Scott Fullenbaum
Naomi Meininger
Grace Hindermann

1 Abstract and Statement of Problem

From pop to theater, music has influenced listeners and artists alike. Its cultural and emotional impact can be seen throughout all genres. An artist's music is shaped by multiple factors, such as social and political controversies, access to the proper tools such as instruments and recording studies, personal lived experiences, and other artists' lives and music. This influence and impact can be modeled by focusing on the similarities of different songs. In our model, we expect to be able to quantify the evolution of music, along with showing how a multitude of influences have shaped music history.

We approached the problem by using three different ways to evaluate and model the influence of music over time: a directed graph among artists, least squares regression looking at data vs time, and Neural Networks classifying music by genre. Each method is detailed in its own section below.

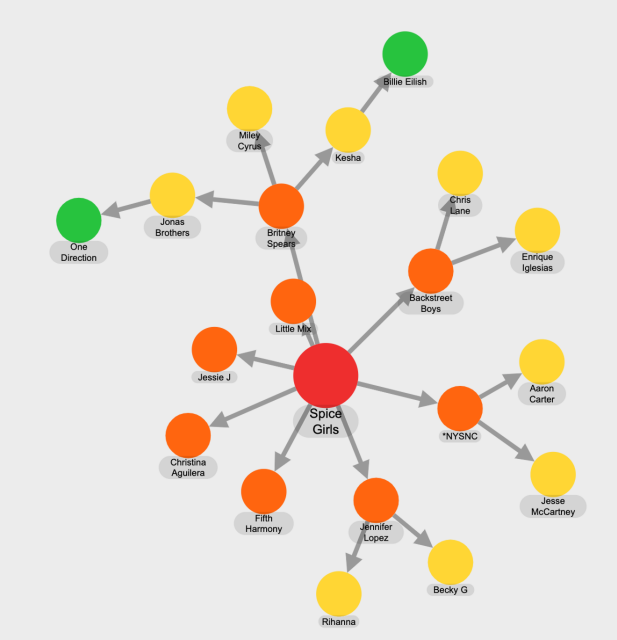
2 Directed Graph

In the pop/rock genre, we noticed you could develop a subnetwork of artists/followers as depicted by the directed graph below, with an outgoing directed edge meaning that an artist influence another. We chose to define similarity among artists through measurements of danceability, energy, valence, tempo, loudness, acousticness, instrumentalness, liveliness, and speechiness. These measurements were all gathered from the provided data set, and we chose to exclude mode and key. We chose those two specifically as the vast majority of songs are written in the same mode, and the key of a song seems arbitrary considering a song can be transposed easily. Inspiration for this method was also drawn from our understanding of how the Spotify algorithm works (source listed below), as we want to draw connections between artist and songs and Spotify is something we all use.

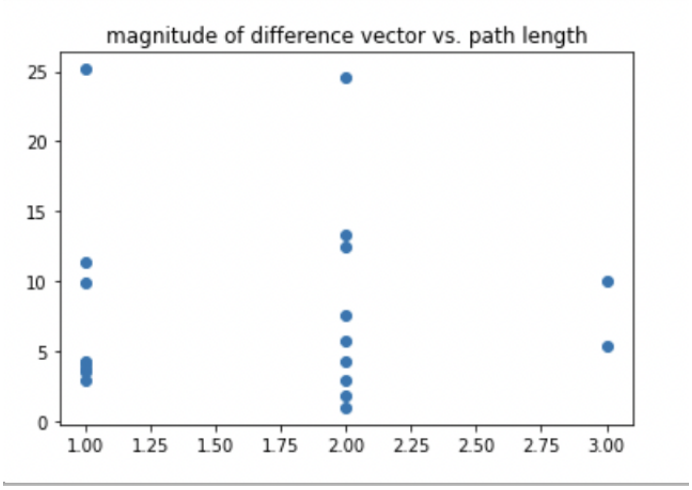
Considering the data set is extremely large, creating an accurate and comprehensive directed graph between every artist and follower would be overwhelming, along with being incredibly difficult to analyze and determine any trends. Even within genres, such as the pop/rock genre, tracing influences between 2800 artists is convoluted. As a result, we determined it would be easier to focus on a subnetwork among a single artists. In our chosen subnetwork, we are comparing the follower's data to that of the Spice Girls. The node corresponding to the Spice Girls is in the red and acts as the root of the network. Every artist on it can be connected to Spice Girls in some way. Any artist that was directly influenced by the Spice Girls is in orange. For each additional node in a path linking the Spice Girls to another artist, the color becomes further from red, with being two edges from the Spice Girls being yellow, and three edges from the Spice Girls being represented by green. In this specific subgraph, we can see that the Spice Girls influenced Britney Spears, Little Mix, Backstreet Boys, Christina Aguilera, Fifth Harmony, Jennifer Lopez, and *NSYNC. Indirect followers include artists such as the Jonas Brothers, Rihanna, and One Direction. It is important to note that this data doesn't represent all influences for each artist listed.

Since we can create a vector for each artist where each element corresponds to a specific element, we thought one way to represent differences between any two artists on a given network would be to create a vector that was the difference between any two artists, then take the magnitude of that vector. This means that the resultant vector from two identical artists would be zero, with the magnitude increasing as artists became further and further apart. We looked to determine if there was a significant difference between artists who were followers of the Spice Girls, or if there was as much influence on others that are indirectly influenced by the Spice Girls (in other words, having a path length of more than 1). We plotted the magnitudes of the difference vectors and immediately saw that there was very little correlation between similarity and path length, as there were followers of the Spice Girls that could be just as different as an artist that is two nodes away. This makes sense because many artists do not have a single inspiration, and many artists take inspiration from other artists who do not fit in the same genre. Our data is fairly limited, so we are not analyzing every single influence on the Spice Girls or their followers, and we are only focusing on a subnetwork of the pop/rock genre. It is possible that many of the Spice Girls' followers draw influences from other genres. A genre such as country could have very different tempo or acousticness, and this could influence specific characteristics of an artist and therefore skew our "difference" vector. Furthermore, the data provided does not measure the magnitude of influence one artist had on another, nor is it fully explained

where that data comes from. For example, one artist could have been heavily inspired for the Spice Girls for a single song, while another could've had their musical identity based around the Spice Girls. If we had been given information about magnitude of influence, we would have developed a weighted directed graph to help model this, and we suspect a trend relating to influence and distance from Spice Girls would become more apparent.



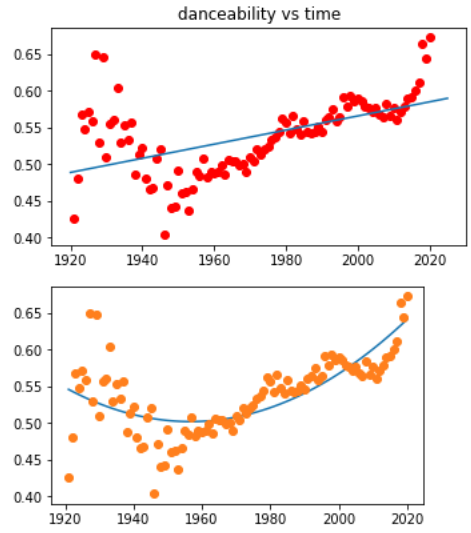
Here is a plot showing the difference:



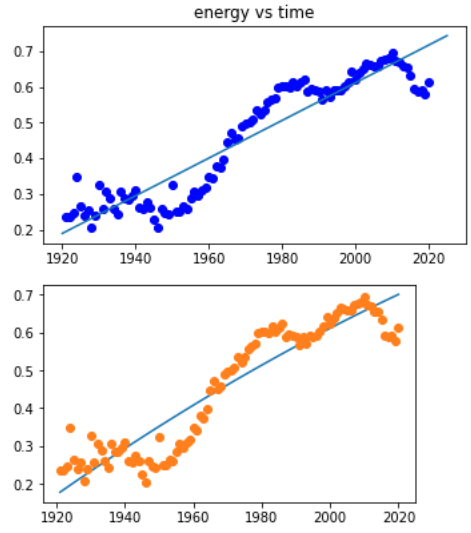
3 Least Squares Regression

Least squares regression is a helpful tool for determining trends in data. We can use least squares regression to determine these numerical components of music have changed over time. The MCM problem provided us with a data set consisting of thirteen musical characteristics, with the data itself ranging from 1921 to 2020. More specifically, we chose to generate two plots, one that showcased the linear least squares regression equation, and one that showed the quadratic least squares regression equation. The code to generate each of these plots is generated under section 5.

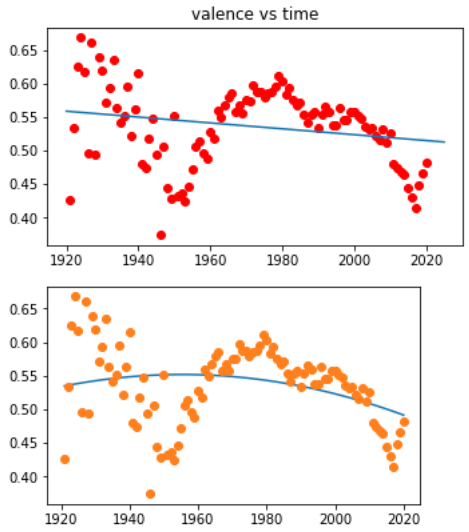
```
danceability vs time equation:
L(x) = 0.000962*x + -1.36
Q(x) = 3.46e-05*x^2 + -0.135*x + 1.33e+02
```



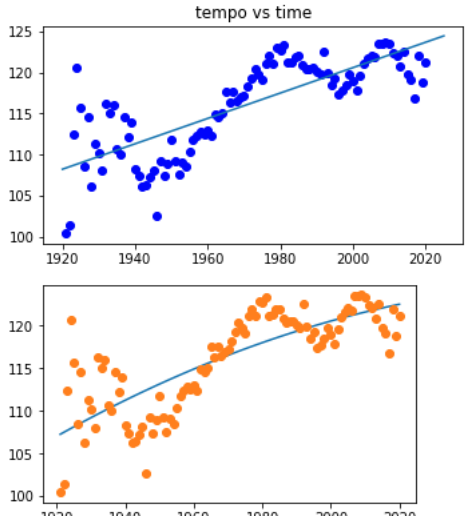

```
energy vs time equation:
L(x) = 0.00527*x + -9.94
Q(x) = -1.02e-05*x^2 + 0.0457*x + -49.7
```



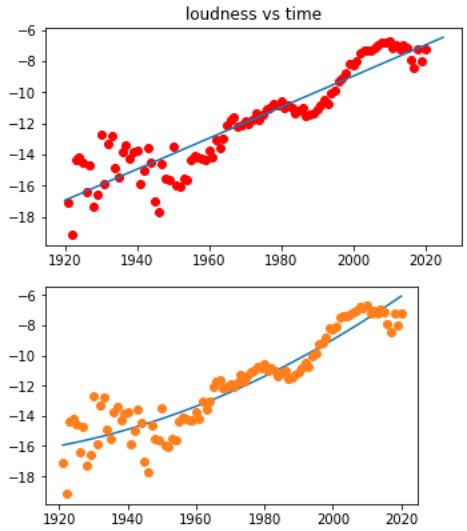

```
valence vs time equation:
L(x) = -0.000438*x + 1.4
Q(x) = -1.45e-05*x^2 + 0.0566*x + -54.8
```



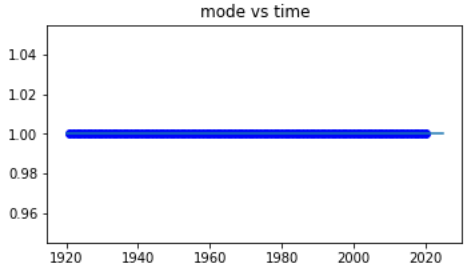

```
tempo vs time equation:
L(x) = 0.154*x + -1.88e+02
Q(x) = -0.000704*x^2 + 2.93*x + -2.92e+03
```

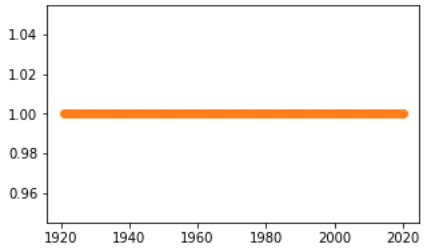


```
loudness vs time equation:
L(x) = 0.0997*x + -2.08e+02
Q(x) = 0.000563*x^2 + -2.12*x + 1.98e+03
```

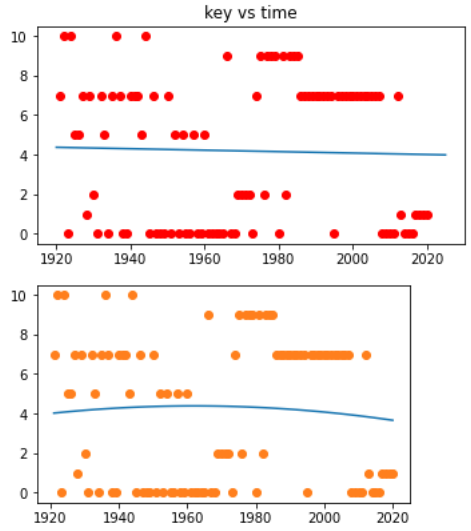



```
mode vs time equation:
L(x) = -9.39e-22*x + 1.0
Q(x) = 9.36e-20*x^2 + -3.71e-16*x + 1.0
```

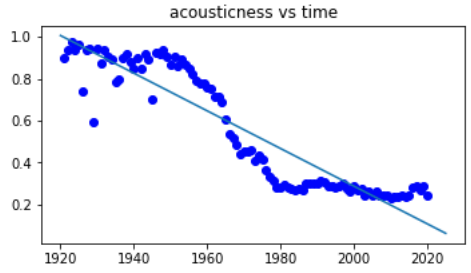


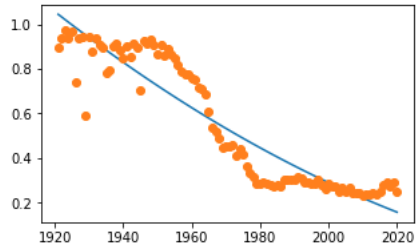


```
key vs time equation:
L(x) = -0.00365*x + 11.4
Q(x) = -0.000216*x^2 + 0.848*x + -8.27e+02
```

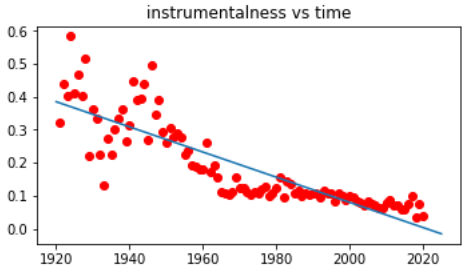


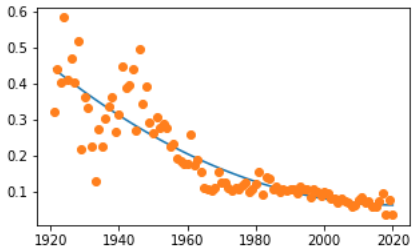

```
acousticness vs time equation:
L(x) = -0.00897*x + 18.2
Q(x) = 2.96e-05*x^2 + -0.126*x + 1.33e+02
```



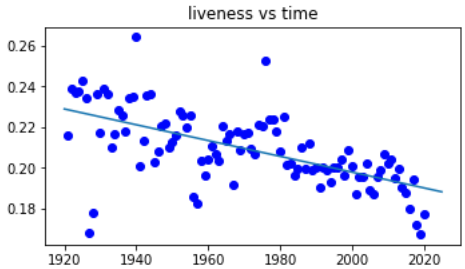


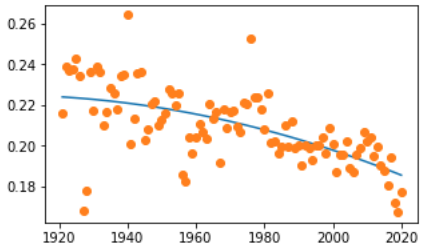
```
instrumentalness vs time equation:
L(x) = -0.00381*x + 7.71
Q(x) = 3.71e-05*x^2 + -0.15*x + 1.52e+02
```



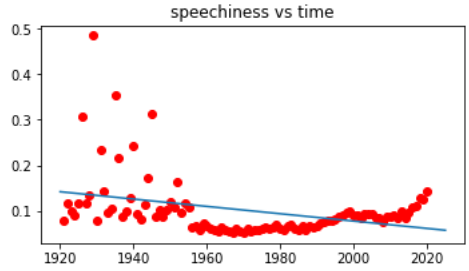


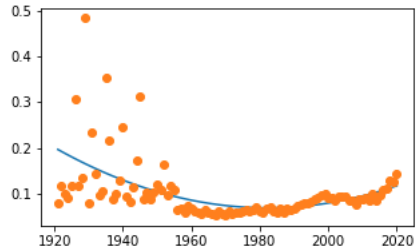
```
liveness vs time equation:
L(x) = -0.000388*x + 0.973
Q(x) = -2.86e-06*x^2 + 0.0109*x + -10.1
```



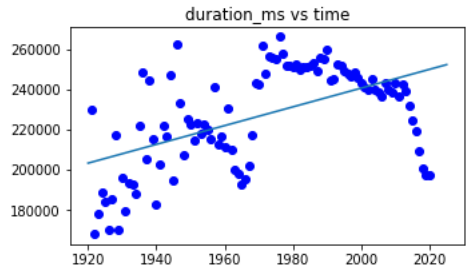


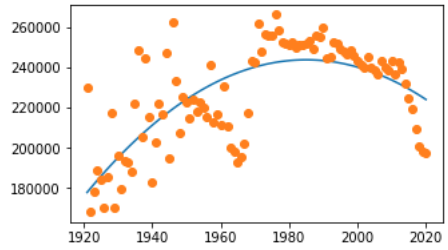
```
speechiness vs time equation:
L(x) = -0.000805*x + 1.69
Q(x) = 3.41e-05*x^2 + -0.135*x + 1.34e+02
```

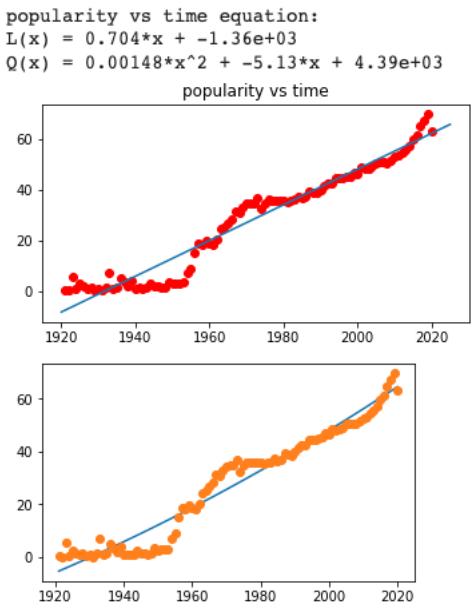




```
duration_ms vs time equation:
L(x) = 4.67e+02*x + -6.94e+05
Q(x) = -16.1*x^2 + 6.4e+04*x + -6.33e+07
```







All of the graphs exhibit interesting trends. Though many showed upward trends, many lacked a direct or obvious linear relationship. Instead, many data sets appeared to oscillate upwards, with some examples beyond energy, tempo, and loudness. The trend here makes sense since music as a whole fluctuates and changes over time, especially as different genres and bands of different influence emerge. For example, the music of the 1940s does not sound the same as the music of the early 2000s.

We can visually see that most of the data does not follow a linear trend. However, utilizing a quadratic least squares regression appears to provide a better curve of best fit to some of the data, with it specifically providing a strong curve when looking at danceability, duration, and instrumentalness appearing to have stronger quadratic rather than linear fits. While we did choose to model the data with linear and quadratic fit, some of the data does not fit these trends, either showing other types of fit or not showing much of a pattern at all.

Additionally, we can see that every characteristic except for key/mode appear to decrease around the 1930s/1940s. From our personal musical experience, when characteristics such as energy, danceability, and tempo increase, that indicates a happier and more upbeat song. Though this may be independent of global conditions, we believe this general trend could be connected to the Great Depression and the WWII, as due to economic issues then the war effort, people may have been pushed away from music in general, and due to the depressing nature of the times, motivated to make less "joyous music".

The characteristics that had the most interesting trends were key and mode because mode remained constant and key exhibited no pattern. The result regarding mode make sense because most music consists of the same few chords and scales, which is the definition of mode. The data about key makes sense as a musical key is dependent on each artist and song, and no song/genre is beholden to a single key.

4 Neural Networks

We wanted to use a Neural Network to attempt to see if we could classify each genre based off the data provided about each artist. All code used for this is listed under section 6.

We decided to use the following genres for classifications: Latin, Electronic, RB, Religious, Blues, Stage Screen, Jazz, Vocal, Classical, Reggae, International, Folk music, and Pop/Rock. To create a classification for a genre, we chose to only include genres that had around at a minimum, 30 artists considered a part of that genre. We chose this as the cutoff because we believed it would be difficult to develop an accurate model on any small data set. Furthermore, the data containing the list of all artists was all the data provided, meaning that it would be our training and testing data set. As a result, we needed to make sure that we had enough artists to fall under both

categories.
One very important thing to note is the subsystem of classifications within the Pop/Rock genre. In our data, the Pop/Rock genre contains about 2800 artists, which is more than half of our entire data set. Furthermore, it appears the creators of the data set used Pop/Rock as a general term, with artists a part of the genre spanning 60-70 years. This means artists like the Beatles and Rihanna are considered a part of the same genre, which though it might be technically correct, in reality, they are incredibly different artists. We thought classifying all of Pop/Rock as a single group would mess up our classification system, so we decided to break Pop/Rock, treating each decade as it's own genre. We chose to get the decade from the decade started, which is explained more below. We also applied the same 30+ artist rule for each decade to get the data set.
Based on the restrictions explained above, we ended up with a total of 19 classifications. To make our model and determine the layers, we had to make a couple of key assumptions about our data. However, that does mean these restrictions necessarily hold true in reality, when they often don't.

First: Involvement in each genre is mutually exclusive. What that means is that each artist can be mapped to a single genre, as in an artist can be in Electric or Latin, not both. This is important because otherwise, you would have to weight involvement with different genres, which isn't something easy to do, especially with musical genres, when we don't have a numerical basis for separating them, as that is our goal. This also transforms our model into a multiclass classification, not multilabel classification.

Second: Artists remain constant throughout their careers. In numerical terms, this means their songs don't numerically change over time. We aren't provided with a formula as to how the data for each artist is calculated, so it's reasonable to assume that those amounts will not drastically change for each new song they make.

Third and lastly: For Pop/Rock, because we classify Pop/Rock by decade, we have to assume each artist can best be represented in a single decade. Since we are only provided with the starting decade, we choose to assign each Pop/Rock artist to that. The main idea behind this assumption is to prevent overlap among different classifications and to also keep the model as multiclass classification rather than multilabel. We think this is reasonable as we generally think of music in decades, so we can map artists to when they start, as their sound is also less likely to change over time.
For our specific model, we chose to use the softmax activation function, because as mentioned in our earlier assumptions, our model is multiclass classification, and the softmax activation function is best for those rather than relu. In practice, we also found that when compared to the relu activation function, softmax produced significantly lower levels of loss, which we want since loss essentially measures the difference between the expected outcome and the outcome from the model, then our model is perfect when loss is zero, so we want loss as low as possible. Across five trials conducted with each the softmax and relu activation functions, we found the average loss be 1.3003 when using softmax, and the average loss across 5 trials when using relu was 525.29.
A little more on our model: we chose to add 20 dense layers because we have 19 classifications. The rest of the model was made with a little bit of experimenting with different layers and levels of dropout. The full model can be seen under the code.
Lastly, before showing results, we split the data into testing and training data sets using the sklearn function train_test_split, which splits the data for us, and makes the training set larger than the test set.
Once the network was created, we performed three trials with 5 epochs for fitting the model and obtained the following results:

```
Epoch 1/5
119/119 [==============================] - 1s 1ms/step - loss: 2.7375 - accuracy: 0.7387
Epoch 2/5
119/119 [==============================] - 0s 1ms/step - loss: 2.3340 - accuracy: 0.9562
Epoch 3/5
119/119 [==============================] - 0s 1ms/step - loss: 1.9500 - accuracy: 0.9562
Epoch 4/5
119/119 [==============================] - 0s 1ms/step - loss: 1.6342 - accuracy: 0.9562
Epoch 5/5
119/119 [==============================] - 0s 1ms/step - loss: 1.3664 - accuracy: 0.9562

Testing data results:
40/40 - 0s - loss: 1.2159 - accuracy: 0.9494 - 141ms/epoch - 4ms/step
```

```
Epoch 1/5
119/119 [==============================] - 1s 1ms/step - loss: 2.8008 - accuracy: 0.5339
Epoch 2/5
119/119 [==============================] - 0s 1ms/step - loss: 2.4002 - accuracy: 0.9541
Epoch 3/5
119/119 [==============================] - 0s 1ms/step - loss: 2.0258 - accuracy: 0.9541
Epoch 4/5
119/119 [==============================] - 0s 1ms/step - loss: 1.6931 - accuracy: 0.9541
Epoch 5/5
119/119 [==============================] - 0s 1ms/step - loss: 1.4172 - accuracy: 0.9541

Testing data results:
40/40 - 0s - loss: 1.2508 - accuracy: 0.9557 - 146ms/epoch - 4ms/step
```

```
Epoch 1/5
119/119 [==============================] - 1s 1ms/step - loss: 2.7936 - accuracy: 0.6978
Epoch 2/5
119/119 [==============================] - 0s 1ms/step - loss: 2.3864 - accuracy: 0.9530
Epoch 3/5
119/119 [==============================] - 0s 1ms/step - loss: 2.0164 - accuracy: 0.9530
Epoch 4/5
119/119 [==============================] - 0s 1ms/step - loss: 1.6941 - accuracy: 0.9530
Epoch 5/5
119/119 [==============================] - 0s 1ms/step - loss: 1.4007 - accuracy: 0.9530

Testing data results:
40/40 - 0s - loss: 1.2429 - accuracy: 0.9589 - 143ms/epoch - 4ms/step
```

Across the three trials, we were able to achieve average loss of around 1.237 and were able to accurately classify genres for each song 95.45% of the time.

5 Applications of Research

By studying the influence of music over time, we are better able to appreciate the music we listen to on a daily basis and understand the vast and complicated connections between different styles and genres of music.
Furthermore, by understanding the connections between different genres of music, and different artists, we are able to help recommend new artists to listeners, like the Spotify algorithm or given different data, could be developed to recommend new songs, with this website, (not made by us), being a practical example of similar types of this research in action. website. A more specific example, based on earlier data would be that an individual who enjoys the Spice Girls might enjoy Jessie J, who were influenced by the Spice Girls, and given our research, we could develop a larger network, where given some artists someone likes, find other artists influenced by those that they should enjoy. On the machine learning end, the same thought process applies, but to genres rather than individual artists.

6 Contributions for Each Group Member

Grace: Grace made the directed graph and analyzed it. She also made a plot of the path sizes and similarities in the directed graph.

Scott: Scott coded the neural network to classify artists by genre.

Naomi: Naomi created the least squares regression code and modularized it to run for all items in the data_by_year.csv file.

7 Code (Directed Graph Plot)

```
import csv
import numpy as np
import numpy.linalg as la
import math
from matplotlib import pyplot as plt
import numpy.linalg as la
```

```
data = np.array([
    [1, 9.87421910224224],
    [1, 2.97887342934977],
    [1, 4.19250304037673],
    [1, 11.3669604942459],
    [1, 4.23518228162527],
    [1, 25.1585864867257],
    [1, 3.77273955158647],
    [1, 3.53412211547925],
    [2, 2.95699174959021],
    [2, 12.4401294923118],
    [2, 4.24638502614565],
    [2, 5.7286764620538],
    [2, 0.971340366540149],
    [2, 24.5188092794298],
    [2, 7.55888698712534],
    [2, 13.3518812899381],
    [2, 1.75827186517958],
    [3, 5.38629361550489],
    [3, 10.0559432807065]

])
x, y = data.T
title_of_graph = 'magnitude of difference vector vs. path length'
plt.title(title_of_graph)
plt.xlabel = 'length of path from Spice Girls'
plt.ylabel = 'magnitude difference vector'
plt.scatter(x,y)
plt.show()
```

8 Code (LSR)

1. Linear Squares Regression Code

   (a)
```
import csv
import numpy as np
import numpy.linalg as la
import math
from matplotlib import pyplot as plt
import numpy.linalg as la

def parse_csv(filename, column):
    data = np.array([])
    line_count = 0
    with open(filename) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        for row in csv_reader:
            if line_count == 0:
                name = (row[column])
                line_count += 1
            else:
                line_count += 1
                data = np.append(data, float(row[column]))
    needed = [name, data]
    return needed
```

```
def year_gen(a, b):
    years = np.array([])
    for i in range (a, b):
        years = np.append(years, i)
    return years

def lin_fit(x,y):
    # solve the least squares problem
    A = np.array([[xx,1] for xx in x])
    ls=la.lstsq(A,y,rcond=None)

    # get the coefficients and report them
    alpha,beta = ls[0]
    print(f"L(x) = {alpha:.03}*x + {beta:.03}")
    res = np.array([alpha, beta])
    # return the linear function determined by these coefficients
    return res

def quad_fit(x,y):
    # solve the least squares problem
    A = [[xx**2,xx,1] for xx in x]
    res=la.lstsq(A,y,rcond=None)

    # extract & report the coefficients
    alpha,beta,gamma=res[0]
    print(f"Q(x) = {alpha:.03}*x^2 + {beta:.03}*x + {gamma:.03}")
    # return the quadratic function determined by these coefficients
    return lambda x:alpha*x**2 + beta*x + gamma

def plot_curve_fit(x0,f,x,y):
    # graph the line with slope alpha and y-intercept beta, and plot the data points
    fig, ax = plt.subplots(figsize=(5, 3))
    #ax.plot(x,alpha1*x + beta1)
    plt.plot(x0,f(x0))
    plt.plot(x,y,'o')
    return fig,ax

def plot_data(x, y, name, color):
    fig, ax = plt.subplots(figsize=(5, 3))
    lin = np.linspace(1920, 2025, 1000)
    title_of_graph = name + ' vs time'
    plt.title(title_of_graph)
    plt.plot(x, y, color)
    print(name, 'vs time equation:')
    coeff = lin_fit(x,y)
    plt.plot(lin, coeff[1] + coeff[0]*lin)

    equation = quad_fit(x,y)
    plot_curve_fit(x, equation, x, y)
    fig.tight_layout(pad=1.0)
    #plots curve_fit
    plt.show()

years = year_gen(1921, 2021)
for i in range(1, 14) :
```

```
if (i % 2 == 0) :
  color = 'bo'
else :
  color = 'ro'
data = parse_csv('data_by_year.csv', i)
plot_data(years, data[1], data[0], color)
```

(b) Since we are modeling the data over time, we first call the year_gen function in order to attain a time period to model our data over. Then, for each characteristic, we call the parse_csv function in order to parse the data for the characteristic stored at a specific index in the array. For instance, danceability is stored at index 1, so for that iteration of the for loop, $i = 1$, data[0] is 'danceability' and data[1] is all the data in the danceability column. Additionally, since i is odd, the plot will be red. Next, we call the plot data function to physically plot the relationship between time and the specific characteristic in question. We set up a 5 by 3 graph for each characteristic, find a line of best fit using the least squares regression function (which performs residual on the data as well), and then plot the data accordingly. We also plotted a quadratic fit equation as well.

9 Code Neural Network

9.1 README

Before the code is shown, there is a little to know about files. Two of the three pieces of code we've attached prepare the files for the Neural Network, with the third creating it. In addition to the start files, there are a couple of files you need to create. Below are instructions on how to recreate them, but they've also been embedded at this link:
Files

9.2 File creation

9.2.1 artist_test.txt

Convert the file full_music_data.csv to a .txt file. .txt files are easier to manipulate then .csv files.

9.2.2 file_genre_*format.csv*

Use the following code to create:

```
import csv

#Reads through the influence data to find each artist and genre
def parse_genre(filename):
    data = set() #Use a set to avoid duplicates
    line_count = 0
    with open(filename) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        for row in csv_reader:
            if line_count == 0:
                line_count += 1
            else:
                data.add((row[0], row[2]))
                data.add((row[4], row[6]))
    return data\subsubsection{pr$\_$data.csv}

data = parse_genre("influence_data.csv")

#creates list where each element is a line from the file
lines = []
```

```
with open("artist_test.txt", "r") as f:
    for row in f:
        words = row.split(",")
        lines.append(row.rstrip())

#Appends genre to each item in the list
count = 0
for line in lines:
    words = line.split(",")
    artist_id = words[1] #Gets artist of word
    for t in data: #Loops through set
        if t[0] == artist_id: #If they match, which happens once
            genre = t[1] #Assign genre
            line_temp = lines[count]
            lines[count] = line_temp + "," + genre
    count = count + 1

#Sends list to file. When done, you must change the file type to csv, and add
#genre to end of first row. I Couldn't write directly to .csv file without
#causing issues. If running multiple times, change "x" to "a" to avoid crash.
with open("artist_genre.txt", "x") as f:
    for l in lines:
        f.write(l + "\n")
```

9.2.3 pr_data.csv

What this code does is create the file pr_data.csv, which consists of all Pop/Rock artists and their data about their music, with the decade they started appended to the end. This was done because we thought grouping all of Pop/Rock together felt ineffective, as it felt incorrect grouping bands like the Beatles with modern artists like Beyonce. Again, the result is stored on link:

```
import csv

#Looks through the csv file and gets all artist ids associated w/ Pop/Rock
def parse_pr(filename):
    data = set() #Want set to avoid duplicates so use set
    line_count = 0
    with open(filename) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        for row in csv_reader:
            if line_count == 0:
                line_count += 1
            else:
                if row[2] == 'Pop/Rock' or row[6] == 'Pop/Rock':
                    if row[2] == 'Pop/Rock':
                        data.add((row[0], row[3]))
                    if row[6] == 'Pop/Rock':
                        data.add((row[4], row[7]))
    return data
data_pop = parse_pr("influence_data.csv")

#Reads in the artist_text.txt and passes each line to file
lines = []
with open("artist_test.txt", "r") as f:
    for row in f:
        lines.append(row.rstrip())
```

```
#Appends year of start to each Pop/Rock artist
count  = 0
for line in lines:
    word = line.split(",")
    artist_id = word[1]
    for t in data_pop:
        if t[0] == artist_id:
            year = t[1]
            line_temp = lines[count]
            lines[count] = line_temp + "," + year
    count = count + 1

#For all artists, if they aren't pop/rock sets updated string to empty
count = 0
for line in lines:
    word = line.split(",")
    try:
        t = word[16]
    except:
        lines[count] = ''
    count = count + 1

#Sends to file, due to nature of .csv files, send as .txt, then can do the
#following prepping: From full_music_data.csv, copy first row w/ column
#names, then add ".year". Then, convert the file type to .csv
with open("pr_data.txt", 'x') as f:
    for l in lines:
        if l != '':
            f.write(l + "\n")
```

9.3 The code for the network

All code for the Neural Network is stored here. All code is stored in the file neuralnet.py, and can run assuming that you have properly created the two files listed above.

10 Sources

1. https://www.mathmodels.org/Problems/2021/ICM-D/2021_ICM_Problem_D.pdf
2. https://www.tensorflow.org/
3. https://medium.com/wearesinch/searching-for-similar-songs-on-spotify-data-science-c7e3faeb66f0