Name

1. Introduction

This project investigates the change of features of popular music in different decades - audio features from the Spotify database (e.g. key and tempo) - and song popularity measured by the number of streams a song has on Spotify. Based on crawler technology, about 4000 musics' metadata are collected on 'top 100 music' from Billboard.com. With the Spotify database and analysis API, these 4000 musics were analyzed from 1970 to 2019 to show the features change of music over time. Practical implications include that Spotify can further develop its database and calculations of the variables for in the future their databases will play an important role in new value creation.

1.1 Music Features Introduction

According to the API document of "Get Audio Features for a Track" provided by Spotify, there are 10 measures to describe a music's features. Acousticness is a confidence measure from 0.0 to 1.0 representing if the track is acoustic. 1.0 represents high confidence the track is acoustic. Danceability is used to describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. Energy is a measure from 0.0 to 1.0 used to represent intensity and activity. The measure is based dynamic range, perceived loudness, timbre, onset rate, and general entropy to represent a perceptual measure of intensity and activity. Spotify provides the example that death metal has high energy while a Bach prelude scores low on the scale. Instrumentalness predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the Instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. The measure Key represents the key the track is in, represented as an integer. Integers map to pitches using standard Pitch Class notation. Spotify data analyst and jazz pianist Kenny Ning explains that G, C and E are convenient keys for guitar and piano. Liveness is a measure from 0.0 to 1.0 that detects the presence of an audience in the recording. Loudness measures the overall loudness of a track in decibels (dB). Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db. We assume that the louder the song, the better it is able to communicate emotions. Mode indicates the modality, major (1) or minor (0) of a track, the type of scale from which its melodic content is derived. Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0. Tempo measures the overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration. Given that rap and hip-hop, which tempo is quite low, are part of main stream chart music. Valence is a measure from 0.0 to 1.1 describing the musical positivity conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).

2. Method

In this section, the methodology of the project is explained. This will be done in the following order: defining the research population and sample, elaborating on the data collection method and data pre-processing. This research will use and analyse data retrieved by Spotify API database as of 2018. Spotify provides this Application Programming Interface (API) as an open and free database for developers to use and build their applications. Spotify's Analysis API allows you to obtain audio features for any song on Spotify based on a search, where a number of parameters can be selected, including genre and year.

2.1 Collecting the Data

For collection, the Spotify database was used. The free database is an online API that is open for developers to use data and build their own applications. Spotify's Search API allows you to obtain audio features for any song on Spotify based on a search, where a number of parameters can be selected, including genre and year. A python script was created to automatically pull the data from the database – the Spotify search can only request data one song at a time and create a dataframe. It was used to write the script to query songs from the Spotify database. Each song's audio features were pulled with the "get audio features" API request. These features will make up the independent variables to research. Next, Python corralled all the data into a pandas dataframe, which is a matrix with labels that can be opened in Excel. Each row contains a track, while each column contains the values for the audio features. The number of streams was tracked by hand for each song in the Spotify application as it is not a feature that is given in the API. However, Spotify is sensitive for excessive requests rates, so each time we call the Spotify API, the program will sleep one second to avoid blocking.

2.2 Data Pre-Processing

The following steps are explained to gain insights into the data and for data pre-processing. 4.3.1 Cleaning the data Each song feature is not measured equally in the Spotify API. Predictors 'Acousticness', 'Danceability', 'Energy', 'Instrumentalness', 'Liveness', 'Speechiness' and 'Valence' are all measured from 0-1 and are (continuous) ratio variables. 'Liveness' was recoded in a dummy variable where only the cases above 0,8 where selected as 1 (As suggested by Spotify), with all other cases as 0. Variables 'Duration' (Ms), 'Loudness' (Db) and 'Tempo' (BPM) are also ratio variables measured in their own units. Secondly, our predictor 'Key' is measured as a (categorical) nominal variable with 0-11 corresponding to the keys according to the standard Pitch Class notation. Similarly measured is 'Mode', which is also a nominal variable (1=pop, 2=rap). A third distinct variable is 'Mode' which is a dichotomous variable, where 0 is defined as the Minor mode and 1 as Major. The representative features of music in one year is calculated by popular songs in this period via aggregation computing method i.e. the average and standard deviation of all features data.

2.3 Data Visualization

To shows the differences of music features in different periods, the visualization is the best tool. The plotly is selected to visualize our data, plotly.py is an interactive, open-source, and browser-based graphing library for Python. Built on top of plotly.js, plotly.py is a high-level, declarative charting library. plotly.js ships with over 30 chart types, including scientific charts, 3D graphs, statistical charts, SVG maps, financial charts, and more. For presenting all the feature's differences in six decades, the boxplot is the best choice. A boxplot is a standardized way of displaying the distribution of data based on a five number summary ("minimum", first quartile (Q1), median, third quartile (Q3), and "maximum"). It can tell you about your outliers and what their values are. It can also tell you if your data is symmetrical, how tightly your data is grouped, and if and how your data is skewed. Besides, to present the change of one single feature over time, the time series is a excellent tool to use.

3. Results

Here shows the visualization results and our insight that can obtained from the visualizations.

1. The measures 'danceability' and 'energy' are volatility increasing overtime, presenting the people increasingly prefer dynamic strong music.

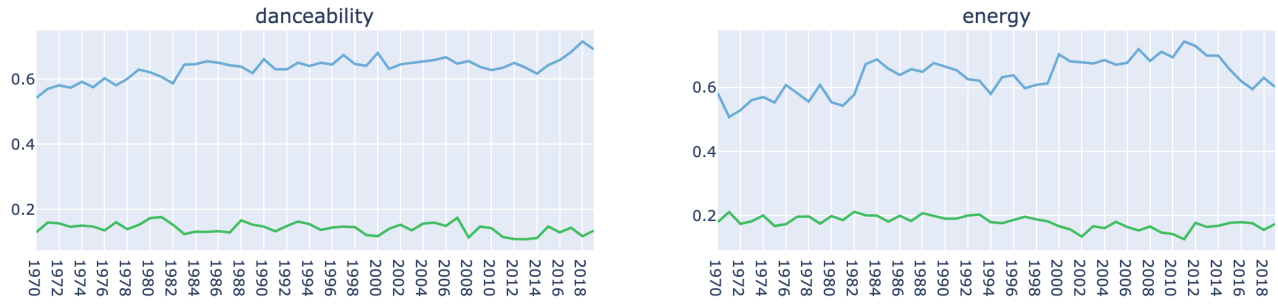

Fig. 1 the values of measures danceability and energy in fifty years

2. The measures 'loudness' and 'speechiness' are keep increasing show the vocal-themed becoming more and more hot and dynamic.

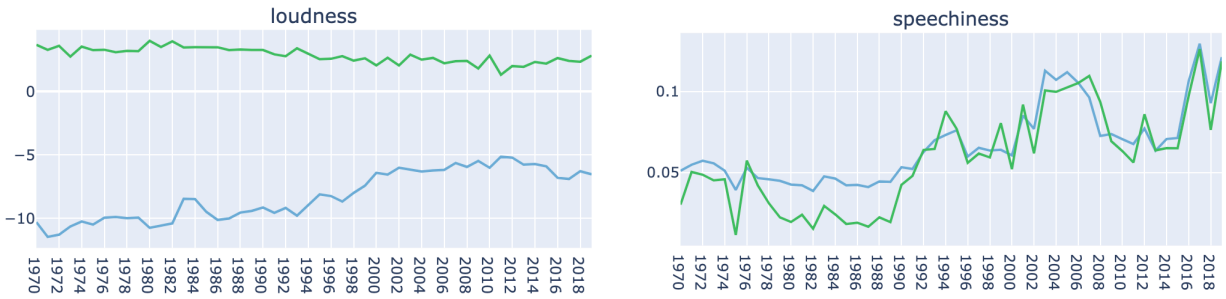

Fig. 2 the values of measure loudness and speechiness in fifty years

3. The indicator 'liveness' is keep decreasing presenting more of the popular music are recording in music studio instead of live house.

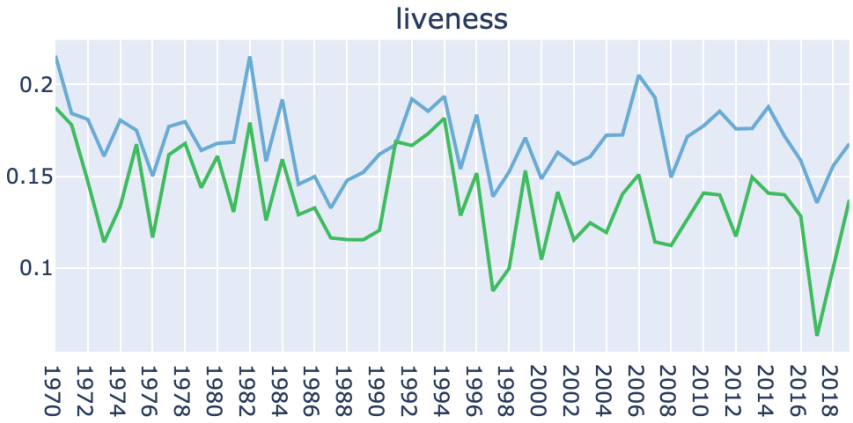

Fig. 3 the values of measure liveness in fifty years

4. The indicator ‘acousticness’ is keep decreasing indicate the songs is most likely to be an acoustic one over time. May because more and more composing tasks are performing on a computer and assisted by intelligent algorithms, so as to meet the requirements of the machines more.

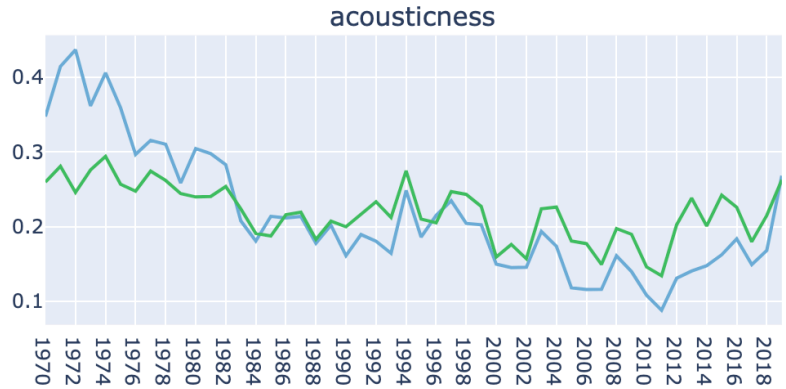

Fig. 4 the values of measure acousticnss in fifty years

5. The measure ‘valence’ is keep decreasing, which shows the emotions expressed by music are keep changing over time, i.e., the positive emotion are gradually transformed to negative.

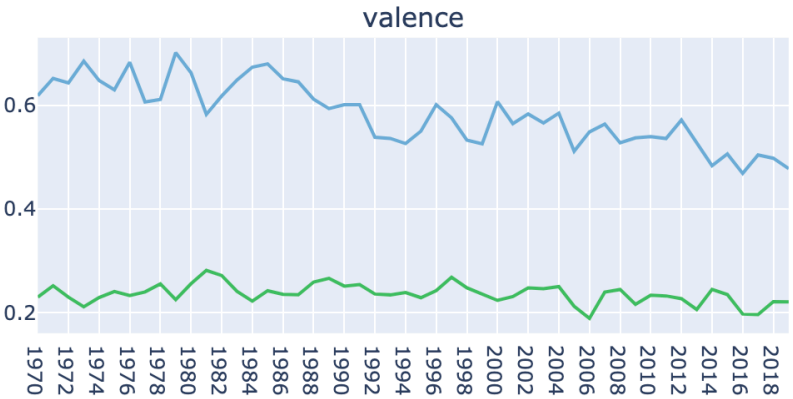

Fig. 5 the value of measure valence in fifty years

In addition to show the change of single measures, six boxplots were used to visualize the features of musics in six periods, which shown in figure 1. There are six findings can be recognized with the

visualization result. We can see the measure ‘instrumentalness’ is very low in all periods.

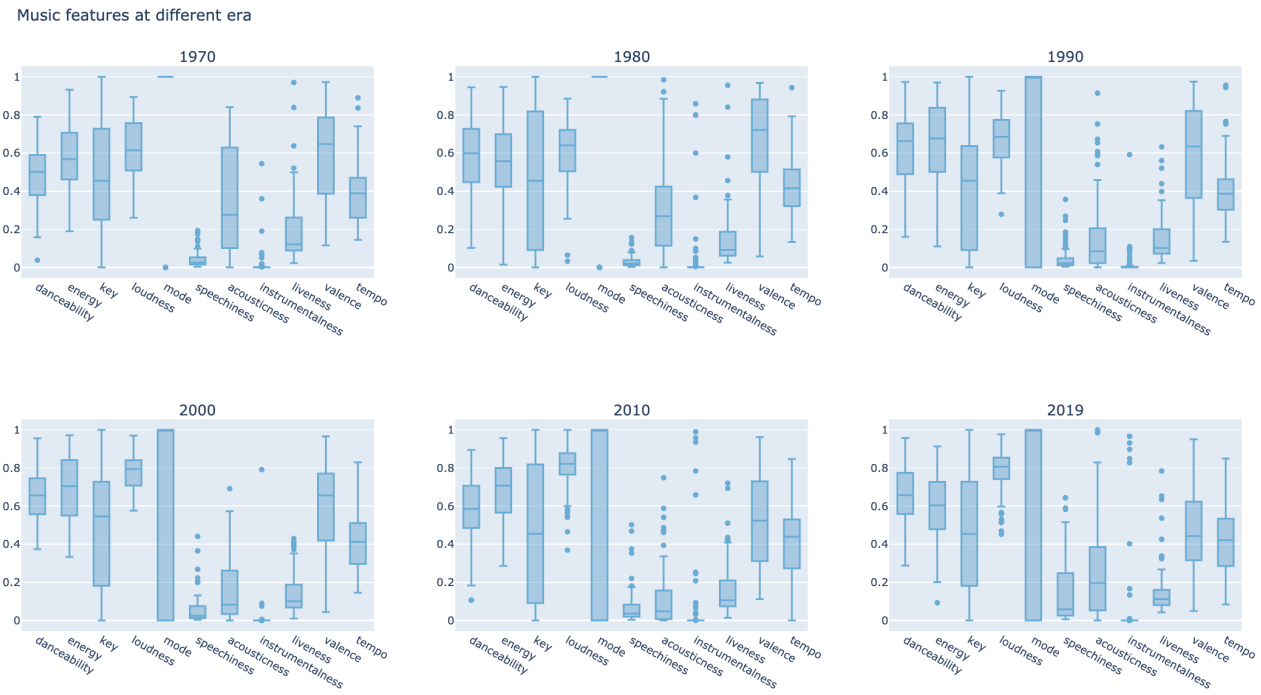

Fig. 6 Boxplots for music features in different periods

4. Discussion

This research intended to answer the research question ‘Have the elements of popular music changed over time in the last fifty years?’. The question was analyzed by using Spotify’s audio features taking an attribute based approach to a success exploration for 4000 musics in the last fifty years on Spotify. The emotions expressed in popular music have also changed year by year, in the early stage (before 2010), popular musics tend to expressing positive emotions while after 2010 they are more about sad tone. Moreover, we also found that over time, people are more prefer louder music, which is more like belong to rock and metal category. According to the value changes of Liveness measure, it is obvious that the purity of music is getting growing, also the live characteristics of popular music is gradually transformed into music studio style. On the one hand, the transformation shows that the audio processing technology is constantly improving, and on the other hand, it also displays that the requirements of a singer now is weakened compared to the pervious periods, because pitch and intonation can be corrected after recording via advanced audio processing techniques.

Appendix

Data Collection - Billboard

```
from urllib.request import urlopen as uRequest
from bs4 import BeautifulSoup as soup

url = 'https://www.billboard.com/charts/hot-100'
```

```
# Opening up connection, grabbing the page
uClient = uRequest(url)
page_html = uClient.read()
uClient.close()
page_soup = soup(page_html, "html.parser")
containers = page_soup.select('article[class*=chart]') # *= means contains
filename = 'billboard_hot_100.csv'
f = open(filename, 'w')
headers = 'Song, Artist, Last Week, Peak Position, Weeks on Chart\n'
f.write(headers)
chart_position = 1

for container in containers:

    song_container = container.find('div', {'class': 'chart-row__title'})
    song = song_container.h2.text
    try:
        artist = song_container.a.text.strip()
    except AttributeError:
        artist = song_container.span.text.strip()
    last_week_container = container.find('div', {'class': 'chart-row__last-week'})
    last_week = last_week_container.find('span', {'class': 'chart-row__value'}).text
    peak_position_container = container.find('div', {'class': 'chart-row__top-spot'})
    peak_position = peak_position_container.find('span', {'class': 'chart-row__value'}).text
    weeks_on_chart_container = container.find('div', {'class': 'chart-row__weeks-on-chart'})
    weeks_on_chart = weeks_on_chart_container.find('span', {'class': 'chart-row__value'}).text
    chart_position += 1
    f.write('\"' + song + '\",\"' + artist.replace('Featuring', 'Feat.') + '\",' + last_week + ',' +
peak_position + ',' + weeks_on_chart + '\n')

f.close()
```

Data Collection - Spotify - Track ID Query

```
import pandas as pd
import requests
import json
import time

token = 'BQBXq32jpkXrDO65sYa7ss1lZ1X9Ehc0h4EMDP-
HmTYPDAO6x1IZfGFeNAlfNnr0tVmrYfPkdEzCEKMvhokU4b6v7tCDaMtw10hK9YjaWlRK3
MozAyM_lT4hKoWEXET_43OGjYFnyDWQ6YpD7ryA5TK8v6KI2cTJJ9sUvvPm_wilD0ShOEY
'

headers ={
    'Accept':'application/json',
```

```
    'Content-Type':'application/json',
    'Authorization':'Bearer ' + token,
}

track_list = open('tracks.csv')
final_artist = ''
final_track = ''
final_year = ''

for line in track_list:
    meta = line.split(',')
    final_year = meta[0]
    final_artist = meta[1]
    final_track = meta[2]

track_list.close()

music_list = open('chartdata.csv')
track_list = open('tracks.csv', 'a')

start = False

for line in music_list:

    meta = line.split(';')
    year = meta[0]
    artist = meta[1]
    track = meta[2].replace('\n', '')

    if artist == final_artist and track == final_track and year == final_year:
        start = True

    if start is False:
        continue

    params = (
        ('q', artist + ' ' + track),
        ('type', 'track'),
        ('limit', '1')
    )

    if not (artist == final_artist and track == final_track and year == final_year):
        response = requests.get('https://api.spotify.com/v1/search', headers=headers, params=params)
        data = json.loads(response.text)
        #print(data)
        if 'tracks' in data and 'items' in data['tracks']:
            if len(data['tracks']['items']) > 0:
```

```
            if 'id' in data['tracks']['items'][0]:
                id = data['tracks']['items'][0]['id']
                track_list.write(year + ',' + artist + ',' + track + ',' + id + '\n')
                print(id)
                time.sleep(1)
```

Data Collection - Spotify - Track Features Query

```
import time
import json
import requests

token = 'BQARki_ztRsCcA0lPrFPEB2seGA_JXucVvv1dz4hzJt-
elLIOrZgqHNegcYFSLzexSUUJxrNyq74Uybg99uflbTGXOLRSAlVTGtYuQh4p3j7BYxcQP7buO
xWiJI_Ew_-GPnkDZHDsgjGHfvPpXPSIvd-Dj1bais2b3gU1Ea8j2sgoH_Xh_k'

headers ={
    'Accept':'application/json',
    'Content-Type':'application/json',
    'Authorization':'Bearer ' + token,
}

feature_list = open('features.csv')
final_id = ''

for line in feature_list:
    meta = line.split(',')
    final_id = meta[11].replace('\n','')

feature_list.close()

track_list = open('tracks.csv')
feature_list = open('features.csv', 'a')
counter = 0
stat = dict()
stat['count'] = 0
useless_field = ['uri', 'track_href', 'analysis_url', 'type', 'duration_ms', 'time_signature']
start = False

for line in track_list:

    meta = line.split(',')
    year = meta[0]
    artist = meta[1]
    track = meta[2]
```

```
    track_id = meta[3].replace('\n', '')

    if final_id == '': start = True
    if track_id == final_id: start = True
    if start is False: continue

    response = requests.get('https://api.spotify.com/v1/audio-features/' + track_id, headers=headers)
    print(response.text)
    write_line = ''
    if len(response.text) > 100:
        data = json.loads(response.text)
        for key in data:
            if key not in useless_field:
                write_line += str(data[key])
                write_line += ','
    if track_id != final_id and write_line != '':
        feature_list.write(write_line[0: -1] + '\n')
    time.sleep(1)
```

Data Visualization

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import statistics

tracksFile = open('tracks.csv')

yearIndicator = dict()

for line in tracksFile:
    meta = line.split(',')
    trackID = meta[3].replace('\n','')
    year = meta[0]
    yearIndicator[trackID] = year

featureFile = open('features.csv')
fields = ['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness',
        'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo']

year_bucket = dict()

for line in featureFile:
    meta = line.split(',')
    trackID = meta[11].replace('\n','')
    year = yearIndicator[trackID]
    block = dict()
```

```
    for i, d in enumerate(meta):
        if i != 11:
            block[fields[i]] = float(meta[i])
            if fields[i] == 'tempo':
                block[fields[i]] = (block[fields[i]] - 60) / 60

    if year not in year_bucket:
        year_bucket[year] = []
    year_bucket[year].append(block)


fig = make_subplots(rows=2, cols=3, subplot_titles=("1970", "1980", "1990", "2000", "2010",
"2019"))
mat = dict()

for year in year_bucket:
    data = year_bucket[year]
    mat[year] = dict()
    for field in fields:
        mat[year][field] = []
    for d in data:
        for field in fields:
            mat[year][field].append(d[field])

maxF = dict()
minF = dict()

for year in mat:
    for field in mat[year]:
        maxV = max(mat[year][field])
        minV = min(mat[year][field])
        if field not in maxF or maxV > maxF[field]:
            maxF[field] = maxV
        if field not in minF or minV < minF[field]:
            minF[field] = minV

for year in mat:
    for field in mat[year]:
        maxV = maxF[field]
        minV = minF[field]
        for i in range(len(mat[year][field])):
            mat[year][field][i] = (mat[year][field][i] - minV) / (maxV - minV)

counter = 0

for year in ['1970', '1980', '1990', '2000', '2010', '2019']:
    for field in mat[year]:
```

```
        fig.add_trace(go.Box(
            y=mat[year][field],
            name=field,
            boxpoints='outliers', # only outliers
            marker_color='rgb(107,174,214)',
            line_color='rgb(107,174,214)'
        ),row=int(counter/3) + 1, col=counter%3 + 1)
    counter += 1

fig.update_layout(title_text="Music features at different era", showlegend=False, height=800,
width=1500)
fig.show()


fig = make_subplots(rows=2, cols=3, subplot_titles=("1970", "1980", "1990", "2000", "2010",
"2019"))

# avg and std

fieldDict = dict()

for year in mat:
    for field in mat[year]:
        if field not in fieldDict:
            fieldDict[field] = dict()
        fieldDict[field][year] = mat[year][field]

year_keys = sorted(list(mat.keys()), key = lambda x: int(x))

avgValueDict = dict()

for field in fieldDict:
    for year in year_keys:
        if field not in avgValueDict:
            avgValueDict[field] = dict()
        avgValueDict[field][year] = statistics.mean(fieldDict[field][year])

stdValueDict = dict()

for field in fieldDict:
    for year in year_keys:
        if field not in stdValueDict:
            stdValueDict[field] = dict()
        stdValueDict[field][year] = statistics.stdev(fieldDict[field][year])

fig = make_subplots(rows=6, cols=2, subplot_titles=list(fieldDict.keys()))
```

```
counter = 0
for field in avgValueDict:
    fig.add_trace(go.Scatter(
        y=list(avgValueDict[field].values()),
        x=list(avgValueDict[field].keys()),
        name='AVG',
        line_color='rgb(107,174,214)'
    ),row=int(counter/2) + 1, col=counter%2 + 1)
    fig.add_trace(go.Scatter(
        y=list(stdValueDict[field].values()),
        x=list(stdValueDict[field].keys()),
        name='STDEV',
        line_color='#44BD62'
    ),row=int(counter/2) + 1, col=counter%2 + 1)
    counter += 1

fig.update_layout(title_text="Music features at different era", showlegend=True, height=2000,
width=1200)
fig.show()
```