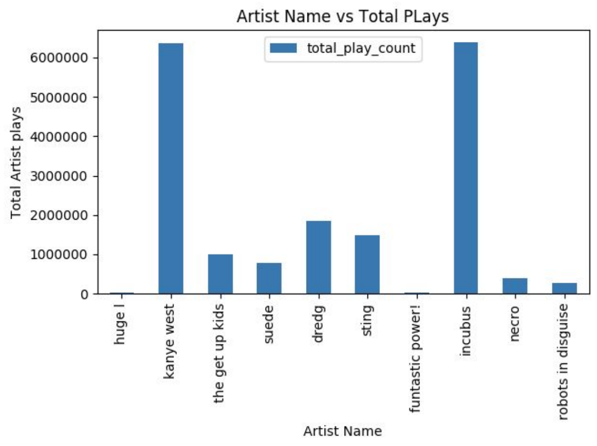
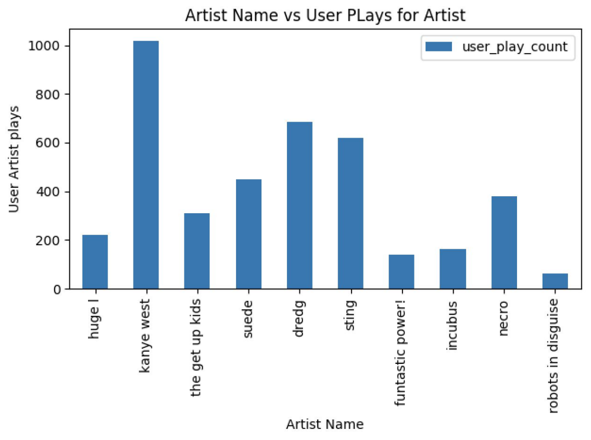
Abstract:

The average American spends 32 hours a week listening to music and, according to Time magazine, there are more people who listen to playlists than those who listen to albums. Self-curated playlists ease the process of listening to music because they are a collection of music that the user enjoys. Today, playlists curated by Spotify like *Today's Top Hits* and *RapCaviar* boast millions of followers and are the go to destinations for many looking to find new music. However, it can be difficult to find an all encapsulating playlist that is personalized to the users that listen to them.

With that being the case, we would like to focus on recommending artists that are personalized to the user by learning from their listening history. Our approach starts off by preprocessing the user and listening history data so that we can compute more efficiently and have more accurate results. We then merged the user data with listening data and built an efficient utility matrix for songs and users. Using a Nearest Neighbor model from scikit, we recommend artists to users either randomly or by using a list of artists from their listening history as a base. We also attempted other approaches, such as alternating least squares, but the performance of our Nearest Neighbor model where k=5 provided the best results. The feature that was most valuable for recommending artists was the number of times a user played an artist.





(figures: Dataset Sample random artists vs their total plays (left), Dataset Sample random artists vs user plays for artist (right). This illustrates the random distribution and an interesting relationship between random user plays and overall artist plays.)

Report:

Our project aimed to create an item-based collaborative fitting model to recommend similar artists based on a user's music listening history. For this project, we are using the Last.FM dataset (https://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/lastfm-360K.html) . The dataset has two main components for the 360,000 Last.FM users, their activity data and profile data. The activity data includes information about the user's listening history. In particular, we chose to use the number of plays as a relative metric for the rating of the song. Their profile details the user's age, gender, user id, and country of residence. We loaded these attributes into a Pandas dataframe in order to preprocess it further.

We did some analysis and found that many artists were not listened to very much. We found that the median number of plays for the artists was about 220. Since the dataset was so large (over 360k entries), we decided to filter to only the popular artists. Through our analysis, we found that our predictor worked best with artists with over 10000 plays. We wanted to look at the top couple percent of artists to produce the best results. We set several thresholds in order to achieve high accuracy and low computational complexity of our model. First, we removed artists with less than 10000 plays. Unpopular artists have a smaller, more spurious dataset to draw from, and thus produce noisy predictions. We also chose to focus on US and UK users only to further reduce the computational complexity.

We chose to format our data using a utility matrix, a popular model for recommendation systems. The rows of this matrix were the artist names, with the columns being the user ids. The values at these indices were the number of times that the user played a given artist. After constructing this utility matrix a large number of the entries were unknown due to the variability in listening history for each user, meaning it was taking up much more space than it needed to. To solve the space issue, we transformed the utility matrix into a sparse matrix to reduce the memory required, since many of the user artist pairs were unfilled. The memory reduction was from 8gb before using the sparse matrix to 100mb after converting to the sparse matrix.

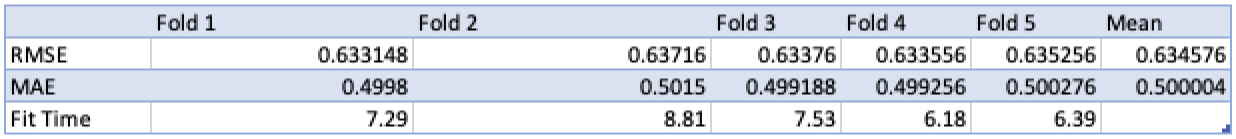
To match similar artists, we used skt-learn library to implement a K-Nearest Neighbor model. The model was fit on the sparse matrix we generated. We also used the Implicit library to create a recommender with it's own built in alternating least squares model. Although we initially tested an Alternating Least Squares method, the nearest neighbor approach consistently outperformed it in our tests. To compute the nearest neighbor, we compared the distance between the artist vectors in our sparse matrix. We researched the most effective distance metric by comparing euclidean, l1, l2, and cosine measurements. By comparing the accuracy of the results of each metric, we chose cosine similarity as the distance measurement in our model because it consistently produced the highest accuracy. We then trained the model to our sparse utility matrix with 10-fold cross validation. We partitioned the data to use 70% for training and 30% for testing. The training and test sets were split up into separate dataframes. After testing various k-values for the nearest neighbor model, we found that a k-value of five produced robust and informative results.

Using the implicit Python library, we are able to test our recommender system. The model was fit on the training set for both the Alternating Least Squares model and the K-Nearest Neighbors model (an item-based model).

Model	Model Recall (%)	Model Average NDCG (%)
KNN	2.98	1.87
ALS	18.32	10.43

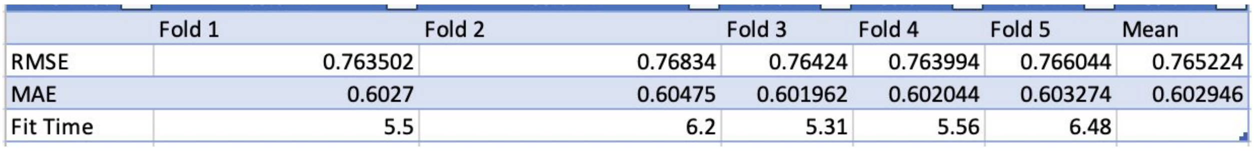
(Figure 1: Model Recall and Model Average Normalized Discounted Cumulative Gain)

We then performed cross validation on our data with training and test datasets altering in size from 10,000 samples to 400,000 samples. We were able to tune the KNN model and the ALS model to get more favorable results. From there, we used the Surprise python library to calculate the RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error) for our model as well as some other models to compare results.



	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
RMSE	0.633148	0.63716	0.63376	0.633556	0.635256	0.634576
MAE	0.4998	0.5015	0.499188	0.499256	0.500276	0.500004
Fit Time	7.29	8.81	7.53	6.18	6.39	

(Figure 2: Cross Validation Test Results using Surprise from Alternating Least Squares Model)



	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
RMSE	0.763502	0.76834	0.76424	0.763994	0.766044	0.765224
MAE	0.6027	0.60475	0.601962	0.602044	0.603274	0.602946
Fit Time	5.5	6.2	5.31	5.56	6.48	

(Figure 3: Cross Validation Test Results using Surprise for K-Nearest Neighbors model)

We noticed that we were able to predict if a user likes a random artist that we recommended pretty well. Our algorithm for our model choice was K-Nearest Neighbors and was able to predict similar artists for a given artist name.

In the future, we would like to gather genre tags for these artists, as well as more song data. If we can figure out which aspects make users like certain songs or genres, we can predict artists or songs with even more accuracy. Our current approach is not scalable since it is tied to a fixed dataset, therefore future iterations would be connected to a scalable database that is constantly updated with new artists and user listening data. It would be interesting to connect recommender results with the Spotify API and generate playlists. Additionally, it'd be interesting to see how the results would look with a KNN, ALS model pair.

Collaboration:

I. Proposal
    A. We worked on this together, everybody brought forth ideas and we voted on one.

II. Status
    A. After coming together and deciding a project we were all excited about, we continued to test ideas collaboratively and wrote the report on our initial findings.

III. Final Report
    A. We had the opportunity to refine our approach during the last part of the class, and each step outside of our comfort zones. Everyone got a chance to work on some aspects of the code and project website, as well as comment on the final output in the report.