**Adam Curry 300-word essay**

The data utilized for the bar, line, and pie charts is from the artist Alcest and is an average of song metrics by album. The data used for the scatter plot is at the song level and is not averaged. Metrics are defined and derived by Spotify.

The pie chart displays the percentage of each album’s position on the pie chart. It doesn't offer any real context to what the data means. For example, is the chart indicating that Kodama has 16.3% beats per minute? The data here is better represented in its raw form. This is the least productive chart to represent this data.

The bar chart displays the data in a much more appealing way. However, the visual seems to indicate a ranking rather than a disparity over time. Bar charts are great for displaying stack rankings, but with this dataset, it appears to be a performance-based measure rather than progression over time. This is the second least productive chart to represent this data.

The line graph displays some interesting findings about the progression of the musical styles of Alcest over the years and is the preferred measurement for this metric. It shows the progression of slower paced (lower beats per minute) on their early albums, to a sharp spike in beats per minute on their third release, and then falls back under an average of 120 beats per minute for their latter two releases. This is the second most productive chart to represent this data.

Finally, a fourth chart was displayed to visualize the correlation between variables. This chart shows the valence, which “measures musical positiveness conveyed by a track”, compared to the energy, which represents a “perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale.” This dataset uses data at a more granular level (song level) to include more datapoints. There appears to be a linear correlation to the valence (positiveness) of a song and the amount energy in a song. Which makes logical sense as low energy songs would tend to be less dynamic in sound range, lower tempo, and lower perceived loudness. This is the least most productive chart to represent this data.

# File: DSC550 Curry Week3 Assignment.py

# Name: Adam Curry

# Date: 03/24/2020[¶](file:///C:\Users\adamp\AppData\Local\Temp\acurry%20DSC550%20Week3.html#Date:-03/24/2020)

# Course: DSC540 - Data Mining

# Desc: This program classifies text in a variety of ways

# Usage: This program should be used when reviewing week 3's assignment

You can create a new analysis scenario or you can use the tutorials you completed this week.

The below analysis uses a Spotify dataset from Spotify's API

# Begin Data Wrangling process from Spotify

import spotipy

from spotipy.oauth2 import SpotifyClientCredentials

import pandas as pd

import requests

import json

# create a spotify object with the connection

# API keys stored in PATH

spotify = spotipy.Spotify(client\_credentials\_manager=SpotifyClientCredentials())

def artist\_uri(s):

"""

function to to get the artist of interest's albums

"""

artist = s

albums = spotify.artist\_albums(s, album\_type = 'album')

return artist, albums

# alcest = artist\_uri('spotify:artist:5YeoQ1L71cXDMpSpqxOjfH')

# localize the data into lists

alcest\_uri, sonata\_artica\_albums = artist\_uri('0d5ZwMtCer8dQdOPAgWhe7')

list\_album\_data = []

for a in sonata\_artica\_albums['items']:

dict\_album = {

'album\_name' : None,

'album\_uri' : None,

'album\_group' : None,

'album\_type' : None,

'album\_name' : None,

'album\_available\_markets' : None,

'album\_release\_date' : None,

'id\_' : None

}

try:

dict\_album['album\_name'] = (a['name'])

dict\_album['album\_uri'] = (a['uri'])

dict\_album['album\_group'] = (a['album\_group'])

dict\_album['album\_type'] = (a['album\_type'])

dict\_album['album\_available\_markets'] = (a['available\_markets'])

dict\_album['album\_release\_date'] = (a['release\_date'])

dict\_album['id\_'] = (a['id'])

list\_album\_data.append(dict\_album)

except Exception:

pass

list\_album\_specific\_data = []

for uri in list\_album\_data:

z\_response = requests.get('https://t4ils.dev:4433/api/beta/albumPlayCount?albumid={}'.format(uri['id\_']))

list\_album\_specific\_data.append(json.loads(z\_response.text))

"""

get the count of track plays into a final list

"""

list\_track\_count = []

for i, track in enumerate(list\_album\_specific\_data):

try:

for t in track['data']:

dict\_track\_count = {

'track\_uri' : None,

'name' : None,

'playcount' : None

}

dict\_track\_count['track\_uri'] = t['uri']

dict\_track\_count['name'] = t['name']

dict\_track\_count['playcount'] = t['playcount']

list\_track\_count.append(dict\_track\_count)

except Exception as e:

print(e)

pass

list\_album\_tracks = []

for track in list\_album\_data:

dict\_album\_tracks = {

'album\_name' : None,

'album\_data' : None,

'album\_uri' : None,

}

try:

dict\_album\_tracks['album\_name'] = (track['album\_name'])

dict\_album\_tracks['album\_uri'] = (track['album\_uri'])

dict\_album\_tracks['album\_data'] = (spotify.album\_tracks(track['album\_uri']))

list\_album\_tracks.append(dict\_album\_tracks)

except Exception:

pass

"""

from the population of the playlist tracks, get the track featers from spotify

"""

list\_audio\_features\_ = []

for i, track in enumerate(list\_album\_tracks):

try:

for t in track['album\_data']['items']:

dict\_audio = {

'album\_name' : None,

'album\_uri' : None,

'audio\_features' : None

}

dict\_audio['audio\_features'] = (spotify.audio\_features(t['uri']))

dict\_audio['album\_uri'] = track['album\_uri']

dict\_audio['album\_name'] = track['album\_name']

list\_audio\_features\_.append(dict\_audio)

except Exception as e:

print(e)

pass

"""

For loop will append each track's audio features to a dict

"""

list\_track\_data = []

for i, t in enumerate(list\_audio\_features\_):

dict\_track\_ = {

'track\_uri' : None,

'acousticness' : None,

'danceability' : None,

'energy' : None,

'instrumentalness' : None,

'key' : None,

'liveness' : None,

'loudness' : None,

'speechiness' : None,

'tempo' : None,

'time\_signature' : None,

'valence' : None,

'album\_uri': None

}

try:

dict\_track\_['track\_uri'] = (t['audio\_features'][0]['uri'])

dict\_track\_['acousticness'] = (t['audio\_features'][0]['acousticness'])

dict\_track\_['danceability'] = (t['audio\_features'][0]['danceability'])

dict\_track\_['energy'] = (t['audio\_features'][0]['energy'])

dict\_track\_['instrumentalness'] = (t['audio\_features'][0]['instrumentalness'])

dict\_track\_['key'] = (t['audio\_features'][0]['key'])

dict\_track\_['liveness'] = (t['audio\_features'][0]['liveness'])

dict\_track\_['loudness'] = (t['audio\_features'][0]['loudness'])

dict\_track\_['speechiness'] = (t['audio\_features'][0]['speechiness'])

dict\_track\_['tempo'] = (t['audio\_features'][0]['tempo'])

dict\_track\_['time\_signature'] = (t['audio\_features'][0]['time\_signature'])

dict\_track\_['valence'] = (t['audio\_features'][0]['valence'])

dict\_track\_['album\_uri'] = (t['album\_uri'])

list\_track\_data.append(dict\_track\_)

except Exception as e:

print(e)

# this function will get the audio analysis, more song related data

#audio\_analysis\_ = spotify.audio\_analysis(Himmelsrand)

df\_track\_data = pd.DataFrame(list\_track\_data)

df\_album\_data = pd.DataFrame(list\_album\_data)

df\_track\_count = pd.DataFrame(list\_track\_count)

# join the two datasets together

df\_final = df\_track\_data.merge(df\_album\_data,how='inner',on=['album\_uri','album\_uri'])\

.merge(df\_track\_count,how='inner',on=['track\_uri','track\_uri'])

# drop Japan releases

indexNames = df\_final[df\_final['album\_uri'] == 'spotify:album:0yB03KUiegLE7N8Gf8Q3GB'].index

df\_final.drop(indexNames , inplace=True)

# drop dups

df\_final.drop\_duplicates(subset ="name",

keep = False, inplace = True)

A. Display the same analysis using 3 different charts (ex. A bar chart, a line chart and a pie chart)

B. Use appropriate, complete, professional labeling

import matplotlib.pyplot as plt

import numpy as np

df\_final['album\_release\_year'] = pd.to\_datetime(df\_final['album\_release\_date'])

df\_final['album\_release\_year'] = df\_final['album\_release\_year'].dt.year

# average each value by album

df\_avg = df\_final.groupby('album\_name').apply(lambda x: x.mean())

# move index back to column var

df\_avg = df\_avg.reset\_index()

# convert the year values to integers

df\_avg['album\_release\_year'] = df\_avg['album\_release\_year'].apply(lambda x: int(x))

# sort the albums by release year

df\_avg = df\_avg.sort\_values(['album\_release\_year'])

# create variables to be plotted

album\_release\_year = np.array(df\_avg['album\_release\_year'])

album\_name = df\_avg['album\_name']

tempo = np.array(df\_avg['tempo'])

"""

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.

"""

'\nThe overall estimated tempo of a track in beats per minute (BPM). \nIn musical terminology, tempo is the speed or pace of a given piece and \nderives directly from the average beat duration.\t\n'

### The charts below are ranked form worst use case to best, in order to answer the below question:

C. Rank your charts from most effective to least effective.

# pie chart

plt.style.use('fivethirtyeight')

plt.figure(figsize=(10,5))

plt.pie(tempo, labels = album\_name,

wedgeprops ={'edgecolor':'black'}

,shadow=True

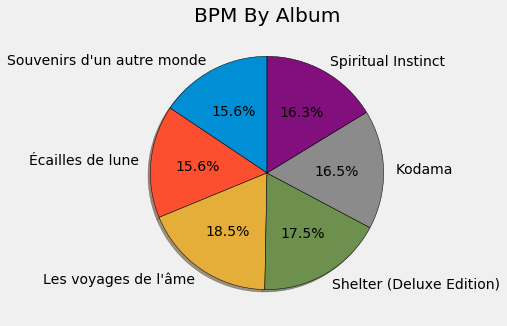
,autopct='%1.1f%%'

# angle the data a touch

,startangle=90)

plt.title('BPM By Album')

Text(0.5, 1.0, 'BPM By Album')



# X bar chart

# use numpy to create album indexes for smoother bar chart

temp\_indx = np.arange(len(album\_release\_year))

plt.style.use('fivethirtyeight')

plt.figure(figsize=(10,5))

plt.bar(temp\_indx, tempo,color='#333444',label = 'tempo')

# sort by album name but include the album title as a label

plt.xticks(temp\_indx, album\_name, rotation=45)

plt.ylabel('Beats Per Min')

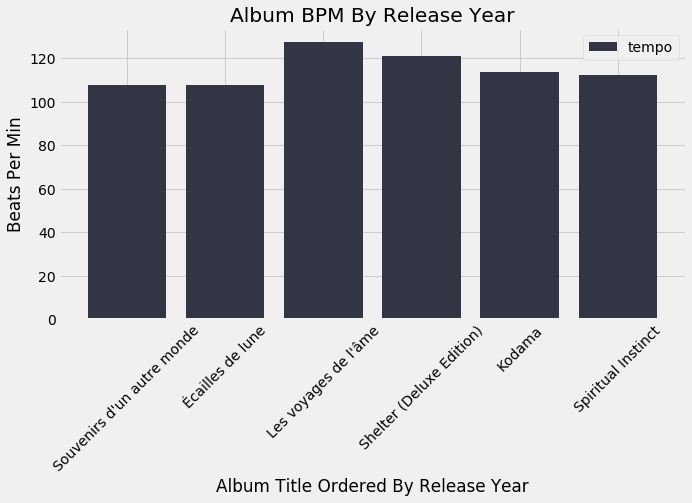
plt.xlabel('Album Title Ordered By Release Year')

plt.title('Album BPM By Release Year')

plt.grid(True)

plt.legend()

<matplotlib.legend.Legend at 0x261b87fea58>



# line chart

# print(plt.style.available)

# utilize predefined styles for final output

plt.style.use('fivethirtyeight')

plt.figure(figsize=(10,5))

plt.plot(album\_release\_year, tempo,marker='o',color='#333444',linestyle='--'\

,label = 'tempo')

# sort by album name but include the album title as a label

plt.xticks(album\_release\_year, album\_name, rotation=45)

plt.ylabel('Beats Per Min')

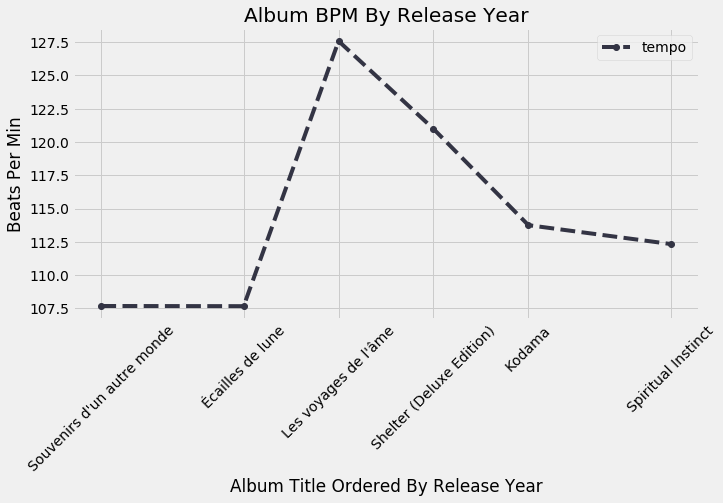
plt.xlabel('Album Title Ordered By Release Year')

plt.title('Album BPM By Release Year')

plt.grid(True)

plt.legend()

<matplotlib.legend.Legend at 0x261b88835f8>



# scatter plot

album\_release\_year = np.array(df\_final['album\_release\_year'])

album\_name = df\_final['album\_name']

y = np.array(df\_final['valence'])

x = np.array(df\_final['energy'])

size = np.array(df\_final['tempo'])

plt.style.use('fivethirtyeight')

plt.figure(figsize=(10,5))

#s = dot size using tempo as a var

# alpha softens the colors

# c differentiates between the albums

plt.scatter(x, y, s=size, c=album\_release\_year, cmap='Greens',

edgecolor='black', linewidth=1,alpha=0.75)

plt.xlabel('Energy')

plt.ylabel('Valence')

# rescale both variables to be on the same scale

plt.xscale('log')

plt.yscale('log')

plt.title('Does higher energy correlate with higher valence?')

# show the bar on thr right to display the what the colors are

cbar = plt.colorbar()

cbar.set\_label('Album Release Year')

plt.show()

