Wrangling the Billboard Top 100

September 14, 2023

[3]:
```
!pip install --upgrade torch torchvision
!pip install word2number
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix, accuracy_score,␣
 ↪classification_report,precision_score, recall_score, f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction import DictVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.metrics.pairwise import cosine_similarity
import torch
import torchvision
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder
import os
import glob
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk import pos_tag
import itertools
import networkx as nx
from collections import Counter
from textblob import TextBlob

pd.set_option('display.max_columns', None)
```

```
pd.set_option('display.max_rows', None)
```

```
Requirement already satisfied: torch in
/Users/barnana/anaconda3/lib/python3.10/site-packages (2.0.1)
Requirement already satisfied: torchvision in
/Users/barnana/anaconda3/lib/python3.10/site-packages (0.15.2)
Requirement already satisfied: jinja2 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from torch) (3.1.2)
Requirement already satisfied: networkx in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from torch) (2.8.4)
Requirement already satisfied: typing-extensions in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from torch) (4.7.1)
Requirement already satisfied: filelock in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from torch) (3.9.0)
Requirement already satisfied: sympy in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from torch) (1.11.1)
Requirement already satisfied: requests in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from torchvision)
(2.28.1)
Requirement already satisfied: numpy in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from torchvision)
(1.25.1)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from torchvision) (9.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from jinja2->torch)
(2.1.1)
Requirement already satisfied: charset-normalizer<3,>=2 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
requests->torchvision) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
requests->torchvision) (2023.5.7)
Requirement already satisfied: idna<4,>=2.5 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
requests->torchvision) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
requests->torchvision) (1.26.14)
Requirement already satisfied: mpmath>=0.19 in
/Users/barnana/anaconda3/lib/python3.10/site-packages/mpmath-1.2.1-py3.10.egg
(from sympy->torch) (1.2.1)
Requirement already satisfied: word2number in
/Users/barnana/anaconda3/lib/python3.10/site-packages (1.1)
```

1 Wrangling the Billboard Top 100

Consider the data in billboard.csv containing every song to appear on the weekly Billboard Top 100 chart since 1958, up through the middle of 2021. Each row of this data corresponds to a single song in a single week. For our purposes, the relevant columns here are:

performer: who performed the song song: the title of the song year: year (1958 to 2021) week: chart week of that year (1, 2, etc) week_position: what position that song occupied that week on the Billboard top 100 chart. Use your skills in data wrangling and plotting to answer the following three questions.

[5]:
```
# Importing the data
billboard=pd.read_csv('billboard.csv')
```

1.0.1 Understanding the data

Looking at the dimensions of the data as well as the records for one specific randomly chosen song to understand each variable properly

[6]:
```
billboard.head()
```

[6]:

	Unnamed: 0	url	week_id	week_position	song	performer	song_id	instance	previous_week_position	peak_position	weeks_on_chart	year	week
0	1	http://www.billboard.com/charts/hot-100/1965-0…	7/17/1965	34	Don't Just Stand There	Patty Duke	Don't Just Stand TherePatty Duke	1	45.0	34	4	1965	29
1	2	http://www.billboard.com/charts/hot-100/1965-0…	7/24/1965	22	Don't Just Stand There	Patty Duke	Don't Just Stand TherePatty Duke	1	34.0	22	5	1965	30
2	3	http://www.billboard.com/charts/hot-100/1965-0…	7/31/1965	14	Don't Just Stand There	Patty Duke	Don't Just Stand TherePatty Duke	1	22.0	14	6	1965	31
3	4	http://www.billboard.com/charts/hot-100/1965-0…	8/7/1965	10	Don't Just Stand There	Patty Duke	Don't Just Stand TherePatty Duke	1	14.0	10	7	1965	32
4	5	http://www.billboard.com/charts/hot-100/1965-0…	8/14/1965	8	Don't Just Stand There	Patty Duke	Don't Just Stand TherePatty Duke	1	10.0	8	8	1965	33

[7]:
```
billboard.shape
```

[7]:
```
(327895, 13)
```

[8]:
```
billboard['week_id'].unique().size
```

[8]:
```
3279
```

[9]:
```
billboard[billboard['song']=="whoknows"]
```

[9]:

	Unnamed: 0	url	\
153364	153365	http://www.billboard.com/charts/hot-100/2004-0…	
153365	153366	http://www.billboard.com/charts/hot-100/2004-0…	
153366	153367	http://www.billboard.com/charts/hot-100/2004-0…	
153367	153368	http://www.billboard.com/charts/hot-100/2004-0…	
153368	153369	http://www.billboard.com/charts/hot-100/2004-0…	
153369	153370	http://www.billboard.com/charts/hot-100/2004-0…	
153370	153371	http://www.billboard.com/charts/hot-100/2004-0…	
153371	153372	http://www.billboard.com/charts/hot-100/2004-0…	
153372	153373	http://www.billboard.com/charts/hot-100/2004-0…	
153373	153374	http://www.billboard.com/charts/hot-100/2004-0…	
153374	153375	http://www.billboard.com/charts/hot-100/2004-0…	
153375	153376	http://www.billboard.com/charts/hot-100/2004-0…	
153376	153377	http://www.billboard.com/charts/hot-100/2004-0…	

	week_id	week_position	song	performer	song_id	instance	\
153364	4/24/2004	75	whoknows	Musiq	whoknowsMusiq	1	
153365	5/1/2004	69	whoknows	Musiq	whoknowsMusiq	1	
153366	5/8/2004	65	whoknows	Musiq	whoknowsMusiq	1	
153367	5/15/2004	72	whoknows	Musiq	whoknowsMusiq	1	
153368	5/22/2004	72	whoknows	Musiq	whoknowsMusiq	1	
153369	5/29/2004	85	whoknows	Musiq	whoknowsMusiq	1	
153370	6/5/2004	86	whoknows	Musiq	whoknowsMusiq	1	
153371	6/12/2004	92	whoknows	Musiq	whoknowsMusiq	1	
153372	6/19/2004	99	whoknows	Musiq	whoknowsMusiq	1	
153373	6/26/2004	92	whoknows	Musiq	whoknowsMusiq	1	
153374	7/3/2004	99	whoknows	Musiq	whoknowsMusiq	1	
153375	7/10/2004	92	whoknows	Musiq	whoknowsMusiq	1	
153376	7/17/2004	100	whoknows	Musiq	whoknowsMusiq	1	

	previous_week_position	peak_position	weeks_on_chart	year	week
153364	NaN	75	1	2004	17
153365	75.0	69	2	2004	18
153366	69.0	65	3	2004	19
153367	65.0	65	4	2004	20
153368	72.0	65	5	2004	21
153369	72.0	65	6	2004	22

153370	85.0	65	7	2004	23
153371	86.0	65	8	2004	24
153372	92.0	65	9	2004	25
153373	99.0	65	10	2004	26
153374	92.0	65	11	2004	27
153375	99.0	65	12	2004	28
153376	92.0	65	13	2004	29

Part A: Make a table of the top 10 most popular songs since 1958, as measured by the total number of weeks that a song spent on the Billboard Top 100. Note that these data end in week 22 of 2021, so the most popular songs of 2021 will not have up-to-the-minute data; please send our apologies to The Weeknd.

Your table should have 10 rows and 3 columns: performer, song, and count, where count represents the number of weeks that song appeared in the Billboard Top 100. Make sure the entries are sorted in descending order of the count variable, so that the more popular songs appear at the top of the table. Give your table a short caption describing what is shown in the table.

(Note: you'll want to use both performer and song in any group_by operations, to account for the fact that multiple unique songs can share the same title.)

[10]:
```
# Get each unique combination of song and performer using groupby and find out how many times it appears in the data
# This tells us how many times this song appeared on the Billboard Top 100
top_songs=billboard.groupby(['song','performer']).size()

# Sort by count values in descending order and keep the top 10 rows
top_10_songs=top_songs.sort_values(ascending=False).head(10)

# Convert to a dataframe
top_10_songs=pd.DataFrame(top_10_songs, columns=['count']).reset_index()
top_10_songs
```

[10]:

	song	performer	count
0	Radioactive	Imagine Dragons	87
1	Sail	AWOLNATION	79
2	I'm Yours	Jason Mraz	76
3	Blinding Lights	The Weeknd	76
4	How Do I Live	LeAnn Rimes	69
5	Party Rock Anthem	LMFAO Featuring Lauren Bennett & GoonRock	68
6	Counting Stars	OneRepublic	68
7	Foolish Games/You Were Meant For Me	Jewel	65
8	Rolling In The Deep	Adele	65
9	Before He Cheats	Carrie Underwood	64

```
3                                  The Weeknd     76
4                                  LeAnn Rimes     69
5  LMFAO Featuring Lauren Bennett & GoonRock     68
6                                  OneRepublic     68
7                                        Jewel     65
8                                        Adele     65
9                            Carrie Underwood     64
```

Part B: Is the "musical diversity" of the Billboard Top 100 changing over time? Let's find out. We'll measure the musical diversity of given year as the number of unique songs that appeared in the Billboard Top 100 that year. Make a line graph that plots this measure of musical diversity over the years. The x axis should show the year, while the y axis should show the number of unique songs appearing at any position on the Billboard Top 100 chart in any week that year. For this part, please filter the data set so that it excludes the years 1958 and 2021, since we do not have complete data on either of those years. Give the figure an informative caption in which you explain what is shown in the figure and comment on any interesting trends you see.

There are number of ways to accomplish the data wrangling here. For example, you could use two distinct sets of data-wrangling steps. The first set of steps would get you a table that counts the number of times that a given song appears on the Top 100 in a given year. The second set of steps operate on the result of the first set of steps; it would count the number of unique songs that appeared on the Top 100 in each year, irrespective of how many times it had appeared.

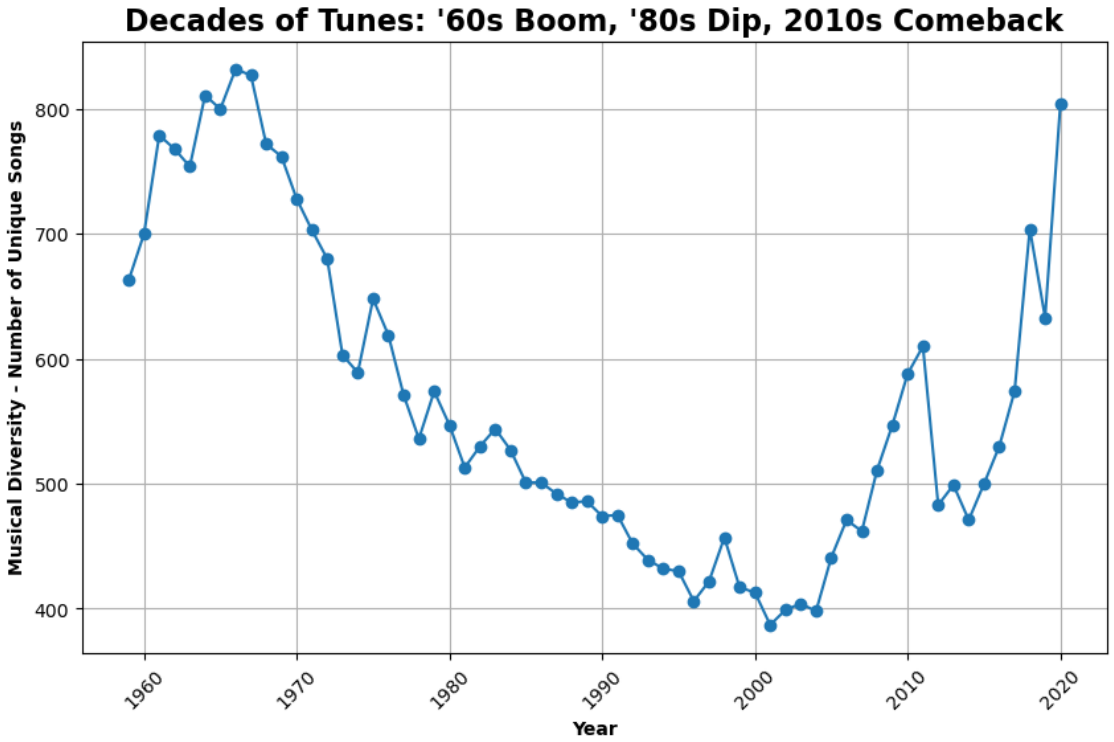
[11]:
```
pd.set_option('display.max_rows', None)

# Grouping the data by year, song, and performer, and then counting the number␣
 ↪of unique combinations
# Using both songs and performers to get distinct songs as recommended in Part A
diversity= billboard.groupby(['year', 'song', 'performer']).size().
 ↪reset_index(name='count')

# Grouping again by year to get the total number of unique combinations of␣
 ↪songs and performers for each year
diversity_year = diversity.groupby('year')['count'].count().reset_index()

# Renaming the columns for clarity
diversity_year.columns = ['year', 'number of unique songs']

# Remove the years 1958 and 2021
diversity_year=diversity_year[~diversity_year['year'].isin([1958,2021])]

diversity_year
```

[11]:
```
   year  number of unique songs
1  1959                     663
2  1960                     700
3  1961                     779
4  1962                     768
```

5	1963	754
6	1964	811
7	1965	800
8	1966	832
9	1967	827
10	1968	772
11	1969	762
12	1970	728
13	1971	703
14	1972	680
15	1973	603
16	1974	589
17	1975	648
18	1976	619
19	1977	571
20	1978	536
21	1979	574
22	1980	547
23	1981	513
24	1982	530
25	1983	544
26	1984	527
27	1985	501
28	1986	501
29	1987	492
30	1988	485
31	1989	486
32	1990	474
33	1991	475
34	1992	452
35	1993	439
36	1994	432
37	1995	430
38	1996	406
39	1997	422
40	1998	457
41	1999	417
42	2000	413
43	2001	387
44	2002	399
45	2003	404
46	2004	398
47	2005	441
48	2006	471
49	2007	462
50	2008	511
51	2009	547

52	2010	588
53	2011	610
54	2012	483
55	2013	499
56	2014	471
57	2015	500
58	2016	530
59	2017	574
60	2018	704
61	2019	633
62	2020	804

[12]:
```
# Plotting the line graph for musical diversity over the years
plt.figure(figsize=(10, 6))
plt.plot(diversity_year['year'], diversity_year['number of unique songs'],␣
 ↪marker='o', linestyle='-')
plt.xlabel('Year',fontdict={'weight': 'bold'})
plt.ylabel('Musical Diversity - Number of Unique Songs',fontdict={'weight':␣
 ↪'bold'})
plt.title("Decades of Tunes: '60s Boom, '80s Dip, 2010s␣
 ↪Comeback",fontdict={'weight': 'bold', 'size': 16})
plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```



Trends and Insights from the data:

- **‘60s Surge:** From 1959 to the mid-1960s, there’s a clear rise in musical diversity, likely driven by cultural shifts and the emergence of genres like rock ’n’ roll.
- **Decades of Decline:** Post the mid-1960s peak, there’s a gradual decline until the late 1980s. Industry consolidation and genre saturation might explain this trend.
- **Stable Late 20th Century:** The period from the late ’80s to early 2000s shows relative stability in song diversity, reflecting the mainstream dominance of genres like pop and hip-hop.
- **Digital Boost:** Sporadic peaks in the 2000s align with the rise of digital music platforms, making production and distribution more accessible.
- **Late 2010s Revival:** A notable increase in musical diversity from the late 2010s into 2020 can be attributed to platforms like Spotify and YouTube, democratizing music creation and sharing.

Part C: Let’s define a “ten-week hit” as a single song that appeared on the Billboard Top 100 for at least ten weeks. There are 19 artists in U.S. musical history since 1958 who have had at least 30 songs that were “ten-week hits.” Make a bar plot for these 19 artists, showing how many ten-week hits each one had in their musical career. Give the plot an informative caption in which you explain what is shown.

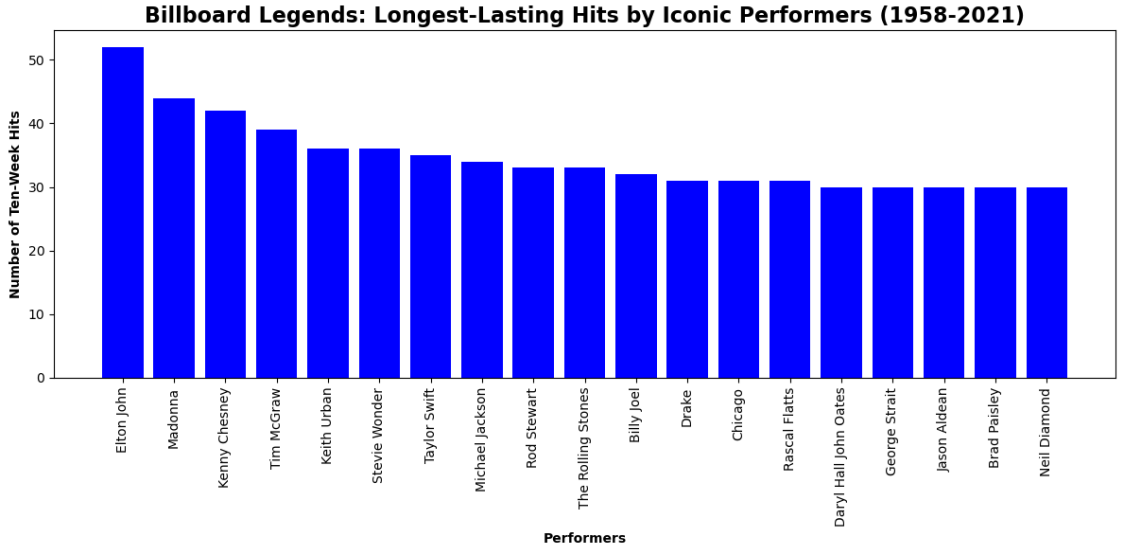
Notes:

You might find this easier to accomplish in two distinct sets of data wrangling steps. Make sure that the individuals names of the artists are readable in your plot, and that they’re not all jumbled together. If you find that your plot isn’t readable with vertical bars, you can add a coord_flip() layer to your plot to make the bars (and labels) run horizontally instead. By default a bar plot will order the artists in alphabetical order. This is acceptable to turn in. But if you’d like to order them according to some other variable, you can use the fct_reorder function, described in this blog post. This is optional.

[13]:
```
# Perform the previous steps as you've done before
twh = billboard.groupby(['song', 'performer'])['week_id'].count().
 ↪reset_index(name='count of weeks')
twh = twh[twh['count of weeks'] >= 10]
twh = twh.groupby('performer')['song'].count().reset_index(name='count of␣
 ↪ten-week hits')

# Filter for performers who had at least 30 songs that were ten-week hits
twh_performers = twh[twh['count of ten-week hits'] >= 30].reset_index()

# Keep only 'performer' and 'count of ten-week hits' columns
twh_performers = twh_performers[['performer', 'count of ten-week hits']].
 ↪sort_values(by='count of ten-week hits', ascending=False)
```

[14]:
```
# Create the bar plot
plt.figure(figsize=(12, 6))
```

```
plt.bar(twh_performers['performer'], twh_performers['count of ten-week hits'],␣
 ↪color='b')
plt.xlabel('Performers',fontdict={'weight': 'bold'})
plt.ylabel('Number of Ten-Week Hits',fontdict={'weight': 'bold'})
plt.title("Billboard Legends: Longest-Lasting Hits by Iconic Performers␣
 ↪(1958-2021)",fontdict={'weight': 'bold', 'size': 16})
plt.xticks(rotation=90)
plt.tight_layout()

# Show the plot
plt.show()
```



[ ]: