How mainstream is Obama's Music Taste?

A Comparison of Obama's Annual Song List and Spotify's Top Songs

Amanda Rudolph and Claire Weadock

Github Repository Link

Goals

-We want to go through Barack Obama's annual "Favorite Song" lists comparing them to Spotify's playlist of top songs of the year for that same year. The similarities between the two will allow us to determine how mainstream Barack Obama's music taste is.

-We want to define "mainstream" by calculating the number of songs Obama has in common in a year with Spotify's playlist. This will allow us to determine which year Obama's music was the most mainstream.

-We want a visualization to show how the commonalities between Obama and Spotify's playlist each year compare (bar plot). We also want a visualization to show the percentage of Obama's songs that are shared with Spotify's playlist that year.

We were able to achieve all of these goals.

Websites and API’s Used

-We used 4 different websites within our project as we had to access Obama’s favorite song lists, which all came from different websites.

-We also accessed the Spotify API to get Spotify’s “Top Hits” playlist for 2017 , 2018 , 2019 , and 2020.

Problems we faced

-Problem getting Obama’s songs in the same format as they came from multiple websites

-Problem with spotify API

-Obama doesn’t have as mainstream of a music taste as we thought, none of his songs have commonalities with Spotify’s 2020 playlist.

-2020’s spotify playlist has a shorter length than the others with only 50 songs.

-The original website we used to access Obama’s favorite songs of 2019 was not formatted in a way that we could work with in comparison to Obama’s other song lists.

Calculation File

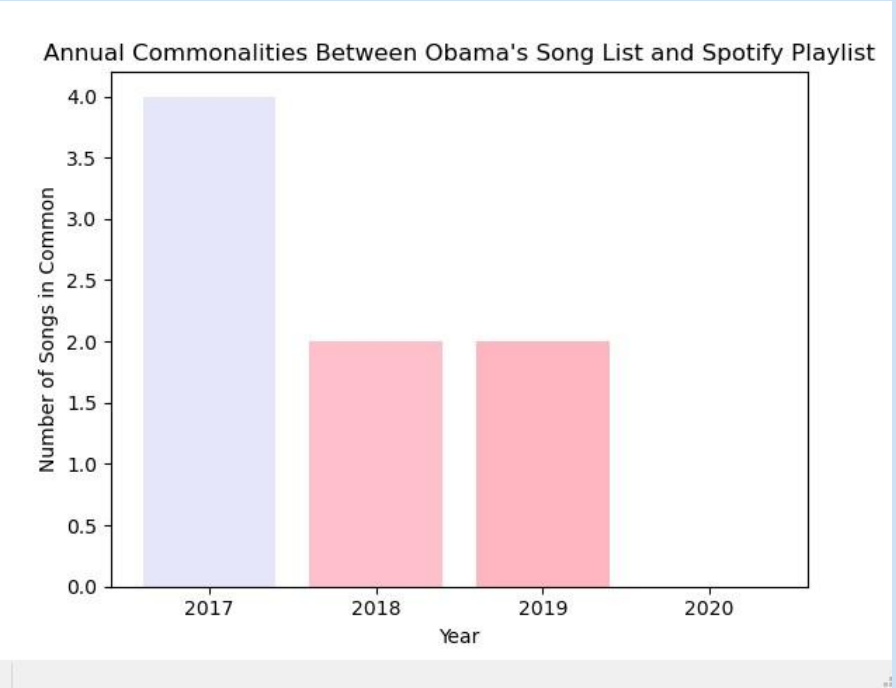
From data stored in the database, we calculated the percentage of songs that Obama has in common with Spotify's "Top Hits" playlist in a given year (2017,2018,2019, or 2020).

5 lines (5 sloc) | 443 Bytes

Raw | Blame

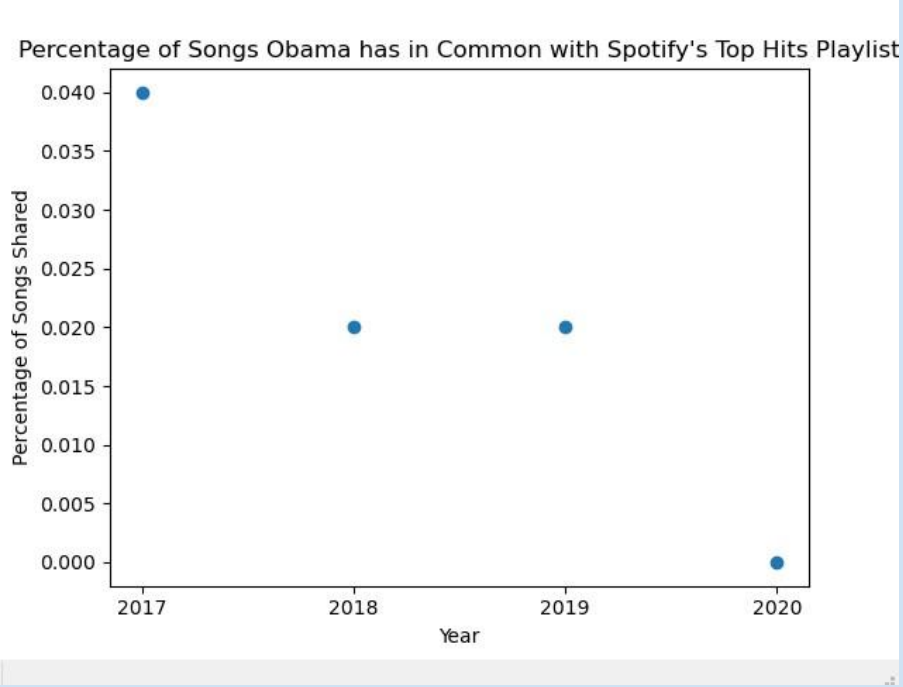
```
1  We calculated the percentage of songs that are in Obama's song list in a given year out of the songs in Spotify's playlist that year.
2  In 2017 , 4.0% of Obama's favorite songs are in Spotify's Top Hits Playlist
3  In 2018 , 2.0% of Obama's favorite songs are in Spotify's Top Hits Playlist
4  In 2019 , 2.0% of Obama's favorite songs are in Spotify's Top Hits Playlist
5  In 2020 , 0.0% of Obama's favorite songs are in Spotify's Top Hits Playlist
```

Visualization 1: Barplot



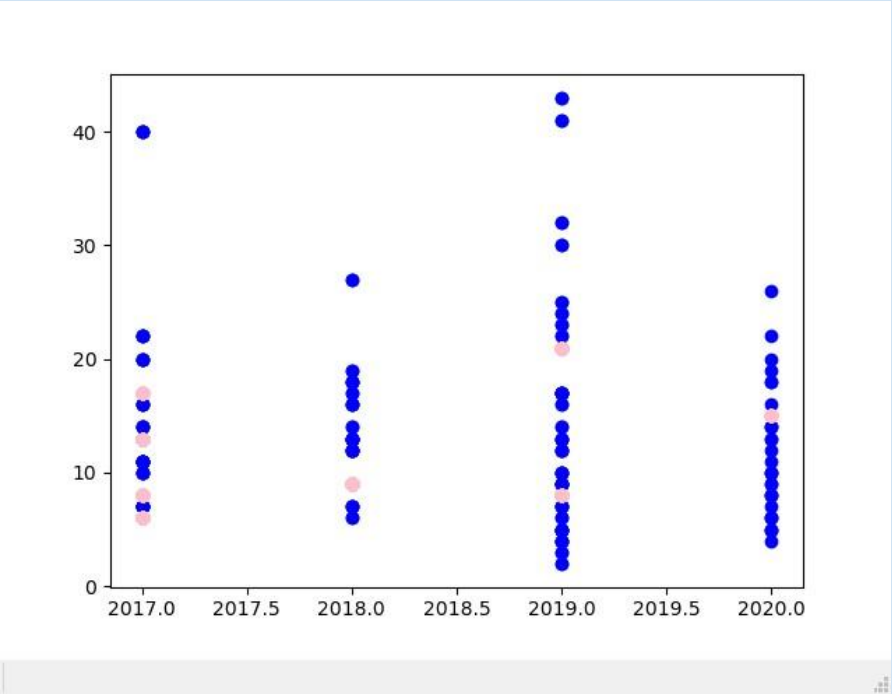
This bar chart shows the number of common songs between Obama's top songs and Spotify's top songs, for each year.

Visualization 2: Scatterplot



This scatterplot shows the percentage of songs that are in common between Obama's top songs and Spotify's top songs, for each year.

Visualization 3: Scatterplot



This scatterplot shows a distribution of the number of characters in Obama's top songs by year. The blue dots are songs that are only in Obama's top songs, while the pink dots are songs that are in both Obama's top songs and Spotify's top songs.

Instructions for running the code

-To view each visualization comment out the one you want to see in the main function.

-Otherwise, nothing else needs to be manipulated before running the code. No other code needs to be run because the values are already inserted into the database.

Code Documentation: get_obama_songs(year)

-def_get_obama_songs(2017)
-def_get_obama_songs(2018)

-def_get_obama_songs_(2019)

-def_get_obama_songs_2020)

Essentially, these 4 functions together are doing the same thing. They are accessing Obama's list for the given year from a website (using BeautifulSoup) and putting them all into their own individual list. We had to use different functions for each year as Obama's songs lists were found on multiple websites.

```
def get_obama_songs_2018():
    response = requests.get('https://www.businessinsider.com/obama-favorite-songs-2018-12')
    if response.ok:
        obamas_songs = []
        soup = BeautifulSoup(response.content, 'html.parser')
        paragraph = soup.find_all('div', class_ = 'slide')
        for i in paragraph:
            div = i.find('h2', class_ = 'slide-title-text')
            obamas_songs.append(div.text)
        regged = []
        reg_exp = r'\"(\b.+\b)\"\sby\s.+'
        for i in obamas_songs:
            x = re.findall(reg_exp, i)
            for i in x:
                regged.append(i)
    return regged
```

Code Documentation: get_playlist_id()

The function get_playlist_id gets the playlist id's for the years 2017,2018,2019, and 2019 from the given Spotify URL's. If a different year's playlist is ran, the function returns "Invalid year".

```
def get_playlist_id(year):
    switcher = {2017: '37i9dQZF1DWTE7dVUebpUW', 2018: '37i9dQZF1DX1HUbZS4LEyL', 2019: '37i9dQZF1DXcz8eC5kMSWZ', 2020: '37i9dQZF1DX7Jl5KP2eZaS'
    return switcher.get(year, "Invalid year")
```

Code Documentation: compare_obama_to_spotify()

```
def compare_obama_to_spotify(obamadict, spotifydict):
    #obamadict is a dictionary where the key is the year, and the values are
    #obamas songs for that year

    #spotifydict is a dictionary where the key is the year, and the values are
    #spotifys top songs for that year

    #key = year, value = list of shared songs
    common_songs = {}

    for i in obamadict.items():
        lst = []
        for j in spotifydict.items():
            for song in i[1]:
                if song in j[1] or song == "I LIke It" or song == "Love Lies":
                    lst.append(song)
        common_songs[i[0]] = lst
    return common_songs
```

In this function, we are comparing Obama's list to Spotify's playlist and creating a dictionary of the common songs. To do this, the function takes in 2 inputs: obamadict and spotifydict. Obamadict is a dictionary where the key is the year, and the values are Obama's songs for that year. Spotifydict is a dictionary where the key is the year, and the values are Spotify's top songs for that year.

Code Documentation: get_playlist_tracks(year)

This function accesses the Spotipy API and returns a list of tracks from a Spotify playlist, given the playlist id and year.

```
def get_playlist_tracks(year):
    tracks = []
    playlist_id = get_playlist_id(year)
    results = sp.playlist_tracks(playlist_id)
    for i in results['items']:
        tracks.append(i['track']['name'])

    return tracks
```

Code Documentation: setUpDatabase(db_name)

This function creates the database file and creates a connection the sqlite3..write more

```
def setUpDatabase(db_name):
    path = os.path.dirname(os.path.abspath(__file__))
    conn = sqlite3.connect(path+'/'+db_name)
    cur = conn.cursor()
    return cur, conn
```

Code Documentation: create_table(curr, conn)

Given cur and conn, this function creates the “Songs” and “Shared” tables in our database file, commonalities.db

```
def create_table(cur, conn):
    cur.execute('CREATE TABLE IF NOT EXISTS Songs ("SongYear" TEXT, "Name" TEXT)') #, "Spotify\'s Top Songs" TEXT, "Songs in Common" TEXT, "Number of Songs in Common" INT
    cur.execute('CREATE TABLE IF NOT EXISTS Shared ("Year" TEXT, "CommonSongs" TEXT, "NumberInCommon" INTEGER, LengthOfSpotifyPlaylist INTEGER)')
    conn.commit()
```

Code Documentation: insert_obama_songs_first_25(obamas_songs, cur, conn)

This function inserts the first 25 songs on Obama's list for a given year into a database. It limits the number of songs inserted using a while loop with an index that updates through each iteration. The function is coded to break once the 25 limit is reached.

```
def insert_obama_first_25(obama_dict, cur, conn):
    index = 0
    #items = obama_dict.values()
    for key in obama_dict:
        for x in obama_dict[key]:
            if index < 25:
                cur.execute('INSERT INTO Songs (SongYear, Name) VALUES (?,?)', (key, x))
                index += 1
            else:
                insert_obama_rest(index, obama_dict, cur, conn)
                break
    conn.commit()
```

Code Documentation: insert_obama_songs_rest(num, obama_dict, cur, conn)

This function inserts the a maximum of 25 more songs from Obama's list into the database by using a for loop.

```
def insert_obama_rest(num, obama_dict, cur, conn):
    index = num
    for key in obama_dict:
        for i in obama_dict[key][25:]:
            cur.execute('INSERT INTO Songs (SongYear, Name) VALUES (?,?)', (key, i))
    conn.commit()
```

Code Documentation: insert_shared(obama_dict, year, cur, conn)

This function inserts the year, common songs, number of common songs, and the length of the spotify playlist for that year. Since 2020 has no shared songs between Obama and Spotify, we had a special if statement to insert 2020's data

```
def insert_shared(obama_dict, year, cur, conn):
    sdict = {year: get_playlist_tracks(year)}
    shared_songs_dict = compare_obama_to_spotify(obama_dict, sdict)
    splaylists_lengths = [len(get_playlist_tracks(2017)), len(get_playlist_tracks(2018)), len(get_playlist_tracks(2019)), len(get_playlist_tracks(2020))]
    for key in shared_songs_dict:
        for i in shared_songs_dict[key]:
            index = 0
            cur.execute('INSERT INTO Shared (Year, CommonSongs, NumberInCommon, LengthOfSpotifyPlaylist) VALUES (?,?,?,?)', (key, i, len(shared_songs_dict[key]), splaylis
            index += 1
    if year == 2020:
        cur.execute('INSERT INTO Shared (Year, CommonSongs, NumberInCommon, LengthOfSpotifyPlaylist) VALUES (?,?,?,?)', (2020, 'No Common Songs', 0, 100))

    conn.commit()
```

Code Documentation: make_barchart(cur)

This function uses Year and NumberInCommon to create a barchart for our Visualization #1.

```
def make_barchart(cur):
    commonalities = []
    years = []
    cur.execute("SELECT Year, NumberInCommon FROM Shared")
    for row in cur:
        if row[0] not in years:
            years.append(row[0])
            commonalities.append(row[1])
    label_name = ['2017', '2018', '2019', '2020']
    labels = (label_name[0], label_name[1], label_name[2], label_name[3])
    plt.bar(labels, commonalities, align = "center", color = ["lavender", "pink", "lightpink", "purple", "hotpink"])
    plt.title("Annual Commonalities Between Obama's Song List and Spotify Playlist")
    plt.ylabel("Number of Songs in Common")
    plt.xlabel("Year")
    plt.savefig("commonalities.png")
    plt.show()
```

Code Documentation: make_scatterplot(cur)

This function takes in cur and uses Year, NumberInCommon, and LengthOfSpotifyPlaylist to create a scatterplot for out Visualization #2.

```
def make_scatterplot(cur):
    percentages = []
    numcommon = []
    years = [] #'2017', '2018', '2019', '2020']
    cur.execute('SELECT Year, NumberInCommon, LengthOfSpotifyPlaylist FROM Shared')
    for row in cur:
        if row[0] not in years:
            years.append(row[0])
            numcommon.append(row[1])
            percentages.append(row[1]/row[2])

    plt.scatter(years, percentages)
    plt.title("Percentage of Songs Obama has in Common with Spotify's Top Hits Playlist")
    plt.xlabel("Year")
    plt.ylabel("Percentage of Songs Shared")
    plt.savefig("ScatterplotPercentages")
    plt.show()
```

Code Documentation: make_char_scatterplot(cur

This function uses Name, Year, and CommonSongs to make a scatterplot of the number of characters for our Visualization #3.

```
def make_char_scatterplot(cur):
    songs_chars = []
    song_years = []
    commons_chars = []
    common_years = []
    cur.execute('SELECT Songs.Name, Songs.SongYear, Shared.CommonSongs, Shared.Year FROM Songs JOIN Shared ON Songs.SongYear = Shared.Year')
    for row in cur:
        songs_chars.append(len(row[0]))
        song_years.append(int(row[1]))
        commons_chars.append(len(row[2]))
        common_years.append(int(row[3]))

    plt.scatter(song_years, songs_chars, c = 'blue')
    plt.scatter(common_years, commons_chars, c = 'pink')
    plt.show()
```

Code Documentation: writefile(cur)

This function writes to an outfile a summary of our calculations. The output is visible in calculationfile.txt.

```
def writefile(cur):

    filename = open("calculationfile.txt", "w")
    filename.write("We calculated the percentage of songs that are in Obama's song list in a given year out of the songs in Spotify's playlist that year.")
    filename.write('\n')
    percentages = []
    numcommon = []
    years = [2017, 2018, 2019, 2020]
    cur.execute('SELECT Year, NumberInCommon, LengthOfSpotifyPlaylist FROM Shared')
    for row in cur:
        if row[0] not in years:
            years.append(row[0])
            numcommon.append(row[1])
            percentages.append(row[1]/row[2])
    year_count = 0
    for x in percentages:
        filename.write("In {} , {}% of Obama's favorite songs are in Spotify's Top Hits Playlist".format(years[year_count], x*100))
        filename.write('\n')
        year_count += 1
    filename.close()
```

Code Documentation: get_dict(year, lst)

This function returns a dictionary given a year and a list, with the year and the key and lst as the value. We needed this function because using Beautiful Soup and Spotipy functions returns lists and we wanted to organize the songs by year as a dictionary.

```
def get_dict(year, lst):
    dict = {year: lst}
    return dict
```

Database Screenshot

	Year ▾1	CommonSongs	NumberInCommon	LengthOfSpotifyPlaylist
	Filter	Filter	Filter	Filter
1	2017	Mi Gente	4	100
2	2017	Havana	4	100
3	2017	Unforgettable	4	100
4	2017	Sign of the Times	4	100
5	2018	I LIke It	2	100
6	2018	Love Lies	2	100
7	2019	Old Town Road - Remix	2	100
8	2019	3 Nights	2	100
9	2020	No Common Songs	0	100

Table: Songs

	SongYear	Name
	Filter	Filter
1	2017	Mi Gente
2	2017	Havana
3	2017	Blessed
4	2017	The Joke
5	2017	First World Problems
6	2017	Rise Up
7	2017	Wild Thoughts
8	2017	Family Feud
9	2017	Humble
10	2017	La Dame et Ses Valises
11	2017	Unforgettable
12	2017	The System Only Dreams in Total Darkness
13	2017	Chanel
14	2017	Feel It Still
15	2017	Butterfly Effect
16	2017	Matter of Time
17	2017	Little Bit
18	2017	Millionaire
19	2017	Sign of the Times
20	2017	Broken Clocks
21	2018	Apes**t
22	2018	Bad Bad News
23	2018	Could've Been
24	2018	Disco Yes

1 - 24 of 106

Resources Used

Date	Issue Description	Location of Resource	Result (Did it solve the issue?)
4/7/2021	We weren't sure how to access the Spotify API's and get the playlists.	Spotify for Developers	Yes, we were able tto access the playlists by learning how to grab playlist IDs.
4/8/21	Needed help with the Spotify API.	Spotipy	Yes, we were able to successfully use the Spotify API.
4/24/21	Needed help with visualizations and choosing which ones to use	Best Python Data Visualization Libraries	Yes, this helped us decide that we wanted to use a bar plot and scatter plot and provided the libraries in order to do so.
4/27/21	We got several error pop-ups throughout the course the project that we could not understand.	Stack Overflow	Stack Overflow helped us understand where the errors