2. Data Acquisition Notebook

Now that we have gone through the OAuth authorization process, we are reading to start collection data from the Spotify API database. But first, we need to develop inputs into the Spotify API. Our interest was to look at trends in the music industry top 100 songs over the history of the Billboard charts. Then in exploring the Spotify API, it was our goal to access the Spotify Audio Features from the above mention specific tracks. With this data, we were interested to see actual numerical values defining the music that we listen to today.

To achieve the above goal, we first had to obtain the Billboard Charts data for the duration of it's history. Then, using the Spotify API search request, we could obtain the unique Spotify ID for the tracks that we wished to observe. Using the Spotify ID, we could the again use the Spotify API to ge the audio features for that specific track.

In [44]:
```
%run functions.py # We first imported the functions.py file to get acces
s to our typical functions.
```

In [41]:
```
def insert(connection, table_name, data):
    '''
    The following function takes and input of a SQL connection, table na
me, and data. It takes these inputs and
    inserts the data into the table denoted by the table name in the SQL
 server via the connection.
    '''

    data.to_sql(name=table_name, con=connection, if_exists='append', ind
ex = True, index_label = "I") # Inserts data into table
```

Web-Scrapped Data Acquisition

To get our webscrapped data from the Billboard charts, we used the website billboardtop100of.com/ to gain the data that we were looking for. This site provided the Rank, Artist Name and Track Name for each song in the charts over the history of the Billboard charts. However, webscrapping this data turned out to be harder than it looked. The range of the history of the Bildboard charts ranges from 1941 to 2017. The problem was was that certain years were formatted differently than others. Additionally in certain years, the charts only were 30 songs while other years were 100. This meant for certain years, we had to go back, look and develop new xpath and parsing tools to get the data that we wanted. To tackle the parsing of the html webscrapped data coming from the billboardtop100of.com/ website, we wrote the following parser function to append a lists of row elements containing year, rank, artist, and song for every year.

In [1]:

```
def parser(htmlroot, year):
    '''
    The following function takes inputs of a htmlroot and year. It takes
 the htmlroot and parses the data and
    appends it in a list. The function is specific to the formatting of
 billboardtop100of.com. The reason for
    the if statements of the years is because of differences in formatti
ng of specific year pages. The output
    is list of lists containing row elements of the tables from the bill
boardtop100of.com site in the following
    order: year, rank of song in the charts, artist name and track name.
    '''
    yearlist = [] # Create yearlist list
    if year == 1942 : # If statement to check if the year is 1942 (Forma
tting of the page is unique)
        matches1 = htmlroot.xpath("//body/div/div/div/div/div/article/di
v/p/text()") # Xpath function to parse song info
        matches = [] # Create matches list
        for match in matches1: # For loop to parse year, rank, artist an
d track name from matches
            if "\n" in match: # Formatting of site captures unneed new l
ine data, if statement to skip appending them
                continue # Continue loop for unneeded new line matches
            number = re.search(r'(?=\d+)\d+', match).group() # Parse ran
k with regular expressions from match
            song_artist = re.search(r'(?<=\.).*', match).group() # Parse
 artist and track with regular expressions from match
            song_artist = song_artist.split(" by ") # Split song_artist
 to separate artist and track
            song = song_artist[0] # Define song from song_artist
            artist = song_artist[1] # Define artist from song_artist
            if song[0] == ' ': # If statement to get rid of spaces at th
e beginning of some song strings
                song = song[1:] # Define song without spaces
            matches.extend([number, artist, song]) # Extend matches list
 with number, artist, song parsed items
    if year == 1943 or year == 1944: # If statement to check if the year
 is 1943 or 1944 (Formatting of these pages is unique)
        matches1 = htmlroot.xpath("//body/div/div/div/div/div/article/di
v/p/text()") # Xpath function to parse song info
        matches = [] # Create matches list
        for match in matches1: # For loop to parse year, rank, artist an
d track name from matches
            if "\n" in match: # Formatting of site captures new lines at
 beginning of matches. If statement to remove \n characters
                match = match.replace("\n", "") # Replace \n character w
ith nothing
            number = re.search(r'(?=\d+)\d+', match).group() # Parse ran
k with regular expressions from match
            song_artist = re.search(r'(?<=\.).*', match).group() # Parse
 artist and track with regular expressions from match
            song_artist = song_artist.split(" – ") # Split song_artist t
o separate artist and track
            song = song_artist[0] # Define song from song_artist
            artist = song_artist[1] # Define artist from song_artist
            if song[0] == ' ': # If statement to get rid of spaces at th
```

```
e beginning of some song strings
                song = song[1:] # Define song without spaces
            matches.extend([number, artist, song]) # Extend matches list
 with number, artist, song parsed items
    if year == 1999: # If statement to check if the year is 1999 (Format
ting of the page is unique)
        matches = htmlroot.xpath(".//tr/td[1]/text() | .//tr/td[2]/text
() | .//tr/td[3]/text()") # Xpath function to parse rank, artist and son
g info
    if year == 2016: # If statement to check if the year is 2016 (Format
ting of the page is unique)
        matches1 = htmlroot.xpath(".//tr/td[1]/text() | .//tr/td[2]/text
() | .//tr/td[3]/text()") # Xpath function to parse rank, artist and son
g info
        matches = []  # Create matches list
        for match in matches1: # For loop to remove \n character from ma
tch strings
            if "\n" in match: # Formatting of site captures new lines at
 beginning of matches. If statement to remove \n characters
                match = match.replace("\n", "") # Replace \n character w
ith nothing
            matches.append(match) # Append matches list
    if year == 2015: # If statement to check if the year is 2015 (Format
ting of the page is unique)
        matches = htmlroot.xpath("(.//tr/td/h6[1]/text())") # Xpath func
tion to parse rank, artist and song info
    if year not in (1999, 2015, 2016, 1942, 1943, 1944): # If statement
 to see if year is not 1999, 2015, 2016, 1942, 1943, 1944 (Formatting of
 these pages are all the same)
        matches = htmlroot.xpath("(.//tr/td/text()) | (.//tr/td/*/text
())") # Xpath function to parse rank, artist and song info
    for i in range(0, len(matches), 3): # For loop to append yearlist as
 a list of lists of year, rank, artist and song for each year
        yearlist.append([year, matches[i], matches[i+1], matches[i+2]])
# Append yearlist
    return yearlist # Return year list
```

With the parser function, we were able to run a for loop for the range of 1941 to 2017 to get the htmlroot of each page, and parse the data from it. This loop is when we ran into some issues again. We printed the year and "Done" each time the loop is completed, but we would find hiccups in the parsing of data. For example, 1942 is uniquely formatted from all the other pages. 1942 and 1943 are formmatted in identical ways, but are different from all the others. The meant a lot of trial and error on our part running the loop to see if it would get all the data we were looking to get.

Once the data is appended in a list of lists of row elements following the [year, rank, artist, song] structure, we create a pandas dataframe of the websrapped data

In [6]:
```
location = 'billboardtop100of.com/' # Location of data for web scrapping
protocol = None # Data for web scrapping has a http protocol, None input
 in buildURL function will define it as http
full = [] # Define new list
for year in range(1941, 2017): # For loop to scrap all data from the bil
lboardtop100of.com/ from all the years (1941, 2016)
    yearlink = str(year)+"-2" # Resource of billboardtop100of.com/ is th
e "year-2", define the yearlink to have correct resource
    url = buildURL(protocol, location, yearlink) # Define url with the b
uildURL function with protocol, location, yearlink
    html = HTTPGet(url, None, "N") # Get html root with the HTTPGet func
tion
    yearlist = parser(html, year) # Get list of all data using the parse
r function
    print(year, "- Done") # Print Year - Done to following status of loo
p
    full.extend(yearlist) # Extend full list of all data for every year
billboard_top_df = pd.DataFrame(full)  # Create pandas dataframe using t
he "full" list of data
billboard_top_df.columns = ["Year", "Rank", "Artist", "Track"] # Define
 the column names of the billboard_top_df dataframe
billboard_top_df.head()
```

1941 – Done
1942 – Done
1943 – Done
1944 – Done
1945 – Done
1946 – Done
1947 – Done
1948 – Done
1949 – Done
1950 – Done
1951 – Done
1952 – Done
1953 – Done
1954 – Done
1955 – Done
1956 – Done
1957 – Done
1958 – Done
1959 – Done
1960 – Done
1961 – Done
1962 – Done
1963 – Done
1964 – Done
1965 – Done
1966 – Done
1967 – Done
1968 – Done
1969 – Done
1970 – Done
1971 – Done
1972 – Done
1973 – Done
1974 – Done
1975 – Done
1976 – Done
1977 – Done
1978 – Done
1979 – Done
1980 – Done
1981 – Done
1982 – Done
1983 – Done
1984 – Done
1985 – Done
1986 – Done
1987 – Done
1988 – Done
1989 – Done
1990 – Done
1991 – Done
1992 – Done
1993 – Done
1994 – Done
1995 – Done
1996 – Done
1997 – Done

```
1998 - Done
1999 - Done
2000 - Done
2001 - Done
2002 - Done
2003 - Done
2004 - Done
2005 - Done
2006 - Done
2007 - Done
2008 - Done
2009 - Done
2010 - Done
2011 - Done
2012 - Done
2013 - Done
2014 - Done
2015 - Done
2016 - Done
```

Out[6]:

	Year	Rank	Artist	Track
0	1941	1	Glenn Miller	Chattanooga Choo Choo
1	1941	2	Sammy Kaye	Daddy
2	1941	3	Artie Shaw	Stardust
3	1941	4	The Andrews Sisters	Boogie Woogie Bugle Boy
4	1941	5	Jimmy Dorsey	Amapola (Pretty Little Poppy)

API Data Acquisition

Now that we have webscrapped the data of the Billboard Charts of songs over it's history, we can now use the track names to perform a Spotify Search API request. This request is similar to searching in the toolbar of the Spotify Application. In the parameters of the search request you can limit your query to one response aka the top search item of the search expression. Since these songs are charting top hits, when we manually searched a number of the tracks, the top result was the intended track. The thing that we were scared about was with the older tracks. Some of the 1940 popular hits have been redone, covered and republished on the Spotify platform. The way that the search request operates is that it is ordered by the most popular or most streamed song. So a cover, could possibly be mistake for the original on the chart. We decided that if this happened, it would only be a very small subset of the data, so we didn't worry too much about it. The other thing that we had to keep in mind was the outcome of the search request not finding anything. Spotify is a streaming platform, but it doesn't mean that is has everything on it. We decided to append an "NA" or null value for failue of the Spotify API search request

In [7]:

```
def search(song):
    '''
    The following function takes an input of song. It takes the song str
ing and uses the Spotify API search request
    to get the specific Spotify ID for the song. The function returns th
e Spotify ID.
    '''
    creds = opencreds() # Use the open creds function to access the cred
s JSON file
    protocol = 'https' # Protocol of the Spotify search request
    location = 'api.spotify.com' # Location of the Spotify search reques
t
    resource = 'v1/search' # Resource of the Spotify search request
    url = buildURL(protocol, location, resource) # Define url of Spotify
 search request
    song = str(song) # Insure song is a string
    song = song.replace(" ", "+") # Format song with + instead of spaces
 as the Spotify search request requires
    params_dic ={"q": song,
          "type": "track",
          "limit": 1} # Define parameters of Spotify search request
    headers_dic = {'authorization': "Bearer " + getToken()} # Define hea
ders of Spotify search request utilizing the getToken function to refres
h authorization
    track_info = requests.get(url, headers=headers_dic, params=params_di
c) # Perform Spotify search request
    if track_info.status_code != 200: # If statement to test if search r
equest is unsuccessful
        track_id = 'NA' # Return null value if request is unsuccessful
    else: # Else statement for when request is successful
        try: # Try statement to get Spotify ID if search is successful
            track_id = track_info.json()['tracks']['items'][0]['id'] # G
et Spotify ID from the search requests returned JSON file
        except IndexError: # Except statement for when search is unsuces
sful meaning the song is not found on Spotify streaming platform
            track_id = 'NA' # Define null value for unfound searches
    return track_id # Return the Spotify ID for inputed song
```

With the search function written to perform a search request on the Spotify API, the next step was to actually run the search request for each and every song in the billboard_top_df dataframe. So that is exactly was we did. We ran the search function for each track appending the unique Spotify ID.

In [8]:
```
track_id_list = [] # Define track_id_list
for i in range (len(billboard_top_df)): # For loop to get every Spotify
 ID for every track in the billboard_top_df dataframe
    print(i) # Print i to monitor progress of loop
    song = billboard_top_df["Track"][i] # Define song from billboard_top
_df dataframe
    track_id = search(song) # Get Spotify ID for each song using the sea
rch function
    track_id_list.append(track_id) # Append the track_id_list list of th
e Spotify ID
print("Success!") # Print "Success!" on completion of loop
```

1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538

1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994

2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792

2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963

3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077

3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476

3534
3535
3536
3537
3538
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590

3705
3706
3707
3708
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3760
3761

3762
3763
3764
3765
3766
3767
3768
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818

3933
3934
3935
3936
3937
3938
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989

4446
4447
4448
4449
4450
4451
4452
4453
4454
4455
4456
4457
4458
4459
4460
4461
4462
4463
4464
4465
4466
4467
4468
4469
4470
4471
4472
4473
4474
4475
4476
4477
4478
4479
4480
4481
4482
4483
4484
4485
4486
4487
4488
4489
4490
4491
4492
4493
4494
4495
4496
4497
4498
4499
4500
4501
4502

4560
4561
4562
4563
4564
4565
4566
4567
4568
4569
4570
4571
4572
4573
4574
4575
4576
4577
4578
4579
4580
4581
4582
4583
4584
4585
4586
4587
4588
4589
4590
4591
4592
4593
4594
4595
4596
4597
4598
4599
4600
4601
4602
4603
4604
4605
4606
4607
4608
4609
4610
4611
4612
4613
4614
4615
4616

4674
4675
4676
4677
4678
4679
4680
4681
4682
4683
4684
4685
4686
4687
4688
4689
4690
4691
4692
4693
4694
4695
4696
4697
4698
4699
4700
4701
4702
4703
4704
4705
4706
4707
4708
4709
4710
4711
4712
4713
4714
4715
4716
4717
4718
4719
4720
4721
4722
4723
4724
4725
4726
4727
4728
4729
4730

4902
4903
4904
4905
4906
4907
4908
4909
4910
4911
4912
4913
4914
4915
4916
4917
4918
4919
4920
4921
4922
4923
4924
4925
4926
4927
4928
4929
4930
4931
4932
4933
4934
4935
4936
4937
4938
4939
4940
4941
4942
4943
4944
4945
4946
4947
4948
4949
4950
4951
4952
4953
4954
4955
4956
4957
4958

4959
4960
4961
4962
4963
4964
4965
4966
4967
4968
4969
4970
4971
4972
4973
4974
4975
4976
4977
4978
4979
4980
4981
4982
4983
4984
4985
4986
4987
4988
4989
4990
4991
4992
4993
4994
4995
4996
4997
4998
4999
5000
5001
5002
5003
5004
5005
5006
5007
5008
5009
5010
5011
5012
5013
5014
5015

5187
5188
5189
5190
5191
5192
5193
5194
5195
5196
5197
5198
5199
5200
5201
5202
5203
5204
5205
5206
5207
5208
5209
5210
5211
5212
5213
5214
5215
5216
5217
5218
5219
5220
5221
5222
5223
5224
5225
5226
5227
5228
5229
5230
5231
5232
5233
5234
5235
5236
5237
5238
5239
5240
5241
5242
5243

5244
5245
5246
5247
5248
5249
5250
5251
5252
5253
5254
5255
5256
5257
5258
5259
5260
5261
5262
5263
5264
5265
5266
5267
5268
5269
5270
5271
5272
5273
5274
5275
5276
5277
5278
5279
5280
5281
5282
5283
5284
5285
5286
5287
5288
5289
5290
5291
5292
5293
5294
5295
5296
5297
5298
5299
5300

5358
5359
5360
5361
5362
5363
5364
5365
5366
5367
5368
5369
5370
5371
5372
5373
5374
5375
5376
5377
5378
5379
5380
5381
5382
5383
5384
5385
5386
5387
5388
5389
5390
5391
5392
5393
5394
5395
5396
5397
5398
5399
5400
5401
5402
5403
5404
5405
5406
5407
5408
5409
5410
5411
5412
5413
5414

5415
5416
5417
5418
5419
5420
5421
5422
5423
5424
5425
5426
5427
5428
5429
5430
5431
5432
5433
5434
5435
5436
5437
5438
5439
5440
5441
5442
5443
5444
5445
5446
5447
5448
5449
5450
5451
5452
5453
5454
5455
5456
5457
5458
5459
5460
5461
5462
5463
5464
5465
5466
5467
5468
5469
5470
5471

5529
5530
5531
5532
5533
5534
5535
5536
5537
5538
5539
5540
5541
5542
5543
5544
5545
5546
5547
5548
5549
5550
5551
5552
5553
5554
5555
5556
5557
5558
5559
5560
5561
5562
5563
5564
5565
5566
5567
5568
5569
5570
5571
5572
5573
5574
5575
5576
5577
5578
5579
5580
5581
5582
5583
5584
5585

5643
5644
5645
5646
5647
5648
5649
5650
5651
5652
5653
5654
5655
5656
5657
5658
5659
5660
5661
5662
5663
5664
5665
5666
5667
5668
5669
5670
5671
5672
5673
5674
5675
5676
5677
5678
5679
5680
5681
5682
5683
5684
5685
5686
5687
5688
5689
5690
5691
5692
5693
5694
5695
5696
5697
5698
5699

5871
5872
5873
5874
5875
5876
5877
5878
5879
5880
5881
5882
5883
5884
5885
5886
5887
5888
5889
5890
5891
5892
5893
5894
5895
5896
5897
5898
5899
5900
5901
5902
5903
5904
5905
5906
5907
5908
5909
5910
5911
5912
5913
5914
5915
5916
5917
5918
5919
5920
5921
5922
5923
5924
5925
5926
5927

6099
6100
6101
6102
6103
6104
6105
6106
6107
6108
6109
6110
6111
6112
6113
6114
6115
6116
6117
6118
6119
6120
6121
6122
6123
6124
6125
6126
6127
6128
6129
6130
6131
6132
6133
6134
6135
6136
6137
6138
6139
6140
6141
6142
6143
6144
6145
6146
6147
6148
6149
6150
6151
6152
6153
6154
6155

6156
6157
6158
6159
6160
6161
6162
6163
6164
6165
6166
6167
6168
6169
6170
6171
6172
6173
6174
6175
6176
6177
6178
6179
6180
6181
6182
6183
6184
6185
6186
6187
6188
6189
6190
6191
6192
6193
6194
6195
6196
6197
6198
6199
6200
6201
6202
6203
6204
6205
6206
6207
6208
6209
6210
6211
6212

6213
6214
6215
6216
6217
6218
6219
6220
6221
6222
6223
6224
6225
6226
6227
6228
6229
6230
6231
6232
6233
6234
6235
6236
6237
6238
6239
6240
6241
6242
6243
6244
6245
6246
6247
6248
6249
6250
6251
6252
6253
6254
6255
6256
6257
6258
6259
6260
6261
6262
6263
6264
6265
6266
6267
6268
6269

6270
6271
6272
6273
6274
6275
6276
6277
6278
6279
6280
6281
6282
6283
6284
6285
6286
6287
6288
6289
6290
6291
6292
6293
6294
6295
6296
6297
6298
6299
6300
6301
6302
6303
6304
6305
6306
6307
6308
6309
6310
6311
6312
6313
6314
6315
6316
6317
6318
6319
6320
6321
6322
6323
6324
6325
6326

6327
6328
6329
6330
6331
6332
6333
6334
6335
6336
6337
6338
6339
6340
6341
6342
6343
6344
6345
6346
6347
6348
6349
6350
6351
6352
6353
6354
6355
6356
6357
6358
6359
6360
6361
6362
6363
6364
6365
6366
6367
6368
6369
6370
6371
6372
6373
6374
6375
6376
6377
6378
6379
6380
6381
6382
6383

```
6555
6556
6557
6558
6559
6560
6561
6562
6563
6564
6565
6566
6567
6568
6569
6570
6571
6572
6573
6574
6575
6576
6577
6578
6579
6580
6581
6582
Success!
```

Following performing this request to get the Spotify ID for each of our Charting tracks, we knew that it was important to export our data as a csv so that we could save our work. To run over 6000 search requests via the Sptify API, it took over 40 mins. So we felt it was a good idea to save our spot in this way. We also decided to define an index this dataframe so that when we developed our relational database between the Charts/Spotify_ID and the Audio Features (coming up soon) dataframe, that we would have unique keys. Because for example some searches failed, there will be multiple NA's in the Spotify ID column not making it unique.

In [ ]:
```
billboard_top_df["Spotify_id"] = track_id_list # Create new column in bi
llboard_top_df dataframe with the list of Spotify IDs
billboard_top_df['I'] = billboard_top_df.index # Create Index column
billboard_top_df.to_csv("Top Songs All Time.csv", sep=',') # Export pand
as dataframe to csv file so that for loop does not have to be run each t
ime kernel is restarted
```

In the process of saving our spot via a csv, we must also reimport the csv file. That is done and the columns are described here.

In [42]:
```
billboard_top_df = pd.read_csv('Top Songs All Time.csv', sep=",", na_values = "NA") # Read in csv file Top Songs All Time.csv
billboard_top_df['I'] = billboard_top_df.index
billboard_top_df = billboard_top_df[["I", "Year", "Rank", "Artist", "Track", "Spotify_id"]] # Selects columns of billboard_top_df dataframe
billboard_top_df.head()
```

Out[42]:

	I	Year	Rank	Artist	Track	Spotify_id
0	0	1941	1	Glenn Miller	Chattanooga Choo Choo	1XogctkYSgvS6R4zSKRLkQ
1	1	1941	2	Sammy Kaye	Daddy	7CUYHcu0RnbOnMz4RuN07w
2	2	1941	3	Artie Shaw	Stardust	0ZciyueIR3Jl1gh4lbiLsk
3	3	1941	4	The Andrews Sisters	Boogie Woogie Bugle Boy	2nDURQ5XAIIbzAmy1F1yKm
4	4	1941	5	Jimmy Dorsey	Amapola (Pretty Little Poppy)	5Zq51T66500WOLxf3KErak

Now that we have the unique Spotify ID's for our charting tracks, we can not perform a Spotify Audio Features request. The response object of this request contains many different variables numerically measuring different aspects of the song from it's acousticness to it's key. Performing this request was pretty straight forward. The Spotify ID just was tacked onto the even of the resource in the request and the access token just had to be provided in a cetain format for the headers. We had to be smart about writing this function to make sure that an inputed NA Spotify Id would be handled correctly. We've been indexing our relational database by an index following the order of the year and rank. Without proper handling of NA inputs, the dataframe of all the audio features for all our tracks could fail to have 1-1 relation as we planned

In [10]:
```
def audiofeatures(track_id):
    '''
    The following function takes an input of a Spotify ID. It uses the S
potify ID to perform a Spotify request to
    get the audio feature of the specific track.
    '''
    creds = opencreds() # Use the open creds function to access the cred
s JSON file
    audiofeaturesnull = {'acousticness': None,
                         'analysis_url': None,
                         'danceability': None,
                         'duration_ms': None,
                         'energy': None,
                         'id': None,
                         'instrumentalness': None,
                         'key': None,
                         'liveness': None,
                         'loudness': None,
                         'mode': None,
                         'speechiness': None,
                         'tempo': None,
                         'time_signature': None,
                         'track_href': None,
                         'type': None,
                         'uri': None,
                         'valence': None} # Null dictionary for null Spo
tify IDs or unsuccessful requests
    if track_id == "NA" or type(track_id) == float: # If statement to ch
eck if Spotify ID is null
        audiofeatures = audiofeaturesnull # Define audiofeatures as audi
ofeaturesnull dictionary
    else: # Else statment for when track_id is not null
        protocol = 'https' # Protocol of the Spotify Audio Features requ
est
        location = 'api.spotify.com' # Location of the Spotify Audio Fea
tures request
        resource = 'v1/audio-features/'+track_id # Resource of the Spoti
fy Audio Features request including the inputed Spotify ID
        headers_dic = {'authorization': "Bearer " + getToken()} # Define
 headers of Spotify search request utilizing the getToken function to re
fresh authorization
        url = buildURL(protocol, location, resource) # Define url of Spo
tify Audio Features request
        resp = requests.get(url, headers=headers_dic) # Perform Spotify
 Audio Features request
        if resp.status_code != 200: # If statement to test if Audio Feat
ures request is unsuccessful
            audiofeatures = audiofeaturesnull # Define audiofeatures as
 audiofeaturesnull dictionary
        else: # Else statement for when request is successful
            audiofeatures = resp.json() # Define audiofeatures as respon
se JSON file
    return audiofeatures # Return audiofeatures dictionary
```

Now that we have have the audiofeatures function written, we want to run it for every item in our billboard_top_df table. To achieve this, a for loop is writen in the range of the length or the billboard_top_df dataframe. We create a list of dictionariees in this loop. To monitor progress, we went ahead and printed the number row that the loop is on and the 'Success' on completion of the loop.

In [11]:
```
audio_features_list = [] # Create audio_features_list list
for i in range (len(billboard_top_df)): # For loop to append list of dic
tionary responses of audio features request
    print(i) # Print i to monitor progress
    track_id = billboard_top_df["Spotify_id"][i] # Define Spotify ID fro
m billboard_top_df dataframe
    audiofeatures = audiofeatures(track_id) # Run audiofeatures function
 to get JSON response
    audio_features_list.append(audiofeatures) # Append audio_features_li
st with audiofeatures response
print("Success!") # Print "Success!" on completion
```

1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196

1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652

1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937

1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994

2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108

2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336

2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621

2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792

2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906

2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963

3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419

3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476

3591
3592
3593
3594
3595
3596
3597
3598
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3640
3641
3642
3643
3644
3645
3646
3647

3705
3706
3707
3708
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3760
3761

3933
3934
3935
3936
3937
3938
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989

4104
4105
4106
4107
4108
4109
4110
4111
4112
4113
4114
4115
4116
4117
4118
4119
4120
4121
4122
4123
4124
4125
4126
4127
4128
4129
4130
4131
4132
4133
4134
4135
4136
4137
4138
4139
4140
4141
4142
4143
4144
4145
4146
4147
4148
4149
4150
4151
4152
4153
4154
4155
4156
4157
4158
4159
4160

4503
4504
4505
4506
4507
4508
4509
4510
4511
4512
4513
4514
4515
4516
4517
4518
4519
4520
4521
4522
4523
4524
4525
4526
4527
4528
4529
4530
4531
4532
4533
4534
4535
4536
4537
4538
4539
4540
4541
4542
4543
4544
4545
4546
4547
4548
4549
4550
4551
4552
4553
4554
4555
4556
4557
4558
4559

5073
5074
5075
5076
5077
5078
5079
5080
5081
5082
5083
5084
5085
5086
5087
5088
5089
5090
5091
5092
5093
5094
5095
5096
5097
5098
5099
5100
5101
5102
5103
5104
5105
5106
5107
5108
5109
5110
5111
5112
5113
5114
5115
5116
5117
5118
5119
5120
5121
5122
5123
5124
5125
5126
5127
5128
5129

5130
5131
5132
5133
5134
5135
5136
5137
5138
5139
5140
5141
5142
5143
5144
5145
5146
5147
5148
5149
5150
5151
5152
5153
5154
5155
5156
5157
5158
5159
5160
5161
5162
5163
5164
5165
5166
5167
5168
5169
5170
5171
5172
5173
5174
5175
5176
5177
5178
5179
5180
5181
5182
5183
5184
5185
5186

5244
5245
5246
5247
5248
5249
5250
5251
5252
5253
5254
5255
5256
5257
5258
5259
5260
5261
5262
5263
5264
5265
5266
5267
5268
5269
5270
5271
5272
5273
5274
5275
5276
5277
5278
5279
5280
5281
5282
5283
5284
5285
5286
5287
5288
5289
5290
5291
5292
5293
5294
5295
5296
5297
5298
5299
5300

5472
5473
5474
5475
5476
5477
5478
5479
5480
5481
5482
5483
5484
5485
5486
5487
5488
5489
5490
5491
5492
5493
5494
5495
5496
5497
5498
5499
5500
5501
5502
5503
5504
5505
5506
5507
5508
5509
5510
5511
5512
5513
5514
5515
5516
5517
5518
5519
5520
5521
5522
5523
5524
5525
5526
5527
5528

5529
5530
5531
5532
5533
5534
5535
5536
5537
5538
5539
5540
5541
5542
5543
5544
5545
5546
5547
5548
5549
5550
5551
5552
5553
5554
5555
5556
5557
5558
5559
5560
5561
5562
5563
5564
5565
5566
5567
5568
5569
5570
5571
5572
5573
5574
5575
5576
5577
5578
5579
5580
5581
5582
5583
5584
5585

5700
5701
5702
5703
5704
5705
5706
5707
5708
5709
5710
5711
5712
5713
5714
5715
5716
5717
5718
5719
5720
5721
5722
5723
5724
5725
5726
5727
5728
5729
5730
5731
5732
5733
5734
5735
5736
5737
5738
5739
5740
5741
5742
5743
5744
5745
5746
5747
5748
5749
5750
5751
5752
5753
5754
5755
5756

5757
5758
5759
5760
5761
5762
5763
5764
5765
5766
5767
5768
5769
5770
5771
5772
5773
5774
5775
5776
5777
5778
5779
5780
5781
5782
5783
5784
5785
5786
5787
5788
5789
5790
5791
5792
5793
5794
5795
5796
5797
5798
5799
5800
5801
5802
5803
5804
5805
5806
5807
5808
5809
5810
5811
5812
5813

5871
5872
5873
5874
5875
5876
5877
5878
5879
5880
5881
5882
5883
5884
5885
5886
5887
5888
5889
5890
5891
5892
5893
5894
5895
5896
5897
5898
5899
5900
5901
5902
5903
5904
5905
5906
5907
5908
5909
5910
5911
5912
5913
5914
5915
5916
5917
5918
5919
5920
5921
5922
5923
5924
5925
5926
5927

6156
6157
6158
6159
6160
6161
6162
6163
6164
6165
6166
6167
6168
6169
6170
6171
6172
6173
6174
6175
6176
6177
6178
6179
6180
6181
6182
6183
6184
6185
6186
6187
6188
6189
6190
6191
6192
6193
6194
6195
6196
6197
6198
6199
6200
6201
6202
6203
6204
6205
6206
6207
6208
6209
6210
6211
6212

6213
6214
6215
6216
6217
6218
6219
6220
6221
6222
6223
6224
6225
6226
6227
6228
6229
6230
6231
6232
6233
6234
6235
6236
6237
6238
6239
6240
6241
6242
6243
6244
6245
6246
6247
6248
6249
6250
6251
6252
6253
6254
6255
6256
6257
6258
6259
6260
6261
6262
6263
6264
6265
6266
6267
6268
6269

6384
6385
6386
6387
6388
6389
6390
6391
6392
6393
6394
6395
6396
6397
6398
6399
6400
6401
6402
6403
6404
6405
6406
6407
6408
6409
6410
6411
6412
6413
6414
6415
6416
6417
6418
6419
6420
6421
6422
6423
6424
6425
6426
6427
6428
6429
6430
6431
6432
6433
6434
6435
6436
6437
6438
6439
6440

6441
6442
6443
6444
6445
6446
6447
6448
6449
6450
6451
6452
6453
6454
6455
6456
6457
6458
6459
6460
6461
6462
6463
6464
6465
6466
6467
6468
6469
6470
6471
6472
6473
6474
6475
6476
6477
6478
6479
6480
6481
6482
6483
6484
6485
6486
6487
6488
6489
6490
6491
6492
6493
6494
6495
6496
6497

6498
6499
6500
6501
6502
6503
6504
6505
6506
6507
6508
6509
6510
6511
6512
6513
6514
6515
6516
6517
6518
6519
6520
6521
6522
6523
6524
6525
6526
6527
6528
6529
6530
6531
6532
6533
6534
6535
6536
6537
6538
6539
6540
6541
6542
6543
6544
6545
6546
6547
6548
6549
6550
6551
6552
6553
6554

```
6555
6556
6557
6558
6559
6560
6561
6562
6563
6564
6565
6566
6567
6568
6569
6570
6571
6572
6573
6574
6575
6576
6577
6578
6579
6580
6581
6582
Success!
```

With the new audio_features_list, we created a pandas dataframe of the list of dictionaries. Then we exported the data to save it.

In [ ]:
```
features_df = pd.DataFrame(audio_features_list) # Create pandas datafram
e from audio_features_list
features_df.to_csv("features.csv", sep=',') # Export features_df datafra
me to csv file
```

With a csv file, we can reimport it at anytime. We reimport here, reformat properly and rename the key column. The reason for this rename will come down the line when it comes to creating a SQL table. The key column doesnt play nice with it.

In [62]:
```
features_df = pd.read_csv('features.csv', sep=",", na_values = "NA") # R
ead in csv file features.csv
features_df= features_df.drop(['Unnamed: 0'], axis = 1) # Remove 'Unname
d: 0' from pandas output to csv of original index
features_df = features_df.rename(columns={'key': 'key_'}) # Rename key v
ariable because SQL wont take key as variable
features_df['I'] = features_df.index
features_df.head()
```

Out[62]:

	acousticness	analysis_url	danceability	duration_ms	energy	
0	0.947	https://api.spotify.com/v1/audio-analysis/1Xog...	0.597	206213.0	0.310	1Xogc
1	0.916	https://api.spotify.com/v1/audio-analysis/0Zci...	0.580	463907.0	0.162	0Zciyu
2	0.357	https://api.spotify.com/v1/audio-analysis/2nDU...	0.718	138573.0	0.593	2nDU
3	0.943	https://api.spotify.com/v1/audio-analysis/5Zq5...	0.596	179640.0	0.284	5Zq51
4	0.539	https://api.spotify.com/v1/audio-analysis/3ou9...	0.608	223040.0	0.405	3ou9r

Creating a Relational Database with the Data

Now that we have webscrapped our data, used the Spotify search requst to get the Spotify ID, and then used the ID to run a request for the audio features, we've got the data that we need. We have a dataframe of the charts of time with the index, year, rank, artist, and song. There is a one to one relation between the charts table and the the audio features table. Now we can use sqlalchemy to put these tables into MySQL to do the heavy lifting of the data. Our goal is to observe trends in the data, so with these tables, we plan on joining, groupping by amoung other operations to observe the data.

We first created a dropcreate function to create a fresh new table based on our inputs of the table just to make things easy in the database.

In [26]:
```
def dropcreate(connection, create, table_name):
    '''
    The followng function takes an input of connection of my MySQL serve
r, a create string and a table name for the
    new table. The function checks to see if the table name input is alr
eady in place, drops it if it is, then
    creates the table. The create input should contain the specification
s for the table.
    '''
    if engine.has_table(table_name): # If statement to check if table al
ready exists and drops the table if so
        drop = 'DROP TABLE'+" "+table_name # Define drop string
        connection.execute(drop) # Executes drop string
    connection.execute(create) # Executes creation of table
```

Next we had to set up our database using our own written database_setup function. We need this done to connect to our personal SQL server.

In [53]:
```
creds = opencreds() # Use the open creds function to access the creds JS
ON file
database = creds['MySQL']["user"] # Define database from creds
user = creds['MySQL']['user'] # Define username from creds
password = creds['MySQL']['password'] # Define password from creds
engine, connection, cstring = database_setup(user, password, database) #
 run the database_setup function to set up SQL database
```

Using the above defined dropcreate function, lets create a new table in our database based on the columns in the billboard_top_df pandas dataframe. Then using a for loop for every item in the billboard_top_df dataframe, we will insert the row into our SQL Table. Again, in our for loop over every row getting inserted to our SQL table, we will print the row index to monitor progress. Then on completion, we print "Success!"

In [56]:
```
table_name = "Top_100_Songs_All_Time" # Define new table name
create = """CREATE TABLE Top_100_Songs_All_Time(
                I  INT,
                Year INT,
                Rank  INT,
                Artist CHAR(100),
                Track CHAR(100),
                Spotify_id CHAR(25),
                PRIMARY KEY(I))""" # Create string of table info for dro
pcreate function
dropcreate(connection, create, table_name) # Use drop create function to
 create table for billboard_top_df dataframe
```

In [57]:
```
table_name = "Top_100_Songs_All_Time" # Define new table name
billboard_top_df = billboard_top_df.replace(np.nan, "NA")
for i in range(len(billboard_top_df)):
    print(i)
    insert = 'INSERT INTO Top_100_Songs_All_Time (I, Year, Rank, Artist,
 Track, Spotify_id) VALUES '+str(tuple(billboard_top_df.iloc[i]))
    connection.execute(insert)
print("Success!")
```

969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025

1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310

1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424

1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652

1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766

1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880

1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937

1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994

2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165

2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906

2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963

2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020

3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362

3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3530
3531
3532
3533

3534
3535
3536
3537
3538
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590

3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3699
3700
3701
3702
3703
3704

4047
4048
4049
4050
4051
4052
4053
4054
4055
4056
4057
4058
4059
4060
4061
4062
4063
4064
4065
4066
4067
4068
4069
4070
4071
4072
4073
4074
4075
4076
4077
4078
4079
4080
4081
4082
4083
4084
4085
4086
4087
4088
4089
4090
4091
4092
4093
4094
4095
4096
4097
4098
4099
4100
4101
4102
4103

4104
4105
4106
4107
4108
4109
4110
4111
4112
4113
4114
4115
4116
4117
4118
4119
4120
4121
4122
4123
4124
4125
4126
4127
4128
4129
4130
4131
4132
4133
4134
4135
4136
4137
4138
4139
4140
4141
4142
4143
4144
4145
4146
4147
4148
4149
4150
4151
4152
4153
4154
4155
4156
4157
4158
4159
4160

4161
4162
4163
4164
4165
4166
4167
4168
4169
4170
4171
4172
4173
4174
4175
4176
4177
4178
4179
4180
4181
4182
4183
4184
4185
4186
4187
4188
4189
4190
4191
4192
4193
4194
4195
4196
4197
4198
4199
4200
4201
4202
4203
4204
4205
4206
4207
4208
4209
4210
4211
4212
4213
4214
4215
4216
4217

4218
4219
4220
4221
4222
4223
4224
4225
4226
4227
4228
4229
4230
4231
4232
4233
4234
4235
4236
4237
4238
4239
4240
4241
4242
4243
4244
4245
4246
4247
4248
4249
4250
4251
4252
4253
4254
4255
4256
4257
4258
4259
4260
4261
4262
4263
4264
4265
4266
4267
4268
4269
4270
4271
4272
4273
4274

4446
4447
4448
4449
4450
4451
4452
4453
4454
4455
4456
4457
4458
4459
4460
4461
4462
4463
4464
4465
4466
4467
4468
4469
4470
4471
4472
4473
4474
4475
4476
4477
4478
4479
4480
4481
4482
4483
4484
4485
4486
4487
4488
4489
4490
4491
4492
4493
4494
4495
4496
4497
4498
4499
4500
4501
4502

4617
4618
4619
4620
4621
4622
4623
4624
4625
4626
4627
4628
4629
4630
4631
4632
4633
4634
4635
4636
4637
4638
4639
4640
4641
4642
4643
4644
4645
4646
4647
4648
4649
4650
4651
4652
4653
4654
4655
4656
4657
4658
4659
4660
4661
4662
4663
4664
4665
4666
4667
4668
4669
4670
4671
4672
4673

4731
4732
4733
4734
4735
4736
4737
4738
4739
4740
4741
4742
4743
4744
4745
4746
4747
4748
4749
4750
4751
4752
4753
4754
4755
4756
4757
4758
4759
4760
4761
4762
4763
4764
4765
4766
4767
4768
4769
4770
4771
4772
4773
4774
4775
4776
4777
4778
4779
4780
4781
4782
4783
4784
4785
4786
4787

4788
4789
4790
4791
4792
4793
4794
4795
4796
4797
4798
4799
4800
4801
4802
4803
4804
4805
4806
4807
4808
4809
4810
4811
4812
4813
4814
4815
4816
4817
4818
4819
4820
4821
4822
4823
4824
4825
4826
4827
4828
4829
4830
4831
4832
4833
4834
4835
4836
4837
4838
4839
4840
4841
4842
4843
4844

4902
4903
4904
4905
4906
4907
4908
4909
4910
4911
4912
4913
4914
4915
4916
4917
4918
4919
4920
4921
4922
4923
4924
4925
4926
4927
4928
4929
4930
4931
4932
4933
4934
4935
4936
4937
4938
4939
4940
4941
4942
4943
4944
4945
4946
4947
4948
4949
4950
4951
4952
4953
4954
4955
4956
4957
4958

4959
4960
4961
4962
4963
4964
4965
4966
4967
4968
4969
4970
4971
4972
4973
4974
4975
4976
4977
4978
4979
4980
4981
4982
4983
4984
4985
4986
4987
4988
4989
4990
4991
4992
4993
4994
4995
4996
4997
4998
4999
5000
5001
5002
5003
5004
5005
5006
5007
5008
5009
5010
5011
5012
5013
5014
5015

5016
5017
5018
5019
5020
5021
5022
5023
5024
5025
5026
5027
5028
5029
5030
5031
5032
5033
5034
5035
5036
5037
5038
5039
5040
5041
5042
5043
5044
5045
5046
5047
5048
5049
5050
5051
5052
5053
5054
5055
5056
5057
5058
5059
5060
5061
5062
5063
5064
5065
5066
5067
5068
5069
5070
5071
5072

5244
5245
5246
5247
5248
5249
5250
5251
5252
5253
5254
5255
5256
5257
5258
5259
5260
5261
5262
5263
5264
5265
5266
5267
5268
5269
5270
5271
5272
5273
5274
5275
5276
5277
5278
5279
5280
5281
5282
5283
5284
5285
5286
5287
5288
5289
5290
5291
5292
5293
5294
5295
5296
5297
5298
5299
5300

5301
5302
5303
5304
5305
5306
5307
5308
5309
5310
5311
5312
5313
5314
5315
5316
5317
5318
5319
5320
5321
5322
5323
5324
5325
5326
5327
5328
5329
5330
5331
5332
5333
5334
5335
5336
5337
5338
5339
5340
5341
5342
5343
5344
5345
5346
5347
5348
5349
5350
5351
5352
5353
5354
5355
5356
5357

5358
5359
5360
5361
5362
5363
5364
5365
5366
5367
5368
5369
5370
5371
5372
5373
5374
5375
5376
5377
5378
5379
5380
5381
5382
5383
5384
5385
5386
5387
5388
5389
5390
5391
5392
5393
5394
5395
5396
5397
5398
5399
5400
5401
5402
5403
5404
5405
5406
5407
5408
5409
5410
5411
5412
5413
5414

5472
5473
5474
5475
5476
5477
5478
5479
5480
5481
5482
5483
5484
5485
5486
5487
5488
5489
5490
5491
5492
5493
5494
5495
5496
5497
5498
5499
5500
5501
5502
5503
5504
5505
5506
5507
5508
5509
5510
5511
5512
5513
5514
5515
5516
5517
5518
5519
5520
5521
5522
5523
5524
5525
5526
5527
5528

5529
5530
5531
5532
5533
5534
5535
5536
5537
5538
5539
5540
5541
5542
5543
5544
5545
5546
5547
5548
5549
5550
5551
5552
5553
5554
5555
5556
5557
5558
5559
5560
5561
5562
5563
5564
5565
5566
5567
5568
5569
5570
5571
5572
5573
5574
5575
5576
5577
5578
5579
5580
5581
5582
5583
5584
5585

5586
5587
5588
5589
5590
5591
5592
5593
5594
5595
5596
5597
5598
5599
5600
5601
5602
5603
5604
5605
5606
5607
5608
5609
5610
5611
5612
5613
5614
5615
5616
5617
5618
5619
5620
5621
5622
5623
5624
5625
5626
5627
5628
5629
5630
5631
5632
5633
5634
5635
5636
5637
5638
5639
5640
5641
5642

5643
5644
5645
5646
5647
5648
5649
5650
5651
5652
5653
5654
5655
5656
5657
5658
5659
5660
5661
5662
5663
5664
5665
5666
5667
5668
5669
5670
5671
5672
5673
5674
5675
5676
5677
5678
5679
5680
5681
5682
5683
5684
5685
5686
5687
5688
5689
5690
5691
5692
5693
5694
5695
5696
5697
5698
5699

5700
5701
5702
5703
5704
5705
5706
5707
5708
5709
5710
5711
5712
5713
5714
5715
5716
5717
5718
5719
5720
5721
5722
5723
5724
5725
5726
5727
5728
5729
5730
5731
5732
5733
5734
5735
5736
5737
5738
5739
5740
5741
5742
5743
5744
5745
5746
5747
5748
5749
5750
5751
5752
5753
5754
5755
5756

5757
5758
5759
5760
5761
5762
5763
5764
5765
5766
5767
5768
5769
5770
5771
5772
5773
5774
5775
5776
5777
5778
5779
5780
5781
5782
5783
5784
5785
5786
5787
5788
5789
5790
5791
5792
5793
5794
5795
5796
5797
5798
5799
5800
5801
5802
5803
5804
5805
5806
5807
5808
5809
5810
5811
5812
5813

5985
5986
5987
5988
5989
5990
5991
5992
5993
5994
5995
5996
5997
5998
5999
6000
6001
6002
6003
6004
6005
6006
6007
6008
6009
6010
6011
6012
6013
6014
6015
6016
6017
6018
6019
6020
6021
6022
6023
6024
6025
6026
6027
6028
6029
6030
6031
6032
6033
6034
6035
6036
6037
6038
6039
6040
6041

6099
6100
6101
6102
6103
6104
6105
6106
6107
6108
6109
6110
6111
6112
6113
6114
6115
6116
6117
6118
6119
6120
6121
6122
6123
6124
6125
6126
6127
6128
6129
6130
6131
6132
6133
6134
6135
6136
6137
6138
6139
6140
6141
6142
6143
6144
6145
6146
6147
6148
6149
6150
6151
6152
6153
6154
6155

6156
6157
6158
6159
6160
6161
6162
6163
6164
6165
6166
6167
6168
6169
6170
6171
6172
6173
6174
6175
6176
6177
6178
6179
6180
6181
6182
6183
6184
6185
6186
6187
6188
6189
6190
6191
6192
6193
6194
6195
6196
6197
6198
6199
6200
6201
6202
6203
6204
6205
6206
6207
6208
6209
6210
6211
6212

6270
6271
6272
6273
6274
6275
6276
6277
6278
6279
6280
6281
6282
6283
6284
6285
6286
6287
6288
6289
6290
6291
6292
6293
6294
6295
6296
6297
6298
6299
6300
6301
6302
6303
6304
6305
6306
6307
6308
6309
6310
6311
6312
6313
6314
6315
6316
6317
6318
6319
6320
6321
6322
6323
6324
6325
6326

6498
6499
6500
6501
6502
6503
6504
6505
6506
6507
6508
6509
6510
6511
6512
6513
6514
6515
6516
6517
6518
6519
6520
6521
6522
6523
6524
6525
6526
6527
6528
6529
6530
6531
6532
6533
6534
6535
6536
6537
6538
6539
6540
6541
6542
6543
6544
6545
6546
6547
6548
6549
6550
6551
6552
6553
6554

```
6555
6556
6557
6558
6559
6560
6561
6562
6563
6564
6565
6566
6567
6568
6569
6570
6571
6572
6573
6574
6575
6576
6577
6578
6579
6580
6581
6582
```

Now that we have the billboard_top_df in SQL, we can carry out a similar process to insert the features data into a new table and then inserting the data from the feature dataframe using a loop

In [65]:
```
table_name = "features" # Define new table name
create = """CREATE TABLE features(
                I INT,
                acousticness DECIMAL(4,3),
                analysis_url CHAR(100),
                danceability DECIMAL(4,3),
                duration_ms INT,
                energy DECIMAL(4,3),
                id CHAR(25),
                instrumentalness DECIMAL(7,6),
                key_ DECIMAL(2,1),
                liveness DECIMAL(4,3),
                loudness DECIMAL(6,3),
                mode DECIMAL(2,1),
                speechiness DECIMAL(5,4),
                tempo DECIMAL(6,3),
                time_signature DECIMAL(3,1),
                track_href CHAR(100),
                type CHAR(100),
                uri CHAR(100),
                valence DECIMAL(4,3),
                PRIMARY KEY(I))""" # Create string of table info for dro
pcreate function
dropcreate(connection, create, table_name) # Use drop create function to
 create table for billboard_top_df dataframe
```

In [ ]:
```
table_name = "features" # Define new table name
features_df = features_df.replace(np.nan, "NA")
for i in range(len(features_df)):
    print(i)
    insert = '''INSERT INTO features (acousticness, analysis_url, dancea
bility, duration_ms, energy, id, instrumentalness, key_, liveness, loudn
ess, mode, speechiness, tempo, time_signature, track_href, type, uri, va
lence, I) VALUES
                '''+str(tuple(features_df.iloc[i]))
    connection.execute(insert)
print("Success!")
```

1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367

1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709

1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823

1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937

1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994

1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051

2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108

2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222

2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450

2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849

3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077

3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362

3762
3763
3764
3765
3766
3767
3768
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818

3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3870
3871
3872
3873
3874
3875

3876
3877
3878
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3889
3890
3891
3892
3893
3894
3895
3896
3897
3898
3899
3900
3901
3902
3903
3904
3905
3906
3907
3908
3909
3910
3911
3912
3913
3914
3915
3916
3917
3918
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
3930
3931
3932

3990
3991
3992
3993
3994
3995
3996
3997
3998
3999
4000
4001
4002
4003
4004
4005
4006
4007
4008
4009
4010
4011
4012
4013
4014
4015
4016
4017
4018
4019
4020
4021
4022
4023
4024
4025
4026
4027
4028
4029
4030
4031
4032
4033
4034
4035
4036
4037
4038
4039
4040
4041
4042
4043
4044
4045
4046

4104
4105
4106
4107
4108
4109
4110
4111
4112
4113
4114
4115
4116
4117
4118
4119
4120
4121
4122
4123
4124
4125
4126
4127
4128
4129
4130
4131
4132
4133
4134
4135
4136
4137
4138
4139
4140
4141
4142
4143
4144
4145
4146
4147
4148
4149
4150
4151
4152
4153
4154
4155
4156
4157
4158
4159
4160

4161
4162
4163
4164
4165
4166
4167
4168
4169
4170
4171
4172
4173
4174
4175
4176
4177
4178
4179
4180
4181
4182
4183
4184
4185
4186
4187
4188
4189
4190
4191
4192
4193
4194
4195
4196
4197
4198
4199
4200
4201
4202
4203
4204
4205
4206
4207
4208
4209
4210
4211
4212
4213
4214
4215
4216
4217

```
4218
4219
4220
4221
4222
4223
4224
```

With that we have a relational database of two tables. One is the charted songs over the history of Billboard, the other is the audio features for those songs. From this point, we can use SQL to do the heavy lifting when it comes to performing operations on the data. Moving forward, we will see how we can perform operations on our collected data to see trends in charting hits over time.

```
In [ ]:
```