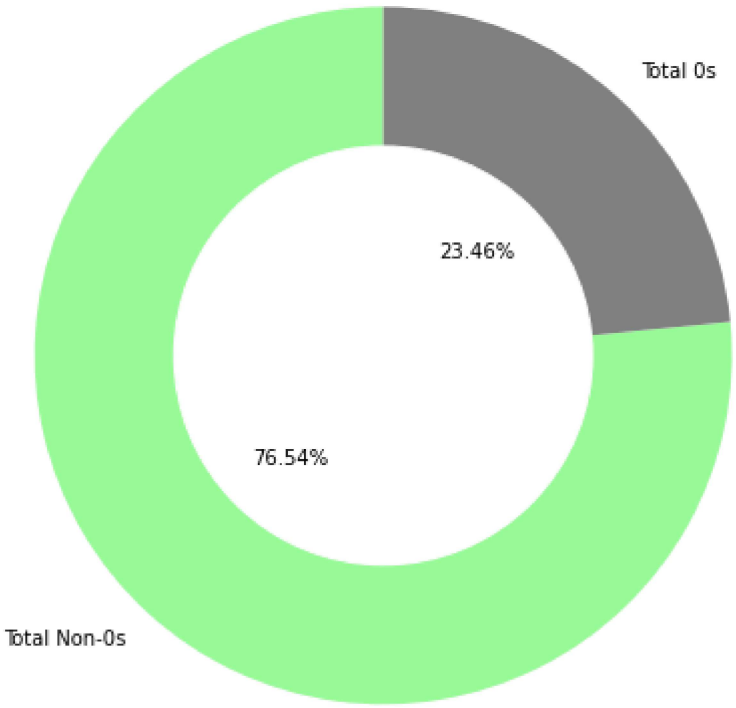
In [30]:
```
#salce the feature value from 0-1 so we can place them in to one chart
scaler = MinMaxScaler()
df_scale = data[['year','instrumentalness','speechiness','energy','danceability','acous
scaler.fit(df_scale)
df_scale = pd.DataFrame(scaler.transform(df_scale))
df_scale.columns=['year','instrumentalness','speechiness','energy','danceability','acou

#transpose columns to row
df_violin = pd.DataFrame(df_scale.T.unstack())
df_violin.index.rename(['ind1','ind2'],inplace=True)

#Add the popularity from main dataset
df_violin=df_violin.join(data['popularity'],on='ind1',how='inner').reset_index()
df_violin.drop('ind1',axis=1,inplace=True)
df_violin.columns=['feature','value','popularity']

#create another column for 0 of non0
df_violin['popularity score']=df_violin.popularity.apply(lambda x:'0'if x ==0 else 'Non
df_violin.drop('popularity',axis=1,inplace=True)

#draw violinplot
plt.figure(figsize=(20,8))
sns.violinplot(x="feature", y="value", hue='popularity score', data=df_violin, palette=
plt.show()
```

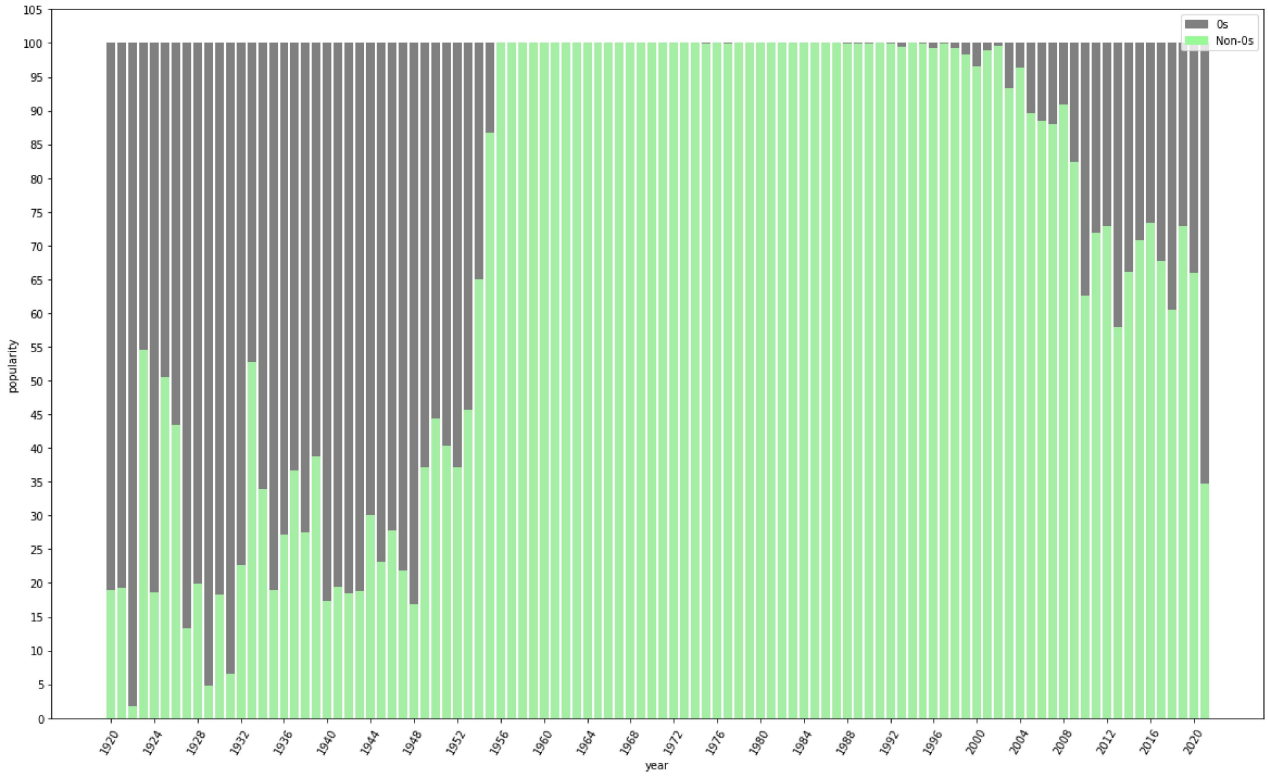


The violin plots present similar results: Less energetic and more acoustic tracks are more likely to have a 0 popularity score. For year, it is obvious that the 0 popular tracks appear to be either older or newer than the average.

In [31]:
```
#Count the 0s and non-0s for donut chart
pop_df = pd.DataFrame(data.popularity>0)
pop_df = pop_df.value_counts().tolist()

#prepare data group by years for the bar charts
non_0 = data[data.popularity>0].groupby('year')['popularity'].count().reset_index()
total = data.groupby('year')['popularity'].count().reset_index()
non_0['popularity'] = [i / j * 100 for i,j in zip(non_0['popularity'], total['popularit
total['popularity'] = [i / j * 100 for i,j in zip(total['popularity'], total['popularit


#Draw a donut chart
names = ['Total Non-0s', 'Total 0s']
plt.figure(figsize=(8,8))

#ax1 = plt.subplot2grid((1,2),(0,0))
my_circle = plt.Circle( (0,0), 0.6, color='white')
plt.pie(pop_df, labels=names, colors=['palegreen','grey'],autopct='%1.2f%%',startangle=
p = plt.gcf()
p.gca().add_artist(my_circle)
#fig.suptitle('Overall 0s Popularity Percentage', fontsize=10)

#Draw a bar chart
# bar chart 1 -> top bars (group of 'Popularity=0')
plt.figure(figsize=(20,12))
#ax2 = plt.subplot2grid((1,2),(0,1))
bar1 = sns.barplot(x="year",y="popularity", data=total, color='grey')
plt.xticks(rotation=90)
plt.locator_params(nbins=30)

# bar chart 2 -> bottom bars (group of 'Popularity>0')
bar2 = sns.barplot(x="year",y="popularity", data=non_0, color='palegreen')
plt.xticks(rotation=60)
plt.locator_params(nbins=30)
# add legend
top_bar = mpatches.Patch(color='grey', label='0s')
bottom_bar = mpatches.Patch(color='palegreen', label='Non-0s')
plt.legend(handles=[top_bar, bottom_bar])
```

```
#fig.suptitle('Overall 0s Popularity Percentage', fontsize=14)
# Show the graph
plt.show()
```





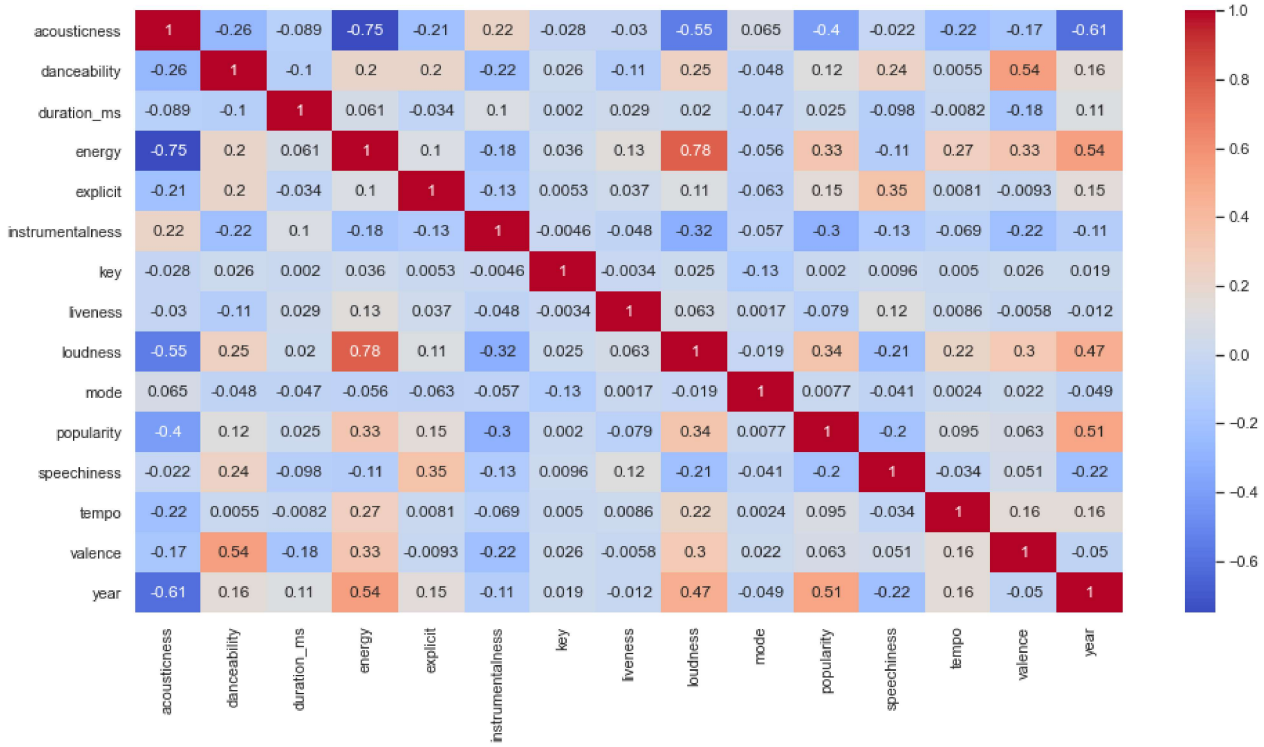
In total, we have about a quarter of tracks(23.46%) that are 0 popular. The bar charts above show that the tracks before 1955 take up the majority of those 0s score. One reason might be that they are 'too old' for people to listen to nowadays. It seems that there is a 'Golden Age' period from 1960

to 2000, where more than 95% of tracks back then have scored some popularity. Nonetheless, the latest tracks that are in 2021 might be too new to have a popularity score assigned.

Feature Correlations

In [32]:
```
plt.figure(figsize=(16, 8))
sns.set(style="whitegrid")
corr = data.corr()
sns.heatmap(corr,annot=True,cmap='coolwarm')
```

Out[32]: <AxesSubplot:>

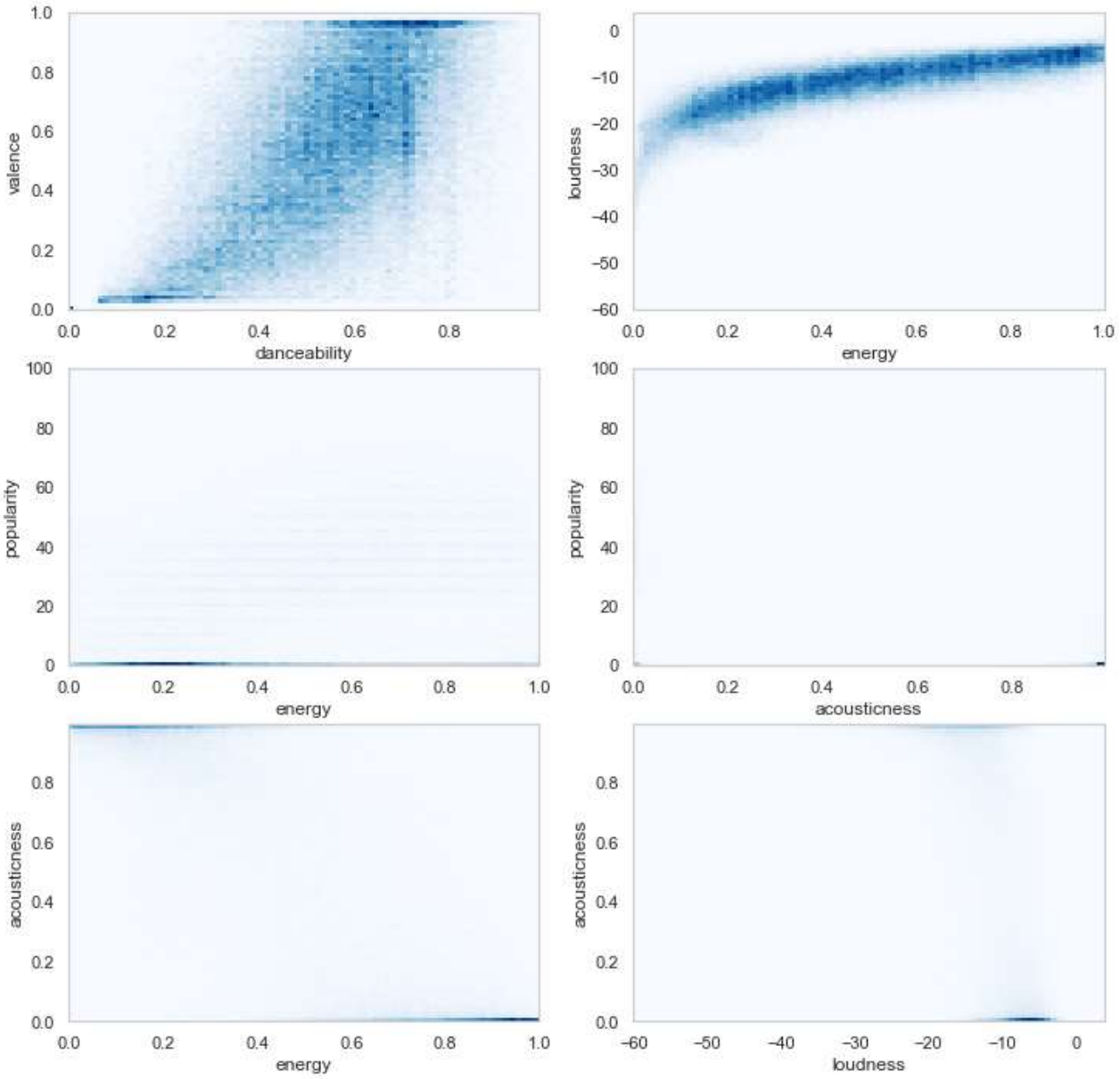


If we look at the popularity column, we notice that it is highly correlated with some of the features. Spotify users seem to prefer the more energetic, louder, less acoustic tracks.
There is also a high correlation of 0.51 with feature 'year.' We notice that the music track has become less acoustic, more energic, and louder over the years.
Some other correlation pairs are inline with what we expected, such as high energy tracks are louder, non-acoustic tracks(R&R, electronic, etc.) are more energetic and louder.

Based on the heatmap, I plot some high correlation feature pairs to examine their relationship in 2d histogram. The darker of bins in the charts, the more points lay at the same position. As we can see, Danceability/Valence, Energy/Lounderness shows a strong positive linear relationship, while others are not very obvious.

In [33]:
```
fig, [(axes, axes1), (axes2, axes3),(axes4, axes5)] = plt.subplots(nrows = 3, ncols = 2
axes.hist2d(x = data['danceability'],y = data['valence'], bins=(80, 80), cmap=plt.cm.Bl
axes.set_xlabel('danceability', fontsize=12)
axes.set_ylabel('valence', fontsize=12)
axes1.hist2d(x = data['energy'],y = data['loudness'], bins=(80, 80), cmap=plt.cm.Blues)
axes1.set_xlabel('energy', fontsize=12)
axes1.set_ylabel('loudness', fontsize=12)
axes2.hist2d(x = data['energy'],y = data['popularity'], bins=(80, 80), cmap=plt.cm.Blue
axes2.set_xlabel('energy', fontsize=12)
```

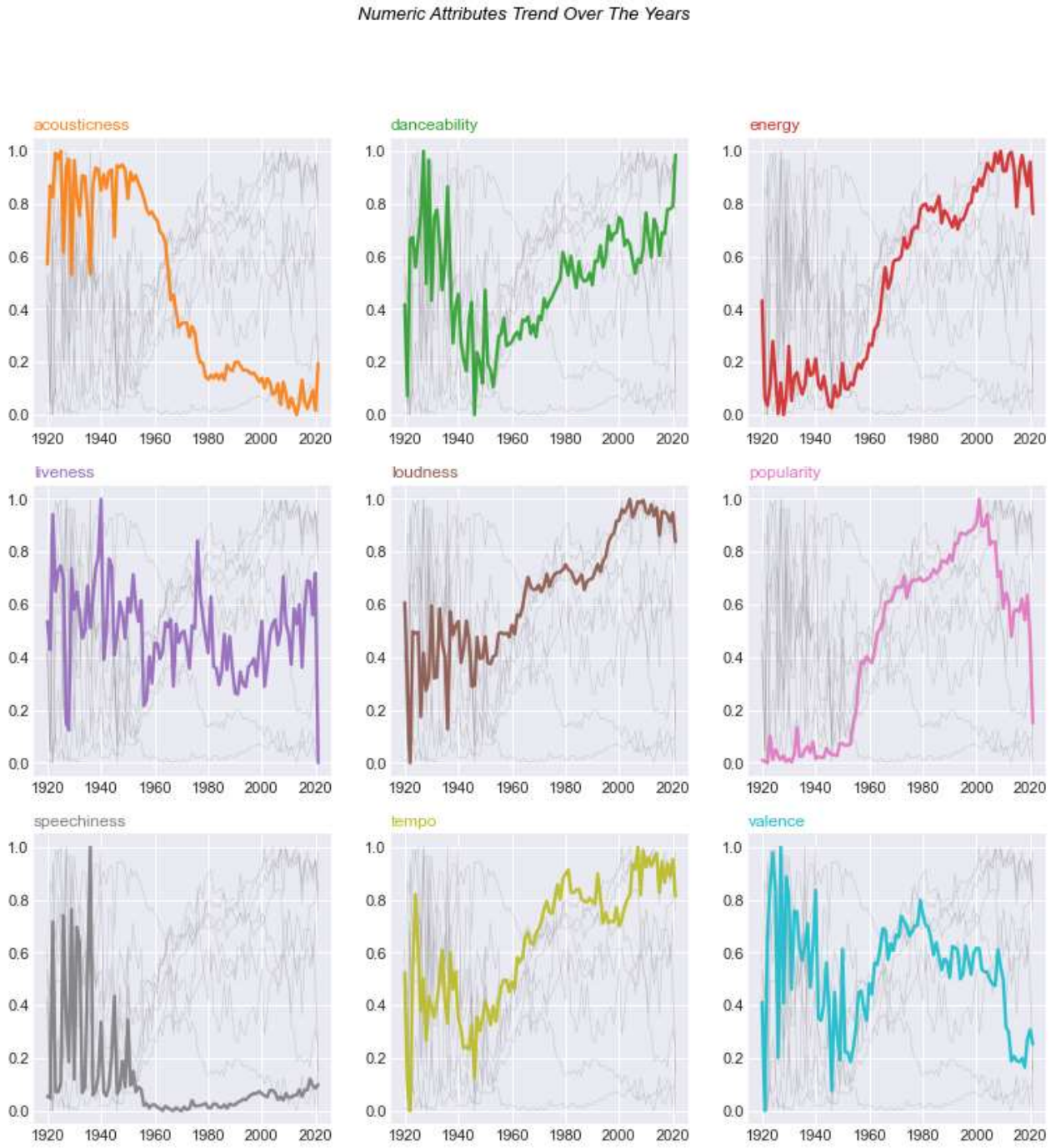
```
axes2.set_ylabel('popularity', fontsize=12)
axes3.hist2d(x = data['acousticness'],y = data['popularity'], bins=(80, 80), cmap=plt.c
axes3.set_xlabel('acousticness', fontsize=12)
axes3.set_ylabel('popularity', fontsize=12)
axes4.hist2d(x = data['energy'],y = data['acousticness'], bins=(80, 80), cmap=plt.cm.Bl
axes4.set_xlabel('energy', fontsize=12)
axes4.set_ylabel('acousticness', fontsize=12)
axes5.hist2d(x = data['loudness'],y = data['acousticness'], bins=(80, 80), cmap=plt.cm.
axes5.set_xlabel('loudness', fontsize=12)
axes5.set_ylabel('acousticness', fontsize=12)
plt.show()
```



Music Trend Analysis

In [34]:
```
#Extract Numeric features and set them on the same scale
year_num = year_df[['acousticness','danceability','energy','liveness','loudness','popul
scaler = MinMaxScaler()
scaler.fit(year_num)
year_scale = pd.DataFrame(scaler.transform(year_num))
year_scale.columns = ['acousticness','danceability','energy','liveness','loudness','pop
year_scale['year'] = year_df['year']
```

```
#Create line charts for each numeric feature across the years
# Initialize the figure style
plt.style.use('seaborn-darkgrid')

# create a color palette
palette = plt.get_cmap('tab10')

f, axes = plt.subplots(3, 3, figsize=(13, 13), sharex=True)

# multiple line plot

num=0
for column in year_scale.drop('year', axis=1):
    num+=1
    x = np.random.rand(9)
    # Find the right spot on the plot
    plt.subplot(3,3, num)
    plt.figsize=(20, 20)

    # plot every group, but discrete
    for v in year_scale.drop('year', axis=1):
        plt.plot(year_scale['year'], year_scale[v], marker='', color='grey', linewidth=

    # Plot the lineplot
    plt.plot(year_scale['year'], year_scale[column], marker='', color=palette(num), lin

    # Same limits for every chart
    #plt.xlim(0,10)
    #plt.ylim(-2,22)

    # Not ticks everywhere
    if num in range(7) :
        plt.tick_params(labelbottom='off')
    if num not in [1,4,7] :
        plt.tick_params(labelleft='off')

    # Add title
    plt.title(column, loc='left', fontsize=12, fontweight=0, color=palette(num) )

# general title
plt.suptitle("Numeric Attributes Trend Over The Years", fontsize=13, fontweight=0, colo

# Show the graph
plt.show()
```



Lines are volatile before 1950 due to the fewer available records. Some general trends spotted are music becomes louder, faster pace, more energetic ,and easier to dance to. There are also more vocal tracks instrumental over the years. It seems that tracks from 1970 to 2000 are more positive, which might lead to the high popularity scores from that period.

The analysis below summarizes tracks' characteristics into different periods (decades). I wonder if every generation of music has its traits that we could recognize and the artists representing that generation.

In [35]:
```
#Convert years to decades:
def convert_year (year):
    if year >=1920 and year < 1930:
        return '20s'
    elif year >=1930 and year < 1940:
        return '30s'
```

```
    elif year >=1940 and year < 1950:
        return '40s'
    elif year >=1950 and year < 1960:
        return '50s'
    elif year >=1960 and year < 1970:
        return '60s'
    elif year >=1970 and year < 1980:
        return '70s'
    elif year >=1980 and year < 1990:
        return '80s'
    elif year >=1990 and year < 2000:
        return '90s'
    elif year >=2000 and year < 2010:
        return '2000s'
    elif year >=2010:
        return '2010 & newer'
data['decades']=data.year.apply(convert_year)
```

In [36]:
```
data[['artists','name','year','decades']]
```

Out[36]:

	artists	name	year	decades
0	Mamie Smith	Keep A Song In Your Soul	1920	20s
1	Screamin Jay Hawkins	I Put A Spell On You	1920	20s
2	Mamie Smith	Golfing Papa	1920	20s
3	Oscar Velazquez	True House Music - Xavier Santos & Carlos Gomi...	1920	20s
4	Mixe	Xuniverxe	1920	20s
...	...	...	...	...
174384	DJ Combo, Sander-7, Tony T	The One	2020	2010 & newer
174385	Alessia Cara	A Little More	2021	2010 & newer
174386	Roger Fly	Together	2020	2010 & newer
174387	Taylor Swift	champagne problems	2021	2010 & newer
174388	Roger Fly	Improvisations	2020	2010 & newer

174389 rows × 4 columns

The most popular artists of each generation:

In [37]:
```
artist_group = pd.DataFrame(data.groupby(['decades','artists'])['popularity'].mean()).s
artist_group.reset_index(inplace=True)
artist_group['rank'] = artist_group.groupby('decades')['popularity'].rank(method='first
artist_group[artist_group['rank']==1.0]
```

Out[37]:

	decades	artists	popularity	rank
0	2010 & newer	24kGoldn, iann dior	96.0	1.0
69	90s	Frank Sinatra, B. Swanson Quartet	84.0	1.0
72	80s	Grover Washington, Jr., Bill Withers	83.0	1.0

	decades	artists	popularity	rank
115	2000s	Shakira, Wyclef Jean	82.0	1.0
531	60s	The Beach Boys, Mark Linett, Sweet, Larry Walsh	75.0	1.0
742	70s	Earth, Wind & Fire, The Emotions	73.0	1.0
3569	50s	Gayla Peevey	62.0	1.0
4764	40s	Bing Crosby, The Andrews Sisters	58.6	1.0
16436	20s	Benny Goodman, Peggy Lee	37.0	1.0
18700	30s	Richard Himber and his Orchestra, Johnny Mercer	34.0	1.0

The six radar charts below show six music generations' features, from the 60s to 2010, following with the most popular artist of that time. The grey hexagon underneath shows the time period's average, and the colored one is the artist's characteristics

In [38]:
```
#Get the numeric featrure of those most poplar artists from th 60s
artist_g_top6 = artist_df[artist_df.artists.isin(['24kGoldn','Frank Sinatra','Grover Wa
artist_g_top6.reset_index(inplace=True)
artist_g_top6['decades'] = ['2010 & newer','70s','90s','80s','2000s','60s']
artist_g_top6_num = artist_g_top6[['acousticness','danceability','energy','loudness','t
artist_g_top6 = artist_g_top6.reset_index()

#Scale those numeric feature based on the main dataset so they can be plotted on the sa
scaler = MinMaxScaler()
scaler.fit(artist_df[['acousticness','danceability','energy','loudness','tempo','valenc
art6_scale = pd.DataFrame(scaler.transform(artist_g_top6_num))
art6_scale.columns = ['acousticness','danceability','energy','loudness','tempo','valenc
art6_scale['artists'] = artist_g_top6['artists']
art6_scale.columns = ['acousticness','danceability', 'energy', 'loudness', 'tempo','val
art6_scale['decades']=artist_g_top6['decades']

#sort the data by decades and make places for the average value for decades
art6_scale=art6_scale.sort_values(by='decades').reset_index().drop('index',axis=1)
art6_scale=art6_scale.reindex([2,3,4,5,0,1]).reset_index().drop('index',axis=1)
art6_scale.index = [0,2,4,6,8,10]

#Calualte the average of numeric value for different decades
avg_df = data.groupby('decades')['acousticness','danceability','energy','loudness','tem
avg_df=avg_df.drop(avg_df.index[2:6])
avg_df['artists'] = 'average'
avg_df = avg_df[['acousticness','danceability','energy','loudness','tempo','valence','a

#Reindex the average and combine them into the artist df
avg_df = avg_df.reindex([2,3,4,5,0,1]).reset_index().drop('index',axis=1)
scaler.fit(artist_df[['acousticness','danceability','energy','loudness','tempo','valenc
avg_df_scale = pd.DataFrame(scaler.transform(avg_df[['acousticness','danceability','ene
avg_df_scale.columns = ['acousticness','danceability','energy','loudness','tempo','vale
avg_df_scale[['artists','decades']]=avg_df[['artists','decades']]
avg_df_scale.index = [1,3,5,7,9,11]
art6_scale=pd.concat([art6_scale,avg_df_scale],axis=0)
art6_scale.sort_index(inplace=True)

# ------- PART 1: Define a function that do a plot for one line of the dataset!

def make_spider( row, title, color):
```

```
    # number of variable
    categories=list(art6_scale)[:-2]
    N = len(categories)

    # What will be the angle of each axis in the plot? (we divide the plot / number of
    angles = [n / float(N) * 2 * pi for n in range(N)]
    angles += angles[:1]

    # Initialise the spider plot
    ax = plt.subplot(2,3,ind, polar=True, )

    # If you want the first axis to be on top:
    ax.set_theta_offset(pi / 2)
    ax.set_theta_direction(-1)

    # Draw one axe per variable + add labels labels yet
    plt.xticks(angles[:-1], categories, color='grey', size=8)

    # Draw ylabels
    ax.set_rlabel_position(0)
    plt.yticks([0.2,0.4,0.6,0.8], ["0.2","0.4","0.6","0.8"], color="grey", size=7)
    plt.ylim(0,1)

    # Ind1
    values=art6_scale.loc[row].drop(['artists','decades']).values.flatten().tolist()
    values += values[:1]
    avg_values = art6_scale.loc[row+1].drop(['artists','decades']).values.flatten().tol
    avg_values+=avg_values[:1]
    ax.plot(angles, avg_values, color='grey', linewidth=0, linestyle='solid')
    ax.plot(angles, values, color=color, linewidth=2, linestyle='solid')
    ax.fill(angles, values, color=color, alpha=0.8)
    ax.fill(angles, avg_values, color='grey', alpha=0.6)

    # Add a title
    plt.title(title, size=11, color=color, y=1.1)


# ------- PART 2: Apply the function to all individuals
# initialize the figure
my_dpi=96
plt.figure(figsize=(1200/my_dpi, 1200/my_dpi), dpi=my_dpi)

# Create a color palette:
my_palette = plt.cm.get_cmap("Set1")

# Loop to plot
ind = 0
for row in range(0, len(art6_scale.index),2):
    ind +=1
    make_spider( row=row, title=art6_scale['decades'][row]+':'+'\n'+art6_scale['artists
```

Key and mode

In [39]:
```
#Map and convert the key and mode
key_mapping = {0:"C",1:"C#",2:"D",3:"D#",4:"E",5:"F",6:"F#",7:"G",8:"G#",9:"A",10:"A#",
mode_mapping = {0:"Minor",1:"Major"}
key_name = data.key.map(key_mapping)
mode_name= data['mode'].map(mode_mapping)
data['key_mode'] = key_name+' '+mode_name
data[['name','key','mode','key_mode']]
```

Out[39]:

	name	key	mode	key_mode
0	Keep A Song In Your Soul	5	0	F Minor
1	I Put A Spell On You	5	0	F Minor
2	Golfing Papa	0	1	C Major
3	True House Music - Xavier Santos & Carlos Gomi...	2	1	D Major
4	Xuniverxe	10	0	A# Minor
...	...	...	...	...
174384	The One	6	0	F# Minor

	name	key	mode	key_mode
174385	A Little More	4	1	E Major
174386	Together	4	0	E Minor
174387	champagne problems	0	1	C Major
174388	Improvisations	7	1	G Major

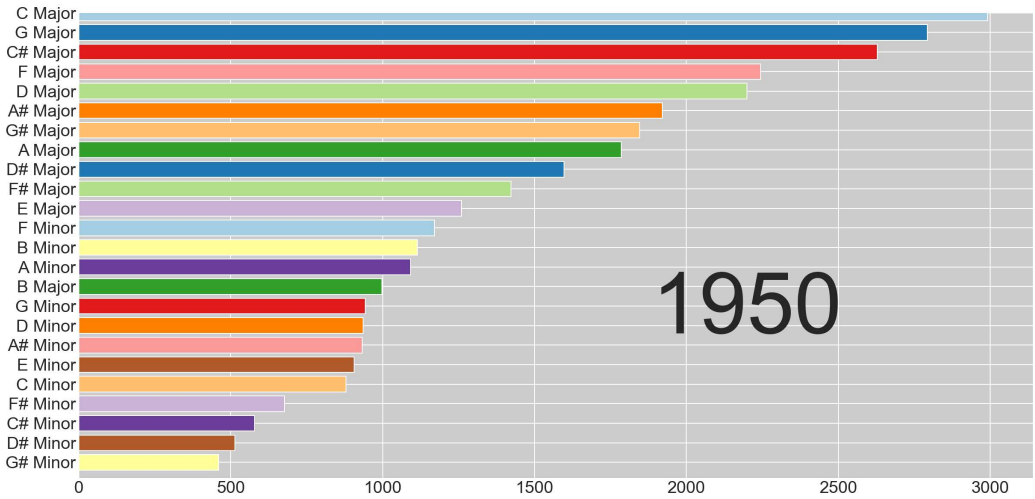
174389 rows × 4 columns

Music keys being used over the years:

In [40]:
```
#Count the number of Key used
key_count = pd.DataFrame(data.key_mode.value_counts()).reset_index()
key_count.columns=['key_name','counts']

#recoder and reformat the key counts by years
column_list = key_count['key_name'].values
ani_bar_df = data.groupby('year')['key_mode'].value_counts().unstack().reset_index()
ani_bar_df_cum = ani_bar_df.cumsum()
ani_bar_df_cum['year']=ani_bar_df['year']
ani_bar_df_cum.set_index('year',inplace=True)
ani_bar_df_cum=ani_bar_df_cum[column_list]

#Select data from 1950 to present
ani_bar_df_cum = ani_bar_df_cum.iloc[30:,:]

#prepare the dat for animation
def prepare_data(df, steps=5):
    df = df.reset_index()
    df.index = df.index * steps
    last_idx = df.index[-1] + 1
    df_expanded = df.reindex(range(last_idx))
    df_expanded['year'] = df_expanded['year'].fillna(method='ffill')
    df_expanded = df_expanded.set_index('year')
    df_rank_expanded = df_expanded.rank(axis=1, method='first')
    df_expanded = df_expanded.interpolate()
    df_rank_expanded = df_rank_expanded.interpolate()
    return df_expanded, df_rank_expanded

df_expanded, df_rank_expanded = prepare_data(ani_bar_df_cum)

#draw a race chart
def nice_axes(ax):
    ax.set_facecolor('.8')
    ax.tick_params(labelsize=20, length=0)
    ax.grid(True, axis='x', color='white')
    ax.set_axisbelow(True)
    [spine.set_visible(False) for spine in ax.spines.values()]

def init():
    ax.clear()
    nice_axes(ax)
    ax.set_ylim(.2, 6.8)
colors = plt.cm.Paired(range(12))
labels = df_expanded.columns
def update(i):
    for bar in ax.containers:
```

```
        bar.remove()
    y = df_rank_expanded.iloc[i]
    width = df_expanded.iloc[i]
    ax.barh(y=y, width=width, color=colors, tick_label=labels)
    date_str = round(df_expanded.index[i])
    ax.set_title(date_str,x=0.7, y=0.3, fontsize=100)

fig = plt.Figure(figsize=(20, 10), dpi=144)
ax = fig.add_subplot()
anim = FuncAnimation(fig=fig, func=update, init_func=init, frames=len(df_expanded),
                     interval=100, repeat=False)
HTML(anim.to_jshtml())
```

Out[40]:



Once Loop Reflect

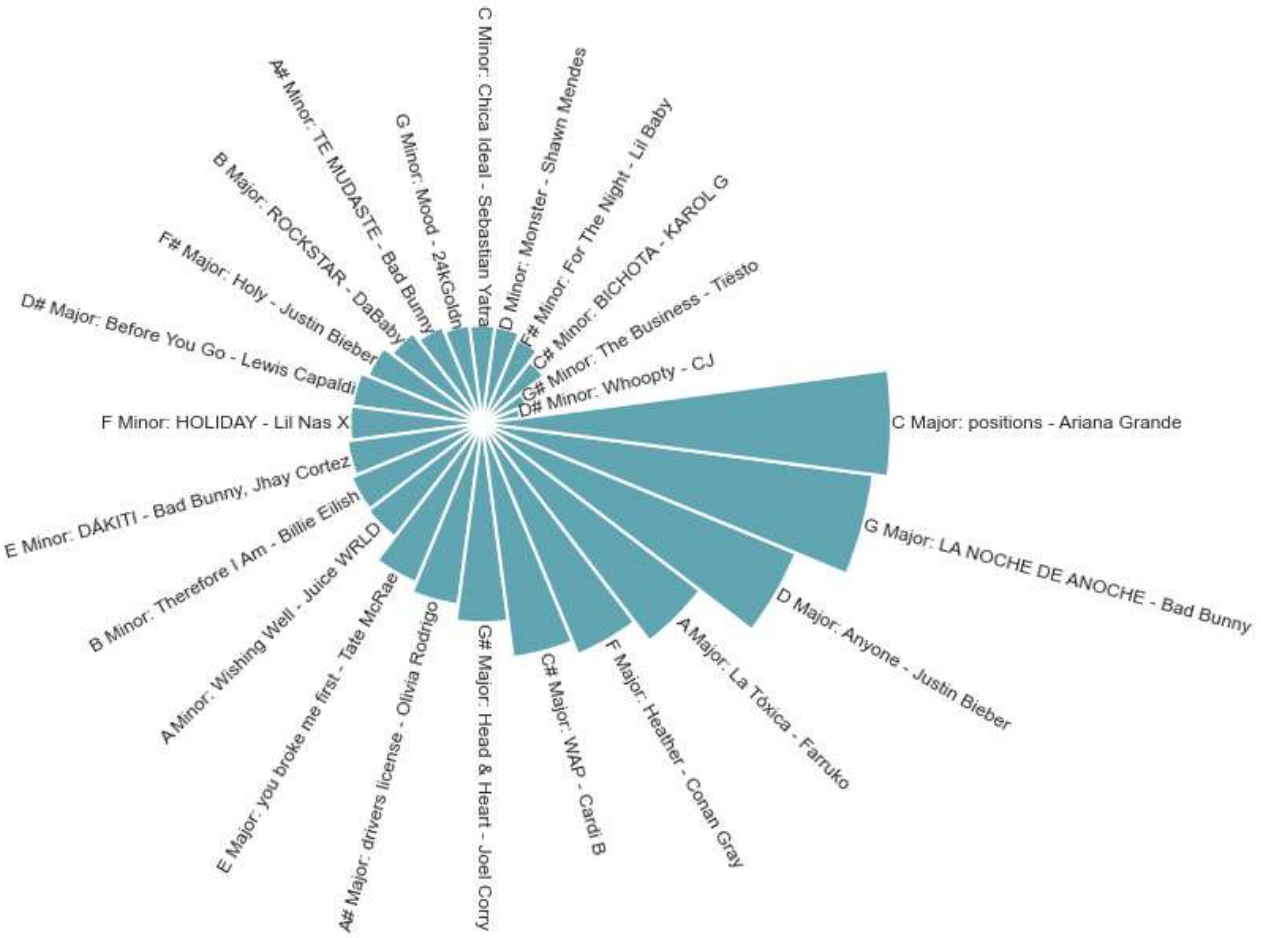
The most popular song of each key:

In [42]:

```
#get the most popular track for each key
data_key_rank = data.copy()
data_key_rank['rank']=data.groupby('key_mode')['popularity'].rank(method='first',ascend
data_key_rank[data_key_rank['rank']==1]
data_key_rank['song_artist'] =data_key_rank.name + ' '+'-'+' '+ data_key_rank.artists.s
data_key_rank = data_key_rank[data_key_rank['rank']==1]

#merge with key counts to show number of tracks with each key
bar_df = pd.merge(key_count,data_key_rank, left_on='key_name',right_on='key_mode')
bar_df = bar_df[['key_name','counts','song_artist']]

#reduce the lenth of artist and tracks name
bar_df = pd.merge(key_count,data_key_rank, left_on='key_name',right_on='key_mode')
bar_df = bar_df[['key_name','counts','song_artist']]
bar_df.loc[bar_df.key_name=='C# Major','song_artist'] = 'WAP - Cardi B'
bar_df.loc[bar_df.key_name=='G Major','song_artist'] = 'LA NOCHE DE ANOCHE - Bad Bunny'
bar_df.loc[bar_df.key_name=='G# Major','song_artist'] = 'Head & Heart - Joel Corry'
bar_df.loc[bar_df.key_name=='F# Major','song_artist'] = 'Holy - Justin Bieber'
bar_df.loc[bar_df.key_name=='B Major','song_artist'] = 'ROCKSTAR - DaBaby'
bar_df.loc[bar_df.key_name=='F# Minor','song_artist'] = 'For The Night - Lil Baby'
```

```
bar_df.loc[bar_df.key_name=='D Minor','song_artist'] = 'Monster - Shawn Mendes'
bar_df.loc[bar_df.key_name=='G Minor','song_artist'] = 'Mood - 24kGoldn'
bar_df.loc[bar_df.key_name=='C Minor','song_artist'] = 'Chica Ideal - Sebastian Yatra'

# Reorder the dataframe
df = bar_df.sort_values(by=['counts'])

# initialize the figure
plt.figure(figsize=(20,10))
ax = plt.subplot(111, polar=True)
plt.axis('off')

# Constants = parameters controling the plot layout:
upperLimit = 100
lowerLimit = 30
labelPadding = 4

# Compute max and min in the dataset
max = df['counts'].max()

# Let's compute heights: they are a conversion of each item value in those new coordina
# In our example, 0 in the dataset will be converted to the lowerLimit (10)
# The maximum will be converted to the upperLimit (100)
slope = (max - lowerLimit) / max
heights = slope * df.counts + lowerLimit

# Compute the width of each bar. In total we have 2*Pi = 360°
width = 2*np.pi / len(df.index)

# Compute the angle each bar is centered on:
indexes = list(range(1, len(df.index)+1))
angles = [element * width for element in indexes]
angles

# Draw bars
bars = ax.bar(
    x=angles,
    height=heights,
    width=width,
    bottom=lowerLimit,
    linewidth=2,
    edgecolor="white",
    color="#61a4b2"
)

# Add labels
for bar, angle, height, label in zip(bars,angles, heights, df["key_name"]+ ':'+' ' +df[

    # Labels are rotated. Rotation must be specified in degrees :(
    rotation = np.rad2deg(angle)

    # Flip some labels upside down
    alignment = ""
    if angle >= np.pi/2 and angle < 3*np.pi/2:
        alignment = "right"
        rotation = rotation + 180
    else:
        alignment = "left"

    # Finally add the labels
    ax.text(
```

```
        x=angle,
        y=lowerLimit + bar.get_height() + labelPadding,
        s=label,
        ha=alignment,
        va='center',
        rotation=rotation,
        rotation_mode="anchor")
```



In [ ]: