



Welcome to the Midterm Project! Projects in DSC 10 are similar in format to homeworks, but are different in a few key ways. First, a project is comprehensive, meaning that it draws upon everything we've learned this quarter so far. Second, since problems can vary quite a bit in difficulty, some problems will be worth more points than others. Finally, in a project, the problems are more open-ended; they will usually ask for some result, but won't tell you what method should be used to get it. There might be several equally-valid approaches, and several steps might be necessary. This is closer to how data science is done in "real life."

It is important that you **start early** on the project! It will take the place of a homework in the week that it is due, but you should also expect it to take longer than a homework, and you want to leave time to get help in [office hours](#).

You are especially encouraged to **find a partner** to work through the project with. If you work in a pair, you must follow the [Project Partner Guidelines](#) on the course website. In particular, you must work together at the same time, and you are not allowed to split up the project and each work on certain problems. If you work with a partner, only one of you needs to upload your notebook to Gradescope; after uploading, you'll see an option to add the other partner to the submission.

**Important:** The `otter` tests don't usually tell you that your answer is correct. More often, they help catch basic mistakes. It's up to you to ensure that your answer is correct. If you're not sure, ask someone (not for the answer, but for some guidance about your approach). Directly sharing answers between groups is not okay, but discussing problems with the course staff or with other students is encouraged.

**Avoid looping through DataFrames unless instructed to do so.** Loops in Python are slow, and looping through DataFrames should usually be avoided in favor of the DataFrame methods we've learned in class, which are much faster. **Please do not import any additional packages.** You don't need them, and our autograder may not be able to run your code if you do.

As you work through this project, there are a few resources you may want to have open:

- [DSC 10 Reference Sheet](#)
- `babypandas` [notes](#)
- Other links in the [Resources](#) and [Debugging](#) tabs of the course website

Start early, good luck, and let's begin! 🏃

```
In [1]: # Please don't change this cell, but make sure to run it.
import babypandas as bpd
import numpy as np
from IPython.display import HTML, display, IFrame, YouTubeVideo, Markdown, clear_output
import ipywidgets as widgets
```

```
import matplotlib.pyplot as plt
plt.style.use('ggplot')
plt.rcParams["figure.figsize"] = (10, 5)

import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

import otter
import numbers
grader = otter.Notebook()

def play_spotify(uri):
    code = uri[uri.rfind(':')+1:]
    src = f"https://open.spotify.com/embed/track/{code}"
    width = 400
    height = 75
    display(IFrame(src, width, height))
```

## Outline

The project is divided into four main sections, each of which contains several questions. Use the outline below to help you quickly navigate to the part of the project you're working on. Questions are worth one point each, unless they contain a ★★ next to them, in which case they are worth two points (e.g. **Question 0.3. ★★**). You can expect questions worth two points to be longer and more challenging than questions worth one point.

- [Welcome to tswift](#), it's been waitin' for you! 🤖
- [Section 1: Data Visualization](#) 🧐
- [Section 2: Song Recommender](#) 🎧
- [Section 3: Lyric Searcher](#) 🔍
- [Section 4: Keywords](#) 🔑

There's also a [Taylor Swift Emoji Quiz](#) 🎯 at the end of the project, just for fun. Try to identify the Taylor Swift song based on an emoji description, and see how many you can get!

## Welcome to tswift, it's been waitin' for you! 🤖

([return to the outline](#))

In case you've been living under a rock, allow us to introduce you to Taylor Swift, a famous singer, songwriter, and cultural icon. She has set all kinds of records in the music industry, including earning the most American Music Awards in history. She is also the number one most-streamed artist on Spotify, a digital music streaming service.

With its origins in country music, Taylor Swift's style has evolved a lot since her 2006 debut album. In her most recent global concert tour, *The Eras Tour*, she guides fans through these

musical "eras" in a three-hour performance which fans are paying thousands of dollars to attend. The film version of *The Eras Tour* became the highest-grossing concert film in history in just one weekend when it was released in October 2023.



Chances are, you probably have heard some of Taylor Swift's songs; you might even know the words to all of them. In this project, we'll look at Taylor Swift's songs through the lens of data science. We have data on both the lyrics and audio qualities of Taylor Swift's musical body of work.

The datasets we will use contain all songs on each of Taylor Swift's [eleven studio albums](#). We've chosen to use deluxe or extended versions of these albums when available, to include more songs, but we've eliminated duplicate versions of songs, such as acoustic versions and remixes. We've also chosen to include the rerecorded "[Taylor's Version](#)" when available. Our datasets don't include Taylor Swift songs that were released as part of movie soundtracks, live recordings, holiday specials, or through any other mechanism.

We'll work with two DataFrames throughout the project:

- The `lyrics` DataFrame contains the lyrics of each Taylor Swift song. The data in `lyrics` comes from [Genius](#), "the world's biggest collection of song lyrics and crowdsourced musical knowledge."
- The `tswift` DataFrame contains information about the audio features of each song. The data in `tswift` comes from [Spotify](#).

Let's start by reading in these DataFrames and taking a look around. Run the cell below to load `lyrics`.

```
In [2]: lyrics = bpd.read_csv('data/lyrics.csv')
        lyrics
```

Out [2]:

	Album	Song	Lyrics
0	The Tortured Poets Department	But Daddy I Love Him	I forget how the West was won\nI forget if thi...
1	The Tortured Poets Department	Cassandra	I was in my new house placing daydreams\nPatch...
2	The Tortured Poets Department	Chloe or Sam or Sophia or Marcus	Your hologram stumbled into my apartment\nHand...
3	The Tortured Poets Department	Clara Bow	"You look like Clara Bow\nIn this light, remar...
4	The Tortured Poets Department	Down Bad	Did you really beam me up\nIn a cloud of spark...
...	...	...	...
224	Taylor Swift	Stay Beautiful	Cory's eyes are like a jungle\nHe smiles, it's...
225	Taylor Swift	Teardrops On My Guitar	Drew looks at me\nI fake a smile so he won't s...
226	Taylor Swift	The Outside	I didn't know what I would find\nWhen I went I...
227	Taylor Swift	Tied Together With A Smile	Seems the only one who doesn't see your beauty...
228	Taylor Swift	Tim McGraw	He said the way my blue eyes shined\nPut those...

229 rows × 3 columns

**Question 0.1.** Choose an appropriate index for `lyrics` and set the index to that column.

```
In [3]: lyrics = lyrics.set_index('Song')
lyrics
```

Out [3]:

Song	Album		Lyrics
But Daddy I Love Him	The Tortured Poets Department	I forget how the West was won\nI forget if thi...	
Cassandra	The Tortured Poets Department	I was in my new house placing daydreams\nPatch...	
Chloe or Sam or Sophia or Marcus	The Tortured Poets Department	Your hologram stumbled into my apartment\nHand...	
Clara Bow	The Tortured Poets Department	"You look like Clara Bow\nIn this light, remar...	
Down Bad	The Tortured Poets Department	Did you really beam me up\nIn a cloud of spark...	
...	...	...	
Stay Beautiful	Taylor Swift	Cory's eyes are like a jungle\nHe smiles, it's...	
Teardrops On My Guitar	Taylor Swift	Drew looks at me\nI fake a smile so he won't s...	
The Outside	Taylor Swift	I didn't know what I would find\nWhen I went I...	
Tied Together With A Smile	Taylor Swift	Seems the only one who doesn't see your beauty...	
Tim McGraw	Taylor Swift	He said the way my blue eyes shined\nPut those...	

229 rows × 2 columns

```
In [4]: grader.check("q0_1")
```

Out [4]: q0\_1 passed!

**Question 0.2.** Set `mastermind` to the lyrics of the song `'Mastermind'`. Compare what happens when you display the value of `mastermind` versus `print mastermind`.

```
In [5]: mastermind = lyrics.loc['Mastermind'].get('Lyrics')
mastermind
```

```
Out[5]: "Once upon a time, the planets and the fates\nAnd all the stars aligned\nYou a  
nd I ended up in the same room\nAt the same time\n\nAnd the touch of a hand li  
t the fuse\nOf a chain reaction of countermoves\nTo assess the equation of you  
\nCheckmate, I couldn't lose\n\nWhat if I told you none of it was accidental?  
\nAnd the first night that you saw me\nNothing was gonna stop me\nI laid the g  
roundwork, and then\nJust like clockwork\nThe dominoes cascaded in a line\nWha  
t if I told you I'm a mastermind?\nAnd now you're mine\nIt was all by design  
\n'Cause I'm a mastermind\n\nYou see, all the wisest women\nHad to do it this  
way\n'Cause we were born to be the pawn\nIn every lover's game\n\nIf you fail  
to plan, you plan to fail\nStrategy sets the scene for the tale\nI'm the wind  
in our free-flowing sails\nAnd the liquor in our cocktails\n\nWhat if I told y  
ou none of it was accidental?\nAnd the first night that you saw me\nI knew I w  
anted your body\nI laid the groundwork, and then\nJust like clockwork\nThe dom  
inoes cascaded in a line\nWhat if I told you I'm a mastermind?\nAnd now you're  
mine\nIt was all my design\n'Cause I'm a mastermind\n\nNo one wanted to play w  
ith me as a little kid\nSo I've been scheming like a criminal ever since\nTo m  
ake them love me and make it seem effortless\nThis is the first time I've felt  
the need to confess\nAnd I swear\nI'm only cryptic and Machiavellian\n'Cause I  
care\n\nSo I told you none of it was accidental\nAnd the first night that you  
saw me\nNothing was gonna stop me\nI laid the groundwork, and then\nSaw a wide  
smirk on your face\nYou knew the entire time\nYou knew that I'm a mastermind\nAnd now you're mine\nYeah, all you did was smile\n'Cause I'm a mastermind"
```

```
In [6]: grader.check("q0_2")
```

```
Out[6]: q0_2 passed!
```

Since the lyrics data is all text, you'll be working a lot with strings throughout this project. Make sure to review the [available string methods](#) so you know how to work with the lyrics data effectively.

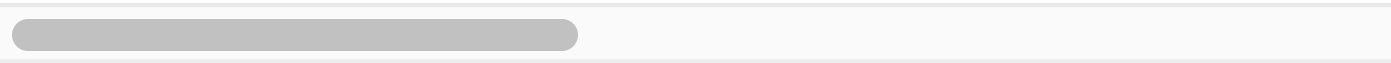
Next, let's look at the `tswift` DataFrame. Run the cell below to load in the DataFrame and take a look around. Songs are ordered by `'Popularity'`.

```
In [7]: tswift = bpd.read_csv('data/tswift.csv')  
tswift
```

Out [7]:

	URI	Album	Song Name	Popularity	Disc Number	Track Number	Explicit
0	1BxfuPKGuaTgP7aM0Bbdwr	Lover	Cruel Summer	90	1	2	False
1	2OzhQISqBEmt7hmkYxfT6m	The Tortured Poets Department	Fortnight (Ft. Post Malone)	86	1	1	False
2	1dGr1c8CrMLDpV6mPblmSI	Lover	Lover	85	1	3	False
3	3hUxzQpSfdDqwM3ZTFQY0K	folklore	august	85	1	8	False
4	4R2kfaDFhslZEMJqAFNpdd	folklore	cardigan	84	1	2	False
...	...	...	...	...	...	...	...
224	2ZoOmCSgj0ypVAmGd1ve4y	Taylor Swift	Stay Beautiful	50	1	8	False
225	6K0CJLVXqbGMeJSmJ4ENKK	Taylor Swift	Tied Together With A Smile	50	1	7	False
226	2QA3lixpRcKyOdG7XDzRgv	Taylor Swift	The Outside	49	1	6	False
227	5OOd01o2YS1QFwdpVLds3r	Taylor Swift	Invisible	49	1	13	False
228	1spLfUJxtyVyiKKTegQ2r4	Taylor Swift	A Perfectly Good Heart	48	1	14	False

229 rows x 20 columns



`tswift` contains a lot of information! We've used the [documentation](#) provided by Spotify to create the table below, which describes the columns present in `tswift` and what they represent. Note that many of these features (such as `'Valence'`) are defined and determined by Spotify. We have no way of knowing exactly how they determine the values of these audio features for each song, as their algorithms are proprietary.

Variable Name	Data Type	Explanation
'URI'	str	Unique identifier for the song in Spotify.
'Album'	str	Album name.
'Song Name'	str	Song name.
'Disc Number'	int	Disc number, usually 1 unless the album contains more than 1 disc.
'Track Number'	int	The number of the track on the specified disc.
'Popularity'	int	0 to 100 scale of the current popularity of the song.

Variable Name	Data Type	Explanation
'Explicit'	bool	True if the song contains explicit words, False otherwise.
'Danceability'	float	0 to 1 scale of how suitable a track is for dancing.
'Energy'	float	0 to 1 scale of a track's activity and intensity.
'Key'	int	The average key/pitch of a track, where 0 = C, 1 = C#/Db, 2 = D, and so on.
'Loudness'	float	The average loudness of a track, measured on a relative scale in decibels. Values typically range between -60 (softer) and 0 (louder).
'Mode'	int	Either 0 for a minor key, or 1 for a major key.
'Speechiness'	float	0 to 1 scale measuring the prevalence of spoken words.
'Acousticness'	float	0 to 1 scale measuring how likely a track is to be acoustic.
'Instrumentalness'	float	0 to 1 scale measuring how likely a track is to be instrumental (without vocals).
'Liveness'	float	0 to 1 scale measuring how likely a track is to have been recorded with a live audience.
'Valence'	float	0 to 1 scale of how positive or happy a track is.
'Tempo'	float	The estimated number of beats per minute.
'Duration_ms'	int	Length of song in milliseconds.
'Time Signature'	int	The number of beats in each bar (or measure).

One piece of information we'd like to have in `tswift`, which is currently missing, is the year in which each album was released. This variable would allow us to explore trends over time. The `albums` DataFrame contains the information we need. Run the next cell to load it in.

```
In [8]: albums = bpd.read_csv('data/albums.csv')
albums
```



Out [8]:

	Album	Release Date
0	Taylor Swift	October 24, 2006
1	Fearless	November 11, 2008
2	Speak Now	October 25, 2010
3	Red	October 22, 2012
4	1989	October 27, 2014
...	...	...
6	Lover	August 23, 2019
7	folklore	July 24, 2020
8	evermore	December 11, 2020
9	Midnights	October 21, 2022
10	The Tortured Poets Department	April 19, 2024

11 rows x 2 columns

**Question 0.3.** ★★ Add a column to `tswift` called `'Year'` that contains the year of each song's release, as an int, based on the data in `albums`.

**\*Note:** This problem has two stars because it's a multi-step, more challenging problem. Take it one step at a time. Feel free to create additional cells.

```
In [9]: def split_date(date):
        return int(date.split(", ")[1])
        Year = albums.get('Release Date').apply(split_date)
```

```
In [10]: album_with_year = albums.assign(Year = albums.get('Release Date').apply(split_date))
        album_with_year
```

Out [10]:

	Album	Year
0	Taylor Swift	2006
1	Fearless	2008
2	Speak Now	2010
3	Red	2012
4	1989	2014
...	...	...
6	Lover	2019
7	folklore	2020
8	evermore	2020
9	Midnights	2022
10	The Tortured Poets Department	2024

11 rows x 2 columns

```
In [11]: tswift = bpd.read_csv('data/tswift.csv')
tswift
```

Out [11]:

	URI	Album	Song Name	Popularity	Disc Number	Track Number	Explicit
0	1BxfuPKGuaTgP7aM0Bbdwr	Lover	Cruel Summer	90	1	2	False
1	2OzhQISqBEmt7hmkYxfT6m	The Tortured Poets Department	Fortnight (Ft. Post Malone)	86	1	1	False
2	1dGr1c8CrMLDpV6mPblmSI	Lover	Lover	85	1	3	False
3	3hUxzQpSfdDqwM3ZTFQY0K	folklore	august	85	1	8	False
4	4R2kfaDFhslZEMJqAFNpdd	folklore	cardigan	84	1	2	False
...	...	...	...	...	...	...	...
224	2ZoOmCSgj0ypVAmGd1ve4y	Taylor Swift	Stay Beautiful	50	1	8	False
225	6K0CJLVXqbGMeJSmJ4ENKK	Taylor Swift	Tied Together With A Smile	50	1	7	False
226	2QA3lixpRcKyOdG7XDzRgv	Taylor Swift	The Outside	49	1	6	False
227	5OOd01o2YS1QFwdpVLds3r	Taylor Swift	Invisible	49	1	13	False
228	1spLfUJxtyVyiKKTegQ2r4	Taylor Swift	A Perfectly Good Heart	48	1	14	False

229 rows x 20 columns

```
In [12]: tswift = tswift.merge(album_with_year, on='Album')
tswift
```

Out [12]:

	URI	Album	Song Name	Popularity	Disc Number	Track Number	Explicit	Danc
0	1BxfuPKGuaTgP7aM0Bbdwr	Lover	Cruel Summer	90	1	2	False	
1	1dGr1c8CrMLDpV6mPblmSI	Lover	Lover	85	1	3	False	
2	3RauEVgRgj1luWdJ9fDs70	Lover	The Man	79	1	4	False	
3	6RRNNciQGZEXnqk8SQ9yv5	Lover	You Need To Calm Down	77	1	14	False	
4	1fzAuUVbzlhZ1lJAx9PtY6	Lover	Daylight	77	1	18	False	
...	...	...	...	...	...	...	...	...
224	2ZoOmCSgj0ypVAmGd1ve4y	Taylor Swift	Stay Beautiful	50	1	8	False	
225	6K0CJLVXqbGMeJSmJ4ENKK	Taylor Swift	Tied Together With A Smile	50	1	7	False	
226	2QA3lixpRcKyOdG7XDzRgv	Taylor Swift	The Outside	49	1	6	False	
227	5OOd01o2YS1QFwdpVLds3r	Taylor Swift	Invisible	49	1	13	False	
228	1spLfUJxtyVyiKKTegQ2r4	Taylor Swift	A Perfectly Good Heart	48	1	14	False	

229 rows x 21 columns

In [13]: `grader.check("q0_3")`

Out [13]: **q0\_3** passed!

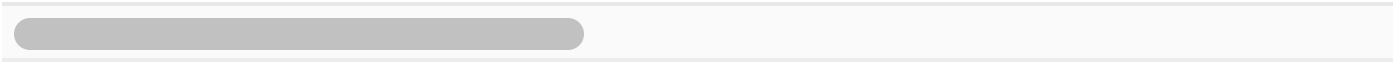
Now that `tswift` has all of the information we need, we'll set its index to `'URI'`, since we were told in the data description table that `'URI'` s are unique.

In [14]: `tswift = tswift.set_index('URI')`  
`tswift`

Out[14]:

URI	Album	Song Name	Popularity	Disc Number	Track Number	Explicit	Danceabi
1BxfuPKGuaTgP7aM0Bbdwr	Lover	Cruel Summer	90	1	2	False	0.9
1dGr1c8CrMLDpV6mPblmSI	Lover	Lover	85	1	3	False	0.9
3RauEVgRgj1luWdJ9fDs70	Lover	The Man	79	1	4	False	0.9
6RRNNciQGZEXnqk8SQ9yv5	Lover	You Need To Calm Down	77	1	14	False	0.9
1fzAuUVbzlhZ1IJAx9PtY6	Lover	Daylight	77	1	18	False	0.9
...	...	...	...	...	...	...	...
2ZoOmCSgj0ypVAmGd1ve4y	Taylor Swift	Stay Beautiful	50	1	8	False	0.9
6K0CJLVXqbGMeJSmJ4ENKK	Taylor Swift	Tied Together With A Smile	50	1	7	False	0.9
2QA3lixpRcKyOdG7XDzRgv	Taylor Swift	The Outside	49	1	6	False	0.9
50Od01o2YS1QFwdpVLds3r	Taylor Swift	Invisible	49	1	13	False	0.9
1spLfUJxtyVyiKKTegQ2r4	Taylor Swift	A Perfectly Good Heart	48	1	14	False	0.9

229 rows × 20 columns



Now we have our data in the format we need for the rest of the project. In the next two sections of the project, we'll work with the `tswift` DataFrame. Then we'll pivot to work with the `lyrics` DataFrame in the last two sections. Let's begin!

# Section 1: Data Visualization 🧐

And you just watched it happen.

[\(return to the outline\)](#)

In this section, we'll use the `tswift` DataFrame to create visualizations that will help us answer questions about Taylor Swift's music, including:

- How many songs did Taylor Swift release each year?

- How do different audio features such as 'Loudness' and 'Energy' relate to one another?
- Are Taylor Swift's songs generally more positive or more negative?
- Which Taylor Swift album has the most songs in a minor key?

**Question 1.1.** ★★ Let's start by determining how many songs Taylor Swift released each year. If we think of 'Year' as a numerical variable, it makes sense to visualize its distribution with a histogram. Create a density histogram showing the distribution of 'Year' in the `tswift` DataFrame.

To get the x-axis labels to display nicely, include the optional `xticks` argument in your call to `.plot`. This optional argument specifies where the x-axis labels should be placed. For this plot, set `xticks = np.arange(2006, 2027, 2)` to get tick marks at even-numbered years. Also use the optional argument `title` to give your histogram a meaningful title.

Set the bins such that each bin represents one year, from 2006 to 2024, inclusive. Be careful with the endpoints; in particular, **make sure 2023 and 2024 are not binned together**.

```
In [15]: tswift
```

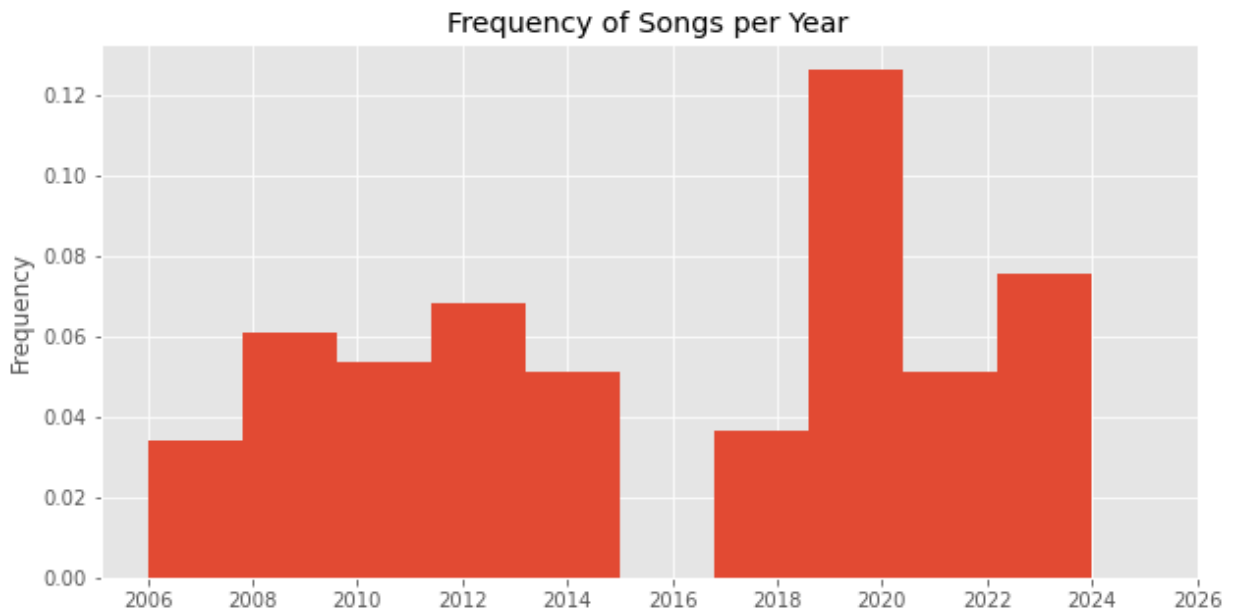
Out[15]:

	Album	Song Name	Popularity	Disc Number	Track Number	Explicit	Danceabi
URI							
1BxfuPKGuaTgP7aM0Bbdwr	Lover	Cruel Summer	90	1	2	False	0.9
1dGr1c8CrMLDpV6mPblmSI	Lover	Lover	85	1	3	False	0.9
3RauEVgRgj1luWdJ9fDs70	Lover	The Man	79	1	4	False	0.9
6RRNNciQGZEXnqk8SQ9yv5	Lover	You Need To Calm Down	77	1	14	False	0.9
1fzAuUVbzlhZ1lJAx9PtY6	Lover	Daylight	77	1	18	False	0.9
...	...	...	...	...	...	...	...
2ZoOmCSgj0ypVAmGd1ve4y	Taylor Swift	Stay Beautiful	50	1	8	False	0.9
6K0CJLVXqbGMeJSmJ4ENKK	Taylor Swift	Tied Together With A Smile	50	1	7	False	0.9
2QA3lixpRcKyOdG7XDzRgv	Taylor Swift	The Outside	49	1	6	False	0.9
50Od01o2YS1QFwdpVLds3r	Taylor Swift	Invisible	49	1	13	False	0.9
1spLfUJxtyVyiKKTegQ2r4	Taylor Swift	A Perfectly Good Heart	48	1	14	False	0.9

229 rows × 20 columns

```
In [16]: # Create your histogram here.
tswift.get('Year').plot(kind='hist', xticks = np.arange(2006, 2027, 2), density=0.01)

Out[16]: <AxesSubplot:title={'center': 'Frequency of Songs per Year'}, ylabel='Frequency'>
```



Use the plot to determine the proportion of songs released *before* 2010. After looking at the plot, store the proportion in the variable `before_2010` by manually typing it in, as a float, to the nearest two decimal places.

```
In [17]: # DOUBLE CHECK
before_2010 = 0.19
before_2010
```

```
Out[17]: 0.19
```

```
In [18]: grader.check("q1_1")
```

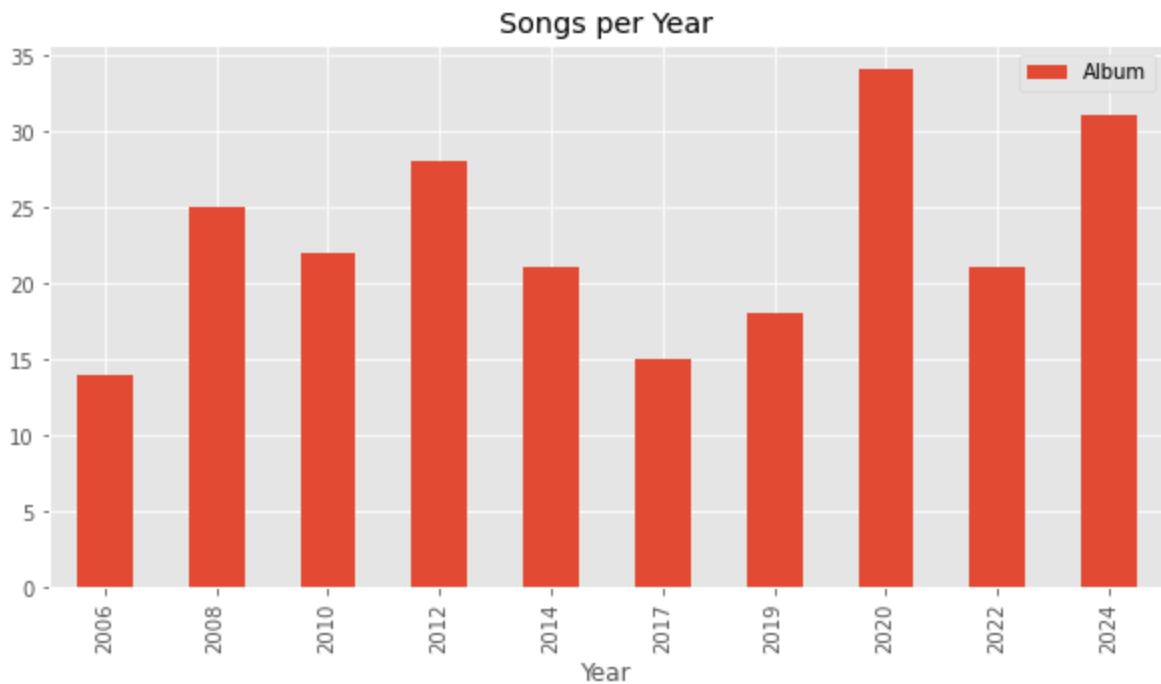
```
Out[18]: q1_1 passed!
```

**Question 1.2.** While `'Year'` can be a numerical variable, it can also be treated as a categorical variable, in which case it would be appropriate to visualize its distribution with a bar chart. Create a vertical bar chart showing the distribution of `'Year'`. Make sure to give your plot a meaningful title, and make sure the bars appear chronologically from left to right.

```
In [19]: # Create your vertical bar chart here.
tswift.groupby("Year").count().plot(kind='bar', y = 'Album', title = 'Songs per
```

```
Out[19]: <AxesSubplot:title={'center':'Songs per Year'}, xlabel='Year'>
```





Use the plot to determine the year with the most songs. After looking at the plot, store the year in the variable `year_with_most_songs` by manually typing it in, as an int.

```
In [20]: year_with_most_songs = 2020
         year_with_most_songs
```

Out[20]: 2020

```
In [21]: grader.check("q1_2")
```

Out[21]: **q1\_2** passed!

**Question 1.3.** Taylor Swift actually released two albums, sometimes called sister albums 🎵, in the `year_with_most_songs`. Use code to find out which two albums were released that year. Specifically, set `sister_albums` to an array containing the names of these albums as strings, in any order. Do not type in the album names manually; write code to get them for you.

**\*Hint:** The Series method `.unique()` might be helpful.

```
In [ ]:
```

```
In [22]: sister_albums = np.array(album_with_year[album_with_year.get('Year') == 2020].c
         sister_albums
```

Out[22]: array(['folklore', 'evermore'], dtype=object)

```
In [23]: grader.check("q1_3")
```

Out[23]: **q1\_3** passed!

**Question 1.4.** Create a visualization or write code to answer the following question:

Which album has the most songs?

Save the name of the album as `most_songs_album`. If you create a visualization to answer the question, it's fine to hardcode the answer as a string; if you write code to answer the question, it's fine to assign `most_songs_album` to an expression that evaluates to a string.

```
In [24]: most_songs_album = tswift.groupby("Album").count().sort_values(by = "Song Name")
most_songs_album
```

```
Out[24]: 'The Tortured Poets Department'
```

```
In [25]: grader.check("q1_4")
```

```
Out[25]: q1_4 passed!
```

The `'Popularity'` column in `tswift` contains a number, on a scale of 0 to 100, that ranks how popular a track currently is, relative to other tracks on Spotify. Every stream, save, share, like, and playlist recommendation contributes to a song's `'Popularity'`. Songs with a higher `'Popularity'` are more likely to be recommended to new listeners and added to algorithmically-generated playlists.

**Question 1.5.** What are the most and least popular Taylor Swift songs right now? Save your answers as `most_pop` and `least_pop`, respectively.

```
In [26]: most_pop = tswift.sort_values(by = "Popularity", ascending = False).get('Song Name')
least_pop = tswift.sort_values(by = "Popularity", ascending = True).get('Song Name')

print(f'The most popular Taylor Swift song right now is {most_pop}.')
print(f'The least popular Taylor Swift song right now is {least_pop}.')
```

The most popular Taylor Swift song right now is Cruel Summer.  
The least popular Taylor Swift song right now is A Perfectly Good Heart.

```
In [27]: grader.check("q1_5")
```

```
Out[27]: q1_5 passed!
```

**Question 1.6.** ★★ Create a DataFrame named `popularity_by_year` that is indexed by `'Year'` and has two columns:

- `'Max_Popularity'` should contain the highest popularity among all songs released that year.
- `'Median_Popularity'` should contain the median popularity of all songs released that year.

```
In [28]: #DOUBLE CHECK
popularity_by_year = bpd.DataFrame().assign(Max_Popularity = tswift.groupby("Year")
                                             Median_Popularity = tswift.groupby("Year")
                                             popularity_by_year
```

```
Out[28]:
```

	Max_Popularity	Median_Popularity
Year		
2006	68	52.0
2008	79	61.0
2010	77	66.0
2012	79	66.0
2014	76	68.0
2017	83	74.0
2019	90	73.0
2020	85	65.0
2022	82	69.0
2024	86	72.0

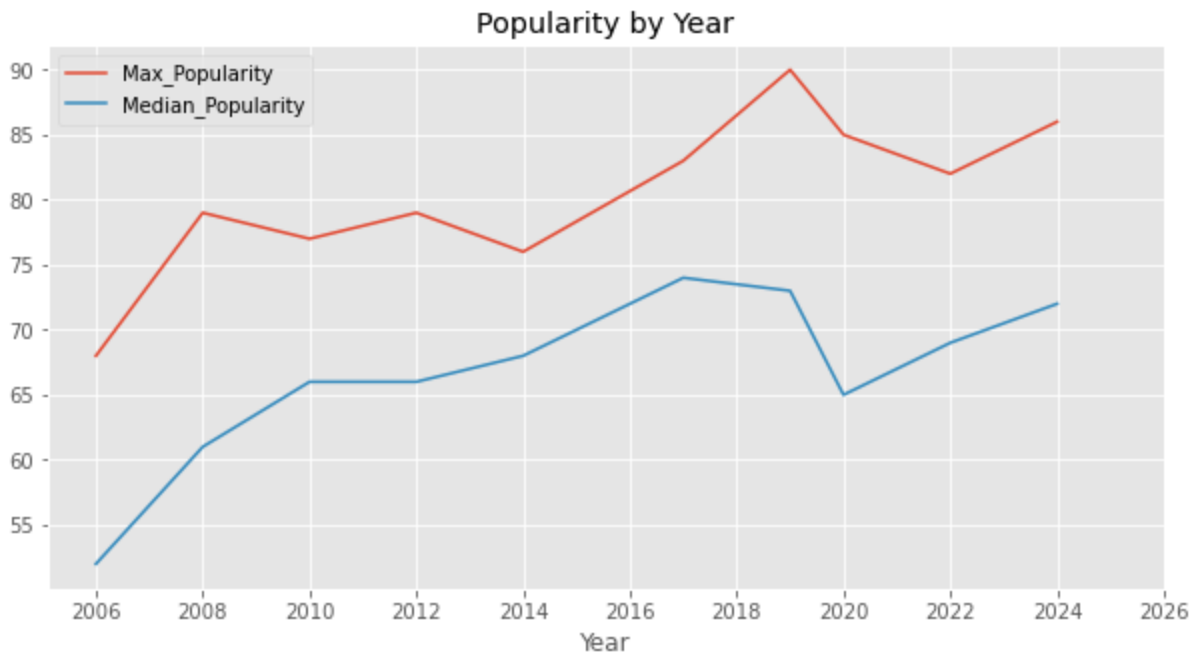
```
In [29]: grader.check("q1_6")
```

```
Out[29]: q1_6 passed!
```

**Question 1.7.** Create an overlaid line plot showing how the 'Max\_Popularity' and 'Median\_Popularity' are related to 'Year'. Give your plot an appropriate title and set `xticks` as you did in Question 1.1.

```
In [30]: # Create your plot here.
popularity_by_year.reset_index().plot(kind = "line", x = 'Year', xticks = np.arange(2006, 2024, 2))
```

```
Out[30]: <AxesSubplot:title={'center':'Popularity by Year'}, xlabel='Year'>
```

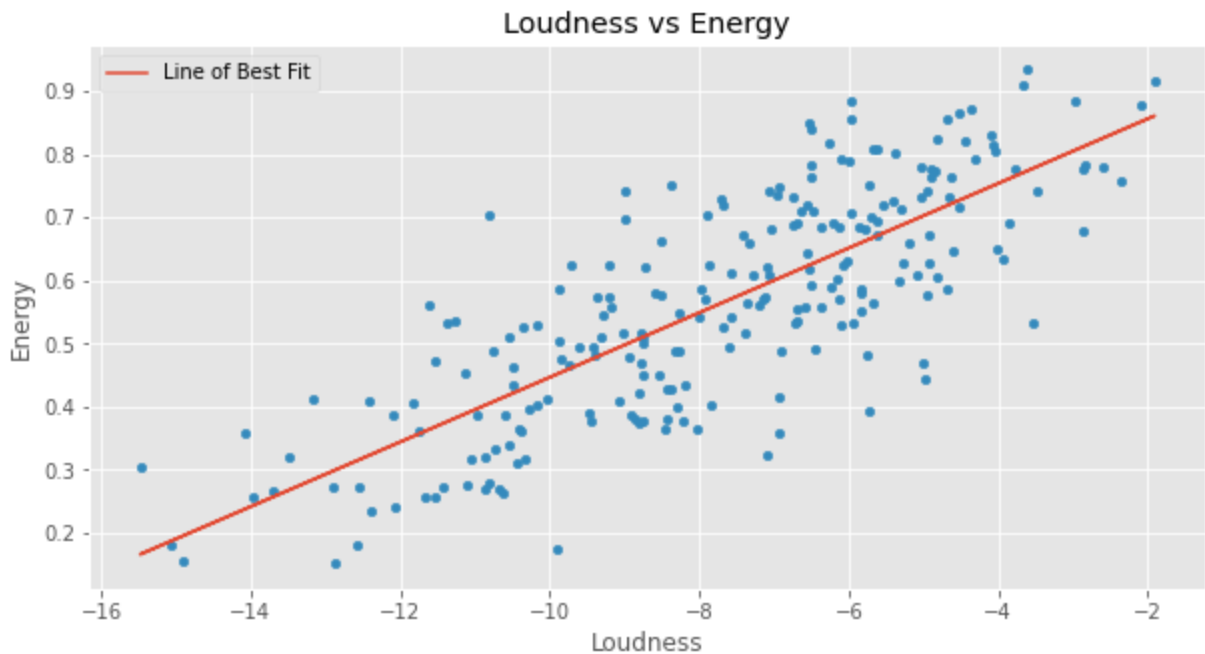


**Question 1.8.** Let's explore the relationship between different variables in the `tswift` dataset. Specifically:

- `'Loudness'` is the average loudness of a track measured on a relative scale in decibels. Values typically range between -60 (softer) and 0 (louder).
- `'Energy'` is a 0 to 1 scale of a track's activity and intensity, where higher values are more energetic.

In the cell below, create a plot that shows the relationship between `'Loudness'` (on the horizontal axis) and `'Energy'` (on the vertical axis) in Taylor Swift's songs. Include an appropriate title. We've included some additional code to draw the best-fitting line to describe the relationship between these variables. We'll learn more about best-fitting lines when we study regression later in the course!

```
In [31]: #DOUBLE CHECK
# Create your plot here.
tswift.get(["Loudness", "Energy"]).plot(kind = 'scatter', x = 'Loudness', y = 'Energy')
# The code below plots the line of best fit; do not alter it!
x = tswift.get('Loudness')
y = tswift.get('Energy')
a, b = np.polyfit(x, y, 1)
plt.plot(x, a * x + b, label='Line of Best Fit')
plt.legend()
plt.show()
```



What is the relationship between 'Loudness' and 'Energy' for Taylor Swift's songs? Set `q1_8` to either 1, 2, or 3, corresponding to your choice from the options below.

1. Louder songs tend to be lower in energy.
2. Louder songs tend to be higher in energy.
3. There is no clear relationship between 'Loudness' and 'Energy'.

In [32]: `q1_8 = 2`

In [33]: `grader.check("q1_8")`

Out [33]: **q1\_8** passed!

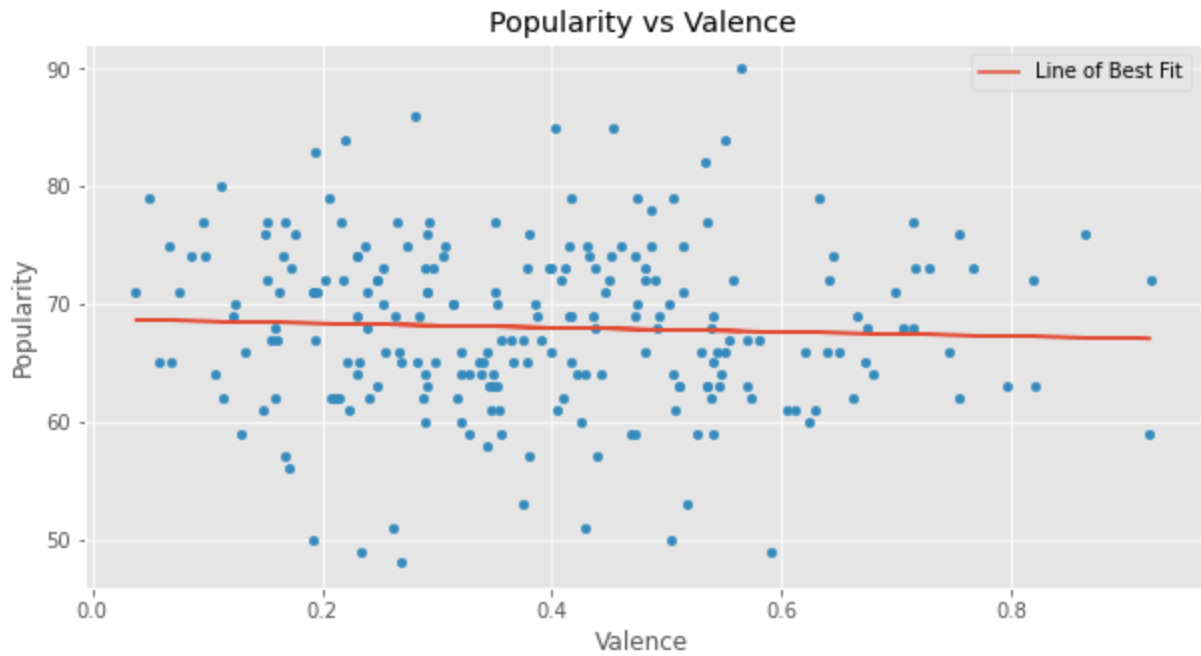
**Question 1.9.** Another one of Spotify's audio features is 'Valence', which measures the positivity, or "happiness", of a track on a 0 to 1 scale. Larger values correspond to more positive songs.

In the cell below, create a plot that shows the relationship between 'Valence' (on the horizontal axis) and 'Popularity' (on the vertical axis) in Taylor Swift's songs. Include an appropriate title. As in the previous question, we've included code to plot the best-fitting line.

```
In [34]: # Create your plot here.
tswift.get(["Valence", "Popularity"]).plot(kind = 'scatter', x = 'Valence', y = 'Popularity')

# The code below plots the line of best fit; do not alter it!
x = tswift.get('Valence')
y = tswift.get('Popularity')
a, b = np.polyfit(x, y, 1)
plt.plot(x, a * x + b, label='Line of Best Fit')
```

```
plt.legend()
plt.show()
```



What is the relationship between 'Valence' and 'Popularity' for Taylor Swift's songs? Set `q1_9` to either 1, 2, or 3, corresponding to your choice from the options below.

1. Positive songs tend to be less popular.
2. Positive songs tend to be more popular.
3. There is no clear relationship between 'Valence' and 'Popularity'.

```
In [35]: q1_9 = 3
```

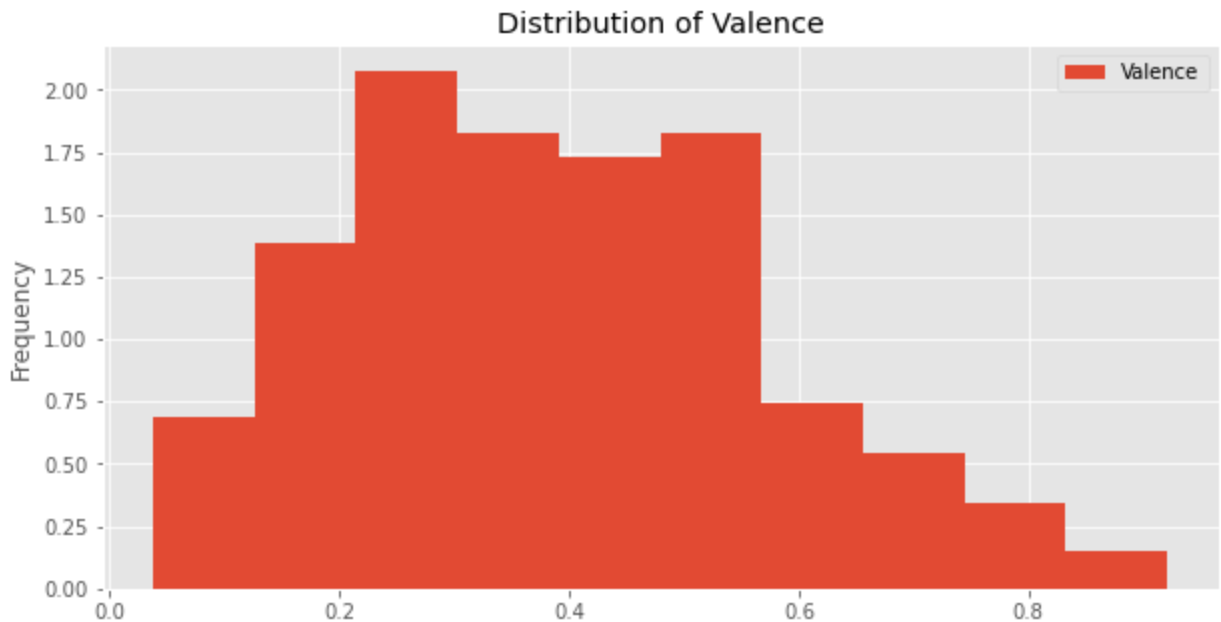
```
In [36]: grader.check("q1_9")
```

```
Out[36]: q1_9 passed!
```

**Question 1.10.** Create a plot that visualizes the distribution of 'Valence' for all of Taylor Swift's songs. Include an appropriate title.

```
In [37]: # Create your plot here.
tswift.get(["Valence", "Song Name"]).plot(kind = "hist", density = True, title
```

```
Out[37]: <AxesSubplot:title={'center': 'Distribution of Valence'}, ylabel='Frequency'>
```



Are Taylor Swift's songs more negative or positive on average? Set `q1_10` to either 1 or 2, corresponding to your choice from the options below.

1. Taylor Swift's songs are more negative on average.
2. Taylor Swift's songs are more positive on average.

In [38]: `q1_10 = 1`

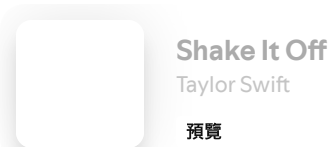
In [39]: `grader.check("q1_10")`

Out [39]: **q1\_10** passed!

In the `tswift` DataFrame, the `'Mode'` of a song is 1 if the song is written in a major key or 0 if it's written in a minor key. Generally, songs in a major key (e.g. G major) sound more upbeat, bright, and fun, while songs in a minor key (e.g. D minor) sound more dark, sad, or serious.

For example, the song `'Shake It Off'`, from the album `'1989'`, has a `'Mode'` of 1 because it was written in G major. Run the cell below and press play to hear a snippet of it right here in your notebook.

In [40]: `play_spotify('3fthfkkvy9av3q3uAGVf7U')`



On the other hand, the song `'Look What You Made Me Do'`, from the album `'Reputation'`, has a `'Mode'` of 0 because it was written in A minor. Again, run the cell below and press play to hear a snippet of it.

```
In [41]: play_spotify('1P17dC1amhFzptugyA07Il')
```



### Look What You Made Me Do

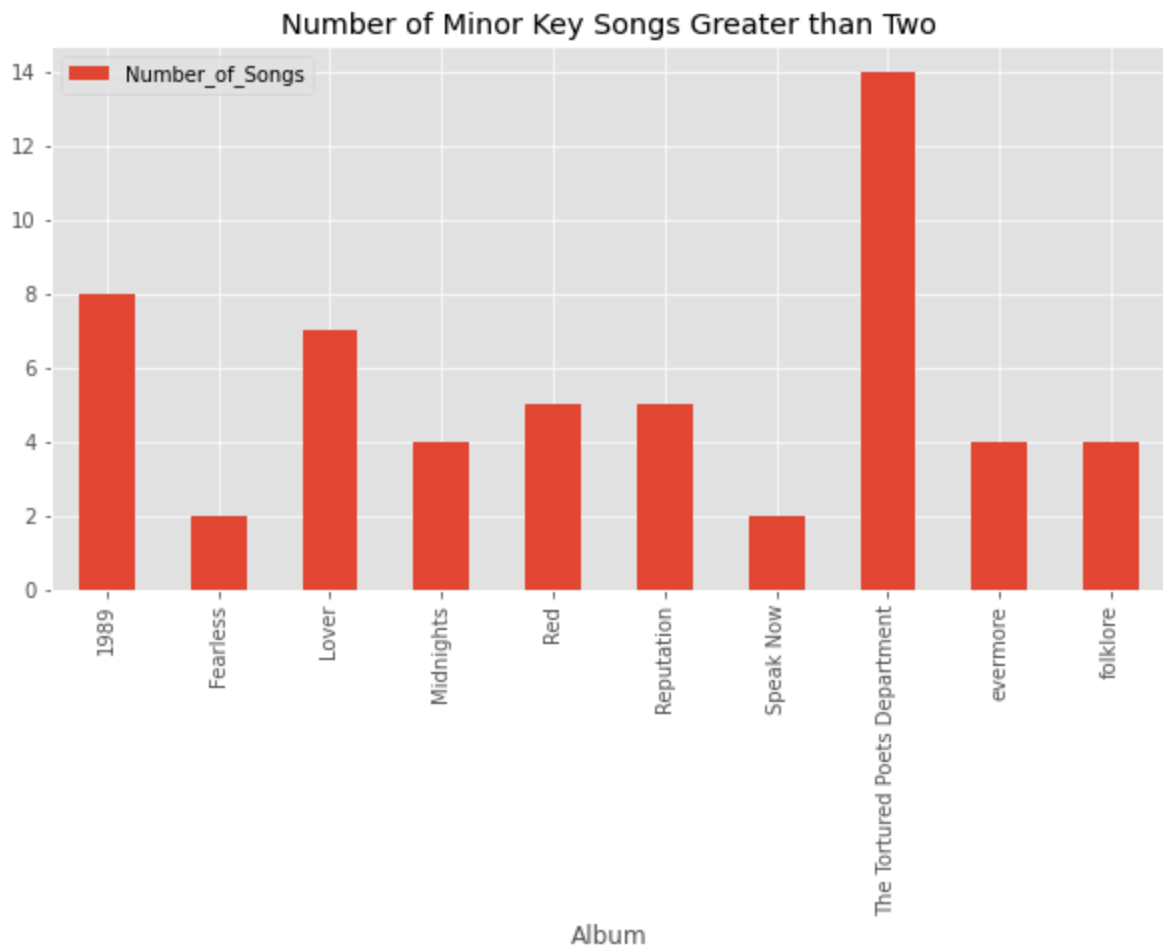
Taylor Swift

預覽

**Question 1.11.** ★★ Create a plot that shows the number of songs in a minor key on each album, but **only among albums with at least two songs in a minor key**. Make sure your plot has a title and a legend that accurately describe what is being shown.

```
In [42]: # Create your plot here.
query = tswift[tswift.get("Key") == 0].groupby("Album").count()
query = query[query.get("Key") >= 2]
query = query.assign(Number_of_Songs = query.get("Key"))
query.get("Number_of_Songs").plot(kind = "bar", title = "Number of Minor Key Songs Greater than Two")
```

```
Out[42]: <AxesSubplot:title={'center': 'Number of Minor Key Songs Greater than Two'}, xlab='Album'>
```



**Question 1.12.** Next, let's explore how certain audio features differ from one album to another. Since we only have audio features for songs, not albums, we'll calculate the values of an audio feature for an album by averaging the values of that audio feature across all



songs on the album. For example, to compute the 'Energy' of the 'Midnights' album, we would compute the mean 'Energy' of all songs on the 'Midnights' album.

Below, complete the implementation of the function `sort_albums_by`, which takes in the column name of an audio feature (e.g. 'Energy') and returns an array of the names of all eleven Taylor Swift albums, with the albums sorted in descending order of the given feature. For example, `sort_albums_by('Energy')` would return an array whose first element is the name of the album with the highest mean 'Energy' across all its songs.

```
In [43]: tswift.groupby('Album').mean().sort_values(by = 'Key', ascending = False).index
```

```
Out[43]: Index(['Midnights', 'Taylor Swift', 'evermore', 'Speak Now', 'Red', 'Fearless',
              'Lover', 'folklore', 'The Tortured Poets Department', 'Reputation',
              '1989'],
              dtype='object', name='Album')
```

```
In [44]: def sort_albums_by(feature):
          return np.array(tswift.groupby('Album').mean().sort_values(by = feature, as
```

```
In [45]: grader.check("q1_12")
```

```
Out[45]: q1_12 passed!
```

Now, run the cell below to see how the albums rank for each audio feature.

```
In [46]: features = ['Popularity', 'Explicit', 'Danceability', 'Energy', 'Loudness',
                    'Speechiness', 'Acousticness', 'Instrumentalness', 'Liveness',
                    'Valence', 'Duration_ms', 'Tempo']

for feature in features:
    # These lines display the output nicely. You don't need to understand how to
    display(Markdown(f"Taylor Swift albums, in descending order of `{feature}`"))
    display(Markdown("- " + ", ".join(sort_albums_by(feature))))
```

Taylor Swift albums, in descending order of 'Popularity' :

- Lover, Reputation, The Tortured Poets Department, folklore, 1989, Midnights, Speak Now, Red, evermore, Fearless, Taylor Swift

Taylor Swift albums, in descending order of 'Explicit' :

- The Tortured Poets Department, evermore, Midnights, folklore, Red, 1989, Fearless, Lover, Reputation, Speak Now, Taylor Swift

Taylor Swift albums, in descending order of 'Danceability' :

- Lover, Reputation, 1989, Midnights, Red, folklore, Taylor Swift, Speak Now, Fearless, The Tortured Poets Department, evermore

Taylor Swift albums, in descending order of 'Energy' :

- 1989, Taylor Swift, Speak Now, Fearless, Red, Reputation, Lover, evermore, Midnights, The Tortured Poets Department, folklore

Taylor Swift albums, in descending order of `'Loudness'` :

- Speak Now, Taylor Swift, Fearless, Red, 1989, Reputation, Lover, The Tortured Poets Department, evermore, folklore, Midnights

Taylor Swift albums, in descending order of `'Speechiness'` :

- Lover, Midnights, Reputation, evermore, The Tortured Poets Department, 1989, Red, folklore, Speak Now, Fearless, Taylor Swift

Taylor Swift albums, in descending order of `'Acousticness'` :

- evermore, folklore, The Tortured Poets Department, Midnights, Lover, Taylor Swift, Fearless, Speak Now, Red, Reputation, 1989

Taylor Swift albums, in descending order of `'Instrumentalness'` :

- Midnights, evermore, 1989, Lover, folklore, The Tortured Poets Department, Red, Taylor Swift, Reputation, Fearless, Speak Now

Taylor Swift albums, in descending order of `'Liveness'` :

- Taylor Swift, Fearless, Reputation, 1989, The Tortured Poets Department, Midnights, Speak Now, Red, Lover, evermore, folklore

Taylor Swift albums, in descending order of `'Valence'` :

- Lover, Red, evermore, Fearless, Taylor Swift, 1989, Speak Now, folklore, The Tortured Poets Department, Reputation, Midnights

Taylor Swift albums, in descending order of `'Duration_ms'` :

- Speak Now, Red, Fearless, evermore, The Tortured Poets Department, folklore, Reputation, 1989, Taylor Swift, Midnights, Lover

Taylor Swift albums, in descending order of `'Tempo'` :

- Fearless, Speak Now, Reputation, The Tortured Poets Department, Taylor Swift, evermore, 1989, Lover, folklore, Red, Midnights

**Question 1.13.** Let's look closely at one result from above, duplicated below. Recall that `'Explicit'` is a Boolean variable that determines whether the song includes explicit words. 🎧

```
In [47]: display(Markdown(f"Taylor Swift albums, in descending order of `Explicit`"))
display(Markdown("- " + ", ".join(sort_albums_by('Explicit'))))
```

Taylor Swift albums, in descending order of **'Explicit'** :

- The Tortured Poets Department, evermore, Midnights, folklore, Red, 1989, Fearless, Lover, Reputation, Speak Now, Taylor Swift

What can you conclude based only on the information displayed by the cell above? Assign a list with the numbers of all the true statements to the variable **q1\_13** .

1. There are more explicit words in **'The Tortured Poets Department'** than in **'evermore'** .
2. A higher fraction of songs on **'The Tortured Poets Department'** use explicit words than on **'evermore'** .
3. There are more songs that use explicit words in **'The Tortured Poets Department'** than on **'evermore'** .
4. If you randomly select a song from **'The Tortured Poets Department'** , the probability it contains explicit words exceeds the probability that a randomly selected song from **'evermore'** contains explicit words.

**\*Note:** You can assume that **'The Tortured Poets Department'** and **'evermore'** are not tied for most explicit album.

In [48]: **q1\_13 = ...**

In [49]: **grader.check("q1\_13")**

Out[49]: **q1\_13 results:**

**q1\_13 - 1 result:**

Trying:

`isinstance(q1_13, list) and set(q1_13) <= {1, 2, 3, 4}`

Expecting:

True

\*\*\*\*\*

Line 1, in q1\_13 0

Failed example:

`isinstance(q1_13, list) and set(q1_13) <= {1, 2, 3, 4}`

Expected:

True

Got:

False

**Question 1.14.** ★★ While Taylor Swift is primarily a solo artist, she has collaborated with other artists on a number of songs. For example, she featured Lana Del Rey in the song **'Snow On The Beach (Ft. Lana Del Rey)'** and Post Malone in **'Fortnight (Ft.**

Post Malone)' . All the collaborative songs in `tswift` are indicated by `'Ft.'` in the name of the song, just like in these examples.

Create an overlaid vertical bar chart that allows you to compare the average values of `'Explicit'`, `'Danceability'` and `'Acousticness'` for collaborative songs versus solo songs. Make sure your plot has a title and a legend that accurately describes what is being shown.

**\*Note:\*** There is more than one way you can approach this question. Any bar chart that allows you to compare the desired quantities will work!

In [50]: `# Create your plot here.`

Do collaborative songs have higher values of `'Explicit'`, `'Danceability'`, and `'Acousticness'` than solo songs, on average? Assign a list with the numbers of all the true statements to the variable `q1_14`.

1. Collaborative songs have a higher value of `'Explicit'` than solo songs, on average.
2. Collaborative songs have a higher value of `'Danceability'` than solo songs, on average.
3. Collaborative songs have a higher value of `'Acousticness'` than solo songs, on average.

In [51]: `q1_14 = ...`

In [52]: `grader.check("q1_14")`

Out [52]: **q1\_14 results:**

**q1\_14 - 1 result:**

Trying:

`isinstance(q1_14, list) and set(q1_14) <= {1, 2, 3}`

Expecting:

True

\*\*\*\*\*

Line 1, in q1\_14 0

Failed example:

`isinstance(q1_14, list) and set(q1_14) <= {1, 2, 3}`

Expected:

True

Got:

False

## Section 2: Song Recommender

Your favorite song was playing from the far side of the gym.

[\(return to the outline\)](#)

In this section, we'll create a Taylor Swift song recommender tool. The tool will allow you to input a song you like, selected from a collection of popular songs on Spotify, and it will recommend to you the songs in Taylor Swift's body of work that are most similar to your input song.

We are already familiar with the `tswift` DataFrame; this contains the Taylor Swift songs from which we will make our recommendations.

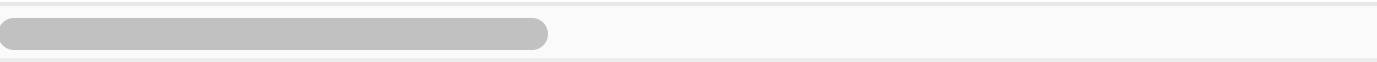
We will also use a separate dataset of allowed input songs. For this, we have a [collection of popular songs with more than one billion streams](#) on Spotify. Let's load in this data and store it in the DataFrame `billions_club`. `billions_club` contains all the same columns as `tswift`, except that it doesn't have a `'Year'` column, and it has an additional column named `'Artist'`, because it includes songs by a variety of artists.

```
In [53]: billions_club = bpd.read_csv('data/billions_club.csv').set_index('URI')
billions_club
```

Out [53]:

	Album	Song Name	Artist	Disc Number	Track Number	Popularity	Ex
URI							
02dRkCEc8Q5ch4TTcnLxOn	Late Night Feelings	Nothing Breaks Like a Heart (feat. Miley Cyrus)	Mark Ronson; Miley Cyrus	1	9	76	
4kV4N9D1iKVxx1KLvtTpjS	thank u, next	break up with your girlfriend, i'm bored	Ariana Grande	1	12	73	
07nH4ifBxUB4lZcsf44Brn	Motion	Blame (feat. John Newman)	Calvin Harris; John Newman	1	3	80	
6dOtVTDdiauQNBQEDOtIAB	HIT ME HARD AND SOFT	BIRDS OF A FEATHER	Billie Eilish	1	4	98	
76JKlSdKrAfWUMjaA0u7v5	Unreal Unearth: Unaired	Too Sweet	Hozier	2	1	78	
...	...	...	...	...	...	...	...
0h1W19pS59KtEd7aDzF58i	Scorpion	In My Feelings	Drake	2	9	47	
2b8fOow8UzyDFAE27YhOZM	Memories	Memories	Maroon 5	1	1	25	
116H0KvKr2ZI4RPuVBruDO	MIA (feat. Drake)	MIA (feat. Drake)	Bad Bunny; Drake	1	1	12	
1pKeFVVUOPjFsOABub0OaV	Dangerous Woman	Side To Side	Ariana Grande; Nicki Minaj	1	5	2	
4sPmO7WMQUAf45kwMOtONw	25	Hello	Adele	1	1	0	

763 rows x 20 columns



The goal in this section is to find a song in the `billions_club` DataFrame that we like and use it as an input to our recommender tool to find Taylor Swift songs with a similar sound. The problem is, in the preview of the DataFrame above, we can only see the first few rows and the last few rows of `billions_club`, so it's hard to see all of the possible songs we can choose from. Let's browse the DataFrame by randomly selecting ten rows to display.

```
In [54]: # Run this cell a few times!
billions_club.sample(10).get(['Song Name', 'Artist'])
```

Out [54]:

Song Name			Artist
URI			
4SFknyjLcyTLJFPKD2m96o	How You Like That		BLACKPINK
7MXVkk9YMctZqd1Srtv4MB	Starboy	The Weeknd; Daft Punk	
0HPD5WQqrq7wPWR7P7Dw1i	TiK ToK		Kesha
7gHs73wELdeycvS48Jflos	Faded		Alan Walker
2glIGP8kEfACgJdZ86kWxhN	Are You With Me - Radio Edit		Lost Frequencies
017PF4Q3l4DBUiWoXk4OWT	Break My Heart		Dua Lipa
7Kt59L2ZZGtOnlhvMwzG6f	Diamonds		Rihanna
4saklk6nie3yiGePpBwUoc	Dynamite		BTS
4B0JvthVoAAuyglLe3n4Bs	What Do You Mean?		Justin Bieber
3ZFTkvIE7kyPt6Nu3PEa7V	Hips Don't Lie (feat. Wyclef Jean)		Shakira; Wyclef Jean

**Question 2.1.** Run the previous cell a few times until you find a song you like; you'll use this song as the input to the recommender tool. Record the `'URI'` of your favorite song in the variable `favorite_uri`. You should just input this value manually based on what you find from browsing the DataFrame. (You can double-click a value in the index to highlight it, and then copy and paste it below.)

Then, using code, find the `'Song Name'` of this song and save it in the variable `favorite_song_name`.

```
In [55]: favorite_uri = ...
         favorite_song_name = ...

         print(f'My favorite song is {favorite_song_name}. It has a URI of {favorite_uri}')
My favorite song is Ellipsis. It has a URI of Ellipsis.

In [56]: grader.check("q2_1")
```

Out [56]:

**q2\_1 results:****q2\_1 - 1 result:**

Trying:

```

instance(favorite_uri, str) and instance(favorite_song_name, str)

```

Expecting:

```

True

```

```

*****

```

Line 1, in q2\_1 0

Failed example:

```

instance(favorite_uri, str) and instance(favorite_song_name, str)

```

Expected:

```

True

```

Got:

```

False

```

You can also listen to a preview of your favorite song in the notebook by running the cell below and pressing play.

In [57]: `play_spotify(favorite_uri)`

```

-----
AttributeError                                Traceback (most recent call last)
/tmp/ipykernel_234/2969436755.py in <module>
----> 1 play_spotify(favorite_uri)

/tmp/ipykernel_234/2134023420.py in play_spotify(uri)
    17
    18 def play_spotify(uri):
----> 19     code = uri[uri.rfind(':')+1:]
    20     src = f"https://open.spotify.com/embed/track/{code}"
    21     width = 400

AttributeError: 'ellipsis' object has no attribute 'rfind'

```

We will allow our recommender tool to base its recommendations on a customizable set of audio features. When not specified, the tool will use *all* of the audio features that are measured on a 0 to 1 scale, which we'll store in the variable `default_features`. But you can choose to use only a subset of these features when asking for recommendations, based on whatever is important to you!

```

In [ ]: default_features = [
        'Danceability',
        'Energy',
        'Speechiness',
        'Acousticness',
        'Instrumentalness',
        'Liveness',

```



```
'Valence'  
]
```

Let's start building our recommender tool!

The first step of the process is to extract the features of our favorite song from a DataFrame of songs. The thing is, each song is stored as a row of a DataFrame, and we don't have any experience accessing full rows of DataFrames. We typically access entire columns of data using `.get`, but we've never needed to access whole rows before.

It turns out that the `.loc` accessor that we've used on Series also works on DataFrames. If we use `.loc` directly on a DataFrame, we can extract the contents of an individual row as a Series. When all the values in a row are of the same data type, we can then convert that Series into an array.

For example, the next cell creates a small example DataFrame.

```
In [ ]: example_df = bpd.DataFrame().assign(x=[1, 2], y=[3, 4], z=[5, 6])  
example_df
```

We can extract the values from the second row of `example_df` as follows.

```
In [ ]: example_df.loc[1]
```

Notice that this is a Series whose index values are the column names of `example_df`. If we want to work with the values in this Series as an array, we can convert the Series to an array.

```
In [ ]: np.array(example_df.loc[1])
```

**Question 2.2.** Use what you just learned about accessing rows to complete the implementation of the function `get_feature_values`. The inputs to `get_feature_values` are:

- `input_uri`, the 'URI' of a song whose features you want to extract as an array.
- `song_df`, a DataFrame that has a row corresponding to the song with the given `input_uri` and columns containing audio features.
- `feature_list`, a list of features to extract.

It should return an **array** containing the values of the specified features for the song, in the order that they appear in `feature_list`. If there is no song with the given `input_uri` in `song_df`, the function should print `'This URI was not found.'` and return `None`. (`None` is a special Python keyword; it should not go in quotes, and it will turn green when you type it.)

Example behavior is given below.

```
# This URI corresponds to the song "Creepin" (with The Weeknd & 21  
Savage)".
```

```
# Query for it in billions_club; you'll see that its 'Valence',
'Instrumentalness', and 'Energy'
# match the values in the array below.
>>> get_feature_values('2dHHgzDwk4BJdRwy9uXhT0', billions_club,
['Valence', 'Instrumentalness', 'Energy'])
array([0.157, 0.    , 0.613 ])
```

After implementing your function, use it to extract some audio features from your favorite song in `billions_club`, whose `'URI'` was stored in `favorite_uri`. You can use any of the audio features in `default_features` that you like.

**\*Hint:\*** Use the `in` keyword to check if `input_uri` is a valid `'URI'`.

**\*Note:\*** We're writing a very general function that allows `song_df` to be any DataFrame of songs. For now, we'll only call the function with `billions_club` as `song_df`, but later in this section, we'll see how to use the function with `song_df` set to something else.

```
In [ ]: def get_feature_values(input_uri, song_df, feature_list):
        ...

# Now call your function to extract some audio features of your favorite song.
get_feature_values(favorite_uri, billions_club, ['Danceability', 'Energy'])
```

```
In [ ]: grader.check("q2_2")
```

We now know how to extract the features of a song (as an array) from a DataFrame using `get_feature_values`. But, what will we do with this feature array once we extract it? We'll want to compare the features of our song to the features of each Taylor Swift song to identify the Taylor Swift songs that are most similar to the song we selected. Since each audio feature is numerical, we need a way to compare two arrays of numbers and measure the similarity between them.

For this, we'll use the Euclidean distance, which you may know more simply as just "distance". Euclidean distance is a fundamental concept in geometry used to measure how far away two points are. We're most familiar with this concept in two dimensions, where it corresponds to the distance between two points in the plane. Here is the distance formula in two dimensions:

**Distance between two points  $(x_1, y_1)$  and  $(x_2, y_2)$  in 2 dimensions:**

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



The picture above shows that the distance formula is really an application of the Pythagorean theorem relating the side lengths of a right triangle:  $c^2 = a^2 + b^2$ .

However, Euclidean distance is not limited to two dimensions; it can be generalized to more dimensions. In general, in  $n$ -dimensional space (where  $n$  can be any positive integer), we can generalize the Euclidean distance formula as follows:

**Distance between two points  $(x_1, x_2, \dots, x_n)$  and  $(y_1, y_2, \dots, y_n)$  in  $n$  dimensions:**

$$d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_n - y_n)^2}$$

What can we do with this? For our application, let's think of the feature array for a given song as a point in  $n$ -dimensional space. For example, if the features we care about are `['Valence', 'Acousticness', 'Energy']`, then for any given song, we have a point in 3-dimensional space. Then, to measure how similar two songs are, we can compute the Euclidean distance between their two points in 3-dimensional space. **The closer this distance is to 0, the more similar the two songs are!**

For instance, suppose we want to compute the similarity between the songs `'Sugar'` by `'Maroon 5'` and `'Halo'` by `'Beyonce'` using the aforementioned three features. Using `get_feature_values`:

- `'Sugar'` has the feature array `np.array([0.884, 0.0591, 0.788])`, and
- `'Halo'` has the feature array `np.array([0.472, 0.272, 0.72])`.

Then, the Euclidean distance between the two songs' points is:

$$\sqrt{(0.884 - 0.472)^2 + (0.0591 - 0.272)^2 + (0.788 - 0.72)^2} = 0.4687...$$

**Question 2.3.** ★★ Now, let's apply the concept of Euclidean distance to calculate the similarity between two songs. Complete the implementation of the function `calculate_similarity` below. This function takes in two arrays of the same length, which we can think of as two points in  $n$ -dimensional space, where  $n$  is the length of each array. The function should output the Euclidean distance between these two points, as a float, with no rounding.

Example behavior is given below.

```
# Note: We've hard-coded the inputs to calculate_similarity just for
this example;
# typically, you'll find feature arrays by calling your
get_feature_values function.
>>> calculate_similarity(np.array([0.287, 0.0885, 0.794]),
np.array([0.888, 0.009, 0.787]))
0.6062757211038555
```

Then, use both the `calculate_similarity` and the `get_feature_values` functions you've defined to calculate the similarity (Euclidean distance) between your favorite song and the Taylor Swift song `'Karma'`, based on the features `'Danceability'`, `'Energy'`, `'Valence'`, and `'Acousticness'`. Store your result in `favorite_vs_karma`.

```
In [ ]: def calculate_similarity(features_1, features_2):
        ...
```

```
favorite_vs_karma = ...
favorite_vs_karma
```

```
In [ ]: grader.check("q2_3")
```

**Question 2.4.** 🌟🌟 So far, we know how to compute the similarity between two particular songs. Now, let's extend what we've learned to compute the similarity between a given song and *each* of the Taylor Swift songs in `tswift`.

Use the two functions you've defined so far in this section ( `get_feature_values` and `calculate_similarity` ) to complete the implementation of the function `calculate_similarity_for_all`. This function should calculate the similarity scores between the input song and each of the Taylor Swift songs in `tswift`, one by one, and output an array of similarity scores. The inputs to `calculate_similarity_for_all` are:

- `input_uri`, the 'URI' of a song whose features you will compare to each song in `tswift`.
- `song_df`, a DataFrame that has a row corresponding to the song with the given `input_uri` and columns containing audio features.
- `feature_list`, a list of features to include in the similarity score.

Make sure your output array has the similarities of all songs in the order they appear in the `tswift` DataFrame. For example, if the first song in `tswift` is 'Anti-Hero' (it may not be), then the first element of your output array should be the similarity between the input song and 'Anti-Hero'.

While we don't normally iterate through DataFrames, Series, or indexes, in this question, you should use a `for`-loop to iterate through the values in the index of `tswift`, extracting the features from one Taylor Swift song at a time and using those features to compute the similarity with the input song.

A sample function call is provided. Feel free to try other example inputs.

```
In [ ]: def calculate_similarity_for_all(input_uri, song_df, feature_list):
        similarity_scores = ...
        return similarity_scores

# Feel free to comment this line out or change the arguments!
calculate_similarity_for_all(favorite_uri, billions_club, ['Danceability', 'Energetic'])
```

```
In [ ]: grader.check("q2_4")
```

**Question 2.5.** Now that we have the similarity scores between our input song and every single Taylor Swift song, let's use them to find the Taylor Swift songs that are most similar to our input song. Complete the implementation of the function `select_top_recommendations`, which takes in an array `similarity_scores`, like the one you just created, and a positive integer `n`. The function should return a DataFrame

of the `n` songs from `tswift` that correspond to the `n` **lowest** values in `similarity_scores`, sorted in increasing order of similarity score.

- Remember, the most similar songs are the ones whose similarity scores are lowest, because songs are more similar when they have a lower Euclidean distance.
- The DataFrame returned by `select_top_recommendations` should have all the same columns as `tswift`, though as mentioned above, it should only have `n` rows.

**\*Hint:\*** You may want to add a new column to your DataFrame containing the contents of `similarity_scores`. Make sure to drop this column at the end, so that your output DataFrame has exactly `n` rows and 20 columns.

```
In [ ]: def select_top_recommendations(similarity_scores, n):
        ...
```

```
In [ ]: grader.check("q2_5")
```

**Question 2.6.** It's time to combine everything we've implemented so far into a single function. Below, complete the implementation of the function `song_recommender`. The inputs to `song_recommender` are:

- `input_uri`, the `'URI'` of a song which you want to find recommendations similar to.
- `song_df`, a DataFrame that has a row corresponding to the song with the given `input_uri` and columns containing audio features.
- `n`, the number of top-matching songs to be shown.
- `feature_list`, a list of features to include in the similarity computation. This is an optional parameter; if not included, the similarity should be based on all features in `default_features`.

The output should be an `n`-row DataFrame whose index contains `'URI'` s and whose only columns are `'Album'`, `'Song Name'`, and the columns included in `feature_list`. Each row in the returned DataFrame should correspond to a song; the songs should be sorted in decreasing order of similarity (that is, in increasing order of similarity score – the same way that the rows are ordered in the output of `select_top_recommendations`).

**\*Note:\*** In the signature of the `song_recommender` function, we set `feature_list=default_features`. This tells Python that `feature_list` is an optional parameter, and if omitted, it should be set to `default_features`. We use functions with default parameter values all the time – for example, in the DataFrame method `.sort_values`, when we don't set the parameter `ascending`, it defaults to having a value of `True`.

**\*Hints:\***

- You've done most of the work already. It's possible to solve this problem in one (long) line of code. We say this not to encourage you to write one long line of code, but to

emphasize that you don't need to write a ton of code within the body of `song_recommender`.

- The `+` symbol, when placed between two lists, concatenates the lists, just like when the `+` symbol is placed between two strings, it concatenates the strings. This will be useful when making sure that your output DataFrame has only certain columns.

```
In [ ]: def song_recommender(input_uri, song_df, n, feature_list=default_features):
        ...

        # The following call to song_recommender finds the 5 Taylor Swift songs
        # that are most similar to your chosen song, in terms of 'Danceability' and 'Energy'
        # We've also include a print statement to help you interpret the output.
        print(f'Taylor Swift songs that are most similar to {favorite_song_name}:')
        song_recommender(favorite_uri, billions_club, 5, ['Danceability', 'Energy'])
```

```
In [ ]: grader.check("q2_6")
```

Nice work! You now have the ability to pick a song you like and audio features that are important to you and use your function to get back recommendations of Taylor Swift songs that are similar to the song you chose.

But wait – it gets cooler.

The cell below is long, but it sets up an interactive widget, which allows you to choose any song in `billions_club` from a dropdown menu and plays for you the 5 most similar songs from Taylor Swift's repertoire, according to all of the features in `default_features`. Run the cell below to check it out!

```
In [ ]: # Run this cell. Don't change any of the code.
        default = 'Bank Account by 21 Savage'

        def get_and_format_recommendations(song_name):
            song, artist = song_name.split(' by ')
            row = billions_club[(billions_club.get('Song Name') == song) & (billions_club.get('Artist') == artist)]
            uri = row.index[0]
            recommendations_df = song_recommender(uri, billions_club, 5, default_features)
            display(HTML(f'<h3>The song you chose was {billions_club.get("Song Name").get("Song Name")}.</h3>'))
            play_spotify(uri)
            display(HTML(f'<h4>Here are the 5 most similar Taylor Swift songs we found.</h4>'))
            for recommended_uri in recommendations_df.index:
                play_spotify(recommended_uri)

        song_options = np.sort(billions_club.get('Song Name') + ' by ' + billions_club.get('Artist')).tolist()
        song_widget = widgets.Dropdown(options=song_options, description='Song', layout={'width': 200})

        def change_rec(change):
            if change['name'] == 'value' and change['new'] != change['old']:
                clear_output()
                display(song_widget)
                get_and_format_recommendations(song_widget.value)

        display(song_widget)
```

```
get_and_format_recommendations(default)
song_widget.observe(change_rec)
```

You did all of the calculations behind-the-scenes to make this widget work – nice job!

Up until now, our goal has been to find the songs in `tswift` that are most similar to our chosen song in `billions_club`. That's what the above widget does.

However, we can also use the functions we've defined to determine the similarity between any pair of songs in `tswift`, since all of the functions we've defined so far in this section take a `song_df` as an input.

Run the cell below to see a similar widget to the one above, but where the possible song options are songs in `tswift`. This time, songs are sorted first by album in alphabetical order, then by song name in alphabetical order. This widget can be used to organize Taylor Swift songs with a similar sound.

```
In [ ]: # Run this cell. Don't change any of the code.
default_tswift = 'Enchanted, from the album Speak Now'

def get_and_format_recommendations_tswift(song_name):
    song, album = song_name.split(', from the album ')

    row = tswift[(tswift.get('Song Name') == song)]
    uri = row.index[0]
    recommendations_df = song_recommender(uri, tswift, 6, default_features)
    display(HTML(f'<h3>The song you chose was {tswift.get("Song Name").loc[uri]}'))
    play_spotify(recommendations_df.index[0])
    display(HTML('<h4>Here are the 5 most similar <b>other</b> Taylor Swift songs'))
    for recommended_uri in recommendations_df.index[1:]:
        play_spotify(recommended_uri)

by_album = tswift.sort_values(['Album', 'Disc Number', 'Track Number'])
song_options_tswift = np.array(by_album.get('Song Name') + ', from the album ')
song_widget_tswift = widgets.Dropdown(options=song_options_tswift, description='Song')

def change_rec_tswift(change):
    if change['name'] == 'value' and change['new'] != change['old']:
        clear_output()
        display(song_widget_tswift)
        get_and_format_recommendations_tswift(song_widget_tswift.value)

display(song_widget_tswift)
get_and_format_recommendations_tswift(default_tswift)
song_widget_tswift.observe(change_rec_tswift)
```

Awesome job! Have you found any new songs to listen to? We have... 🎵

## Section 3: Lyric Searcher 🔍

Dark side, I search for your dark side.



[\(return to the outline\)](#)

In this section, we'll create a Taylor Swift lyric searcher similar to [this online tool](#) created by Shayna Kothari, a software engineer at Facebook. Run the next cell and try inputting some search terms to explore what it does!

```
In [ ]: display(IFrame(src="https://shaynak.github.io/taylor-swift", width=800, height=
```

Creating our own version of this search tool will involve multiple steps, but we'll start simple and gradually increase the complexity of our search. Our search tool will have some differences from the online tool, but it will be similar.

To implement our lyric searcher, we won't need any of the musical data in `tswift`, just lyrics of the songs themselves. At this point in the project, we'll switch our attention to the `lyrics` DataFrame.

```
In [ ]: lyrics
```

**Question 3.1.** To start, create a DataFrame named `casually_cruel` that has the same index and columns as `lyrics` and a row for each song that has the exact string `'casually cruel'` in the lyrics.

```
In [ ]: casually_cruel = ...
        casually_cruel
```

```
In [ ]: grader.check("q3_1")
```

If you search the phrase `'casually cruel'` on the lyric searcher website, or if you're a big Swiftie, you'll recognize that there's another song, `'Mr. Perfectly Fine'`, that uses this same phrase, but for some reason, it's not appearing among our search results. Try printing out the lyrics to `'Mr. Perfectly Fine'` and see if you can figure out why it doesn't appear in our `casually_cruel` DataFrame.

```
In [ ]: # Why is Mr. Perfectly Fine not included?
```

**Question 3.2.** Now, write a function called `phrase_match_df` with one parameter, a string `phrase`. The function should return a DataFrame with the same index and columns as `lyrics`, with a row for each song that includes the given `phrase` in the lyrics, regardless of capitalization.

For example, on the input phrase of `'casually cruel'`, the function should return a DataFrame with two rows; the same two rows should be returned on the input phrase `'CASUALLY CRUEL'`.

```
In [ ]: def phrase_match_df(phrase):
        ...
```



```
# Feel free to change the argument in the function call below to experiment!
phrase_match_df('casually cruel')
```

```
In [ ]: grader.check("q3_2")
```

Let's try to figure out exactly where in a song a certain phrase appears. For example, we know the phrase 'casually cruel' appears in 'All Too Well (10 Minute Version)' but what line(s) is it a part of? Does it appear several times or just once?

First, when we refer to a line of a song, we're referring to what gets printed on its own line when the lyrics are printed. Lines *can* be blank, which usually happens as a separator between different parts of the song, such as the chorus and the verse. For example, the fifth line of 'Mastermind' is blank, as you can see below.

```
In [ ]: print(mastermind)
```

**Question 3.3.** Set the variable `fine_lines` to a list of all the lines of 'Mr. Perfectly Fine'. Some of these lines will be blank (meaning they will appear as empty strings).

**\*Hint:** Lines are separated by newline characters `'\n'`.

```
In [ ]: fine_lines = ...
        fine_lines
```

```
In [ ]: grader.check("q3_3")
```

**Question 3.4.** Loop through the list `fine_lines`, and when you encounter a line that contains the phrase 'casually cruel' (with any capitalization), append that line to the array `cruel_fine_lines`, which we have already initialized to be empty.

Note that when you append a line to `cruel_fine_lines`, the line should maintain the capitalization of the original lyrics. They should also appear in the same order they appear in `fine_lines`. If there are duplicate lines, include all occurrences of the line.

```
In [ ]: phrase = 'casually cruel'
        cruel_fine_lines = np.array([])
        for line in fine_lines:
            ...
        cruel_fine_lines
```

```
In [ ]: grader.check("q3_4")
```

**Question 3.5.** Now, we're ready to generalize our work so that we can search for *any* phrase in *any* song. Complete the implementation of the function `isolate_phrase`, which takes as input the name of a song in the `lyrics` DataFrame and a phrase to search for, and returns an array of all lines in the song containing the phrase.

As in the previous question, this should be a case-insensitive search, meaning the phrase is considered a match even if it appears with different capitalization, though the line you add

to the output array should have the original capitalization of the song lyrics.

The search should also include results where the target phrase appears as part of a longer word or phrase. For example, the output of `isolate_phrase('Mastermind', 'plan')` should include the line `'Once upon a time, the planets and the fates'` because `'plan'` is a part of `'planets'`. Note that this is not necessarily desirable behavior, but it's the most straightforward to implement.

Be careful: sometimes the same phrase can appear multiple times in a single line. When that happens, the line should only appear in the output array once.

```
In [ ]: def isolate_phrase(song_title, phrase):
        ...

        isolate_phrase('Mastermind', 'plan')
```

```
In [ ]: grader.check("q3_5")
```

When we search the lyric searcher website for a specific phrase, notice that it displays not only the line where the search term was found, but also the lines immediately before and after, if they exist.

Let's look closely at the four instances of the word `'time'` in the song `'Mastermind'` and see how these search results are displayed on the website.



If there is a match for `'time'` in the first line, like `'Once upon a time'`, there is no previous line of the song. As a result, this line and the next are the only two lines in the output.

If there is a match anywhere besides the first and last lines, there is a previous line and a next line, so all three lines should get added to the output. It's possible that the previous line or next line is blank; for instance, this happens in the second match in the example above, `'At the same time'`, which is followed by a blank line because it is at the end of the first verse.

If there is a match in the last line, the previous line and the matching line should be output, but there is no next line.

**Question 3.6.** ★★ Let's now define a function, `surround_phrase`, that works similarly to `isolate_phrase` except instead of only appending to the output array the lines where a match occurs, it should also append the previous and next lines, if they exist (even if they are blank lines).

Start with the code you wrote for `isolate_phrase` and modify as needed. Instead of looping through all the lines, you'll want to handle matches in the first line and last line separately from matches in the middle.

**\*Hint:\*** Think about three cases of matches: a match in the first line, a match in the middle, and a match in the last line. Our solution first checks for matches in the first line using an `if`-statement. Then it loops through all the middle lines, looking for matches in each such line (again, using an `if`-statement). Finally, it checks for matches in the last line using another `if`-statement.

Example behavior is given below.

```
>>> surround_phrase('Mastermind', 'time')

array(['Once upon a time, the planets and the fates',
      'And all the stars aligned',
      'You and I ended up in the same room',
      'At the same time',
      '',
      'To make them love me and make it seem effortless',
      'This is the first time I've felt the need to confess',
      'And I swear',
      'Saw a wide smirk on your face',
      'You knew the entire time',
      'You knew that I'm a mastermind'],
      dtype='<U52')
```

```
In [ ]: def surround_phrase(song_title, phrase):
        ...
        surround_phrase('Mastermind', 'time')
```

```
In [ ]: grader.check("q3_6")
```

Our lyric searcher is looking more like the website, but it's hard to parse the output. Let's make the output array easier to understand by doing what the website does: including the name of the song and album after each match.

**Question 3.7.** Write a function called `one_song_search` that works similarly to `surround_phrase`, except it should also append to the array a string that includes the song name and album name, after each match. Format this string like this: `'Song Name, Album Name'`, where the capitalization and spelling of song and album names are exactly as they appear in `lyrics`.

**\*Hint:\*** We solved this problem by taking our code from `surround_phrase` and adding four lines of code to it. The first calculated the name of the album the song was from. The other three lines of code each appended a string with the song and album. We needed three such appends to deal with the three cases: a match in the first line, a match in a middle line, and a match in the last line.

```
In [ ]: def one_song_search(song_title, phrase):
        ...
```

```
one_song_search('Mastermind', 'time')
```

```
In [ ]: grader.check("q3_7")
```

We now have search results for a single song, but we need to extend our search to the entire body of Taylor Swift's work. To do this, we'll first use our `phrase_match_df` function from earlier to create a smaller DataFrame of just the songs that contain our target phrase. Then we'll loop through the entries in the `'Lyrics'` column of this smaller DataFrame and call our `one_song_search` function on each song's lyrics.

**Question 3.8.** Fill in the blanks below to try out the strategy outlined above to search for `'perspective'` in all of Taylor Swift's songs. We've initialized an empty array called `perspective_array`, where you should store all of your matches from all songs.

**\*Hint:\*** We usually use `np.append` to append a single item to an array. However, you can also use it to append a whole array of items to an array.

```
In [ ]: perspective_songs = phrase_match_df('perspective')
perspective_array = np.array([])
...
perspective_array
```

```
In [ ]: grader.check("q3_8")
```

**Question 3.9.** Generalize the previous example by writing a function `search_for` that takes a phrase as input and searches all of Taylor Swift's songs for that phrase, returning an output array as we've discussed. For example, `search_for('perspective')` should return an array with the same contents as `perspective_array` above.

```
In [ ]: def search_for(phrase):
    ...

# Feel free to change the function call below.
# Make sure to try some other words and phrases.
search_for('perspective')
```

```
In [ ]: grader.check("q3_9")
```

Our search tool works pretty well! However, it's still very hard to read the output in this format. Let's display it more nicely, and at the same time, implement something that the website has: a count of the total number of matches (usages), and the number of songs with a match. For example, the search term `'perspective'` is used 3 times in 3 songs throughout Taylor Swift's repertoire.



**Question 3.10.** Complete the implementation of the function `search_and_display`, which takes as input a phrase to search for, calls the function `search_for` on that input

phrase, and then nicely displays the output. In addition, the `search_and_display` function should calculate the total number of usages of the input phrase across all songs, as well as the number of songs in which the input phrase appears, and return both of these values in a list whose first element is the number of usages and whose second element is the number of songs.

We've provided the code to do the displaying; you don't need to understand how this works. Your job is to calculate the total number of usages in `num_usages` and the number of matching songs, and to return a list of two elements as described. You'll see a comment that says `# TODO` above every line you need to complete.

```
In [ ]: def search_and_display(phrase, to_display=True):
    # Ignore the optional to_display argument.
    # By default, we will display all of the lyrics as done in search_for.

    match_array = search_for(phrase)

    num_usages = 0
    matching_songs = np.array([])

    for line in match_array:

        # If the line represents a song name and album name, display it nicely.
        if line in np.array(lyrics.reset_index().get('Song') + ", " + lyrics.re
            if to_display:
                display(HTML(f'<center><b><i>{line}</i></b></center>')) # Disp
                display(Markdown('___')) # Add horizontal line between matches.

        # TODO: Update matching_songs.
        matching_songs = ...

        # Otherwise, if the line is not blank, print it.
        elif len(line) > 0:
            if to_display:
                display(HTML(f'<center>{line}</center>'))

        # TODO: Update num_usages.
        num_usages = ...

    # TODO: Create a list of two elements to output.
    output_list = ...

    if to_display:
        display(HTML('<h3><center><span style="color:#888">Found ' + str(output
    return output_list

# An example function call. Feel free to change it.
perspective_stats = search_and_display('perspective')
```

```
In [ ]: grader.check("q3_10")
```

We now have a search tool that can search Taylor Swift's body of work for any phrase and display the results in much the same format as the online search tool we tried to replicate.

Well done!

Run the cell below to play around with an interactive version of `search_and_display`. It'll show you a text box; type a phrase and hit "enter" to see the value of `search_and_display` when called on your input.

```
In [ ]: # Run this cell. Don't change any of the code.
default_lyric = 'casually cruel'

lyric_box = widgets.Text(
    value=default_lyric,
    placeholder='Type a phrase here and hit enter.',
    description='Phrase:',
    layout={'width': '525px'},
    disabled=False
)

def change_matches(change):
    clear_output()
    display(lyric_box)
    search_and_display(lyric_box.value)

display(lyric_box)
search_and_display(default_lyric)
lyric_box.on_submit(change_matches)
```


You may notice that for certain search terms, our search tool gives different results than the online search tool. There are a few reasons why. First, the set of songs we are searching is not exactly the same. Our `lyrics` DataFrame includes all the songs from Taylor Swift's studio albums, and the online search tool includes a few additional songs, like songs from movie soundtracks.

Moreover, our search results always include matches where the search term is part of a longer word or phrase, like how `'plan'` is part of `'planets'`. The online search tool handles this more carefully, with options to search for an exact phrase, to include plurals, or to do a wildcard search which is similar to how we've chosen to implement the search. We'll stop here with our lyric searcher, but if you're interested in extending your lyric searcher further, you can try implementing some of these features from the online tool after you submit the project. There are lots of details to consider, such as how to pluralize words (it's not always as simple as adding an `'s'`)!

## Section 4: Keywords

I didn't know you were keeping count, but, oh, you were keeping count.

[\(return to the outline\)](#)

In this section, we'll identify keywords that summarize each song in Taylor Swift's album `'Lover'` .

For example, you'll discover, after completing Section 4, that the single word that best summarizes the song 'London Boy' is 'fancy'. Listen to the song below and see if you agree!

```
In [ ]: play_spotify('1LLXZFeAHK9R4xUramtUKw')
```

Pretty cool, right? You'll soon learn how to find the best keywords for each song yourself. The first step in calculating keywords is to identify every unique word used on the 'Lover' album.

**Question 4.1.** To start, create a DataFrame called `lover_df` with the same columns as `lyrics`, but with only the songs from the 'Lover' album.

```
In [ ]: lover_df = ...  
lover_df
```

```
In [ ]: grader.check("q4_1")
```

**Question 4.2.** Now, we want to determine the words used in all the lyrics in `lover_df`. Store all such unique words in an array called `unique_words_raw`. Deal with capitalization and punctuation as follows:

- Words with the same letters that are capitalized differently are considered the same. For example, 'talk', 'Talk', and 'talk' should all be counted as the same word. All words in `unique_words_raw` should be in lowercase, so all of these words should be counted as 'talk'.
- Words that use the same letters but have different punctuation are considered different. For example, '"for' and 'for' will both be words in `unique_words_raw`, as they're not considered the same word.

**\*Hints:**

- Words may be separated by spaces or by newline characters, `\n`. The string method `.split()` when called with no arguments will separate on both of these; this is what you want!
- You can use the `.sum()` method on a Series of lists to concatenate the lists into a single, larger list.

```
In [ ]: unique_words_raw = ...  
unique_words_raw
```

```
In [ ]: grader.check("q4_2")
```

As mentioned above, some of the words in `unique_words_raw` contain punctuation. For example, both '"for' and 'for' are in `unique_words_raw`, although they correspond to the same English word.

```
In [ ]: 'for' in unique_words_raw and '"for' in unique_words_raw
```

Let's reconcile this by removing punctuation from the words in `unique_words_raw` and then keeping only the unique words that remain.

**Question 4.3.** Complete the implementation of the function `drop_punctuation`, which takes in a string `word` and performs the actions below to address the aforementioned issues (there may be other issues with the data, but don't worry about them).

1. Remove quotations ( `'` and `"` ) at the beginning and end of strings.
2. Replace all double quotes ( `"` ) with single quotes ( `'` ).
3. Remove the following punctuation symbols: `( , ) , ? , . , , , ; , - , _ , \ .`

The function should return a version of the input word with these changes.

**\*Hint:\*** When removing `\`, use `\\` instead of `\`. You need to "escape" the backslash.

```
In [ ]: def drop_punctuation(word):
        ...
```

```
In [ ]: grader.check("q4_3")
```

**Question 4.4.** Use the `drop_punctuation` function to create an array of all the unique words used in the `'Lover'` album, without punctuation. Store that array in `unique_words`. Notice that after removing punctuation, there may be duplicate words (for example, `'for'` and `'"for'` become the same word after punctuation is dropped), but make sure there are no duplicates in `unique_words`.

```
In [ ]: unique_words = ...
        unique_words
```

```
In [ ]: grader.check("q4_4")
```

Now that we've determined which words appear in the `'Lover'` album, we'll attempt to describe how important each word is to each song. To do this, we'll use a method from natural language processing called the **term frequency-inverse document frequency (TF-IDF)**.

The purpose of TF-IDF is to measure how important a term is in a document relative to a collection of documents. In our case, terms are words, documents are songs, and the collection of documents we're interested in is the collection of songs on the album `'Lover'`.

Given a word, **term**, in a document, **doc**, the TF-IDF of the word in the document is the product:



$$\begin{aligned} \text{TF-IDF}(\mathbf{term}, \mathbf{doc}) &= \text{TF}(\mathbf{term}, \mathbf{doc}) \cdot \text{IDF}(\mathbf{term}) \\ &= \frac{\text{number of occurrences of } \mathbf{term} \text{ in } \mathbf{doc}}{\text{total number of words in } \mathbf{doc}} \cdot \ln\left(\frac{\text{total number of documents}}{\text{number of documents containing } \mathbf{term}}\right) \end{aligned}$$

Let's look at the term frequency (TF) first. This is just the proportion of words in document **doc** that are equal to **term**.

- **Example:** What is the term frequency (TF) of "Taylor" in the following document?
  - "My friend named her baby **Taylor** because she is a huge **Taylor** Swift fan."
- **Answer:**  $\frac{2}{14}$ , because two of the fourteen words in the document are "Taylor".

Remember that usually, a document will consist of an entire song, not just of a single line (like above).

Now, let's look at the inverse document frequency (IDF). This is the natural logarithm of the reciprocal of the proportion of documents in the collection containing **term**. One way to think of the IDF is as a "rarity factor" – words that appear frequently in the full collection of documents are not very rare, and hence have low IDF, while words that rarely appear in the full collection of documents have high IDF.

- **Example:** What is the inverse document frequency (IDF) of "Taylor" in the following four documents?
  - "My friend named her baby **Taylor** because she is a huge **Taylor** Swift fan."
  - "I wanted to see **Taylor** Swift on the Eras Tour, but the tickets were way too expensive, so I saw the movie instead."
  - "I can't even afford a movie ticket."
  - "Tailor and **Taylor** are homophones."
- **Answer:**  $\ln\left(\frac{4}{3}\right) \approx 0.288$  because three of the four documents include the word "Taylor".

Putting these together, we can now compute the TF-IDF.

- **Example:** What is the term frequency-inverse document frequency (TF-IDF) of "Taylor" in the first document in this collection of four documents?
- **Answer:**  $\frac{2}{14} \cdot \ln\left(\frac{4}{3}\right) \approx 0.041$ , by multiplying the term frequency with the inverse document frequency.

The idea behind TF-IDF is that for a word to be a good summary of a document, it should appear frequently in that document, but not too frequently in the full collection of documents. This means words like "I" or "and" won't have high TF-IDF, because while they may appear frequently in a document (high TF), they appear too often in general to be considered good summary words (low IDF).

In our case, the words that best summarize a song will be the words within that song with the highest TF-IDFs.

Run the cell below to load in a DataFrame named `counts_df` that has a row for each word in the `'Lover'` album and a column for each song on the album. Each entry counts the number of instances of a word in a song.

```
In [ ]: counts_df = bpd.read_csv('data/word_counts.csv').set_index('word')
counts_df
```

For instance, the preview above tells us that the word `"you're"` appears 4 times in `'Cruel Summer'`.

The `counts_df` DataFrame will be useful in the next several questions.

**Question 4.5.** 🌟🌟 Now, it's time for some calculations 🧮. Let's begin by computing the TF-IDF for the word `'about'` in the song `'You Need To Calm Down'`. To start, make sure you understand the "Taylor" TF-IDF example given above, and consider how you will use the data available in `counts_df`.

We've provided four variables for you to fill in: `tf_numerator_about`, `tf_denominator_about`, `idf_numerator_about`, and `idf_denominator_about`. We then compute the TF-IDF from those four variables and store the result in `tfidf_about`.

```
In [ ]: tf_numerator_about = ...
tf_denominator_about = ...
idf_numerator_about = ...
idf_denominator_about = ...
tfidf_about = (tf_numerator_about / tf_denominator_about) * np.log(idf_numerator_about / idf_denominator_about)
tfidf_about
```

```
In [ ]: grader.check("q4_5")
```

Imagine you wanted to calculate the TF-IDF for the word `'about'` in the song `'False God'`. You could do that by taking the code you just wrote and changing every instance of `'You Need To Calm Down'` to `'False God'`. Notice that you may get a different value for the term frequency (TF) because the word `'about'` may be more prevalent in one song than another. However, you'd get the same exact value for the inverse document frequency (IDF) because IDF is calculated based only the term (word) not the document (song), as we can see in the formula introduced earlier:

$$\begin{aligned} \text{TF-IDF}(\text{term}, \text{doc}) &= \text{TF}(\text{term}, \text{doc}) \cdot \text{IDF}(\text{term}) \\ &= \frac{\text{number of occurrences of term in doc}}{\text{total number of words in doc}} \cdot \ln\left(\frac{\text{total number of documents}}{\text{number of documents containing term}}\right) \end{aligned}$$

This means that if we wanted to calculate the TF-IDF for the word 'about' in every song on the 'Lover' album, we could save some time and energy by just computing the IDF for 'about' one time, in advance. Then for each song, we'd just need to calculate the TF of 'about' in that song and multiply with the IDF for 'about' that we'd already pre-computed.

In the next question, we'll pre-compute the IDF values for all words and store them in an array so that we can more easily compute TF-IDF values later.

**Question 4.6.** Fill in the blanks in the code below to calculate IDF values for each word on the 'Lover' album. The strategy is as follows:

1. Begin by storing all the words used on the album in an array called `word_array`. You should get these words from `counts_df`.
2. Loop through this array, one word at a time, calculating the IDF, and storing the result in `idf_array`, which has been initialized to an empty array.

At the end, `idf_array` should have the IDF values for every word on the album, in the same order that they appear in `counts_df`. That is, the first element of `idf_array` should be the IDF of '16th'.

```
In [ ]: idf_array = np.array([])
word_array = ...
# Loop through each word and compute the IDF of that word.
for word in word_array:
    ...

# Display the resulting array of IDF values.
idf_array
```

```
In [ ]: grader.check("q4_6")
```

**Question 4.7.** ★★ Now, we are ready to calculate the TF-IDF for each word in each song. We've already calculated the IDF values and stored them in `idf_array`, so we'll want to make use of them here!

We'll approach this problem one song at a time. For each song, we'll calculate the TF-IDF values for that song, for every word that appears on the album. We can calculate all the TF-IDF values for a given song at the same time using Series arithmetic, which works element-wise. Once we obtain a Series of all the TF-IDF values for a song, we can add that Series as a column to a DataFrame, gradually building up the DataFrame one column (one song) at a time.

We've provided an outline of the strategy and some code to handle the assignment of new columns, which includes details that you don't need to worry about. Your job is to fill in the blanks below.

Once you've filled in the blanks, run the code cell to create a DataFrame called `every_tfidf` where the columns are the songs on the album `'Lover'`, the rows are the words that appear in the lyrics of the album, and the entries are the TF-IDF values for each song and each word.

```
In [ ]: # Create a new empty DataFrame to store TF-IDF values.
every_tfidf = bpd.DataFrame()

# Create an array with the names of all songs on the Lover album.
songs_array = ...

# Loop through the songs, and for each song, compute a Series of TF-IDF values
for song in songs_array:
    # Assign tf_numerators to a Series of the numerators of the TF values of each word
    tf_numerators = ...

    # Assign tf_denominator to the denominator of all TF values, for this song.
    # Note that this is a single number, not a Series or array.
    # We use the same denominator when calculating the TF of each word, for this song.
    tf_denominator = ...

    # Assign tfs to a Series of the TF values of each word, for this song.
    tfs = ...

    # Assign tfidfs to a Series of the TF-IDF values of each word, for this song.
    # Remember that you've already calculated the IDF of each word, so use those values.
    tfidfs = ...

    # Add a new column to the DataFrame every_tfidf.
    # The column name is the song title and the contents are the values in tfidfs.
    # Don't worry about how the line of code below works.
    every_tfidf = every_tfidf.assign(**{song: tfidfs})

every_tfidf
```

```
In [ ]: grader.check("q4_7")
```

It's a good idea to verify that the TF-IDF of the word `'about'` in the song `'You Need To Calm Down'` according to the DataFrame above is the same as you calculated in Question 4.5. Let's do that using code. The following cell should evaluate to `True`.

```
In [ ]: every_tfidf.get('You Need To Calm Down').loc['about'] == tfidf_about
```

Now that we've done the hard work of calculating all these TF-IDF values, it's time to reap the benefits and find our keywords!

**Question 4.8.** Find the 10 words with the highest TF-IDF values for the song `'Cruel Summer'`. Store these words in an array called `top_10_summer`. Sort the words in decreasing order of TF-IDF, breaking ties any way you like. These words should be a good summary for `'Cruel Summer'`. 🌈

```
In [ ]: top_10_summer = ...
top_10_summer
```

```
In [ ]: grader.check("q4_8")
```

**Question 4.9.** Complete the implementation of the function `ten_keywords`, which takes in the name of a song on the `'Lover'` album and returns an array of the 10 words with the highest TF-IDF values, sorted in decreasing order of TF-IDF. Again, ties can be broken in any way.

```
In [ ]: def ten_keywords(song_name):
    ...

# Here's one sample call, but try some more!
ten_keywords('London Boy')
```

```
In [ ]: grader.check("q4_9")
```

Let's have some fun visualizing the keywords for each song in a word cloud. The code provided below, adapted from an [article by Tia Plagata](#), creates a word cloud for any song on the `'Lover'` album. A word cloud is a type of data visualization for text data, showing more important words as bigger and bolder. Try it out, and feel free to have fun with it!

```
In [ ]: # We need to import some packages to make word clouds.
from wordcloud import WordCloud
from PIL import Image

# This function creates a word cloud for a given song.
def generate_lyrics_wordcloud(song_name):
    cloud = WordCloud(scale=3,
                      max_words=150,
                      colormap='RdPu',
                      mask=np.array(Image.open('data/images/heart.jpeg')),
                      background_color='white',
                      collocations=False).generate(lover_df.get('Lyrics').loc[song_name])
    plt.figure(figsize=(7, 5), dpi=100)
    plt.imshow(cloud)
    plt.axis('off')
    plt.show()

default_lover = 'I Forgot That You Existed'

song_options_lover = np.array(lover_df.index)
song_widget_lover = widgets Dropdown(options=song_options_lover, description='Song')

def change_rec_lover(change):
    if change['name'] == 'value' and change['new'] != change['old']:
        clear_output()
        display(song_widget_lover)
        display(HTML('Note: It may take a few seconds for the updated word cloud to load'))
        generate_lyrics_wordcloud(song_widget_lover.value)

display(song_widget_lover)
```

```
generate_lyrics_wordcloud(default_lover)
song_widget_lover.observe(change_rec_lover)
```

## Parting Thoughts

Big congratulations on finishing the Midterm Project! 🎉 We hope this experience gave you a taste of what doing data science is really like: frustrating at times, but incredibly satisfying when you produce a finished product you're proud of! If you feeling like telling everyone you know about the cool song recommender you created or the beautiful word clouds you generated, that's a good sign!

You're likely a stronger programmer and data scientist now than you were before you started this project, after all the effort you put into completing it. As a reminder of the growth you can achieve through hard work, and the fact that success does not come by accident, here's a spoof of Taylor Swift's song **'Mastermind'** about succeeding in DSC 10.

Once upon a time,  
pandas were just bears, Jupyter was in the sky.  
I took a ten week course in data science,  
And learned otherwise.

In the front row of class, 9am.  
Asking questions on Ed yet again.  
Churning through extra practice problems.  
Solutions? Don't need them.

What if I told you none of it was accidental?  
From the first class that you taught me, nothing was gonna stop me?  
I laid the groundwork and then did the hard work.  
The test cases all passed one at a time.

What if I told you I'm a mastermind?  
I'm good at data science.  
I just worked all the time.  
And now I'm a mastermind.

You see, all the wisest students had to do it this way.  
'Cause we were born to understand things as we toil away.

If you fail to plan, you plan to fail.  
Strategy sets the scene for the tale.  
So I cleared time in my schedule  
For a lot of nights in Geisel.

What if I told you none of it was accidental?  
From the first class that you taught me, nothing was gonna stop me?  
I laid the groundwork and then did the hard work.

The test cases all passed one at a time.

What if I told you I'm a mastermind?  
 I'm good at data science.  
 It was all by design.  
 And now I'm a mastermind.

No one wanted to work with me on the big project.  
 So I've been getting all the tutor help I can get.  
 I think they love me for all the effort I have spent.  
 This is the first time I felt the need to confess.  
 And I swear, I'm only understanding this material  
 'Cause I care.

So I told you none of it was accidental.  
 And when finals were graded, nothing was gonna stop me.  
 I peeked at Gradescope, exclaimed with newfound hope,  
 "My final exam grade is ninety nine!"

Because I am a mastermind.  
 I'm good at data science.  
 Yeah, all I can do is smile.  
 'Cause I'm a mastermind.

## Taylor Swift Emoji Quiz <sup>100</sup>

Just for fun, here are some emojis that describe particular Taylor Swift songs. See how many you can identify! We'll post the answers on Ed after the project is due.

1. 🙌🙌🧡
2. 🚫🎮
3. 🍷❤️
4. 🖼️🔥
5. 🕒☁️
6. 🍀📁1
7. 📄💍
8. ❄️🔄
9. 🤔➡️🎸
10. 🐉
11. 🔍🏀
12. 🖤🐶
13. 🏹
14. ❄️☂️
15. 🚗💨
16. 🙌🩸
17. 🎡🚫
18. 🍷🤔

19. 🍦 🌴
20. 🍷 🗑️
21. 📺 🔄
22. 🌟 💡
23. 📦 ➡️ 🍼
24. 🛒 ✈️
25. 💕 📖
26. 🗣️ ⏳
27. 🐼 🧑🎤
28. 🗑️ 👉
29. 🚫 🧑🎤
30. 🙌 🧑🎤 📺

## References and Data Sources 📖

If you're interested in learning more about the data and analysis that inspired this project, check out Alice Zhao's blog post [A Data Scientist Breaks Down All 10 Taylor Swift Albums \(The Extended Version\)](#). Here's a plot from Alice's analysis showing that Taylor Swift's music is getting more experimental over time.



Below are links to all the resources we used in developing this project. Thanks to all the people who provided these resources!

- Shayna Kothari, [Taylor Swift Lyric Searcher](#) and [source code](#)
- Tia Plagata, [How to Create Beautiful Word Clouds in Python](#)
- Melanie Walsh, [Introduction to Cultural Analytics & Python](#)
- Cameron Watts, [Extracting Song Data From the Spotify API Using Python](#)
- Alice Zhao, A Dash of Data, [A Data Scientist Breaks Down All 10 Taylor Swift Albums \(The Extended Version\)](#) and [source code](#)
- Lyrics data from [Genius API](#)
- Musical data from [Spotify API](#)

**Citations:** Did you use any generative artificial intelligence tools to assist you on this assignment? If so, please state, for each tool you used, the name of the tool (ex. ChatGPT) and the problem(s) in this assignment where you used the tool for help.

---

Please cite tools here.

---

## Submission Instructions 📧

As usual, follow these steps to submit your assignment:



Select **Kernel -> Restart & Run All** to ensure that you have executed all cells, including the test cells.

1. Read through the notebook to make sure everything is fine and all tests passed.
2. Run the cell below to run all tests, and make sure that they all pass.
3. Download your notebook using **File -> Download as -> Notebook (.ipynb)**, then upload your notebook to Gradescope.
4. If working with a partner, don't forget to add your partner as a group member on Gradescope!
5. Stick around while the Gradescope autograder grades your work. Make sure you see that all tests have passed on Gradescope.
6. Check that you have a confirmation email from Gradescope and save it as proof of your submission.

If running all the tests at once causes a test to fail that didn't fail when you ran the notebook in order, check to see if you changed a variable's value later in your code. Make sure to use new variable names instead of reusing ones that are used in the tests.

Remember, the tests here and on Gradescope just check the format of your answers. We will run correctness tests after the due date has passed.

```
In [ ]: grader.check_all()
```