Music Recommendations with AIML

MUSIC & last.fm

Alline Ayala, Jonathan Haile, & Noah Teckle
USC Apple NACME AIML Bootcamp 2024
RecSetters Capstone Project

TEAM

Alline Ayala
https://www.linkedin.com/in/alline-ayala/
Texas A&M - Junior
Electronic Systems & Minor in Cybersecurity

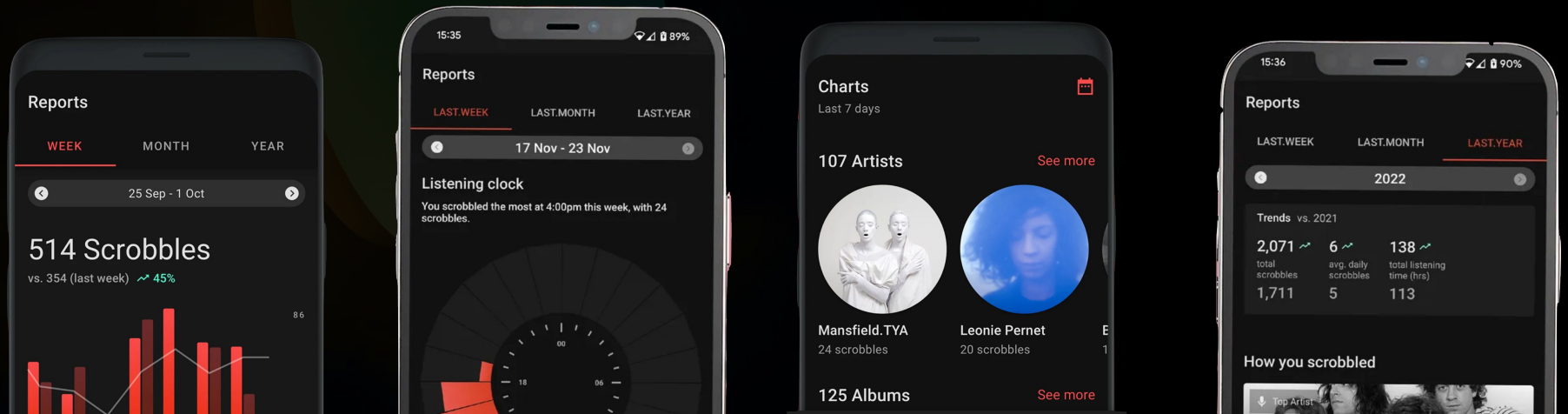
Noah Tekle
https://www.linkedin.com/in/noah-tekle/
University of Southern California- Junior
Electrical Engineering

Jonathan Haile
https://www.linkedin.com/in/jonathan1haile/
University of Southern California - Junior
Computer Science & Business Administration

GUIDE: Abdulla Alshabanah PhD in Computer Engineering at USC

Last.fm

Last.fm is a music tracking app that can sync to users' music services. Last fm is compatible with Spotify, Apple Music, Youtube, Tidal, SoundCloud, and Deezer. It gives each user's details on their streaming times on certain devices and services, and allows them to keep track of their listening as well as suggest new music. Our team will be using Last.fm's API to gather users for our dataset.



Apple Music

Apple Music is an audio streaming service developed by Apple Inc. It has a collection of over 100 million songs and allows its subscribers to listen to spatial and lossless audio. For our project, we will be using a dataset of 10,000 Apple Music songs collected and uploaded to Kaggle. The Kaggle dataset contains a variety of diverse genres, artists, and songs.This Kaggle dataset was in compliance with legal frameworks and has a usability of 9.41.

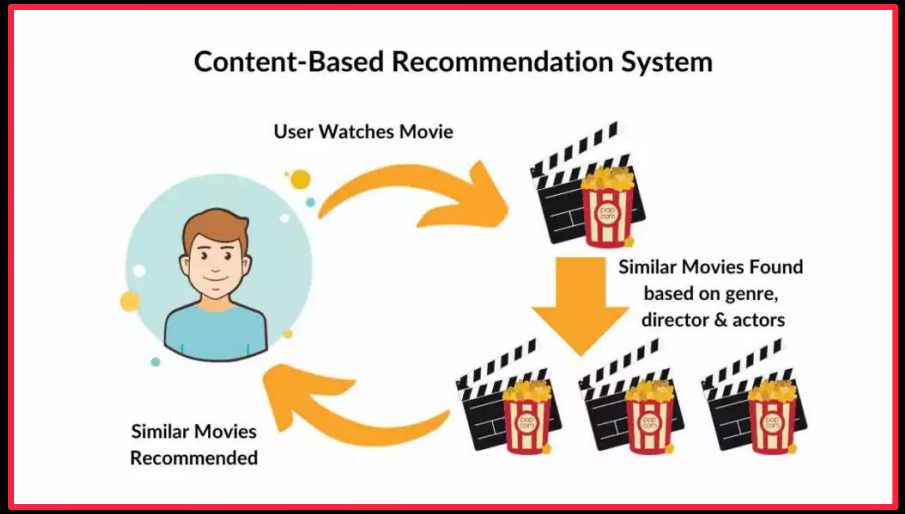
Our Task

Team RecSetter's task is to create a study and create a music recommendation system designed for an Apple Music dataset. We will be using Deep Learning to make this possible with the use of a model-based approach with an MLP (Multi-Layer Perceptron) that will learn representations.

How do Recommendation Systems Work?

Content-Based Filtering

- This approach attempts to guess the features or behavior of each user according to the features of the items they interact with.
    - This method can benefit from deep learning by using neural networks to extract complex feature **representations** from user interactions with items.
        - The original data is not directly used in making predictions
- Estimate the rating a user will give other items based on their ***individual*** behavior.

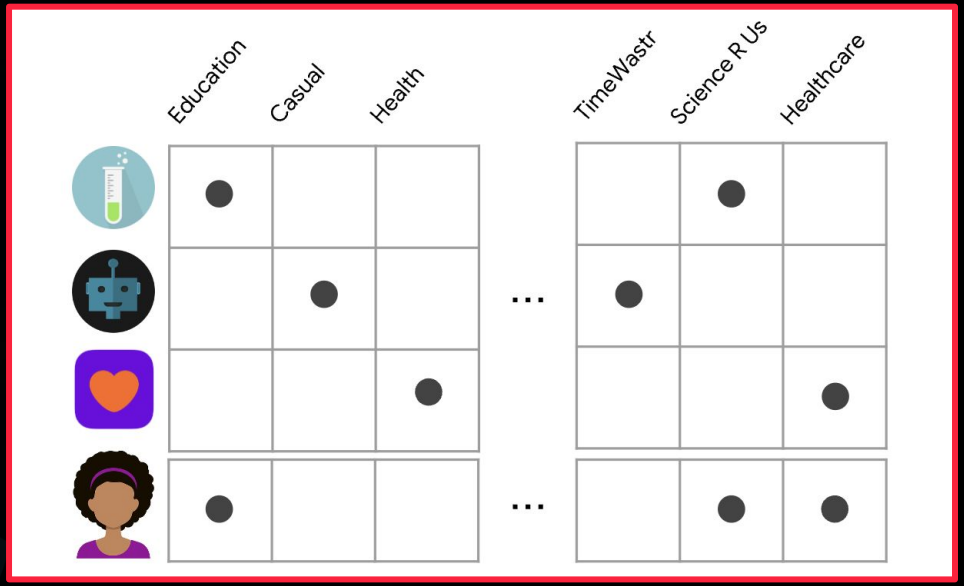


E-commerce, movie recommendations, app stores

How do Recommendation Systems Work?

Content-Based Filtering

- Uses the dot product similarity
  - This method can benefit from deep learning by using neural networks to extract complex feature representations from user interactions with items.
- Assigns items a binary value based on interaction
  - Deep learning can learn more interactions beyond binary data
- If a feature appears in both x and y vectors, there is a higher similarity.
- A higher dot product means more common features

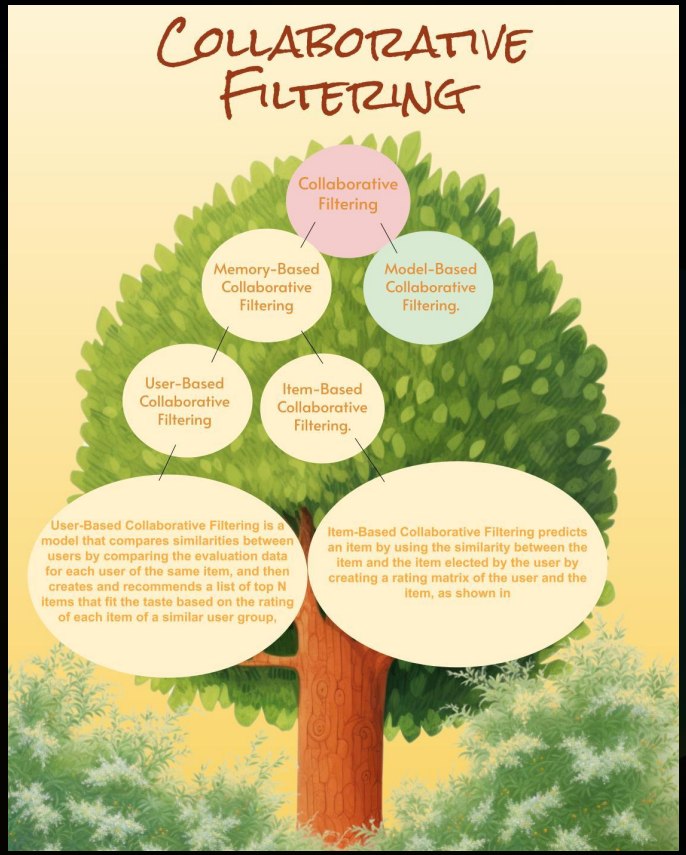


Dot Product

$$\langle x, y \rangle = \sum_{i=1}^{d} x_i y_i$$

https://developers.google.com/machine-learning/recommendation/content-based/basics

How do Recommendation Systems Work?

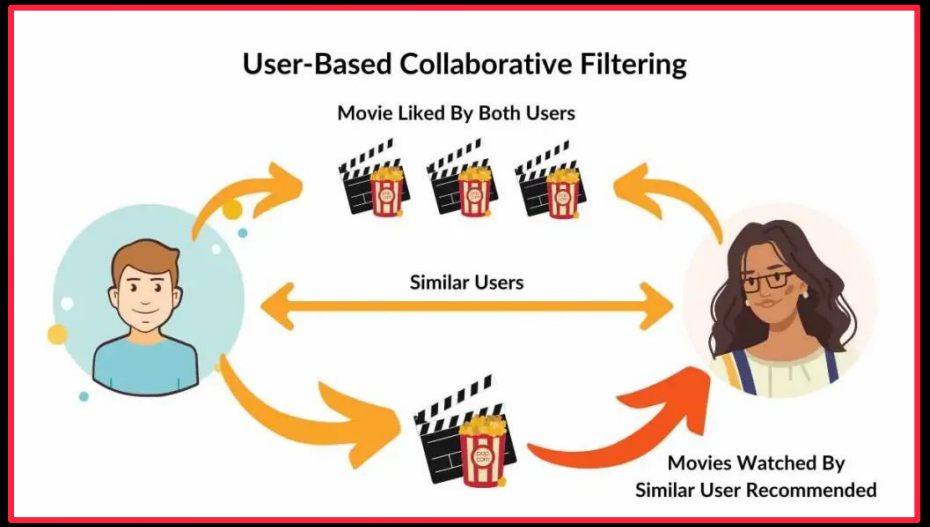


- User Based Filtering:
  - "People with similar interests also viewed…"
  - “Users with similar tastes also liked…”
- Item Based FIltering
  - "Customers who bought this item also bought…"
  - "Recommended based on your shopping history"

How do Recommendation Systems Work?

Collaborative-Based Filtering

- This approach attempts to improve upon the limitations of content-based filtering.
    - MLPs can model more complex, non-linear interactions
- It uses similarities between users and items simultaneously.
- It can recommend items to one user based on similar interests of another user.



Spotify, Youtube, Etsy, Reddit

How do Recommendation Systems Work?

Collaborative-Based Filtering

- Has implicit or explicit feedback
  - Explicit: Numerical rating
  - Implicit: If the user interacted with the item, they are interested in it
  - Deep learning can learn both types of feedback
- Embeddings can be learned automatically
  - Autoencoders and RNNs can create embeddings that capture more complex data

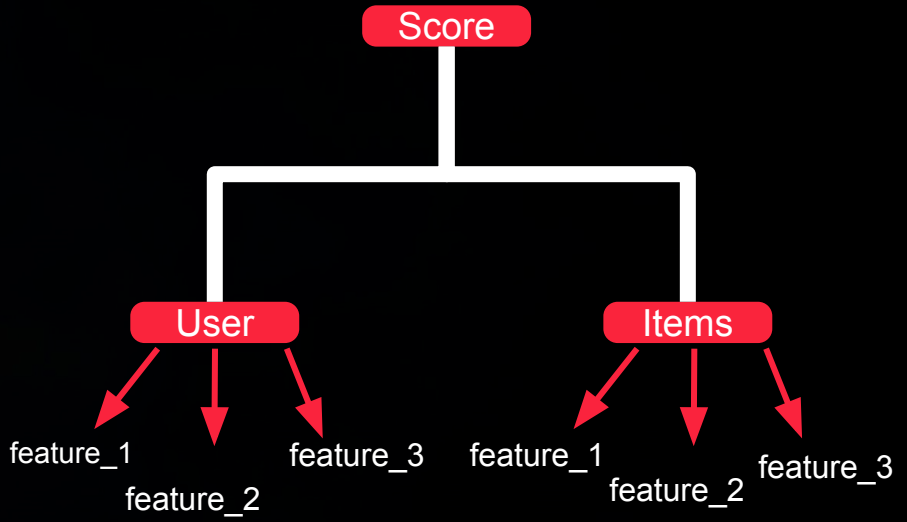
Weighted Sum

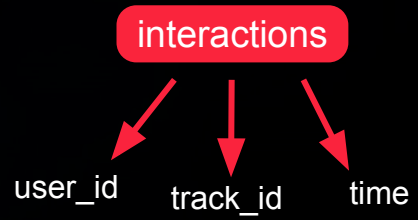
$$z = f(b + x \cdot w) = f\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

https://developers.google.com/machine-learning/recommendation/collaborative/basics

Two-Tower Model: Hybrid Filtering

- Content: User features and item features allows the model to make recommendations
  - User tower
  - Item tower
- Collaborative: Operates on user-item interactions
  - Interaction data -> determined by user ID and track ID
    - Implicit interactions
  - Dot product
- Allows for scalability and flexibility
  - Testing with a new dataset and with a bigger dataset
  - Deep Learning models can do this effectively to improve recommendation quality

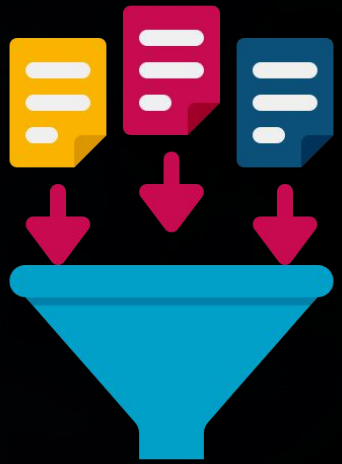


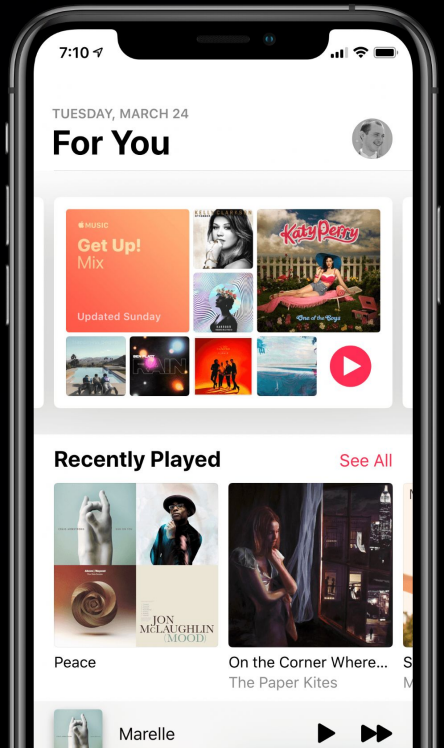


Data Preparation

Data Collection → User and Item Core Filtering

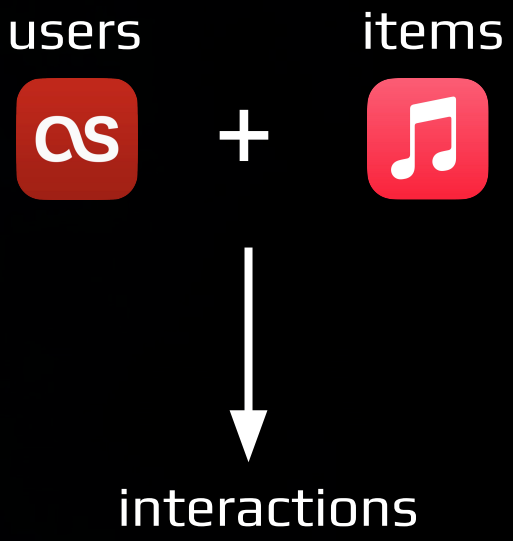
EDA → Data Reading





Data Collection

- Users.csv
  - Last fm api
    - Collect users, their country, and their loved track info
- Apple_items.csv
  - Add the song tag found from last fm to the songs in apple music dataset
- Apple_interactions.csv
  - Add every user's loved songs from last fm that match with the apple_items
  - Check both artist name and track name(apple music)
  - Username, track name, and time(last fm)



User and Item Core Filtering

- Improves the quality of the dataset by removing low interactions
- Needed to train the model with sufficient data
- Reduces noisy data
- Allows the model to make accurate predictions

```
# Create DataFrame from the extracted data
df_result = pd.DataFrame(df_lst, columns=['username', 'track_name', 'date_time'])

# Remove rows where track_name is "Unknown"
df_result = df_result[df_result['track_name'] != 'Unknown']

# Drop duplicates
df_result = df_result.drop_duplicates(subset=['username', 'track_name'], keep='last').reset_index(drop=True)

# Convert 'date_time' to Unix time (assuming 'date_time' is already in Unix format as string)
df_result['date_time'] = pd.to_numeric(df_result['date_time'], errors='coerce')

# Filter interactions based on the given criteria
index_count = df_result['track_name'].value_counts().to_dict()

while True:
    song_lst = [track for track, count in index_count.items() if count < 5]
    df_result = df_result[~df_result['track_name'].isin(song_lst)]

    user_interaction_count = df_result['username'].value_counts()
    low_interaction_users = user_interaction_count[user_interaction_count < 5].index.tolist()

    if not low_interaction_users and not song_lst:
        break

    df_result = df_result[~df_result['username'].isin(low_interaction_users)]

    updated_track_count = df_result['track_name'].value_counts()
    index_count = updated_track_count.to_dict()
```

User and Item Core Filtering

username	
rony_sp	50
juan_melo	46
everdeeen	46
AmeliePoulain7	44
	..
primalstatik	5
carlohilton	5
sadeyedprophet	5
cabeshpash	5
tedbassman	5

Track Name	
Skinny Love	246
There Is a Light That Never Goes Out	208
Pumped Up Kicks	195
Midnight City	187
Kids	178
	...
You should be sad	5
Rose Tattoo	5
Lonely No More	5
Be Here Now	5
Let It Die	5

Data Collection

users.csv + apple_items.csv = interactions

```
name,realname,country,loved_tracks
RJ,Richard Jones ,United Kingdom,"[{'track_name': 'Fire', 'artist_
franhale,Fran,United Kingdom,"[{'track_name': 'Mayhem', 'artist_nam
eartle,Michael Coffey,United Kingdom,"[{'track_name': 'Tester', 'ar
massdosage,Mass Dosage,United Kingdom,"[{'track_name': 'Rock Lobste
Knapster01,Matt Knapman,United Kingdom,"[{'track_name': 'Asimov', '
jonocole,Jono Cole,United Kingdom,"[{'track_name': 'Mix Tape', 'art
isaac,Isaac,United Kingdom,"[{'track_name': 'Do', 'artist_name': '[
lobsterclaw,Laura Weiss,United Kingdom,"[{'track_name': 'Where Is M
mremond,Mickaël,France,"[{'track_name': 'Le retour du roi Katché /
Orlenay,Orlena,United States,"[{'track_name': 'The Night Last Night
schlagschnitzel,,United Kingdom,"[{'track_name': 'The Mandalorian',
Edouard,Édouard,France,"[{'track_name': 'Mango', 'artist_name': 'Sa
naniel,Lyndsey,United Kingdom,"[{'track_name': 'Jiyuu no Tsubasa',
dunk,Duncan,United Kingdom,"[{'track_name': ""Blue Monday '88"", 'a
RUPERT,,United Kingdom,"[{'track_name': 'Together in Electric Dream
mxcl,Max Howell,United States,"[{'track_name': 'Buildings Began to
jwheare,James Wheare,United Kingdom,"[{'track_name': 'God Only Know
nancyvw,Nancy Walker,United Kingdom,"[{'track_name': 'Thank You for
underpangs,David Singleton,United Kingdom,"[{'track_name': 'Goooo',
p_wheel,Paul Wheeler,United Kingdom,"[{'track_name': 'Riot In My Ho
```

```
artistName,trackName,tags
Jason Mraz,I'm Yours,acoustic
Shakira,Hips Don't Lie (feat. Wyclef Jean),pop
P!nk,Just Give Me a Reason (feat. Nate Ruess),pop
Fun.,Some Nights,indie
Foster the People,Pumped Up Kicks,indie
Mark Ronson,Uptown Funk (feat. Bruno Mars),funk
AWOLNATION,Sail,electronic
Bruno Mars,When I Was Your Man,pop
Ed Sheeran,Thinking Out Loud,soul
One Direction,What Makes You Beautiful,pop
Christina Perri,A Thousand Years,pop
Kelly Clarkson,Stronger (What Doesn't Kill You),pop
Bruno Mars,Locked Out of Heaven,pop
Nickelback,Rockstar,rock
Hozier,Take Me to Church,indie
Wiz Khalifa,See You Again (feat. Charlie Puth),Hip-Hop
DJ Snake & Lil Jon,Turn Down for What,party
Iyaz,Replay,iyaz
Sara Bareilles,Love Song,pop
Leona Lewis,Bleeding Love,pop
Pitbull,Timber (feat. Ke$ha),pop
One Direction,Story of My Life,pop
twenty one pilots,Stressed Out,indie pop
```

```
username,track_name,date_time
RJ,Hypnotize,1412295881
RJ,I Kissed a Girl,1393609579
RJ,Toxic,1393608840
RJ,I Wish,1393601271
RJ,Butterfly,1382626421
RJ,I Only Have Eyes for You,1351342526
RJ,Skyfall,1349532860
RJ,Pumped Up Kicks,1342877126
franhale,Ignition (Remix),1377880399
franhale,You Got It,1375718297
franhale,Fake Empire,1365780785
franhale,The Chain,1364492171
franhale,Two Weeks,1360170473
franhale,Anyone Else But You,1359374752
franhale,Personal Jesus,1349361564
franhale,Edge of Seventeen,1349352332
franhale,You Spin Me Round (Like a Record),1349282115
franhale,Go Slow,1348585611
franhale,"Hey, Soul Sister",1338454080
franhale,Afterlife,1334830377
franhale,Dear God,1334761909
franhale,Nightmare,1334761905
franhale,Bittersweet Memories,1334658013
eartle,Harder Than You Think,1477045804
```

Exploratory Data Analysis

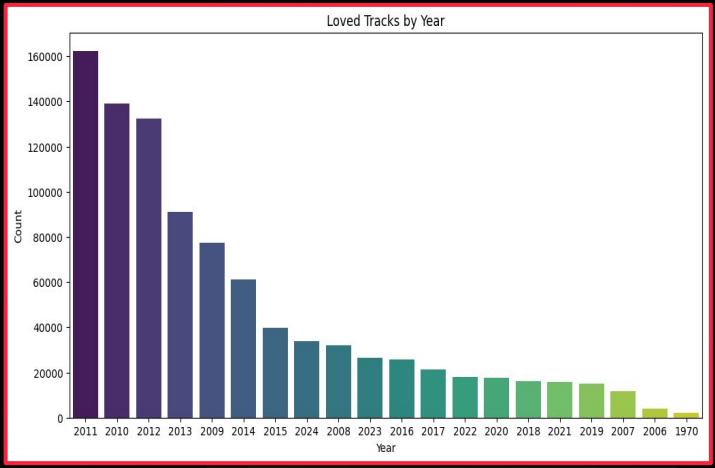
EDA - user.csv

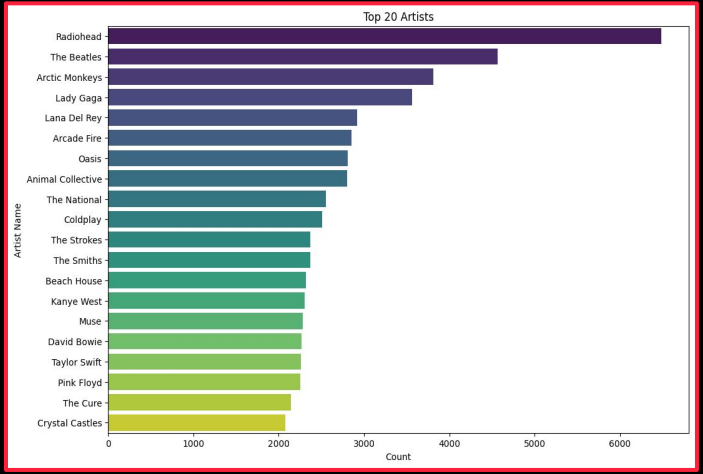
	name	realname	country	loved_tracks
0	RJ	Richard Jones	United Kingdom	[{'track_name': 'Fire', 'artist_name': 'The Bl...
1	franhale	Fran	United Kingdom	[{'track_name': 'Mayhem', 'artist_name': 'Imel...
2	eartle	Michael Coffey	United Kingdom	[{'track_name': 'Tester', 'artist_name': 'Hind...
3	massdosage	Mass Dosage	United Kingdom	[{'track_name': 'Rock Lobster', 'artist_name':...
4	Knapster01	Matt Knapman	United Kingdom	[{'track_name': 'Asimov', 'artist_name': '65da...

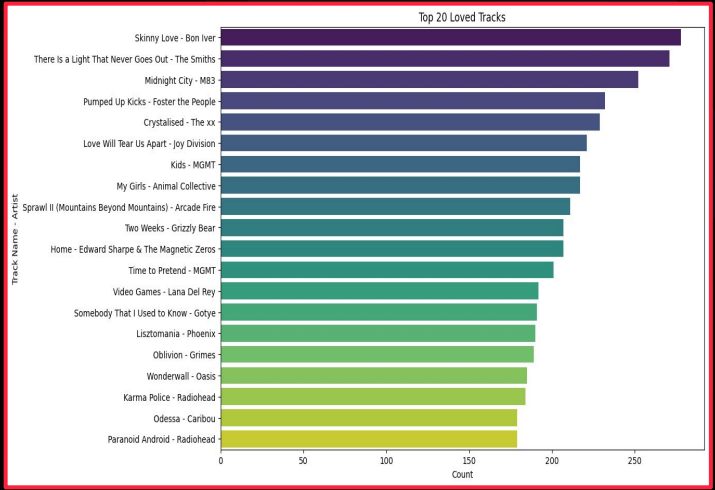
	username	realname	country	user_id	loved_song_name	loved_artist_name	year
0	RJ	Richard Jones	United Kingdom	1	Fire	The Black Seeds	2016
0	RJ	Richard Jones	United Kingdom	1	Just Jammin'	Gramatik	2016
0	RJ	Richard Jones	United Kingdom	1	Giddy Up	Tahuna Breaks	2015
0	RJ	Richard Jones	United Kingdom	1	The Nod	Fat Freddy's Drop	2015
0	RJ	Richard Jones	United Kingdom	1	Lady Day and John Coltrane	Gil Scott-Heron	2015
...	...	...	...	...	...	...	...
6828	LeoMetal965	Leonardo Marques	Brazil	6829	Cryin´	Aerosmith	2013
6828	LeoMetal965	Leonardo Marques	Brazil	6829	Agony	James LaBrie	2013
6828	LeoMetal965	Leonardo Marques	Brazil	6829	Maquina do tempo	Rosa de Saron	2013
6828	LeoMetal965	Leonardo Marques	Brazil	6829	Hanging by a Moment	Lifehouse	2013
6828	LeoMetal965	Leonardo Marques	Brazil	6829	Sacred & Wild	Powerwolf	2013

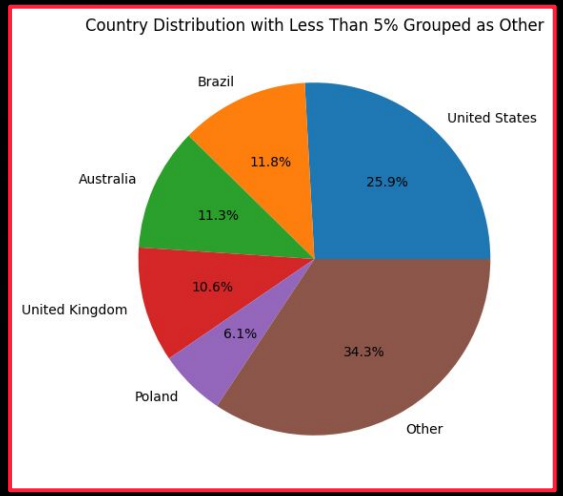
942671 rows × 7 columns

EDA - user.csv

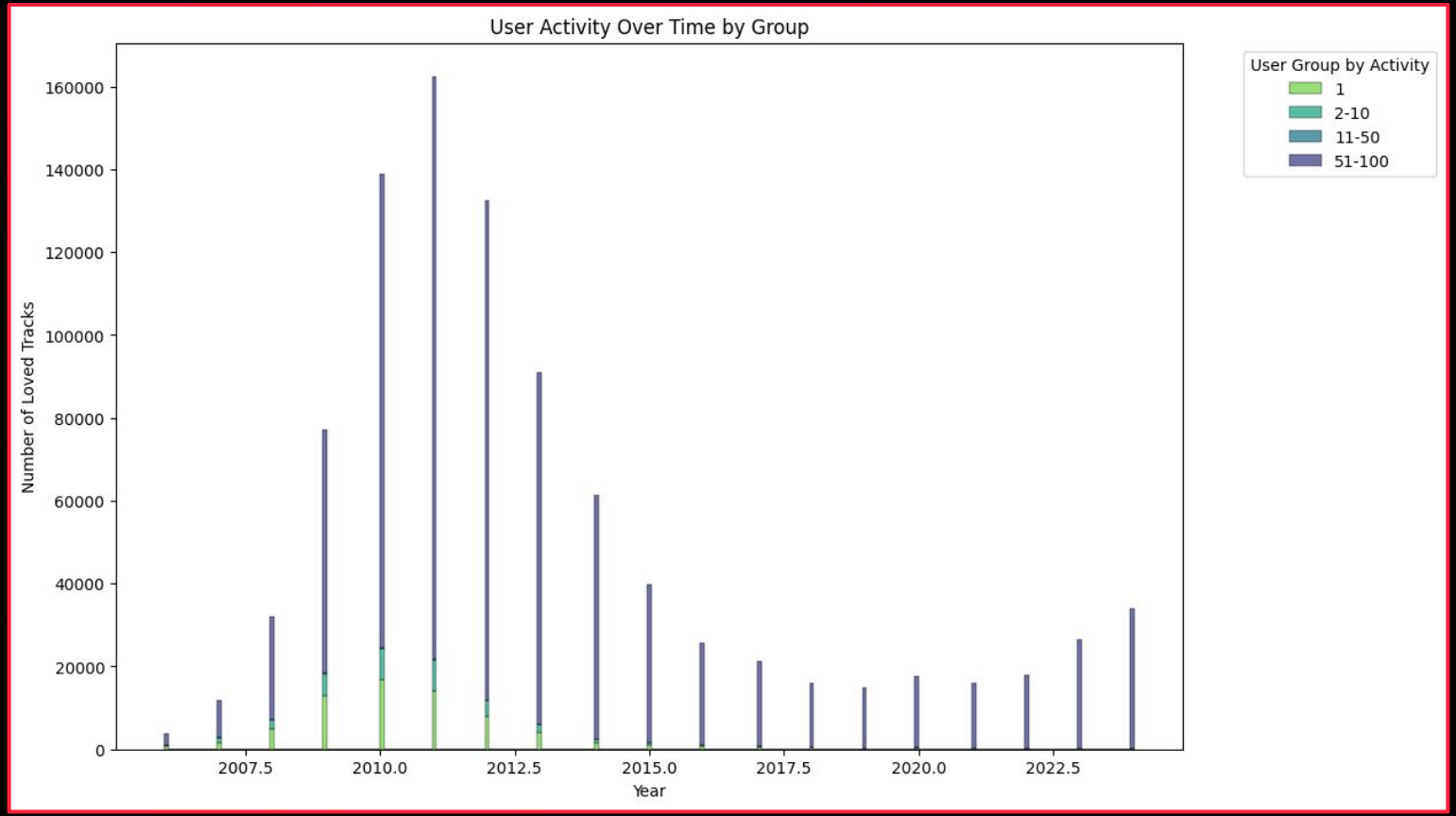

Loved Tracks by Year
Count
Year
160000
140000
120000
100000
80000
60000
40000
20000
0
2011
2010
2012
2013
2009
2014
2015
2024
2008
2023
2016
2017
2022
2020
2018
2021
2019
2007
2006
1970


Top 20 Artists
Artist Name
Count
Radiohead
The Beatles
Arctic Monkeys
Lady Gaga
Lana Del Rey
Arcade Fire
Oasis
Animal Collective
The National
Coldplay
The Strokes
The Smiths
Beach House
Kanye West
Muse
David Bowie
Taylor Swift
Pink Floyd
The Cure
Crystal Castles
0
1000
2000
3000
4000
5000
6000


Top 20 Loved Tracks
Track Name - Artist
Count
Skinny Love - Bon Iver
There Is a Light That Never Goes Out - The Smiths
Midnight City - M83
Pumped Up Kicks - Foster the People
Crystalised - The xx
Love Will Tear Us Apart - Joy Division
Kids - MGMT
My Girls - Animal Collective
Sprawl II (Mountains Beyond Mountains) - Arcade Fire
Two Weeks - Grizzly Bear
Home - Edward Sharpe & The Magnetic Zeros
Time to Pretend - MGMT
Video Games - Lana Del Rey
Somebody That I Used to Know - Gotye
Lisztomania - Phoenix
Oblivion - Grimes
Wonderwall - Oasis
Karma Police - Radiohead
Odessa - Caribou
Paranoid Android - Radiohead
0
50
100
150
200
250


Country Distribution with Less Than 5% Grouped as Other
Brazil
United States
Australia
United Kingdom
Poland
Other
11.8%
25.9%
11.3%
10.6%
6.1%
34.3%

EDA - user.csv

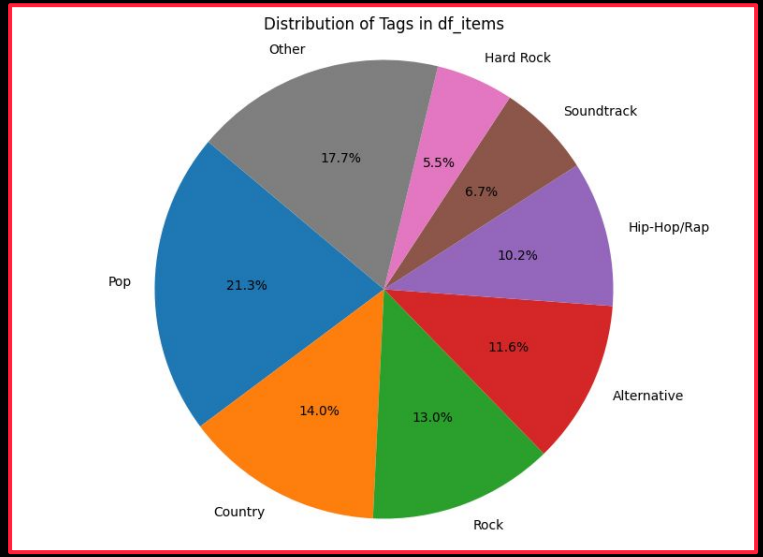

User Activity Over Time by Group
Number of Loved Tracks
Year
User Group by Activity
1
2-10
11-50
51-100
0
20000
40000
60000
80000
100000
120000
140000
160000
2007.5
2010.0
2012.5
2015.0
2017.5
2020.0
2022.5

EDA - apple_items.csv

```python
# Compare apple music dataset songs and every user's loved songs
matching_songs = pd.merge(expanded_tracks, df_filtered_items, on=['Artist', 'Track Name'], how='inner')

# Print only the matching songs (track name and artist name)
matching_songs[['Track Name', 'Artist']].value_counts()
```

✓ 0.4s Python

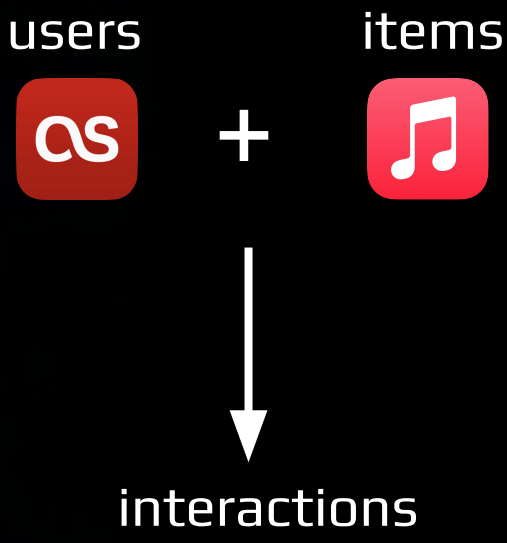
```
Track Name                          Artist
Skinny Love                         Bon Iver                  278
There Is a Light That Never Goes Out  The Smiths              271
Midnight City                       M83                       252
Pumped Up Kicks                     Foster the People         232
Kids                                MGMT                      217
                                                             ...
Heart Skips a Beat (feat. Chiddy Bang)  Olly Murs               1
This Means War                      Nickelback                1
Mad World                           Michael Andrews & Gary Jules  1
He Loves U Not                      Dream                     1
Open Wide (feat. Big Sean)          Calvin Harris             1
Name: count, Length: 3703, dtype: int64
```



EDA - apple_interactions.csv

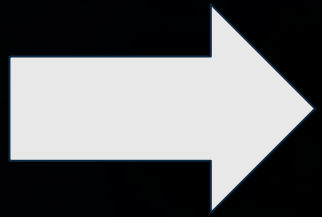
	username	Track Name	date_time
0	RJ	Hypnotize	1412295881
1	RJ	I Kissed a Girl	1393609579
2	RJ	Toxic	1393608840
3	RJ	I Wish	1393601271
4	RJ	Butterfly	1382626421
...	...	...	...
31036	LeoMetal965	Africa	1439132982
31037	LeoMetal965	Belief	1396883356
31038	LeoMetal965	People Are Strange	1384100431
31039	LeoMetal965	My Curse	1380844015
31040	LeoMetal965	Take My Breath Away	1380074155

31041 rows × 3 columns



Data Reading

- Users Data
  - User core filtering
    - The only valid users are users in interactions
    - Keep users with at least 5 interactions
    - Assign users IDs
- Item Data
  - Item core filtering
    - The only valid items are items in interactions
    - Keep items with at least 5 interactions
    - Assign tracks IDs
- Apple_interactions.csv
  - Drop duplicates
  - Map the track IDs and user IDs to their interactions



Dictionaries:
User_data
Item_data
sorted_reindexed_items

Data Reading

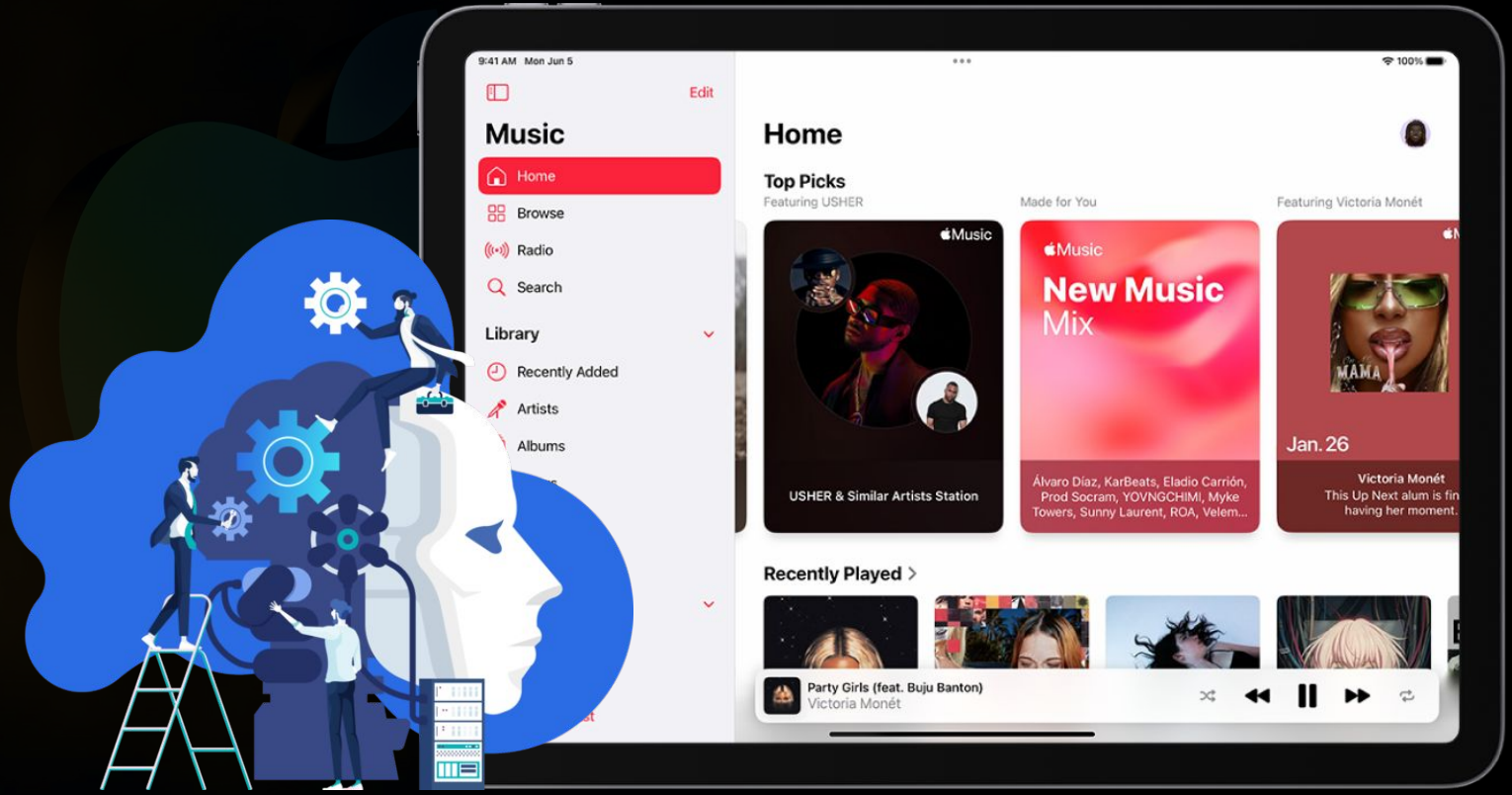
```
def readmusic(df_user, df_items, df_interactions):
```

```
    #-------- User data processing:
    df_user['country'] = df_user['country'].apply(lambda x: x.split(',')[-1].strip()) # to get country only
    df_user = df_user[df_user['country'].apply(is_valid_country)] # filter out wrong values
    df_user = df_user.drop_duplicates(subset=['id'], keep='last').reset_index(drop=True)

    # Ensure valid users are in df_user
    df_user = df_user[df_user['name'].isin(valid_usernames)]

    # Drop interactions whose username is not in df_user
    df_interactions = df_interactions[df_interactions['username'].isin(df_user['name'])]

    # Ensure valid track IDs are in df_items
    df_interactions = df_interactions[df_interactions['track_id'].isin(df_items['id'])]


    # Convert user data to categorical
    username = df_user['name'].to_numpy().astype(str).tolist()
    userid = df_user['id'].to_numpy().astype(str).tolist()
    countries = df_user['country'].to_numpy().astype(str).tolist()

    username_reindexer = convert_to_categorical(username)
    userid_reindexer = convert_to_categorical(userid)
    countries_reindexer = convert_to_categorical(countries)

    user_data = {}
    for i in range(len(username)):
        reindexed_username = username_reindexer[username[i]]
        reindexed_userid = userid_reindexer[userid[i]]
        reindexed_country = countries_reindexer[countries[i]]
        user_data[reindexed_userid] = {
            'username': reindexed_username,
            'country': reindexed_country
        }
```

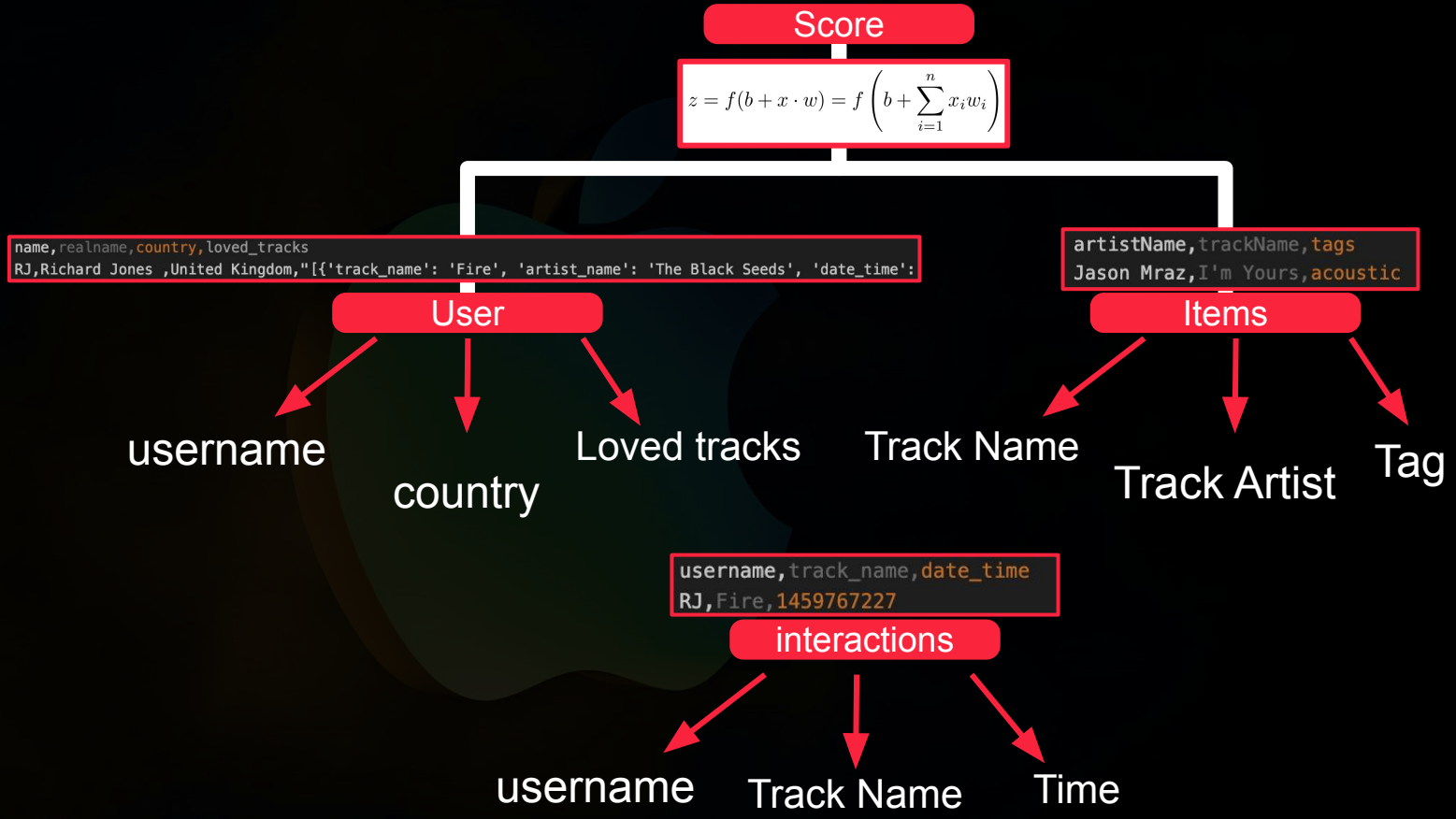
```
    #-------- Item data processing:
    track_ids = df_items['id'].to_numpy().astype(str).tolist()
    artist_list = df_items['artistName'].to_numpy().astype(str).tolist()
    tags_list = df_items['tags'].to_numpy().astype(str).tolist()
    track_list = df_items['trackName'].to_numpy().astype(str).tolist()

    track_ids_reindexer = convert_to_categorical(track_ids, offset=len(userid_reindexer))
    tags_reindexer = convert_to_categorical(tags_list)
    artist_reindexer = convert_to_categorical(artist_list)

    # Embed track names using SentenceTransformer
    model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')
    title_embeddings = model.encode(track_list)

    item_data = {}
    for i in range(len(track_list)):
        reindexed_track_id = track_ids_reindexer[track_ids[i]]
        reindexed_tags = tags_reindexer[tags_list[i]]
        reindexed_artist = artist_reindexer[artist_list[i]]
        embedding = title_embeddings[i]
        item_data[reindexed_track_id] = {
            'Track Name': embedding,
            'Artist': reindexed_artist,
            'Tags': reindexed_tags
        }
```

```
#-------- Interaction data processing:
reindexed_ratings = defaultdict(list)
for interaction in df_interactions.itertuples():
    try:
        user_id = userid_reindexer[str(interaction.user_id)]
        item_id = track_ids_reindexer[str(interaction.track_id)]
        date_time = interaction.date_time
        reindexed_ratings[user_id].append((item_id, date_time))
    except KeyError as e:
        print(f"KeyError: {e} - User ID: {interaction.user_id}, Track ID: {interaction.track_id}")

# Sort Reindexed Ratings by key to ensure reproducibility
sorted_reindexed_interactions_by_key = {k: reindexed_ratings[k] for k in sorted(reindexed_ratings)}
```

Model


9:41 AM Mon Jun 5
100%
Edit
Music
Home
Browse
Radio
Search
Library
Recently Added
Artists
Albums
Home
Top Picks
Featuring USHER
Made for You
Featuring Victoria Monét
Music
USHER & Similar Artists Station
Music
New Music
Mix
Álvaro Díaz, KarBeats, Eladio Carrión, Prod Socram, YOVNGCHIMI, Myke Towers, Sunny Laurent, ROA, Velem...
MAMA
Jan. 26
Victoria Monét
This Up Next alum is fin
having her moment.
Recently Played
Party Girls (feat. Buju Banton)
Victoria Monét

Two-Tower Model: Collaborative-Filtering



Two-Tower Model - Structs

- __init__: initializes user, item, interactions, and warm threshold
- create_data_split: training, validation, and test
  - Edges
  - Select one item in the tuple of interactions to be used for testing and another for validation
  - warm _threshold determines the proportion of data to be identified as cold items

```python
class MusicInteractionGraph(InteractionGraph):
    def __init__(self, user_data, item_data, interaction_data, warm_threshold=0.2) -> None:
        super().__init__(user_data, item_data, interaction_data)
        self.create_data_split()
        self.create_bipartite_graph()
        assert (warm_threshold < 1.0 and warm_threshold > 0.0)
        self.warm_threshold = warm_threshold
        self.compute_tail_distribution()

    def create_data_split(self):
            # Leave one out validation
            self.all_edges = set()
            self.train_edges_set = set()
            for user_id in tqdm(self.interaction_data):
                    sorted_interactions = sorted(self.interaction_data[user_id], key=lambda x : x[1])
                    test_edge = [user_id, sorted_interactions[-1][0]]
                    val_edge = [user_id, sorted_interactions[-2][0]]
                    self.all_edges.add((user_id, sorted_interactions[-2][0]))


                    train_edges = [[user_id, interaction[0]] for interaction in sorted_interactions[:-2]]
                    for interaction in sorted_interactions[:-2]:
                        self.all_edges.add((user_id, interaction[0]))
                        self.train_edges_set.add((user_id, interaction[0]))

                    self.train_edges += train_edges
                    self.validation_edges.append(val_edge)
                    self.test_edges.append(test_edge)

            self.train_edges = np.array(self.train_edges)
            self.validation_edges = np.array(self.validation_edges)
            self.test_edges = np.array(self.test_edges)
```

Two-Tower Model - Structs

- create_bipartite_graph: creates the node matrix of size (user x item)
  - Will be filled with boolean (0 or 1)
  - Use CSR (Compress Sparse Row) to save the entry that has 1s
- compute_tail_distribution: find the cold items
  - Calculate the interactions per user and item
  - Sort and store the bottom 80% of items
  - Adjust indices

```python
def create_bipartite_graph(self):
    num_nodes = len(self.user_data) + len(self.item_data) # Num users + num items = size of matrix
    self.adj_matrix = scipy.sparse.dok_matrix((num_nodes, num_nodes), dtype=bool)   # initialize as empty, will have boolean values of 0 or 1

    for edge in self.train_edges:
        self.adj_matrix[edge[0], edge[1]] = 1   # Edge of user to item
        self.adj_matrix[edge[1], edge[0]] = 1   # Edge of item to user

    self.adj_matrix = self.adj_matrix.tocsr()   # .toscr(), compress the sparse row and save the non zero elements

def compute_tail_distribution(self, warm_threshold):    # This function is used to store cold items
    self.is_cold = np.zeros((self.adj_matrix.shape[0]), dtype=bool)
    self.start_item_id = len(self.user_data)

    self.user_degrees = np.array(self.adj_matrix[:self.start_item_id].sum(axis=1)).flatten()
    self.item_degrees = np.array(self.adj_matrix[self.start_item_id:].sum(axis=1)).flatten()

    cold_items = np.argsort(self.item_degrees)[:int((1 - warm_threshold) * len(self.item_degrees))] + self.start_item_id
    self.is_cold[cold_items] = True
```

Two-Tower Model - Structs

Split Statistics

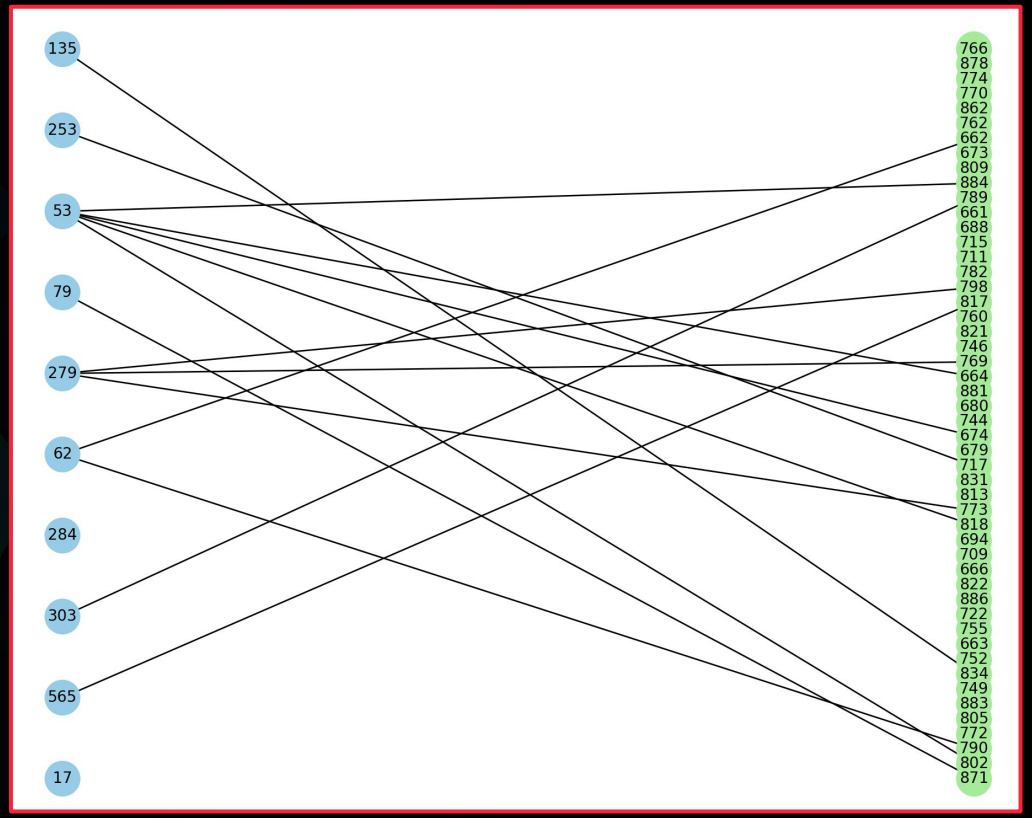
```
Train edges:
 [[  0 707]
 [  0 825]
 [  0 687]
 [  0 832]
 [  0 871]
 [  0 683]
 [  0 788]
 [  0 699]
 [  0 751]
 [  1 868]
 [  1 827]
 [  1 830]
 [  2 721]
 [  2 846]
 [  2 795]]
```

```
Validation edges:
 [[  0 663]
 [  1 690]
 [  2 831]
 [  3 863]
 [  4 805]]
```

```
Test edges:
 [[  0 801]
 [  1 685]
 [  2 834]
 [  3 840]
 [  4 696]]
```

```
Adjacency matrix:
 <Compressed Sparse Row sparse matrix of dtype 'bool'
        with 27 stored elements and shape (5, 889)>
  Coords        Values
  (0, 707)      True
  (0, 825)      True
  (0, 687)      True
  (0, 832)      True
  (0, 871)      True
  (0, 683)      True
  (0, 788)      True
  (0, 699)      True
  (0, 751)      True
  (1, 868)      True
  (1, 827)      True
  (1, 830)      True
  (2, 721)      True
  (2, 846)      True
  (2, 795)      True
  (2, 828)      True
  (2, 759)      True
```

Bipartite graph



Two-Tower Model - Dataloader.py

- Overview
  - Purpose is to handle the loading and preparation of data for training and testing the recommendation model.
  - Ensures data consistency and prepares data for efficient processing.
- 1. class TrackDataset:
  - Manages user and item data through the TrackDataset class, ensuring they are organized for easy access and manipulation (self.user_data, self.item_data).
  - Divides data into training, validation, and test sets based on the mode ('train', 'val', 'test'), crucial for evaluating model performance across different scenarios (self.edges).

```python
class TrackDataset(Dataset):
    def __init__(self, igraph : MusicInteractionGraph, mode='train') -> None:
        self.igraph = igraph
        self.user_data = igraph.user_data
        self.item_data = igraph.item_data
        self.adj_matrix = igraph.adj_matrix
        if mode == 'train':
            self.edges = igraph.train_edges
        elif mode == 'val':
            self.edges = igraph.validation_edges
        else:
            self.edges = igraph.test_edges
    def __len__(self):
        return self.edges.shape[0]
    def __getitem__(self, index):
        return self.edges[index]
```

Two-Tower Model - Dataloader.py

- class TrackCollator:
  - Generates negative samples (generate_in_and_oob_ negatives) to train the model, teaching it to recognize which items are not relevant to users.
  - Prepares batches of data, including user details and item characteristics, which are essential for training and testing the model's accuracy in making recommendations (__call__ method).

```python
class TrackCollator:
    def __init__(self, igraph: MusicInteractionGraph, mode: str, num_neg_samples=1) -> None:
        self.igraph = igraph
        self.user_data = igraph.user_data
        self.item_data = igraph.item_data
        self.adj_matrix = igraph.adj_matrix
        self.num_neg_samples = num_neg_samples
        self.mode = mode
        self.rng = np.random.Generator(np.random.PCG64(seed=0))
```

```python
def _generate_in_and_oob_negatives(self, positive_edges):
    item_start_id = len(self.user_data)
    pos_edges = np.array(positive_edges)

    negative_edges = []
    for i, (user_id, _) in enumerate(positive_edges):
        # Out of batch negative Sampling
        candidate_item_probs = np.asarray(self.adj_matrix[user_id, item_start_id:].todense()).flatten() == 0
        candidate_item_probs = candidate_item_probs / candidate_item_probs.sum()

        valid_samples = False
        while not valid_samples:
            neg_items = np.random.choice(candidate_item_probs.shape[0], (self.num_neg_samples,), p=candidate_item_probs)
            for neg_item in neg_items:
                if (user_id, neg_item + item_start_id) in self.igraph.all_edges:
                    valid_samples = False
                    break
                valid_samples = True
        # In batch negative sampling
```

```python
    def __call__(self, positive_edges):
        if self.mode != 'train':
            true_edges = np.array(positive_edges)
            edges_to_score, offsets = self._get_edges_to_score(true_edges)
            return torch.as_tensor(true_edges, dtype=torch.int32), torch.as_tensor(edges_to_score, dtype=torch.int32), torch.as_tensor(offsets, dtype=torch.int32)

        negative_edges = self._generate_in_and_oob_negatives(positive_edges)
        edges = positive_edges + negative_edges

        user_feats, item_feats, item_title_embeddings = self._fetch_data(edges)

        user_feats_by_type = list(zip(*user_feats))
        user_ids = torch.as_tensor(user_feats_by_type[0], dtype=torch.int64)
        user_country = torch.as_tensor(user_feats_by_type[1], dtype=torch.int64)
        user_name = torch.as_tensor(user_feats_by_type[2], dtype=torch.int64)
```

Two-Tower Model - Dataloader.py

- Track Inference Datasets and Collators
- class TrackInferenceItemsDataset:
  - Initializes with all item IDs sorted and creates an index mapping (self.item_reindexer) for efficient data retrieval.
- class TrackItemsCollator
  - Extracts and batches item features, including Artist and Tags, and computes embeddings for track names using track_name_embeddings.
  - Aggregate and batch features from items and users, making them ready for model inference (__call__ method).

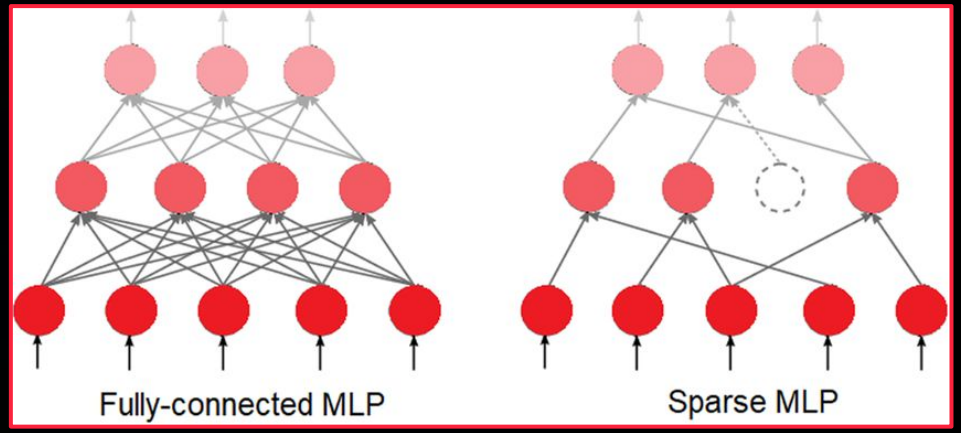
```python
class TrackInferenceItemsDataset(Dataset):
    def __init__(self, igraph : MusicInteractionGraph) -> None:
        self.all_item_ids = sorted(list(igraph.item_data.keys()))
        self.item_reindexer = {}
        for item_id in self.all_item_ids:
            self.item_reindexer[item_id] = len(self.item_reindexer)
        self.reverse_item_indexer = {v : k for k, v in self.item_reindexer.items()}
    def __len__(self):
        return len(self.all_item_ids)
    def __getitem__(self, index):
        return self.all_item_ids[index]
```

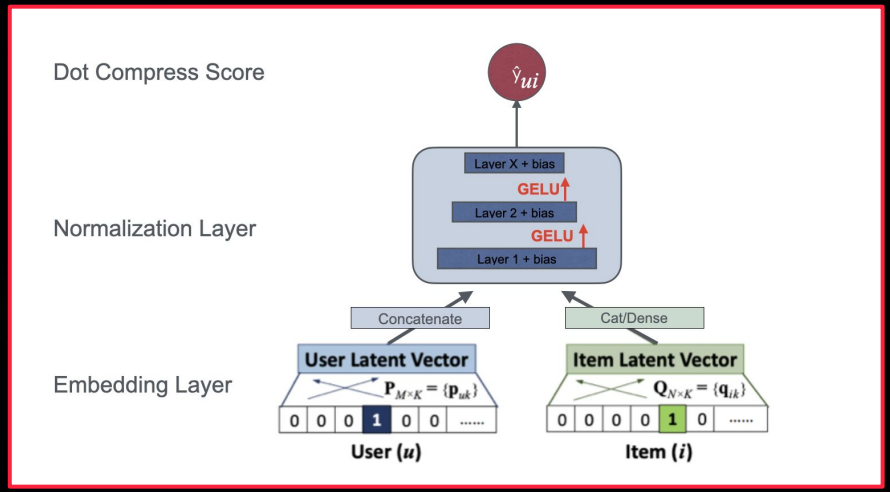
This process is similarly applied to users.

```python
class TrackItemsCollator:
    def __init__(self, igraph : MusicInteractionGraph) -> None:
        self.igraph = igraph
        self.item_data = igraph.item_data

    def __call__(self, batch):
        item_feats = [(track_id, self.item_data[track_id]['Artist'], self.item_data[track_id]['Tags']) for track_id in batch]
        item_title_embeddings = [self.item_data[track_id]['Track Name'] for track_id in batch]
        item_feats_by_type = list(zip(*item_feats))
        track_ids = torch.as_tensor(item_feats_by_type[0], dtype=torch.int64)
        zero_indexed_track_ids = track_ids - len(self.igraph.user_data)
        track_artists = torch.as_tensor(item_feats_by_type[1], dtype=torch.int64)
        tag_embeddings = torch.as_tensor(item_feats_by_type[2],  dtype=torch.int64)
        track_name_embeddings = torch.as_tensor(np.asarray(item_title_embeddings), dtype=torch.float32)
        return (track_ids, track_artists, tag_embeddings, track_name_embeddings, zero_indexed_track_ids)
```

Two-Tower Model - Sparse Neural Network

- MLP (Multi-Layer Perceptron): The sparse MLP will be used because our data consists of a lot of zeros
- SparseNN:
  - Sparse User: Embed the user features and concatenate them as a single representation
  - Sparse Item: Embed the item features as a single representation
- Dot Compress Scoring: uses the dot product on the user and item and compresses the combined representation to pass it to the MLP





Two-Tower Model - Sparse Neural Network

```python
class TrackSparseNNUserModel(nn.Module):    # embed the user items
    def __init__(self,
        num_user_ids,
        num_user_countries,
        num_user_names,
        feat_embed_dim=64,
        output_embed_dim=128,
        combine_op='cat',
    ):
        super(TrackSparseNNUserModel, self).__init__()
        self.id_embeddings = nn.Embedding(num_user_ids, embedding_dim=feat_embed_dim)   # embed the user ids
        self.countries_embeddings = nn.Embedding(num_user_countries, embedding_dim=feat_embed_dim)  # embed the countries
        self.user_name_embeddings = nn.Embedding(num_user_names, embedding_dim=feat_embed_dim)  # embed the usernames

        self.output_embed_dim = output_embed_dim
        self.combine_op = combine_op
        self.act = nn.GELU()

        self.output_mlp = self._create_output_mlp(3*feat_embed_dim if combine_op == 'cat' else feat_embed_dim, output_embed_dim)

    def _create_output_mlp(self, first_layer_dim, output_embed_dim):
        return nn.Sequential(nn.Linear(first_layer_dim, 128),
                             nn.LayerNorm(128, elementwise_affine=False),
                             self.act,
                             nn.Linear(128, 64),
                             nn.LayerNorm(64, elementwise_affine=False),
                             self.act,
                             nn.Linear(64, output_embed_dim))


    def forward(self, user_ids, user_countries, user_names):
        id_embeddings = self.id_embeddings(user_ids)
        countries_embeddings = self.countries_embeddings(user_countries)
        user_name_embeddings = self.user_name_embeddings(user_names)

        combined_rep = torch.cat([id_embeddings, countries_embeddings, user_name_embeddings], dim=1) if self.combine_op == 'cat' else \
            id_embeddings + countries_embeddings + user_name_embeddings

        return self.act(self.output_mlp(combined_rep))
```

```python
class TrackSparseNNItemModel(nn.Module):    # embed the item features
    def __init__(self,
        num_track_ids,
        num_track_artists,
        num_track_tags,
        num_track_names,
        feat_embed_dim=96,
        dense_feat_input_dim=384,
        output_embed_dim=192,
        combine_op='cat'
    ):
        super(TrackSparseNNItemModel, self).__init__()
        self.track_id_embeddings = nn.Embedding(num_track_ids, feat_embed_dim)  # embed track ids
        self.artists_embeddings = nn.Embedding(num_track_artists, feat_embed_dim)   # embed artists
        self.tags_embeddings = nn.Embedding(num_track_tags, feat_embed_dim) # embed tags
        self.dense_transform = nn.Linear(dense_feat_input_dim, feat_embed_dim) # turn the higher dimension track names into a lower dimension

        self.output_embed_dim = output_embed_dim
        self.combine_op = combine_op
        self.act = nn.GELU()

        self.output_mlp = self._create_output_mlp(4*feat_embed_dim if combine_op == 'cat' else feat_embed_dim, output_embed_dim)

    def _create_output_mlp(self, first_layer_dim, output_embed_dim):
        return nn.Sequential(nn.Linear(first_layer_dim, 128),
                             nn.LayerNorm(128, elementwise_affine=False),
                             self.act,
                             nn.Linear(128, 64),
                             nn.LayerNorm(64, elementwise_affine=False),
                             self.act,
                             nn.Linear(64, output_embed_dim))

    def forward(self, track_ids, track_artists, track_tags, track_names):
        track_id_embeddings = self.track_id_embeddings(track_ids)
        artists_embeddings = self.artists_embeddings(track_artists)
        tags_embeddings = self.tags_embeddings(track_tags)
        dense_embeddings = self.act(self.dense_transform(track_names))

        combined_rep = torch.cat([track_id_embeddings, artists_embeddings, tags_embeddings, dense_embeddings], dim=1) if self.combine_op == 'cat' else \
            track_id_embeddings + artists_embeddings + tags_embeddings + dense_embeddings

        return self.act(self.output_mlp(combined_rep))
```

Two-Tower Model - Sparse Neural Network

```python
class MLP(nn.Module):   # multi layer perceptron, takes input features. Each node represents one feature
    def __init__(self, dims: List[int], add_bias=True, act="gelu", apply_layernorm=False, elemwise_affine=False):
        super().__init__()
        self._activation = self._get_activation(act)    # apply activation function
        self._apply_layernorm = apply_layernorm # normalize the layers
        self._elemwise_affine = elemwise_affine # boolean to determine the normalization of each layer
        self._add_bias = add_bias   # boolean to add bias to each layer
        self._model = self._create_model(dims)

    def _create_model(self, dims):
        layers = nn.ModuleList()    # to store the layers of the model
        for i in range(1, len(dims)):
            layer = nn.Linear(dims[i-1], dims[i]) if self._add_bias else nn.Linear(dims[i-1], dims[i], bias=False)
            layers.append(layer)

            if i < len(dims) - 1:
                if self._apply_layernorm:
                    layers.append(nn.LayerNorm(dims[i], elementwise_affine=self._elemwise_affine))

                layers.append(self._activation)

        return nn.Sequential(*layers)
```

```python
class DotCompressScoringModel(nn.Module):
    def __init__(self, input_dim: int, hidden_dims: List[int], act='gelu'):
        super(DotCompressScoringModel, self).__init__()
        self.dot_compress_weight = nn.Parameter(torch.empty(2, input_dim // 2))
        nn.init.xavier_normal_(self.dot_compress_weight)

        self.dot_compress_bias = nn.Parameter(torch.zeros(input_dim // 2))

        self.dims = [input_dim] + hidden_dims + [1]
        self.output_layer = MLP(self.dims, apply_layernorm=True, elemwise_affine=True)

    def forward(self, set_embeddings, item_embeddings):
        all_embeddings = torch.stack([set_embeddings, item_embeddings], dim=1)  # stack the user and item embeddings as a single vector of 1D
        combined_representation = torch.matmul(all_embeddings, torch.matmul(all_embeddings.transpose(1, 2), self.dot_compress_weight) + self.dot_compress_bias).flatten(1)
        output = self.output_layer(combined_representation) # pass the output to the MLP
        return output
```

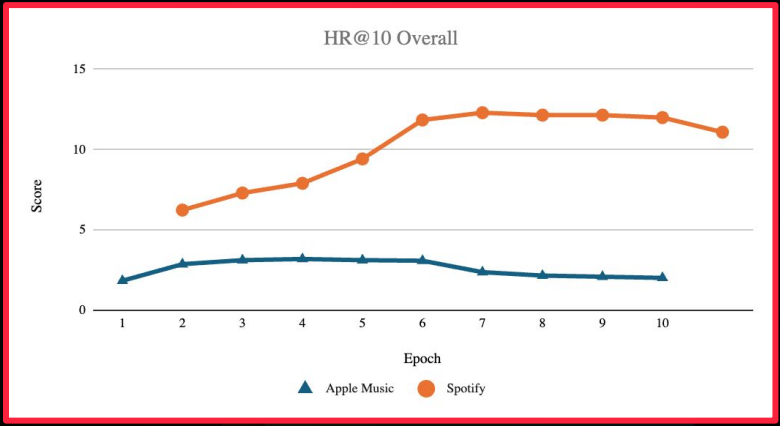
Two-Tower Model - Main

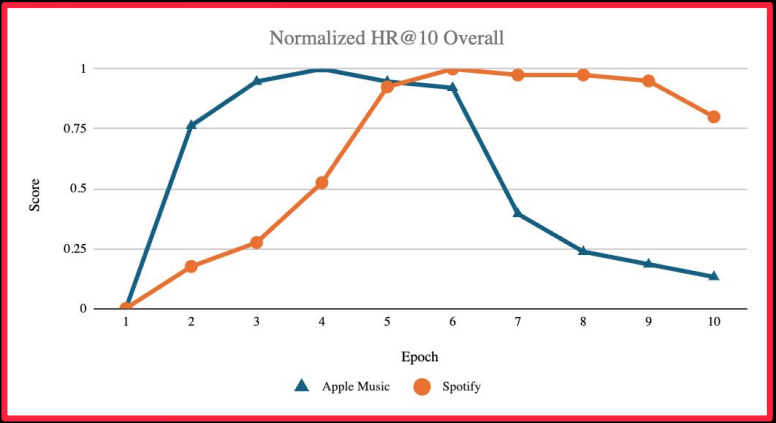
- Sent the batch data to device
- Trained and tested the model
- Called the readmusic and dataloader functions
- Called sparseNN
- Calculated the scores of the recommendations

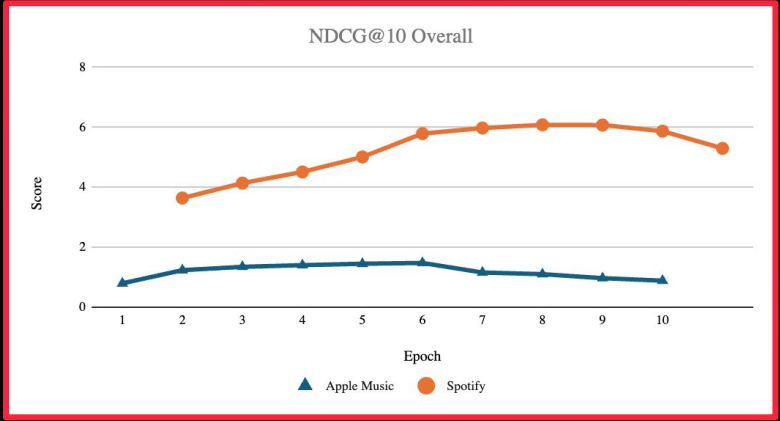
- Top-k: the metric for the top 10 (k) items
- HR@10 Overall: the proportion of the true item being found in the interactions
  - Cold: Hit rate for cold items
  - Warm: hit rate for warm items
- NDCG: the ranking considering the position of the item

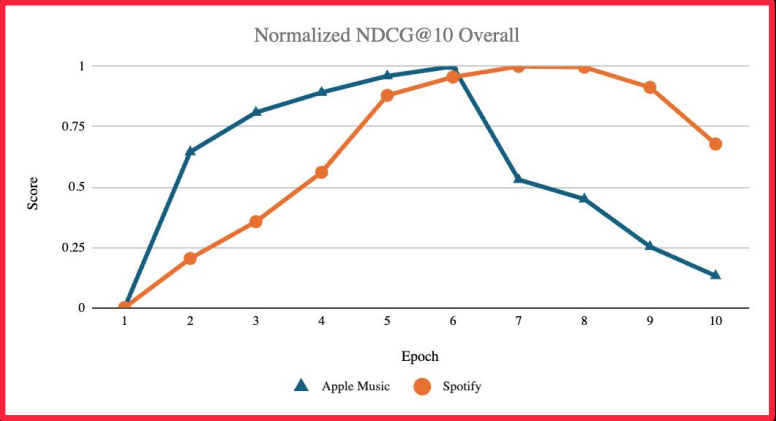
```
print(f"HR@{k} Overall = {hitrate_k_combined}, NDCG@{k} Overall = {ndcg_k_combined}")
print(f"HR@{k} Cold = {hitrate_k_cold}, NDCG@{k} Cold = {ndcg_k_cold}")
print(f"HR@{k} Warm = {hitrate_k_warm}, NDCG@{k} Warm = {ndcg_k_warm}")
```

Two-Tower Model (general) - Results









Two-Tower Model (general) - Results for Apple Music

```
Total number of items = 1519
Number of items present across training edges = 1519
Number of items present across val edges = 1038
Number of items present across test edges = 1052
Average item degree = 16.60434496379197
Average user degree = 8.985393658710366
Number of items common between train and validation edges = 1038
Number of items common between train and test edges = 1052
Number of cold items in validation set = 742
Number of cold items in test set = 770
```

Best Epoch:
4

```
  0%|          | 0/3 [00:00<?, ?it/s]
 33%|███       | 1/3 [00:05<00:11,  5.94s/it]
 67%|██████    | 2/3 [00:09<00:04,  4.30s/it]
100%|█████████| 3/3 [00:11<00:00,  3.40s/it]
100%|█████████| 3/3 [00:11<00:00,  3.83s/it]
HR@10 Overall = 3.1706448165301033, NDCG@10 Overall = 1.3929433215401565
HR@10 Cold = 0.4961020552799433, NDCG@10 Cold = 0.2721466449169873
HR@10 Warm = 5.873925501432665, NDCG@10 Warm = 2.5257829423963827
Best HR so far = 3.1706448165301033, Best NDCG so far 1.3929433215401565
Corresponding (Warm, Cold) Hit Rate = (5.873925501432665, 0.4961020552799433), Corresponding
(Warm. Cold) NDCG = (2.5257829423963827,0.2721466449169873)
```

Two-Tower Model (general) - Results for Spotify

```
Total number of items = 230
Number of items present across training edges = 230
Number of items present across val edges = 195
Number of items present across test edges = 194
Average item degree = 15.521739130434783
Average user degree = 5.417298937784522
Number of items common between train and validation edges = 195
Number of items common between train and test edges = 194
Number of cold items in validation set = 150
Number of cold items in test set = 148
```

Best Epoch:
6

```
HR@10 Overall = 12.291350531107739, NDCG@10 Overall = 5.96867473689393
HR@10 Cold = 0.0, NDCG@10 Cold = 0.0
HR@10 Warm = 27.0, NDCG@10 Warm = 13.111188838710332
Best HR so far = 12.291350531107739, Best NDCG so far 5.96867473689393
Corresponding (Warm, Cold) Hit Rate = (27.0, 0.0), Corresponding (Warm. Cold) NDCG = (13.111188838710332,0.0)
```

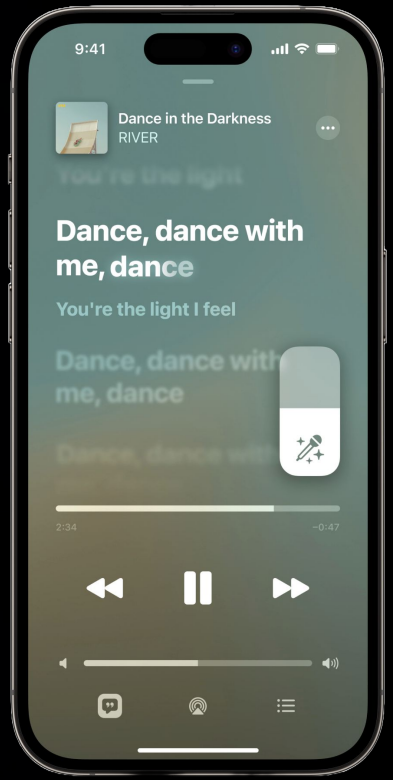
Enhancing Our Two Tower Model

Introduction to Enhancement:

- Our Objective is to enhance our model's accuracy by refining the feature set.
- Our model currently uses artist names and song titles for recommendations.

Proposed Enhancement:

- Replace artist names and song titles with lyric summaries using Groq (an advanced AI model) that runs LLM's .
- Lyrics provide deeper insights into song content, offering more concrete data for comparing and recommending songs, thus enhancing context and relevance.



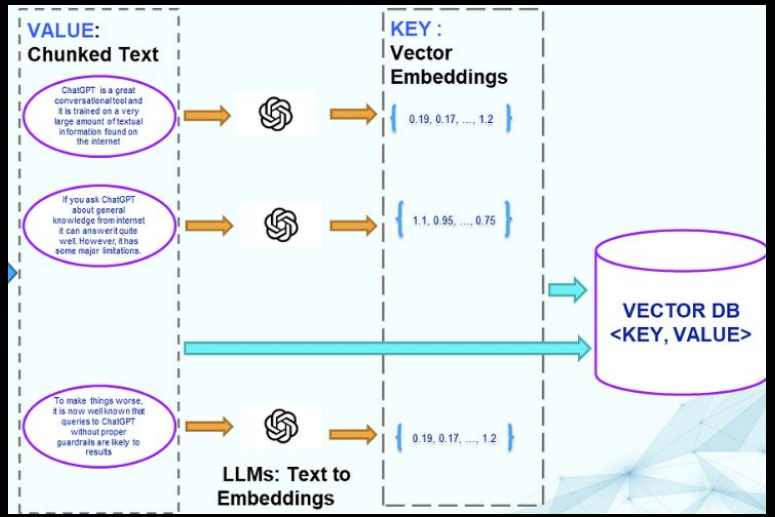
What is an LLM and how does it work

What is an LLM?

- LLM is a large Language model designed to interpret, understand and generates human language.
- It starts with engineers collecting a large amount of data (in the billions to trillions of data points) and then using all this data to train the LLM (expose it to the data) so it can start looking at patterns to eventually predict, look at, and respond to regular human language

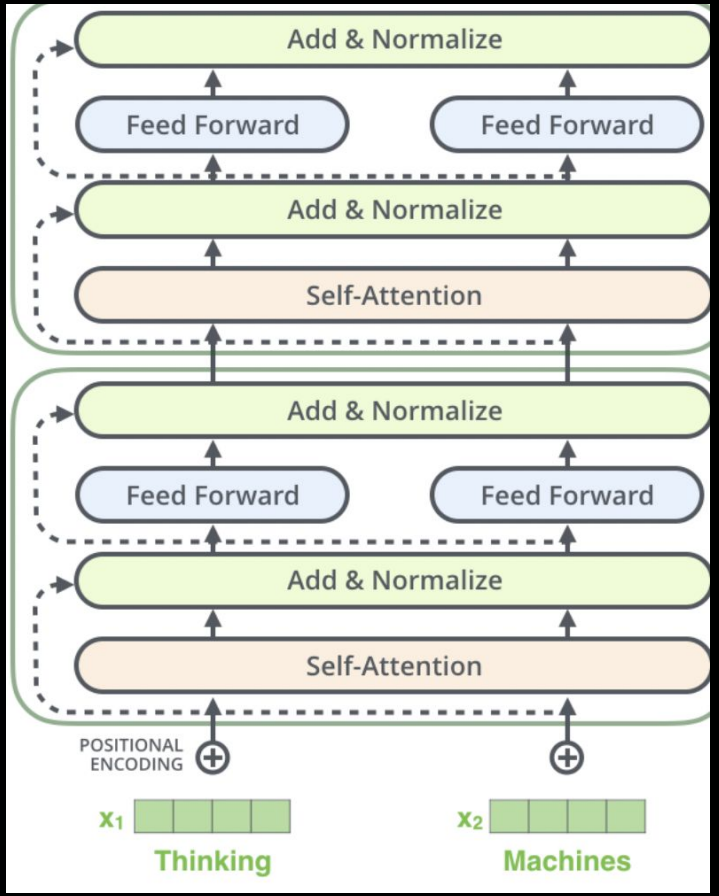
How do LLM's work?

- An LLM usually receives some kind of text input usually symbolizing the question or statement that the LLM is to respond to
- When looking at these text inputs LLM's tokenize (split) the sentence into individual words ("summarize", "the", "important",)
- They then embed (convert) each word into a vector of numbers representing the meaning of a word. Like "summarize" could equal the numbers [-0.5,0.2,0.5,-0.1, …..]]



How exactly does an LLM produce a response

- Then the vectors are passed through transformer layers which utilizes a self-attention mechanism
- This self-attention mechanism is where each token looks at all the other tokens and sees how relevant they are to itself
- Based off its relevance to itself the token assigns every other token a score.
- This score gets fed through feedforward neural networks constantly recalculate the score to make it as accurate as possible
- By reading the scores the all tokens give each other the model can start guessing the correct tokens (words) to respond with
- It constantly chooses new words until it can form an accurate sentence to answer the question/statement it received as an input



Summary Generator

- To run our LLM we got a list of the songs to summarize with the corresponding artist for each song
- As an LLM requires a large amount of processing power we lack, we run our LLM through Groq's API which allowed us to run the LLM utilizing Groq's hardware

```
df=pd.read_csv('apple_songs.csv')
song_lst=df['trackName'].tolist()
artist_lst=df['artistName'].tolist()
summary=[]


client = Groq(
    api_key=API_KEY,
)
```

- We then looped through each song and created a chat request which allowed us to send a message to the LLM
- We requested a 25-word summary for each song and then stored said summary into a list that got sent.

```
for i in range(len(song_lst)):
    chat_completion = client.chat.completions.create(
        messages=[
            {
                "role": "user",
                "content": "Summarize the important menaing of the song "+
                (song_lst[i]) + "by the artist" + (artist_lst[i]) +
                "in 25 words without mentioning the song or artist",
            }
        ],
        model="llama3-8b-8192",
    )

    summary.append(chat_completion.choices[0].message.content)
```

Recreating Data

- Afterwards we had to recreate our interaction, user, and item csv to swap the summaries with the Song name

Create interactions with summaries

```
items['track_id'] = range(1, len(items) + 1)
users['user_id'] = range(0, len(users))

id_to_track_name = pd.Series(items['trackName'].values, index=items['track_id']).to_dict()
id_to_user_name = pd.Series(users['name'].values, index=users['user_id']).to_dict()

interactions['trackName'] = interactions['track_id'].map(id_to_track_name)

# Drop the old track_id column if desired
interactions = interactions.drop(columns=['track_id'])

interactions['name'] = interactions['user_id'].map(id_to_user_name)

# Drop the old track_id column if desired
interactions = interactions.drop(columns=['user_id'])

items['song_summary'] = summary['Song Title']
new_df=items.drop(columns=['trackName'])
new_df.to_csv('apple_items_with_lyrics.csv', index=False)

dct={}
for i in range(len(items)):
    dct[items['trackName'][i]]=new_df['song_summary'][i]

interactions['song_summary'] = interactions['trackName'].map(dct)
```

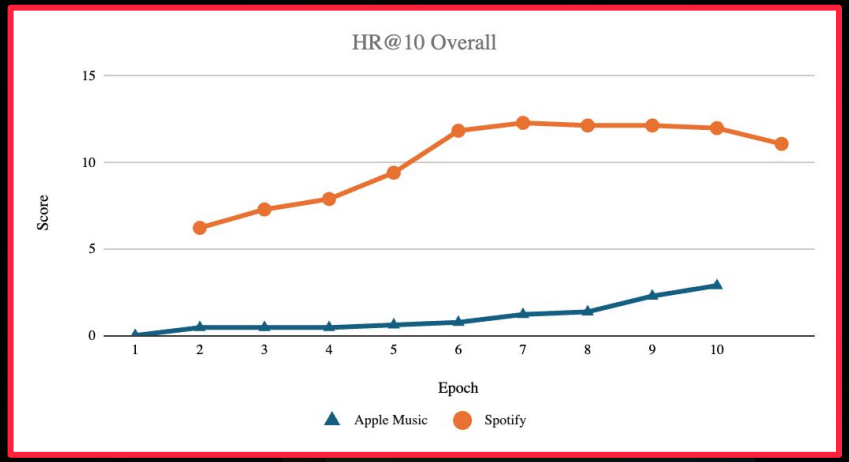
Create users with summaries

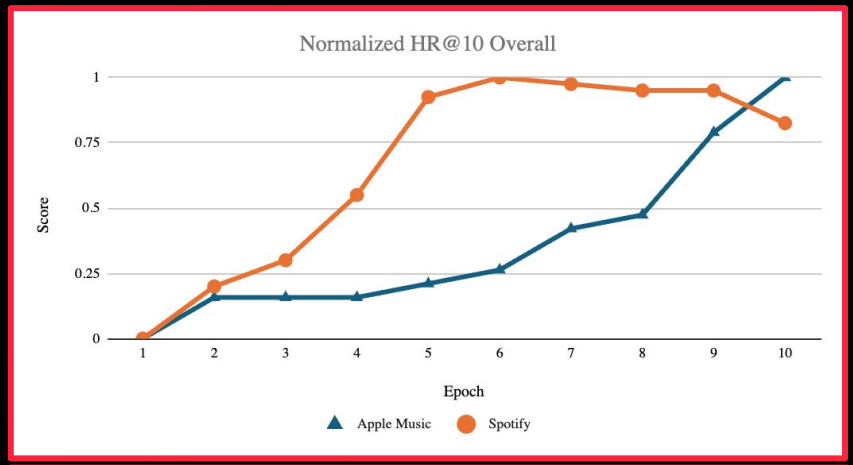
```
user_data=pd.read_csv('s-user.csv')

track_summaries=pd.read_csv('updated_user_track_data.csv')

# Create a dictionary for the track summaries
summary_dict = dict(zip(track_summaries['trackName'], track_summaries['song_summary']))

# Function to replace track names with summaries in the list of tracks
def replace_track_names_with_summaries(track_list_str, summary_dict):
    track_list = ast.literal_eval(track_list_str)  # Convert string representation of list
    for track in track_list:
        if track['track_name'] in summary_dict:
            track['track_name'] = summary_dict[track['track_name']]
    return str(track_list)

user_data['loved_tracks'] = user_data['loved_tracks'].apply(lambda x:
replace_track_names_with_summaries(x, summary_dict))

# Save the updated DataFrame to a new CSV
user_data.to_csv('apple_users.csv', index=False)
```

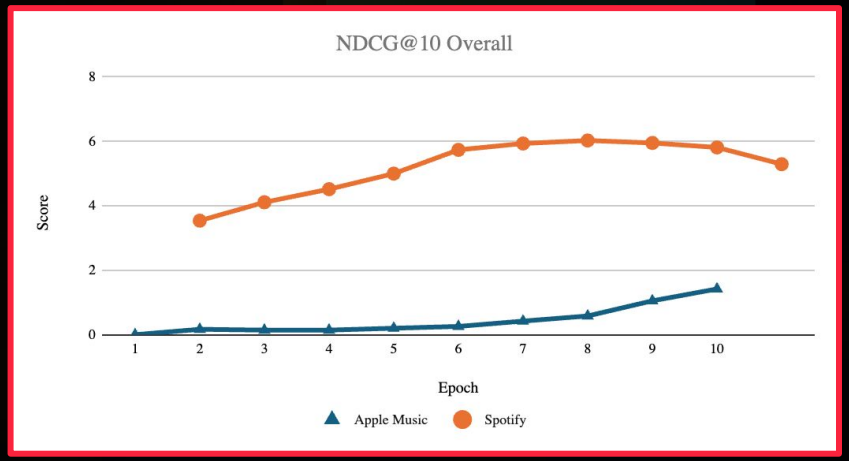
Create items with summaries

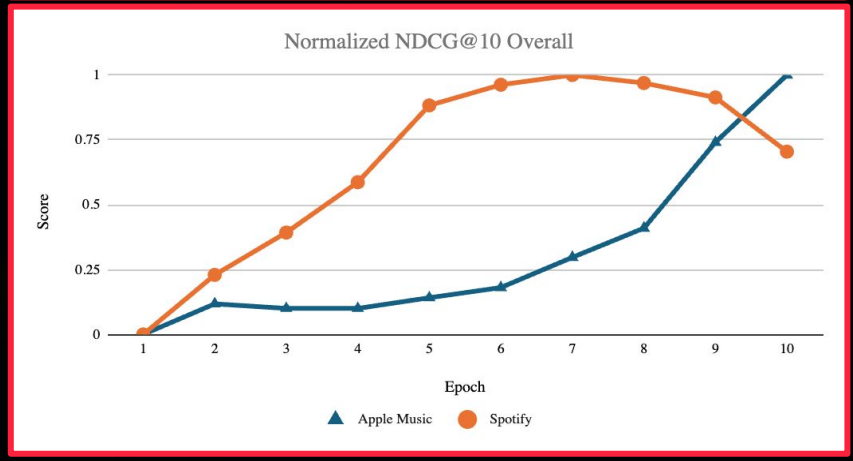
```
items_data=pd.read_csv('spotify_songs.csv')

for index, row in items_data.iterrows():
    track_name = row['trackName']
    if track_name in summary_dict:
        items_data.at[index, 'trackName'] = summary_dict[track_name]


items_data.to_csv('apple_items.csv', index=False)
```

Two-Tower Model (general) - Results









Two-Tower Model (summary) - Results for Apple Music

```
  0%|          | 0/1 [00:00<?, ?it/s]
100%|██████████| 1/1 [00:08<00:00,  8.13s/it]
100%|██████████| 1/1 [00:08<00:00,  8.22s/it]
HR@10 Overall = 6.22154779969651, NDCG@10 Overall = 2.903682754505036
HR@10 Cold = nan, NDCG@10 Cold = nan
HR@10 Warm = 6.22154779969651, NDCG@10 Warm = 2.903682754505036
Best HR so far = 6.22154779969651, Best NDCG so far 2.903682754505036
```

Best Epoch: 26

```
Total number of items = 3629
Number of items present across training edges = 230
Number of items present across val edges = 195
Number of items present across test edges = 194
Average item degree = 0.9837420777073574
Average user degree = 0.5275602187084381
Number of items common between train and validation edges = 195
Number of items common between train and test edges = 194
Number of cold items in validation set = 0
Number of cold items in test set = 0
```

Two-Tower Model (summary) - Results for Spotify

```
100%|██████████| 1/1 [00:01<00:00,  1.33s/it]
100%|██████████| 1/1 [00:01<00:00,  1.42s/it]
HR@10 Overall = 12.139605462822459, NDCG@10 Overall = 5.93189040720068
HR@10 Cold = 0.0, NDCG@10 Cold = 0.0
HR@10 Warm = 26.666666666666668, NDCG@10 Warm = 13.030385927817495
Best HR so far = 12.139605462822459, Best NDCG so far 5.93189040720068
```

Best Epoch: 6

```
Total number of items = 230
Number of items present across training edges = 230
Number of items present across val edges = 195
Number of items present across test edges = 194
Average item degree = 15.521739130434783
Average user degree = 5.417298937784522
Number of items common between train and validation edges = 195
Number of items common between train and test edges = 194
Number of cold items in validation set = 150
Number of cold items in test set = 148
```

Backing up our enhancement

- While our model did see improvements, we wanted to ensure the improvements came because the lyrics meaning had an effect as opposed to better results simply because the summary had more words than the Song title

- To prove this we decided to create a third model, a model where we took the first 25 words of the song and see if it had how good the recommendation system would be with that.

Lyrics Generator

- To ethically grab the lyrics as opposed to web scraping the lyrics of the songs we ran an API called Lyrics OVH

- By passing in the Song name and the Artist name, the API would return the lyrics of the song in which we would take the first 25 words and pass it into a list

```
df=pd.read_csv('apple_songs.csv')
song_lst=df['trackName'].tolist()
artist_lst=df['artistName'].tolist()
summary=[]
print(len(song_lst))

def get_lyrics(song_title, artist_name):
    base_url = "https://api.lyrics.ovh/v1"
    song_url = f"{base_url}/{artist_name}/{song_title}"
    response = requests.get(song_url)

    if response.status_code == 200:
        lyrics = response.json().get("lyrics", None)
        return lyrics
    return None

def get_first_25_words(lyrics):
    words = lyrics.split()[:25]
    return ' '.join(words)
```

Loop through every song

```
for i in range(len(song_lst)):
    print(i)
    song_title = song_lst[i]
    artist_name = artist_lst[i]

    lyrics = get_lyrics(song_title, artist_name)
    if lyrics:
        first_25_words = get_first_25_words(lyrics)
        print(f"First 25 words of '{song_title}' by {artist_name}:")
        summary.append(first_25_words)
    else:
        summary.append(song_title)
song_summary_df = pd.DataFrame(summary, columns=["Song Title"])
song_summary_df.to_csv("song_lyrics_spotify_2000-end.csv")
```

Recreating Data

- Similarly to before we have to recreate our interaction, user, and item csv to swap the summaries with the Song name

Create interactions with lyrics

```
items['track_id'] = range(1, len(items) + 1)
users['user_id'] = range(0, len(users))

id_to_track_name = pd.Series(items['trackName'].values, index=items['track_id']).to_dict()
id_to_user_name = pd.Series(users['name'].values, index=users['user_id']).to_dict()

interactions['trackName'] = interactions['track_id'].map(id_to_track_name)

# Drop the old track_id column if desired
interactions = interactions.drop(columns=['track_id'])

interactions['name'] = interactions['user_id'].map(id_to_user_name)

# Drop the old track_id column if desired
interactions = interactions.drop(columns=['user_id'])

items['song_summary'] = summary['Song Title']
new_df=items.drop(columns=['trackName'])
new_df.to_csv('apple_items_with_lyrics.csv', index=False)

dct={}
for i in range(len(items)):
    dct[items['trackName'][i]]=new_df['song_summary'][i]

interactions['song_summary'] = interactions['trackName'].map(dct)
```

Create users with summaries

```
user_data=pd.read_csv('s-user.csv')

track_summaries=pd.read_csv('updated_user_track_data.csv')

# Create a dictionary for the track summaries
summary_dict = dict(zip(track_summaries['trackName'], track_summaries['song_summary']))

# Function to replace track names with summaries in the list of tracks
def replace_track_names_with_summaries(track_list_str, summary_dict):
    track_list = ast.literal_eval(track_list_str)  # Convert string representation of list
    for track in track_list:
        if track['track_name'] in summary_dict:
            track['track_name'] = summary_dict[track['track_name']]
    return str(track_list)

user_data['loved_tracks'] = user_data['loved_tracks'].apply(lambda x:
replace_track_names_with_summaries(x, summary_dict))

# Save the updated DataFrame to a new CSV
user_data.to_csv('apple_users.csv', index=False)
```

Create users with new format

```
items_data=pd.read_csv('spotify_songs.csv')

for index, row in items_data.iterrows():
    track_name = row['trackName']
    if track_name in summary_dict:
        items_data.at[index, 'trackName'] = summary_dict[track_name]


items_data.to_csv('apple_items.csv', index=False)
```

Lyric Summary Results for Apple Music

```
HR@10 Overall = 17.667679364337463, NDCG@10 Overall = 8.548743145334772
HR@10 Cold = 0.33751205400192863, NDCG@10 Cold = 0.10097021898305215
HR@10 Warm = 33.96825396825397, NDCG@10 Warm = 16.494630242501877
Best HR so far = 17.667679364337463, Best NDCG so far 8.548743145334772
```

Best Epoch: 4

```
Total number of items = 241
Number of items present across training edges = 241
Number of items present across val edges = 239
Number of items present across test edges = 239
Average item degree = 23.921161825726140
Average user degree = 1.347277401261977
Number of items common between train and validation edges = 239
Number of items common between train and test edges = 239
Number of cold items in validation set = 190
Number of cold items in test set = 190
```

Lyric Summary Results for Spotify

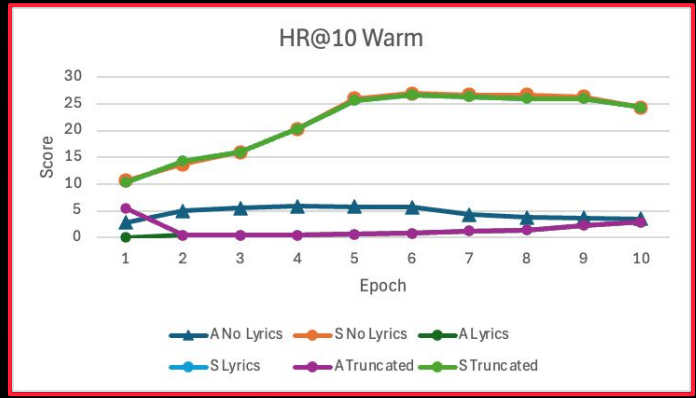
```
HR@10 Overall = 15.629742033383915, NDCG@10 Overall = 6.9137578223582645
HR@10 Cold = 1.1142061281337048, NDCG@10 Cold = 0.49773920478590955
HR@10 Warm = 33.0, NDCG@10 Warm = 14.591593434719847
Best HR so far = 15.629742033383915, Best NDCG so far 6.9137578223582645
```

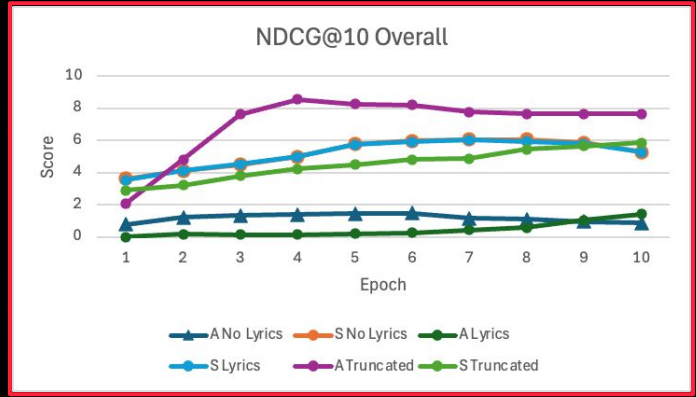
Best Epoch: 18

```
Total number of items = 227
Number of items present across training edges = 227
Number of items present across val edges = 192
Number of items present across test edges = 192
Average item degree = 15.691629955947137
Average user degree = 5.405159332321699
Number of items common between train and validation edges = 192
Number of items common between train and test edges = 192
Number of cold items in validation set = 147
Number of cold items in test set = 146
```

Lyric Truncated vs Summary vs. Song Name

- Overall, our results showed that truncated lyrics improved the model by a significant amount, and even tripled the HR.
- In order from best to worst
  - Spotify: 1. Truncated 2. Song Name 3. Summary
    - Song name and summary were almost on par with each other
  - Apple Music: 1. Truncated 2. Summary 3. Song Name
- These results could've been influenced by AI hallucination and the sizes of our datasets



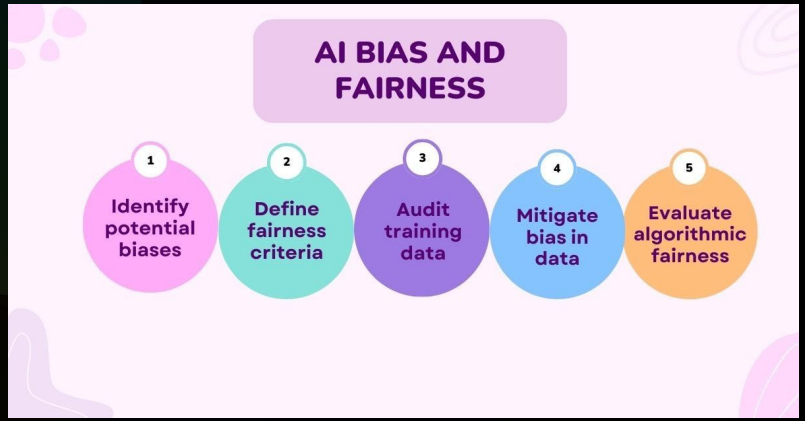


Conclusion & Future Work

- In conclusion, this challenge gave us the opportunity to dive into deep learning and understand how recommender systems work
- We experimented with several different features and found that truncated lyrics, instead of song names, improved the recommendation of songs for both Apple Music and Spotify
- This work inspired our team to write a paper that will be submitted to a workshop (MuRS - Music Recommendation Systems Workshop 2024)
- If we had more time to work on this project, we would be interested in giving the LLM the truncated lyrics to summarize instead of having it return summaries based on artist and song since that caused some confusion for it and produced vague summaries
- We also think it would be interested to find a larger sentence transformer for our data, and look for other item/user features

Fairness vs. Bias

In machine learning, **fairness** refers to the principle of ensuring that algorithms and models provide equitable outcomes for all individuals and groups, avoiding unjust or prejudiced results. **Bias**, on the other hand, occurs when there is a systematic error or deviation in the data, model, or predictions that unfairly favors certain individuals or groups over others.



Fairness in ML

-There are many definition in machine learning in regards to fairness

-Machine learning fairness is the process of correcting and eliminating algorithmic bias (of race and ethnicity, gender, sexual orientation, disability, and class) from machine learning models.

-It's important to have both quantitative and qualitative data when building a machine learning model to ensure a comprehensive understanding of the data and to capture both measurable metrics and nuanced perspectives for a more balanced and effective model.

https://www.coursera.org/articles/machine-learning-fairness

Biases in Our Project

Addressing **Sampling Bias.** Ensures that our datasets reflect a wide range of demographic groups helping prevent over-representation or under-representation of certain groups, thus reducing the risk of biased recommendations.

To address sampling bias, we ensure our datasets from Last.fm, Apple Music, and Spotify include diverse demographics, genres, and artists.

Addressing **Outcome Bias.** By using fairness metrics like demographic parity or equal opportunity, we can identify disparities in recommendations across different demographic groups, ensuring that the model's outcomes are equitable.

To address outcome bias, we implement fairness metrics such as demographic parity and equal opportunity. This allows us to detect and correct disparities in recommendations, ensuring the model treats all demographic groups equitably.

Addressing **Subjectivity Bias**. Gathering qualitative feedback from diverse stakeholders helps capture nuances and perspectives that might not be evident through quantitative metrics alone, providing a more holistic view of fairness.

To address subjectivity bias in our dataset, we would collect qualitative feedback from users of diverse backgrounds. This feedback helps identify if any groups feel underrepresented, allowing us to refine the model for a more inclusive experience.

Q&A

Sources

1. Google. “Content-Based Filtering | Machine Learning | Google for Developers.” *Google*, developers.google.com/machine-learning/recommendation/content-based/basics. Accessed 31 July 2024.
2. He, Xiangnan, et al. “Neural Collaborative Filtering.” *arXiv preprint arXiv:1708.05031*, 2017. Available at arXiv.
3. McAuley, Julian. *Personalized Machine Learning*. University of California, San Diego, 2023. Available at CSE UCSD.
4. Reference Repo: link
5. Slides from SlideChef

THANKS!