Billboard Hot 100 Music Recommender

By Alex Gifford

Problem Statement

As growth in the accessibility of technology with high computation power, along with the availability of free education, has skyrocketed, music creation is easier to get into then ever before. This has led to a boom in the number of artists putting out music on various platforms such as Apple Music, Spotify, and Soundcloud, which only compounds year after year.

As more and more new artists are added every year, older artists tend to be left in the past. Most music discovery platforms focus on current trends so they will rarely recommend anything that wouldn't' be considered somewhat recent. I want to solve this problem, by creating a recommendation model that can be used to suggest songs that made it to the Billboard Hot 100 in the years 1960-1999 given a user input song.

Types of Recommenders

Content Based Recommender

- Built around information/features of items or users (i.e. song metadata)
- Each item is recorded in a row with features describing that item in the columns
- Idea is to score similarities/differences between all individual items in a matrix and provide recommendation based on that ranked score

Collaborative Based Recommender

- Built around the past interactions that have been recorded between users and items
- Data would be stored in a User-Item Interaction Matrix
- Idea is that past user-item interactions is enough to find patterns for similar users and/or recommended items

Project Flow

- 01 – Web Scrape Wikipedia
- 02 – Spotify API Client
- 03 – Gathering Metadata
- 04 – EDA
- 05 – Music Recommender: Production
- 06 – Music Recommender: In Action

Wikipedia Web Scrape: Billboard Hot 100

- I started with a web scrape of the Wikipedia entries for the Billboard Hot 100 number ones for each year 1960-99
- First consideration was that each song was broken up into multiple entries depending on when it was issued and when it left the charts
- Decided to keep 1 occurrence of each entry

List of *Billboard* Hot 100 number ones of 1960

From Wikipedia, the free encyclopedia

These are the ***Billboard* Hot 100** number one hits of 1960.

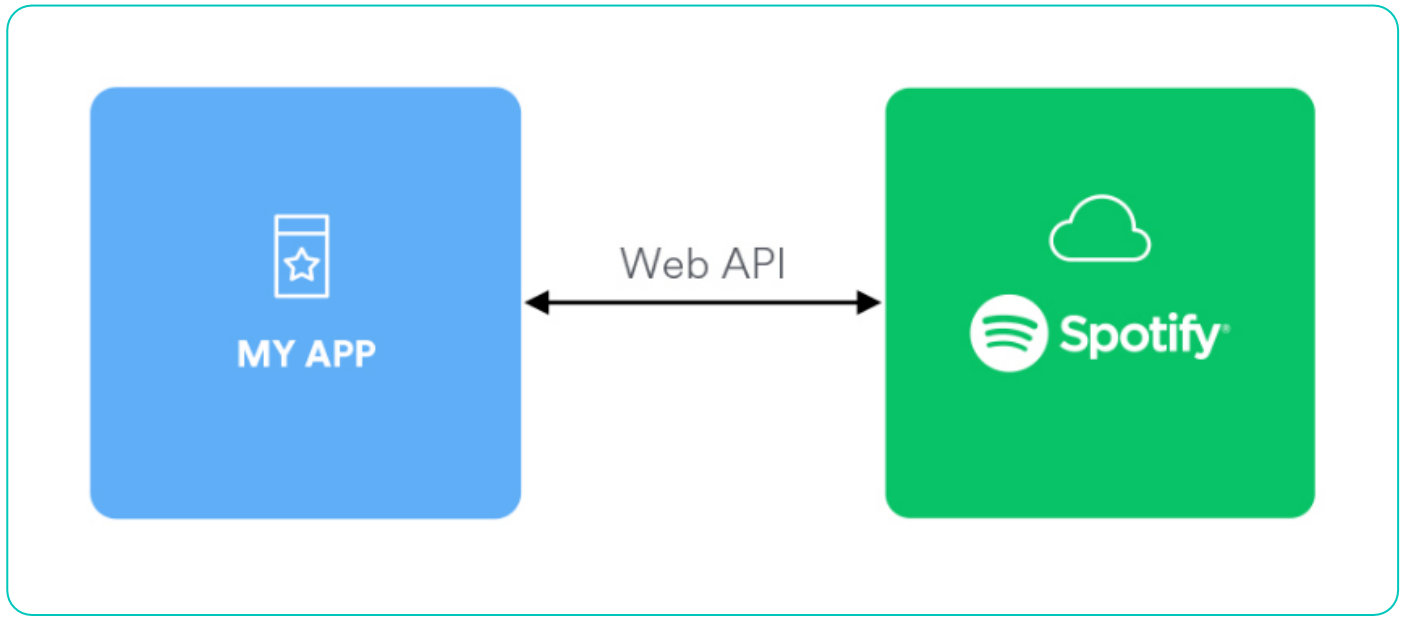
That year, 12 acts achieved their first number ones, such as Marty Robbins, Johnny Preston, Mark Dinning, Connie Francis, T
Larry Verne, The Drifters, Ray Charles, and Maurice Williams and the Zodiacs. Percy Faith and The Everly Brothers, despite t
creation of the Hot 100, also achieved their first number one songs on the chart. Elvis Presley, Connie Francis, and Brenda Le

Key

The yellow background indicates the #1 song on *Billboard*'s 1960 Year-End Chart of Pop Singles.

No.	Issue date	Song	Artist(s)	Reference
24	January 4	"El Paso"	Marty Robbins	[1]
	January 11			[2]
25	January 18	"Running Bear"	Johnny Preston	[3]
	January 25			[4]
	February 1			[5]
26	February 8	"Teen Angel"	Mark Dinning	[6]
	February 15			[7]
	February 22			[8]
	February 29			[9]
	March 7			[10]
	March 14			[11]
				[12]

Spotify WebAPI Client: BillboardHot 100



- In order to recommend songs I needed a way to measure similarity between songs
- Spotify WebAPI provides song metadata that can be pulled, however navigating to the correct song can be a challenge
- I wanted to build a client from the ground up to configure the search to make it easier to loop through the web scraped data

Quick Look at Spotify Client

```python
class SpotifyAPI(object):
    access_token = None
    access_token_expires = datetime.datetime.now()
    access_token_did_expire = True
    client_id = None
    client_secret = None
    token_url = 'https://accounts.spotify.com/api/token'



    def __init__(self, client_id=os.environ['HOT100_REC_CID'],                client_secret=os.environ['HOT100_REC_CS'], *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.client_id = client_id
        self.client_secret = client_secret

    def get_client_credentials(self):
        """
        Returns a base64 encoded string
        """
        client_id = self.client_id
        client_secret = self.client_secret
        if client_secret == None or client_id == None:
            raise Exception("You must set client_id and client_secret")
        client_creds = f'{client_id}:{client_secret}'
        client_creds_b64 = base64.b64encode(client_creds.encode())
        return client_creds_b64.decode()

    def get_token_headers(self):
        client_creds_b64 = self.get_client_credentials()
        return {
    'Authorization': f'Basic {client_creds_b64}'
              }

    def get_token_data(self):
        return {
    'grant_type': 'client_credentials'
```

```python
    def get_album(self, _id):
        return self.get_resource(_id, resource_type='albums')

    def get_artist(self, _id):
        return self.get_resource(_id, resource_type='artists')

    def get_track(self, _id):
        return self.get_resource(_id, resource_type='tracks')

    def get_features(self, _id):
        return self.get_resource(_id, resource_type='audio-features')

    def get_analysis(self, _id):
        return self.get_resource(_id, resource_type='audio-analysis')

    def base_search(self, query_params):
        access_token = self.get_access_token()
        headers = self.get_resource_header()
        endpoint = "https://api.spotify.com/v1/search"
        lookup_url = f"{endpoint}?{query_params}"
        r = requests.get(lookup_url, headers=headers)
        if r.status_code not in range(200, 299):
            return {}

        return r.json()

    def search(self, query=None, operator=None, operator_query=None, search_type='artist'):
        if query == None:
            raise Exception("A query is required")
        if isinstance(query, dict):
            query = " ".join([f"{k}:{v}" for k,v in query.items()])
        if operator != None and operator_query != None:
            if operator.lower() == "or" or operator.lower() == "not":
                operator = operator.upper()
                if isinstance(operator_query, str):
                    query = f"{query} {operator} {operator_query}"
        query_params = urlencode({"q": query, "type": search_type.lower()})
        return self.base_search(query_params)
```

Data Collection and Search Validation

- In order to get the right song from the Spotify API I built a validation algorithm to make sure the search returned the song with the same artist released in the same year or earlier then the year it appeared on the Hot 100
- One choice I made was to not limit the song to only 1 version and include any version of the song the search turned up with. This means live versions, re-recordings, re-works, and re-masters will be included.

```
for song in df['song'][c1:]:
    # Collect search query for each song in the loop
    track_search = spotify.search({"track": str(song)}, search_type="track")

    if len(track_search['tracks']['items']) > 0:
        # The limit for each search query is 20 so the i counter will
        # loop through a range of 20 to check each item in the query against
        # the song from hot100.
        for i in range(len(track_search['tracks']['items'])):

            song_info = []

            # Check if hot100 song matches query items song name
            # replace method was added due to name issues with
            # 'Theme from A Summer Place'. Based on this more cleaning
            # may be needed as I go through the rest of the hot100 dataframe.
            if song in track_search['tracks']['items'][i]['name'].replace('"', ''):

                # Check if hot100 song's artist matches the name of the
                # query items artist as well as the artist listed under
                # the query items album. Some items match artist name but are
                # from compilation albums which are listed as having various artists.
                if (df['artist'][c1] in track_search['tracks']['items'][i]['artists'][0]['name']) and\
                (df['artist'][c1] in track_search['tracks']['items'][i]['album']['artists'][0]['name']):

                    # Check to see if query item was released in the
                    # same year or earlier as the hot100 song. Some albums get
                    # reissued at later dates so this check helps filter those out
                    if df['year'][c1] <= int(track_search['tracks']['items'][i]['album']['release_date'][:4]):
                        song_info.append(track_search['tracks']['items'][i]['name'])
                        song_info.append(track_search['tracks']['items'][i]['album']['name'])
                        song_info.append(track_search['tracks']['items'][i]['album']['artists'][0]['name'])
                        song_info.append(track_search['tracks']['items'][i]['popularity'])
                        song_info.append(track_search['tracks']['items'][i]['id'])
                        song_info.append(track_search['tracks']['items'][i]['explicit'])

                        track_list.append(song_info)
```

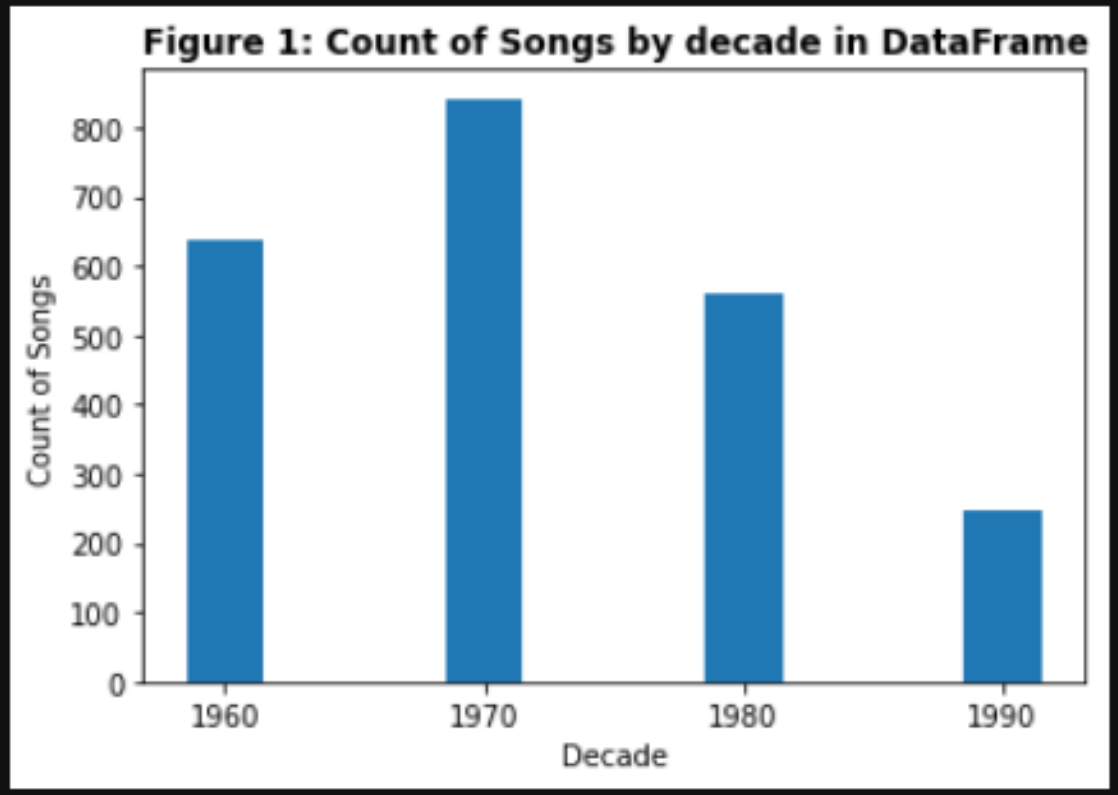
Song Features and Analysis

The recommender would need each song to have metadata that it could compare against itself to determine similarity. Here are the main features used by the recommender model ---------->

Song Features:

- **Danceability**
- **Energy**
- **Key**
- **Loudness**
- **Mode**
- **Speechiness**
- **Acousticness**
- **Instrumentalness**
- **Liveness**
- **Valence**
- **Tempo**

Understanding Decade Distribution



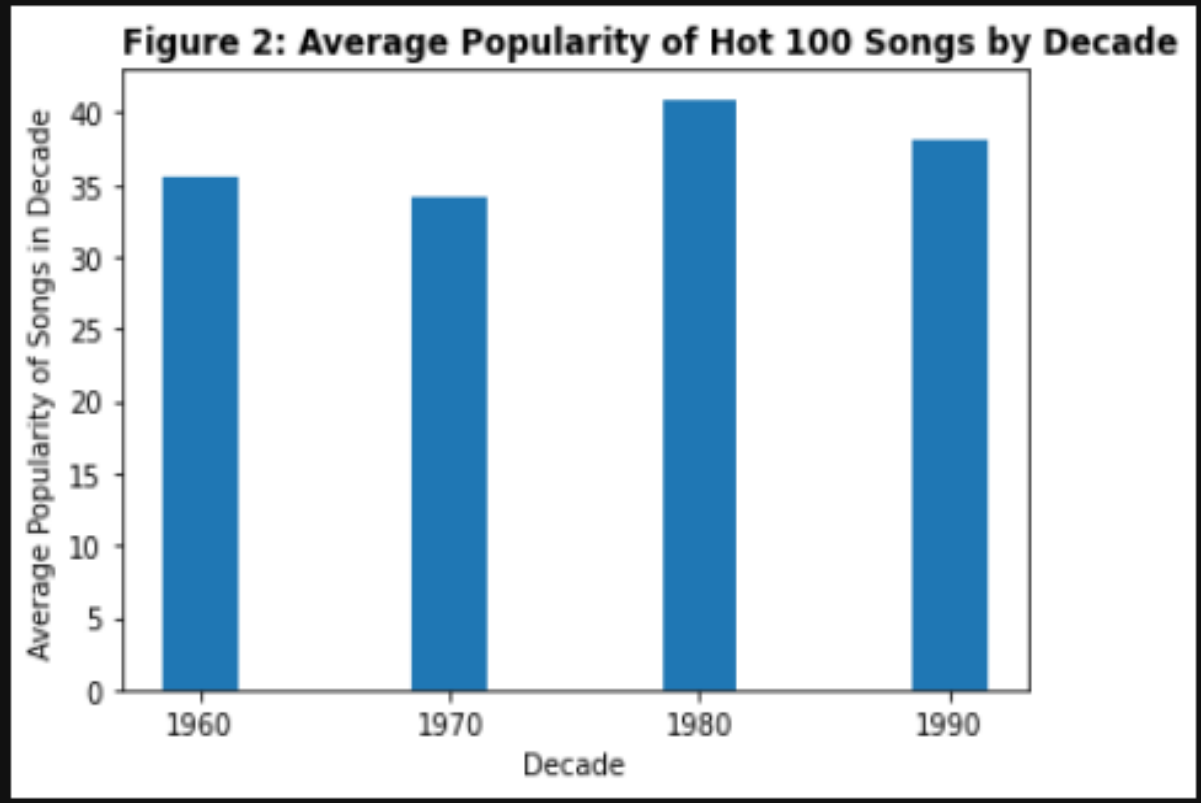
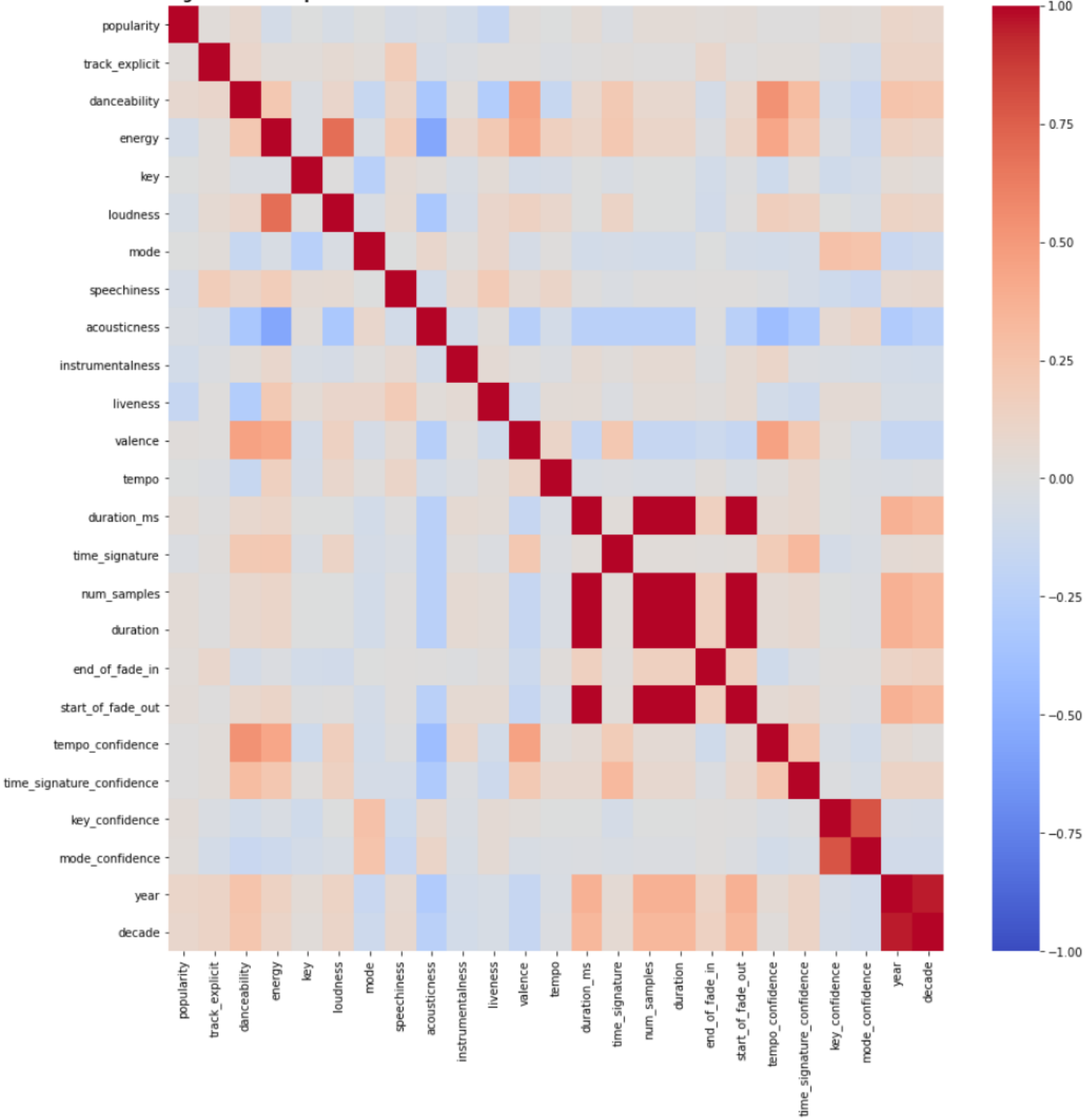


Figure 3. Heatmap of Track Features



Recommendation Model

- Steps to Recommendation Model:
  - Find User Input song and gather metadata from Spotify WebAPI
  - Match new user song dataframe with format of Main Reference dataframe and concatenate
  - Find and sort on the cosine similarity score comparing main dataframe against itself
  - Print out top 10 recommendations with no artist being repeated

```
class RecommendSong:
    col_name = [
        'song', 'album', 'artist', 'popularity', 'track_id',
        'track_explicit', 'danceability', 'energy', 'key', 'loudness', 'mode',
        'speechiness', 'acousticness', 'instrumentalness', 'liveness',
        'valence', 'tempo', 'duration_ms', 'time_signature', 'num_samples',
        'duration', 'end_of_fade_in', 'start_of_fade_out', 'tempo_confidence',
        'time_signature_confidence', 'key_confidence', 'mode_confidence']
    df = pd.read_csv('../data/spotify_final.csv').drop('Unnamed: 0', axis=1)

    def __init__(self, song_name, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.song_name = song_name
        self.song_id = None
        self.track_data = []

    def find_song(self):
        spotify = SpotifyAPI()
        song_id = 0
        song_name = self.song_name
        i = 0
        song_search = spotify.search({"track": song_name}, search_type="track")
        if len(song_search['tracks']['items']) > 0:
            while i < len(song_search['tracks']['items']):
                if self.song_name == song_search['tracks']['items'][i]['name']:
                    song_id = song_search['tracks']['items'][i]['id']
                    return song_id
                else:
                    i += 1
            if song_id != 0:
                return song_id
            else:
                return 404
        else:
            return 404
        return 404

    def get_track_data(self):
        song_id = self.song_id
        spotify = SpotifyAPI()
        track = spotify.get_track(song_id)
        self.track_data.append(track['name'])
        self.track_data.append(track['album']['name'])
        self.track_data.append(track['artists'][0]['name'])
        self.track_data.append(track['popularity'])
        self.track_data.append(track['id'])
        self.track_data.append(int(track['explicit']))

    def get_track_features(self):
        song_id = self.song_id
        spotify = SpotifyAPI()
        track_features = spotify.get_features(song_id)
        self.track_data.append(track_features['danceability'])
        self.track_data.append(track_features['energy'])
        self.track_data.append(track_features['key'])
        self.track_data.append(track_features['loudness'])
        self.track_data.append(track_features['mode'])
        self.track_data.append(track_features['speechiness'])
        self.track_data.append(track_features['acousticness'])
        self.track_data.append(track_features['instrumentalness'])
        self.track_data.append(track_features['liveness'])
        self.track_data.append(track_features['valence'])
        self.track_data.append(track_features['tempo'])
        self.track_data.append(track_features['duration_ms'])
        self.track_data.append(track_features['time_signature'])
```

```
    def get_track_analysis(self):
        song_id = self.song_id
        spotify = SpotifyAPI()
        track_analysis = spotify.get_analysis(song_id)
        self.track_data.append(track_analysis['track']['num_samples'])
        self.track_data.append(track_analysis['track']['duration'])
        self.track_data.append(track_analysis['track']['end_of_fade_in'])
        self.track_data.append(track_analysis['track']['start_of_fade_out
        self.track_data.append(track_analysis['track']['tempo_confidence
        self.track_data.append(track_analysis['track']['time_signature_co
        self.track_data.append(track_analysis['track']['key_confidence'])
        self.track_data.append(track_analysis['track']['mode_confidence'])

    def search(self):
        self.song_id = self.find_song()
        if self.song_id == 404:
            raise Exception("Song not found.")
        else:
            self.get_track_data()
            self.get_track_features()
            self.get_track_analysis()
        return pd.DataFrame([self.track_data], columns=self.col_name)

    def print_recommendations(self, indi, rec_df):
        print(f"For the song {self.song_name} by {rec_df['artist'].iloc[i
        artist_already_featured = []
        c = 0
        for i in indi[1:]:
            if rec_df['artist'].iloc[i] in artist_already_featured:
                pass
            else:
                print(f"{c+1}. {rec_df['song'].iloc[i].title()} by {rec_d
                artist_already_featured.append(rec_df['artist'].iloc[i])
                c += 1
                if c >= 10:
                    break


    def recommend(self):
        search_df = self.search()
        rec_df = pd.concat([df, search_df], ignore_index=True)
        features = [x for x in df.columns if x not in ['song', 'album',
        cosine_similarities = cosine_similarity(rec_df[features])
        indicies = pd.Series(rec_df.index, index=rec_df['song'])
        idx = indicies[self.song_name]
        sim_scores = list(enumerate(cosine_similarities[idx]))
        sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
        indi = [i[0] for i in sim_scores]
        return self.print_recommendations(indi, rec_df)
```

---------- Begin Here ----------

---------- Example Recommendation Searches ----------

Example 1: New Light by John Mayer

[2]:
```
ex_1 = RecommendSong("New Light")
ex_1.recommend()
```

```
For the song New Light by John Mayer, we recommend you check out:

1. I Want You Back by The Jackson 5

2. Brand New Key by Melanie

3. Against All Odds Take A Look At Me Now  2016 Remaster by Phil Collins

4. Sweet Dreams Are Made Of This  Remastered by Eurythmics

5. I Got You Babe by Sonny & Cher

6. Take On Me  2017 Acoustic by Aha

7. Believe by Cher

8. Let Your Love Flow by The Bellamy Brothers

9. Heaven by Bryan Adams

10. Together Forever by Rick Astley
```

Example 2: Lose Yourself by Eminem

[4]:
```
ex_2 = RecommendSong("Lose Yourself")
ex_2.recommend()
```

```
For the song Lose Yourself by Eminem, we recommend you check out:

1. Every Rose Has Its Thorn by Poison

2. Brown Sugar by The Rolling Stones

3. Rhinestone Cowboy by Glen Campbell

4. Beat It by Michael Jackson

5. Boogie Oogie Oogie  Remastered by A Taste Of Honey

6. Another One Bites The Dust  2011 Remaster by Queen

7. Flashdance... What A Feeling by Irene Cara

8. Oh Sheila by Ready For The World

9. Centerfold by The J. Geils Band

10. Money For Nothing by Dire Straits
```

Example 3: good 4 u by Olivia Rodrigo

[7]:
```
ex_3 = RecommendSong("good 4 u")
ex_3.recommend()
```

```
For the song good 4 u by Olivia Rodrigo, we recommend you check out:

1. I Want To Hold Your Hand  Remastered 2015 by The Beatles

2. Its My Party by Lesley Gore

3. The Locomotion by Little Eva

4. Hello, I Love You by The Doors

5. I Get Around Mono by The Beach Boys

6. Baby Love by The Supremes

7. Everyday People by Various Artists

8. Please Mr. Postman by The Marvelettes

9. Big Girls Dont Cry by Frankie Valli & The Four Seasons

10. Stuck On You by Elvis Presley
```

Recommender Model Search Examples