Music Recommendation System

Prepared By:

PARVEJ ALAM MAHAMMADDIN ANSARI
Intern ID: SMI65929

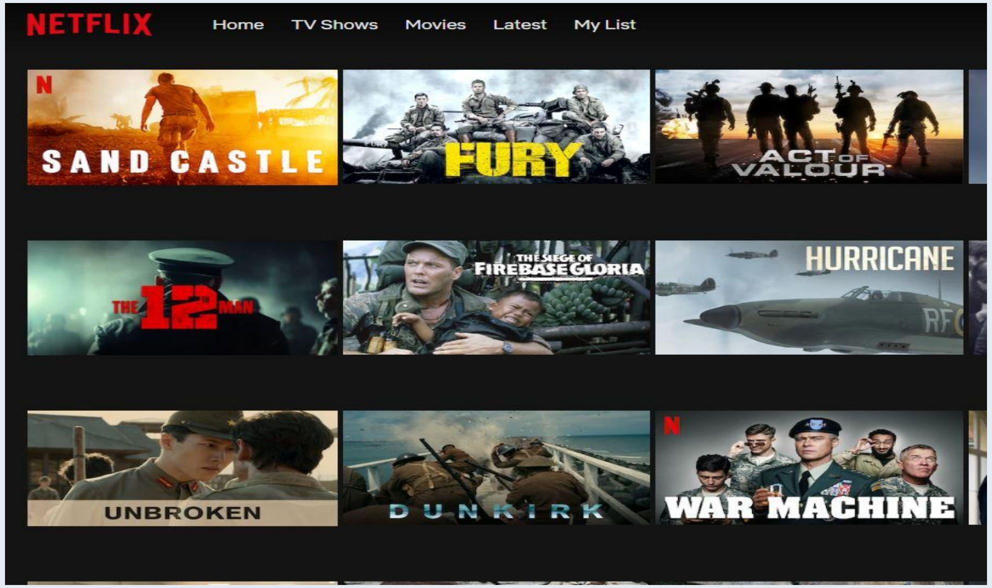
Music Recommender System

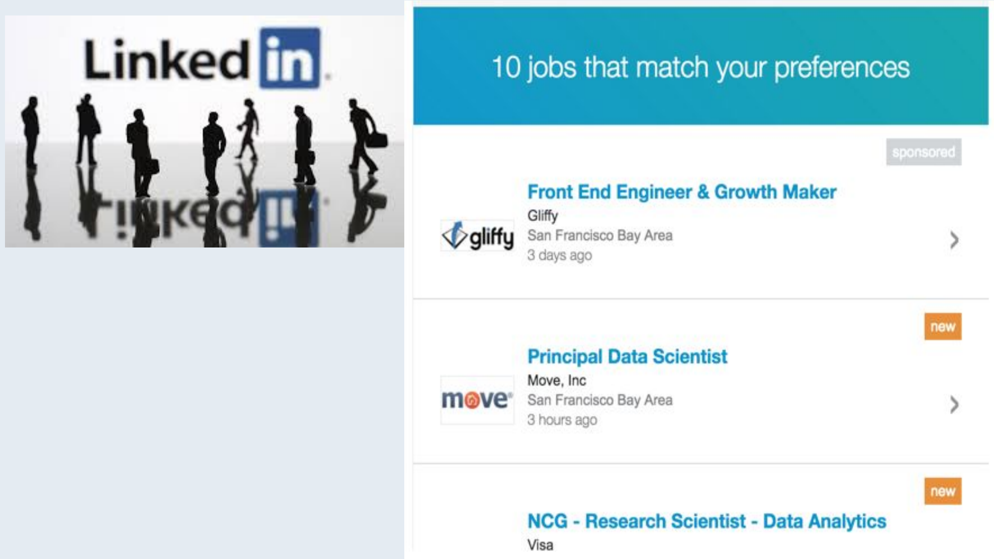
1. Why Recommender Systems ?
2. How to build a Recommender System ?
    a. Popularity Based Filtering
    b. Content Based Filtering (Classification Based)
    c. Collaborative Based Filtering
        i. Nearest Neighbor
        ii. Matrix factorization
3. About K-Means Clustering
4. Dataset Information
5. Collaborative Recommendation System
6. Content-based Recommendation System
7. Conclusion

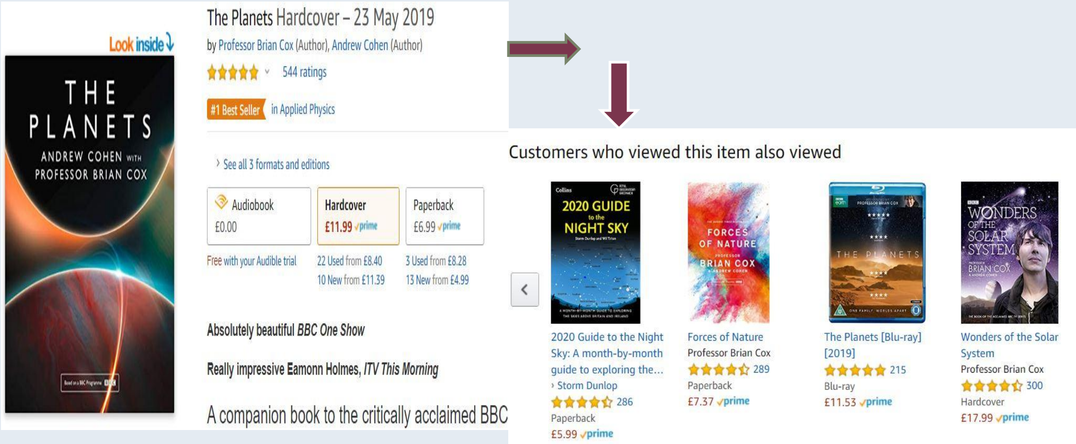
1. Why Recommender Systems ?

- Personalized Experience: A recommender system makes prediction that are tailored and specific to a user
- Customer Satisfaction: Users are left satisfied as they do not need to spend time searching for what they might like as the system does it for them
- Discovery: Users can discover new items that are similar to their taste, making it an engaging experience
- Increased Profit: Users then to spend more time on sites that make adequate prediction of what they might like or want, which could translate to spending more money on purchases.
- Business Analysis: A recommender system make it easy for a business to analysis its customer behaviour and put up product that their customers want.

Examples : Movie/TV show Recommendations, Friend Recommendations, Job Recommendations, Product Recommendations

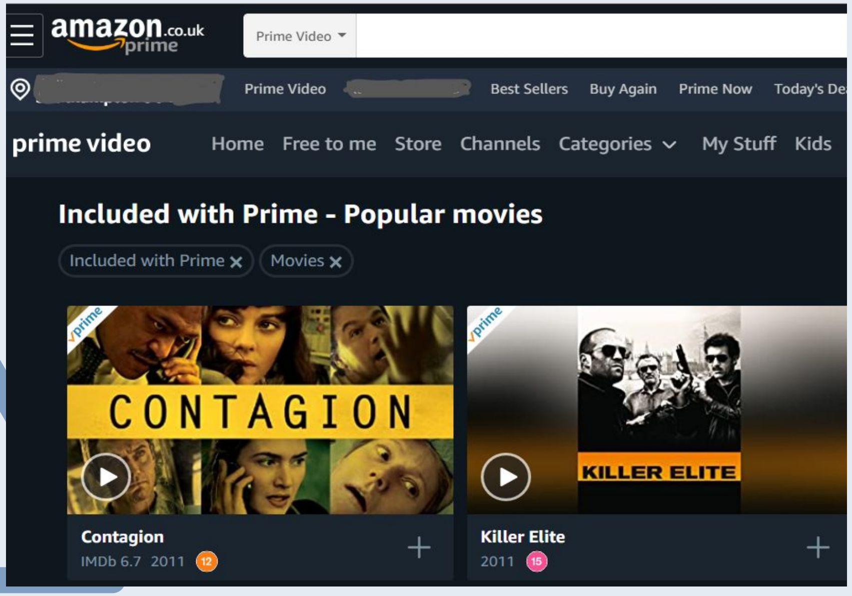






2. Building a Recommender System
a. Popularity Based Recommender System
b. Content Based Filtering

a. Popularity Based Recommender System : Recommend Items viewed/purchased by most people, Recommender ranks list of items by their purchase count



b. Content Based Filtering: Use features of both products and users in order to predict whether a user will like a product or not.

User Features
Eg: Age, Gender

Product Features
Eg: Cost, Quality

Purchase History

→ Classifier → Like/not like

Limitations:
1. It is difficult to collect high quality information about products and users.

c. Nearest Neighbor Collaborative Filtering

User-based Collaborative Filtering

Find users who have a similar taste of songs as the current user .

Similarity is based upon similarity in user's listening mood.

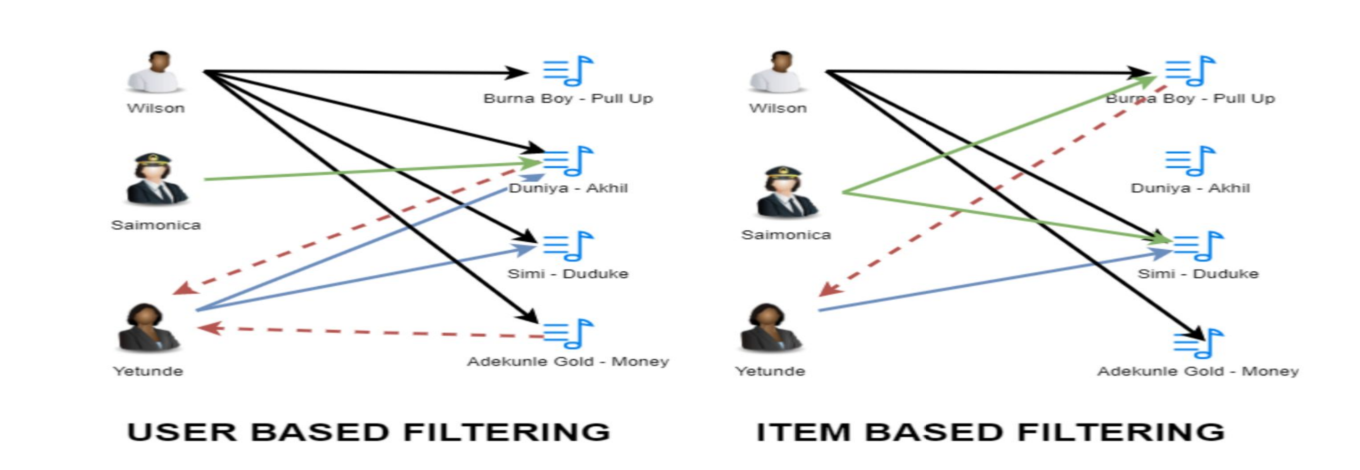
Eg: "User A is similar to user B because both listened items are X,Y and Z."

Item - Based Collaborative Filtering

Recommend songs that are similar to the songs the user listened to.

Similarity is based upon co-occurrence of songs.

Eg: "Songs A and B were listened by both users X and Y, so they are similar."



Item-Based Collaborative Filtering:

An Example (People who listened A this also listened C)

History Matrix

Co-occurrence Matrix items by items

	A	B	C
A		1	2
B	1		1
C	2	1	

Effect of Popular songs : Example, if everybody has listened to X song, then it is not a very good indicator of what to recommend next, the recommender would become similar to a popularity based recommender engine.

Normalize Co-occurrence Matrix : Normalize by popularity , number of users common for I and j and number of users for either I and j.

Effect of Multiple Items : Weighted sum , Ranked Recommendations

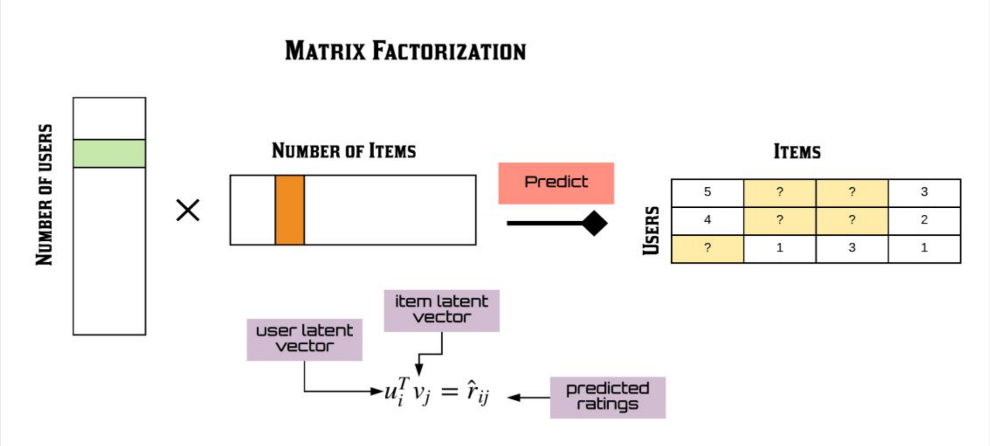
	A	B	C
A		1/3	2/2
B	1/3		1/2
C	2/3	1/2	

x

10,000/10002

Model Based Collaborative Filtering (Matrix Factorization Method)

Identity latent(hidden) features from the input user X song Ratings Matrix to represent users and songs as vectors in N dimensional space.



(Serious/Escapist?) Geared towards Males or Females?

User Vector (u) = [1.3 2.8]

Item Vector (v) = [2.5 -1.9]

New user (Known ratings): [4 5 ….3]

Training: Use Matrix Factorization approaches (eg. Singular Value Decomposition (SVD)) to split the Rating Matrix into constituent User Matrix and Item Matrix with minimum (Sum is squared error (SSE)).

Goal : Predict unknown ratings for the remaining set of songs using the learned User Matrix and Item Matrix

SOME IMPORTANT POINTS :

1. Popularity Based recommender system can use purchase history and its scalable.
2. Content Based (Classification Model) Recommender System
   a. Can generate the Personalized Recommendations
   b. Can use User and Item Features
   C. Can use the purchase History and it is not scalable
3. User X item Ratings sparse matrix(any huge data), example: size 480,189 X 17,770 –. Model Based Collaborative Filtering (MATRIX FACTORIZATION) method used.

Which model can handle the brand new items ??

COMPARISION OF RECOMMENDATION SYSTEMS

	Popularity Based	Content Based	(Nearest Neighbor-based CF)	(Matrix Factorization based CF)
Personalized Recommendations		YES	YES	YES
Uses Context (Eg. time of day)	YES	YES		
User Features		YES		YES
Item Features		YES		YES
Can handle brand new Items?	NO	YES	NO	YES
Purchase History		YES	YES	YES
Scalable	YES	NO		YES

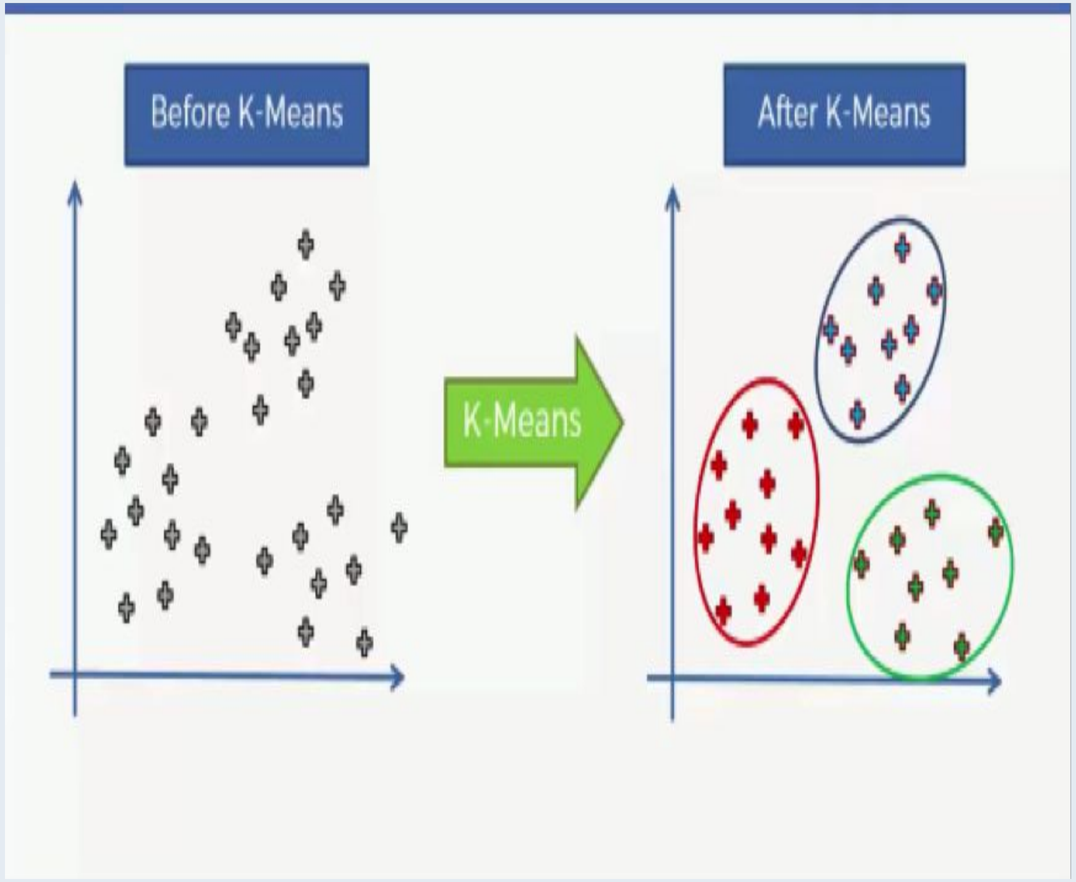
3. About K-Means Clustering

Clustering is the process of dividing the datasets into groups, consisting of similar data-points and its main aim is to group similar elements or group points into a cluster.

1. Clustering is unsupervised learning – It will have unknown number of classes , no prior knowledge and finds the natural groupings between objects.
2. Clustering methods are basically used to automatically group the retrieved documents into a list of meaningful categories

K-MEANS in Recommendation Systems:

K-Means clustering was implemented by creating a cluster of songs in the dataset. Based on the cluster, recommendation was then made. One of the advantages of this method is that it scales well with very large dataset.



4. Dataset Information

We are going to use the **Million Song Dataset**, a freely-available collection of audio features and metadata for a million contemporary popular music tracks.

There are two files that will be interesting for us. The first of them will give us information about the songs. Particularly, it contains the user ID, song ID and the listen count. On the other hand, the second file will contain song ID, title of that song, release, artist name and year. We need to merge these two DataFrames. For that aim, we'll use the `song_ID`

Dataset Links:

1. https://static.turi.com/datasets/millionsong/10000.txt
2. https://static.turi.com/datasets/millionsong/song_data.csv

5. Collaborative Recommendation System

In [9]:
```
df_songs.head()
```
Out[9]:

	user_id	song_id	listen_count	title	release	artist_name	year
0	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAKIMP12A8C130995	1	The Cove	Thicker Than Water	Jack Johnson	0
1	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBBMDR12A8C13253B	2	Entre Dos Aguas	Flamenco Para Niños	Paco De Lucia	1976
2	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBXHDL12A81C204C0	1	Stronger	Graduation	Kanye West	2007
3	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBYHAJ12A6701BF1D	1	Constellations	In Between Dreams	Jack Johnson	2005
4	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SODACBL12A8C13C273	1	Learn To Fly	There Is Nothing Left To Lose	Foo Fighters	1999

Then, we'll check how many observations there are in the dataset.

In [10]:
```
#Get total observations
print(f"There are {df_songs.shape[0]} observations in the dataset")
```
```
There are 2000000 observations in the dataset
```

Basic Exploratory Data Analysis

continue...

In [11]:
```
df_songs.isnull().sum()
```

Out[11]:
```
user_id        0
song_id        0
listen_count   0
title          0
release        0
artist_name    0
year           0
dtype: int64
```

And most of the columns contain strings.

In [12]:
```
df_songs.dtypes
```

Out[12]:
```
user_id        object
song_id        object
listen_count    int64
title          object
release        object
artist_name    object
year            int64
dtype: object
```

Basic Exploratory Data Analysis (Continue..)

continue…

```
In [16]:
#count how many rows we have by song, we show only the ten more popular songs
ten_pop_songs = df_songs.groupby('title')['listen_count'].count().reset_index().sort_values(['listen_count', 'tit
ten_pop_songs['percentage']  = round(ten_pop_songs['listen_count'].div(ten_pop_songs['listen_count'].sum())*100,
```

```
In [17]:
ten_pop_songs = ten_pop_songs[:10]
ten_pop_songs
```

Out[17]:

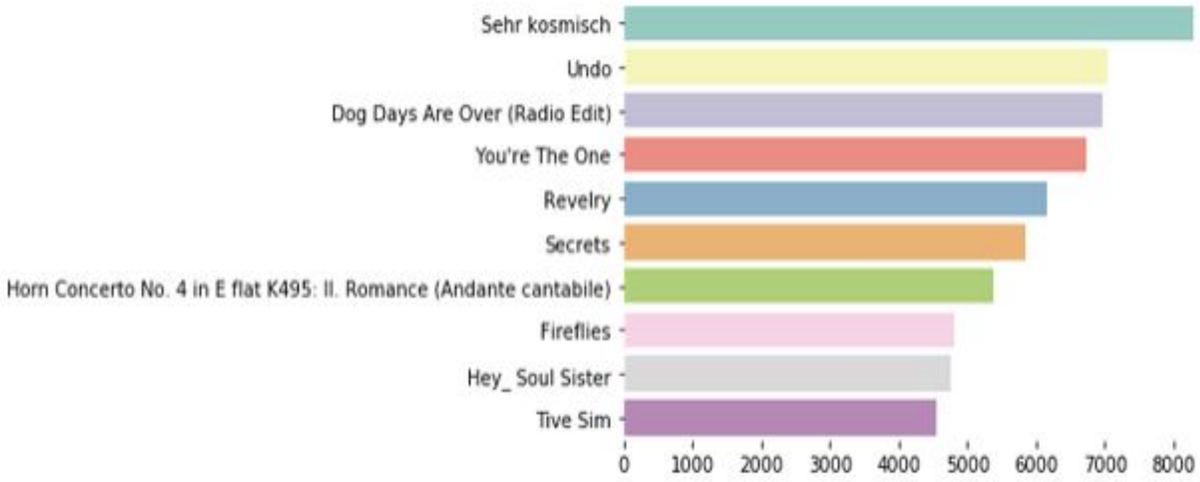
	title	listen_count	percentage
6836	Sehr kosmisch	8277	0.41
8725	Undo	7032	0.35
1964	Dog Days Are Over (Radio Edit)	6949	0.35
9496	You're The One	6729	0.34
6498	Revelry	6145	0.31
6825	Secrets	5841	0.29
3437	Horn Concerto No. 4 in E flat K495: II. Romanc...	5385	0.27
2595	Fireflies	4795	0.24
3322	Hey_ Soul Sister	4758	0.24
8494	Tive Sim	4548	0.23

Most popular songs

continue…

In [18]:
```
labels = ten_pop_songs['title'].tolist()
counts = ten_pop_songs['listen_count'].tolist()
```

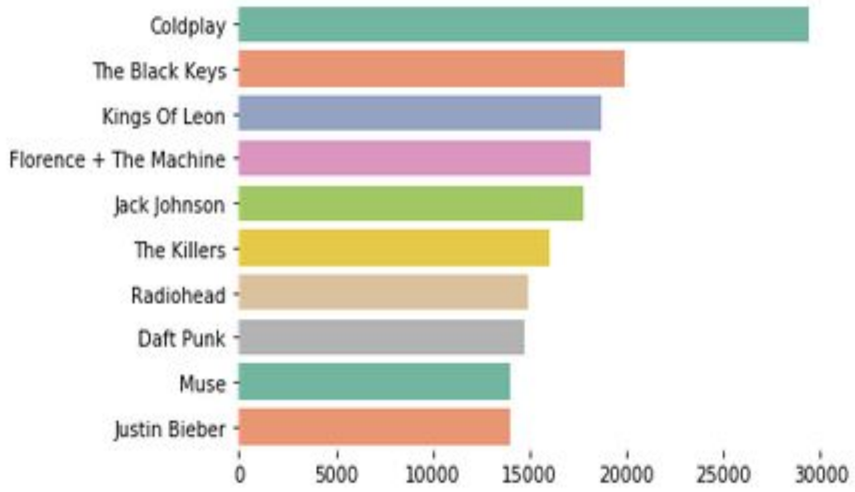
In [19]:
```
plt.figure()
sns.barplot(x=counts, y=labels, palette='Set3')
sns.despine(left=True, bottom=True)
```



Most popular songs

continue...

In [22]:
```
plt.figure()
labels = ten_pop_artists['artist_name'].tolist()
counts = ten_pop_artists['listen_count'].tolist()
sns.barplot(x=counts, y=labels, palette='Set2')
sns.despine(left=True, bottom=True)
```



Most popular artists

continue...

```
In [42]:  model = Recommender(metric='cosine', algorithm='brute', k=20, data=mat_songs_features, decode_id_song=decode_id_s

In [43]:  song = 'I believe in miracles'

In [44]:  new_recommendations = model.make_recommendation(new_song=song, n_recommendations=10)

I believe in miracles
Starting the recommendation process for I believe in miracles ...
... Done

In [62]:  print(f"The recommendations for {song} are:")
          print(f"{new_recommendations}")

The recommendations for I believe in miracles are:
Nine Million Bicycles
If You Were A Sailboat
Shy Boy
I Cried For You
Spider's Web
Piece By Piece
On The Road Again
Blues In The Night
Blue Shoes
Thank You Stars
```

Model and Recommendations

6. Content-based Recommendation System

```
In [3]: songs = pd.read_csv('content based recommedation system/songdata.csv')
```

```
In [4]: songs.head()
```

Out[4]:

	artist	song	link	text
0	ABBA	Ahe's My Kind Of Girl	/a/abba/ahes+my+kind+of+girl_20598417.html	Look at her face, it's a wonderful face \nAnd...
1	ABBA	Andante, Andante	/a/abba/andante+andante_20002708.html	Take it easy with me, please \nTouch me gentl...
2	ABBA	As Good As New	/a/abba/as+good+as+new_20003033.html	I'll never know why I had to go \nWhy I had t...
3	ABBA	Bang	/a/abba/bang_20598415.html	Making somebody happy is a question of give an...
4	ABBA	Bang-A-Boomerang	/a/abba/bang+a+boomerang_20002668.html	Making somebody happy is a question of give an...

Basic Exploratory Data Analysis

continue…

```
In [5]: songs = songs.sample(n=5000).drop('link', axis=1).reset_index(drop=True)
```

We can notice also the presence of `\n` in the text, so we are going to remove it.

```
In [6]: songs['text'] = songs['text'].str.replace(r'\n', '')
```

Data Selection and Data Cleaning

continue…

After that, we use TF-IDF vectorizerthat calculates the TF-IDF score for each song lyric, word-by-word.

Here, we pay particular attention to the arguments we can specify.

In [7]:
```
tfidf = TfidfVectorizer(analyzer='word', stop_words='english')
```

In [8]:
```
lyrics_matrix = tfidf.fit_transform(songs['text'])
```

TF-IDF Vectorization Technique (Word to Vector Conversion)

continue…

We now need to calculate the similarity of one lyric to another. We are going to use **cosine similarity**.

We want to calculate the cosine similarity of each item with every other item in the dataset. So we just pass the lyrics_matrix as argument.

In [12]:
```
cosine_similarities = cosine_similarity(lyrics_matrix)
```

Once we get the similarities, we'll store in a dictionary the names of the 50 most similar songs for each song in our dataset.

In [70]:
```
similarities = {}
```

In [133]:
```
for i in range(len(cosine_similarities)):
    # Now we'll sort each element in cosine_similarities and get the indexes of the songs.
    similar_indices = cosine_similarities[i].argsort()[:-50:-1]
    # After that, we'll store in similarities each name of the 50 most similar songs.
    # Except the first one that is the same song.
    similarities[songs['song'].iloc[i]] = [(cosine_similarities[i][x], songs['song'][x], songs['artist'][x]) for
```

After that, all the magic happens. We can use that similarity scores to access the most similar items and give a recommendation.

Applying Concept of Cosine Similarity

continue...

For that, we'll define our Content based recommender class.

In [134]:
```
class ContentBasedRecommender:
    def __init__(self, matrix):
        self.matrix_similar = matrix

    def _print_message(self, song, recom_song):
        rec_items = len(recom_song)

        print(f'The {rec_items} recommended songs for {song} are:')
        for i in range(rec_items):
            print(f"Number {i+1}:")
            print(f"{recom_song[i][1]} by {recom_song[i][2]} with {round(recom_song[i][0], 3)} similarity score")
            print("--------------------")

    def recommend(self, recommendation):
        # Get song to find recommendations for
        song = recommendation['song']
        # Get number of songs to recommend
        number_songs = recommendation['number_songs']
        # Get the number of songs most similars from matrix similarities
        recom_song = self.matrix_similar[song][:number_songs]
        # print each item
        self._print_message(song=song, recom_song=recom_song)
```

Now, instantiate class

In [135]:
```
recommedations = ContentBasedRecommender(similarities)
```

Creating Recommendation Class

continue…

Then, we are ready to pick a song from the dataset and make a recommendation.

In [136]:
```
recommendation = {
    "song": songs['song'].iloc[10],
    "number_songs": 4
}
```

In [137]:
```
recommedations.recommend(recommendation)
```

```
The 4 recommended songs for Jehovah And All That Jazz are:
Number 1:
Sing by Glen Campbell with 0.166 similarity score
--------------------
Number 2:
Devil Cried by Black Sabbath with 0.149 similarity score
--------------------
Number 3:
Angelique by Kenny Loggins with 0.141 similarity score
--------------------
Number 4:
Up The Devil's Pay by Old 97's with 0.131 similarity score
--------------------
```

Use Case: 01_ Recommending Song

continue...

And we can pick another random song and recommend again:

```
In [138]:
recommendation2 = {
    "song": songs['song'].iloc[120],
    "number_songs": 4
}
```

```
In [139]:
recommedations.recommend(recommendation2)
```

```
The 4 recommended songs for I Do It For Your Love are:
Number 1:
I Love You by Lionel Richie with 0.189 similarity score
--------------------
Number 2:
Just One Love by Michael Bolton with 0.187 similarity score
--------------------
Number 3:
I'm Gonna Sit Right Down And Write Myself A Letter by Nat King Cole with 0.184 similarity score
--------------------
Number 4:
If I Love Again by Barbra Streisand with 0.183 similarity score
--------------------
```

Use Case: 02_ Recommending Song

7. Conclusion

- Popularity-Based Model works in cases popularity of an items is only considered which is not suited for the type of recommender system.
- Collaborative Filtering Model using Matrix Factorization gave a good result, it was able to make personalized recommendations, which is better suited for our model.
- KMeans- Clustering Model can also be explored as an alternative to building a recommender system, as the built model was able to cluster songs based on listen_count.

THANK YOU