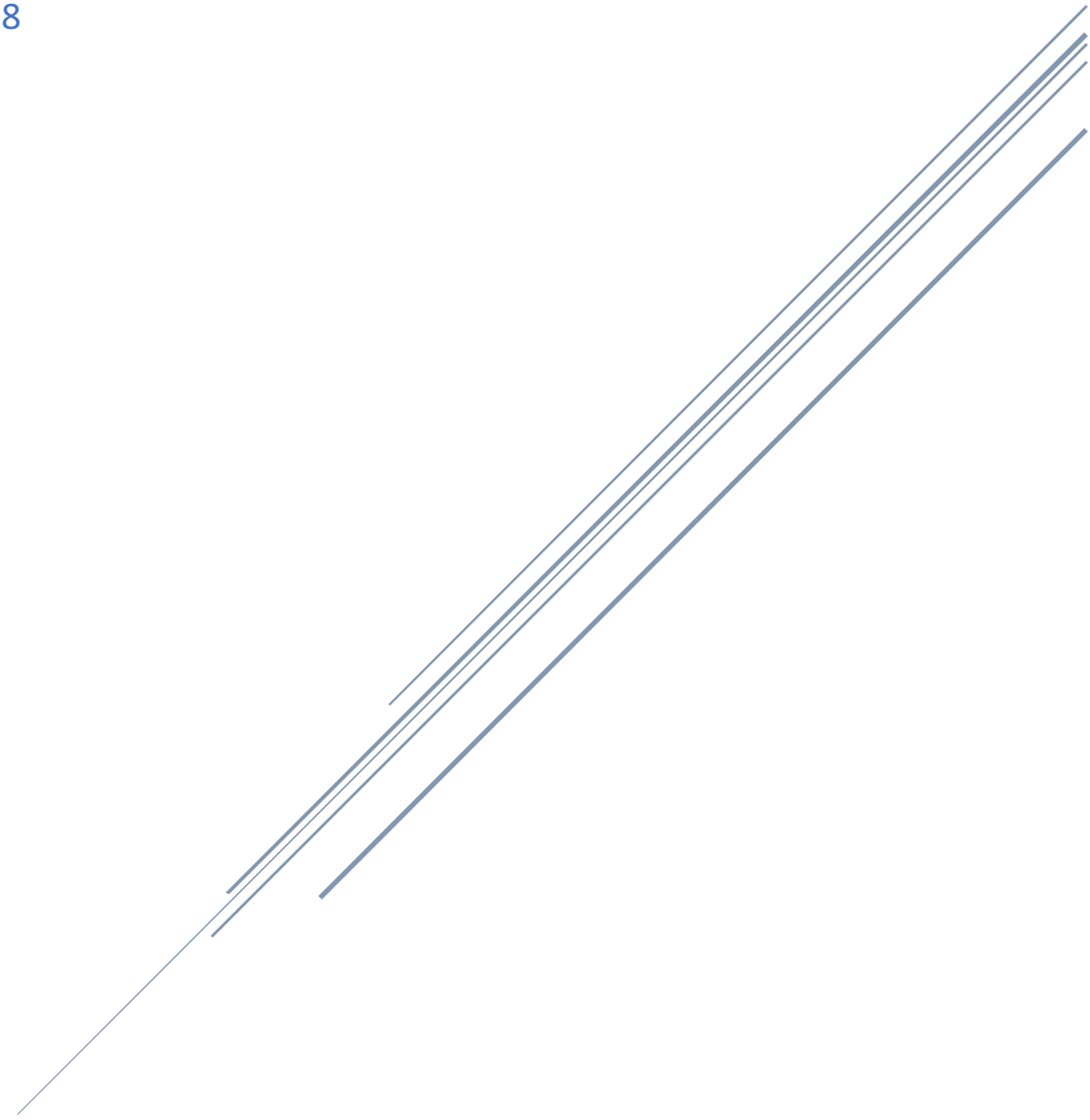
RECOMMENDATION SYSTEM BUILD

Detailed Project Report

Ivette M Tapia

9.2.2018



Springboard
Data Science Career Track

Table of Contents

Introduction 3

Motivation 3

What are recommendation systems? 4

Brief Overview of Recommendation Systems 5

Dataset Creation Background 8

Data Wrangling 10

Dataset Features 18

Exploratory Data Analysis and Statistical Inference 19

- Analytic Goals 19
- Major Findings in Exploratory Data Analysis and Statistical Inference 20
- Part 1. Song Features Exploration 21
- Part 2. User Listen Counts 23
- Part 3. User Listen Counts per Song 27
- Part 4. User Listen Counts per Artist 31
- Part 5. Relationships between basic song features and song listens 35
- Part 6. Artist Tags and Terms 39

Recommendation Algorithms Implementation 40

- A. Python Libraries Used 40
- B. Data Preparation and Preprocessing 40
- C. Recommendation Algorithms 42
- D. Evaluation 46
- E. Limitations 48
- F. Recommendations and Next Steps 49

Sources 50

Introduction

This report summarizes the overall process taken to build preliminary recommendation algorithms for music recommendation. The data used to train and validate the recommendation system was the million song dataset and the user taste profile. This dataset was created and is distributed by Columbia University's LabRosa. The Million song dataset contains metadata for one million songs; while the user taste profile dataset contains user song listening counts and for 1.019M EchoNest users.

The report starts by outlining the motivation for the recommender system, and follows with a brief overview of recommendation systems. After this background, a broad overview of the process LabRosa took to create the datasets is outlined followed by the data wrangling steps taken to prepare the data for further analysis and recommender system creation. After this, exploratory data analysis results are discussed. Finally, the recommender system is described and predictive performance results are discussed. The coding language used throughout this project is python and the coding interface is Jupyter notebooks. The deliverables for the project is python code, a detailed report, and presentation slides.

Motivation[1]

The Great Music Service company has a 1M catalog of songs to listen to available to their users. Given the large catalog of songs, no one user can be reasonably expected to go through all the content and choose what they want to listen to.

The Great Music Service conducted with survey of their active users found that: (1) [2]users are not aware of all the music available, (2) the catalog is too big and is hard to choose what to listen to, (3) they do not always know what they are looking for, and (4)

[1] This scenario is used to illustrate a realistic use case for recommendation system and frame the project.
[2] https://www.businessinsider.com/netflixs-recommendation-engine-drives-75-of-viewership-2012-4

they often use the music service as a song discovery tool in which they are explicitly interested in listening to new things.

Recommendations engines have shown to have significant benefit to commercial enterprises. Seventy-five percent of content watched in Netflix comes from its recommendation engine. Moreover, thirty-five percent of Amazon's revenue comes from its recommendation system.[3] Finally, studies show that 86% of consumer's purchases are significantly influenced by personalized recommendations.

Due to the survey results and recommender system statistics, the Chief Technology Officer and the Chief Data Scientist have decided to build a recommendation system for the music service platform to assist each user in finding and engaging with content that is relevant to them. If users engage with content relevant to them, they are more likely to spend more time in the service and less likely to churn. Higher engagement directly leads to higher revenue for the music service. The chief data scientist has tasked a member of the data science team to build a prototype recommendation system and recommend next steps.

What are recommendation systems?

Recommendation systems are a type of retrieval information technology. Recommendations engines differ from search in that: (1) users will not necessarily tell the engine what they are looking for, or (2) users may not know what they are looking for. These types of engine use individual's explicit preferences or behaviors (implicit preferences) to retrieve information that is relevant to the specific active user. The next section gives a brief overview of key recommender goals, types and evaluation metrics

[3] http://rejoiner.com/resources/amazon-recommendations-secret-selling-online/

Brief Overview of Recommendation Systems

Goals of Recommendation Systems

- Prediction Version: Predict values a user will give to an unrated item.
- Ranking Version: Determine top-k items for a particular user.

Types of User Data Used in Recommendation Systems

1- **Explicit Rating:** Ratings are specified by the user in regards their level of preference. Like and dislike for the item is explicit. The task is to predict the rating for the unrated items in the matrix.

2- **Implicit Rating:** Tracking of user behavior. A missing value does not mean a dislike for an item. As such, it is recommended that the matrix is converted to a unary matrix (1,0) and missing values are filled with zeros to prevent overfitting.

Considerations

- ✓ Relevance
- ✓ Novelty
- ✓ Serendipity
- ✓ Diversity

Challenges

1- Sparsity
2- Cold Start
3- Fraud

Evaluation

1- MSE
2- RMSE
3- Precision
4- Recall
5- AUC
6- ROC
7- Kendall's Tau (for rakings)

Recommender Type	Description	Approaches	Advantages	Disadvantages
Content-Based	It recommends items that are similar in content to those the user has liked in the past or matches their 'profile'.	• Naïve Bayes classifier • K-nearest neighbor • Decision Trees • Neural networks	• Easy to explain. • Provides personalized recommendations.	• Leads to recommendations that are 'too' similar. • Does dot have serendipity effects.
Collaborative Filtering	A user is recommended items based on past ratings of a group of users. These types of recommenders can be divided in two broad groups: memory based and model –based.	**User-Based:** the ratings provided by like-minded users of target A are used to make recommendations for A (i.e. neighborhood based)	• Easy to implement and explain. • Exploits the logic of 'word of mouth' and user similarities.	• Does not scale well. • Computationally complex. • Highly engaged users can have a large impact. • Does not work well in sparse matrixes.
		Item-Based: First determine the set of items S that are most similar to target item B. The ratings of set S items are used to determine whether a user will like target item B.	• Easy to implement and explain. • Recommendations make sense. • Can be used as product association tool. • More efficient than user-based.	• Can produce recommendations that are too 'obvious'. • Hard to discover items that are different, user-based tends to be better at this.
		Model Based: Decision trees, rule based, Bayesian & Latent Factor (matrix factorization, SVM, SVD). They work within an optimization framework.	Tend to work reasonably well with sparse data. Works within optimization framework. Latent models assume similarity between users is induced by a 'unknown' factor in the data.	Need to regularize. They are parametric models. Can get stuck on local optimum cannot know if it has reached global optimum.
Hybrid	Combine several approaches in providing recommendations.	Usually is done by combining content-based and collaborative filtering methods through weighting, cascading or zipping.	• Exploit strengths of several approaches and minimize weaknesses of other approaches.	• Complex to build and explain.

Knowledge Based	Recommendations based on explicit user requirements.	There are several approaches: conversational, search-based and navigation based.	• Works well when the user is knowledgeable about the item domain. • Works well for high-price ticket items.	• Need a lot of information about the item.
Demographic Based	Recommendations based on user demographic attributes.	Use social graph and demographic features to build recommender.	• Uses demographic data.	• Can lead to assumptions about certain demographic groups which may not be accurate for an individual.
Location Based	Take location into account when making recommendations.	Include location and geographical considerations into on other recommender approaches.	• Takes time into account when making recommendations potentially leading to higher relevance and/or accuracy.	• Need location data.
Time Sensitive	Take time into account when making recommendations.	Include time and longitudinal considerations on other recommender approaches. Can be combined with location depending the domain.	• Takes time into account when making recommendations potentially leading to higher relevance and/or accuracy.	• Needs time data. • It is longitudinal in nature. Need to be aware of time-effects.
Group Based	Recommendations to a group (i.e. find least miserable options for the group).	Instead of targeting recommendation to individuals, target group preferences.	• Makes recommendations for an entire group. • Attempts to find the most acceptable option for the group.	• Works well in specific domains.

Dataset Creation Background[4]

- The million-song dataset was created by using python wrapper on the ECHO Nest API for 10 days in December of 2010.

- The collection criteria for the songs dataset was as follows:

    1. Getting the most 'familiar' artists according to The Echo Nest, then downloading as many songs as possible from each of them.
    2. Getting the 200 top terms from The Echo Nest, then using each term as a descriptor to find 100 artists, then downloading as many of their songs as possible.
    3. Getting the songs and artists from the CAL500 dataset.
    4. Getting 'extreme' songs from The Echo Nest search params, e.g. songs with highest energy, lowest energy, tempo, song hotttnesss, etc...
    5. A random walk along the similar artists links starting from the 100 most familiar artists.

- The resulting dataset is a collection of popular western songs. The number of songs was approximately 8,950 after step 1), step 3) added around 15,000 songs, and then added approx. 500,000 songs before starting step 5.

- The user dataset was created in October 2011 by the Echo Nest. The user data include a shuffled hash of persistent session identifiers from a small random selection of the musical universe and only play counts associated with Echo Nest song IDs that overlap with the song set. No usernames, listener details, original IDs, dates, IPs, locations were released.

[4] https://labrosa.ee.columbia.edu/millionsong/faq, http://blog.echonest.com/post/11992136676/taste-profiles-get-added-to-the-million-song

- The user data is composed of solely of implicit feedback. This means that I only know what the user listened to, and a listen to a song does not mean the user liked or disliked the song.

Data Wrangling

At a basic level, three pieces of information are needed to build a recommendation system: (1) information about the items, (2) information about item users, and (3) information about the relationship between items and users. As such, the goal of the data wrangling task was to obtain these three pieces of information. I have divided the data wrangling process into 12 parts. See figure 1 for a broad summary of the data wrangling steps.

Part 1. Data Extraction

File	Description	Source	Type	Quantity of Files
Million Song Dataset	Collection of 1M .h5 files. Each file represents data for one song.	Using the terminal: rsync -avzuP publicdata.opensciencedatacloud.org::ark:/31807/osdc-c1c763e4/remotefile /path/to/local_copy	.h5	1M
Million Song Summary File	Song metadata information.	http://labrosa.ee.columbia.edu/millionsong/sites/default/files/AdditionalFiles/msd_summary_file.h5	.h5	1
Artist Terms and Tags	Artist terms and tags obtained from the Echo Nest API and the Music Brainz.	Using the terminal: rsync -avzuP publicdata.opensciencedatacloud.org::ark:/31807/osdc-c1c763e4/remotefile /path/to/local_copy **Located in file A	SQLite	1
Million Song Duplicates	List of known song duplicates.	http://labrosa.ee.columbia.edu/millionsong/sites/default/files/AdditionalFiles/msd_duplicates.txt	.txt	1
Million Song Mismatches	List of known song mismatches.	http://labrosa.ee.columbia.edu/millionsong/sites/default/files/tasteprofile/sid_mismatches.txt	.txt	1

Part 2. Song Metadata Summary File

The song metadata summary file is an .h5 file that has metadata information for 1M song files. The file is divided into three groups: (1) metadata, (2) analysis and (3) musicbrainz. Each group contains numpy arrays for each song.[5]

The **song metadata** group contains arrays that describe basic features of a song such as id, song title, song hotness, artist id, and release album. The **analysis metadata** group contains song's musical features such as key, length, danceability. The **musicbrainz** group includes the year of song release. The wrangling steps for each group of arrays are as follows:

- **Song Metadata Dataset:** The group's arrays were converted into a pandas dataframe. Once in the dataframe, object type columns were converted to strings and columns with b‘ ’ string were removed. These strings were the result of python parsing the string data into Unicode from another format. There were also removed to make song identifying features consistent since these strings are not found in other complimentary datasets. The resulting data had missing values in some of the columns. The dataframe has 1M entries.

- **Analysis Metadata Dataset:** The group's arrays were converted into a pandas dataframe. Once in the dataframe, object type columns were converted to strings and columns with b‘ ’ string were removed. These strings were the result of python parsing the string data into Unicode from another format. There were also removed to make song identifying features consistent. The resulting data had no missing values. The dataframe has 1M entries.

[5] The aggregate song metadafile was created by the researches of Columbia's LabRosa throught iteration. The code they used can be found here:
https://github.com/tbertinmahieux/MSongsDB/blob/master/PythonSrc/create_aggregate_file.py

- **Musicbrainz:** The group's arrays were converted into a pandas dataframe. The dataset has two columns, with one of these columns representing year of song release. Zeroes in the year column represent missing. As such, zeroes were replaced with 'NaN' using df.replace and np.nan. The dataframe has 1M entries.

An alternative approach to using the song metadata summary file was to use an iterator to extract the information of interest from each of the individual song's .h5 files. The difference between the summary file and the full dataset, is that the full dataset contains artist similarity analysis and in-depth song analysis. This approach was explored; and a function, as well for loop were built to extract the data from the larger dataset.

However, during testing the amount of time that it took to complete iterating through the files was substantial. Using the summary metadata file showed to be more efficient since the iteration work for the song's metadata has already been done. Moreover, for the purposes of building a recommendation system prototype, song metadata is sufficient and using in-depth song analysis or artist similarity is not required.

Part 3. Artist Terms and Tags

The artist terms and tags are contained in a SQLite database. The database has five tables. The tables artist_mbtag and and artist_term were queried and converted to a pandas dataframe. The resulting dataframes were merged on artist id and using a left merge approach. Missing values in the merged dataframe were filled using the pandas method fillna.

Part 4. User Listening Data

The user listening data text file was converted to a dataframe. . The dataframe read the several columns of data as one. Separating by delimiter was used to separate data into several columns. The result was a dataframe with user_id, song_id and play_count. A

column called “play” with value 1 was added to indicate the user played the song. The resulting dataframe has ~48.4M entries.

Part 5. Song Mismatches List

There is known song mismatches list. This means songs that were labeled incorrectly and matched incorrectly when the dataset was created. The song mismatches text file was converted to a dataframe. The dataframe read the several columns of data as one. Separating by delimiter was used to separate data into several columns. As a result, a dataframe with four columns was created: song_id, track_id, song_name, and does_not_equal_to.

Part 6. Song Duplicates List

There is known song duplicates list. The song duplicate text file was converted to a dataframe. The dataframe read the several columns of data as one. The structure of this data column is a song name followed by track_ids below. The method series.str.split was used to split the track_ids and the song sames.

As a result of this a blank column was created, which was dropped. Two columns remained: track_id and song names. However, blanks were created in the track_id column because song names were previously there. These blanks were filled with NaN using replace method and np.nan. Further, the first five rows were dropped because they had dataset contact information and not actual duplicate information.

Moreover, the song names column has blanks cells below song names (i.e. they are parallel to track_ids). Using the pandas fillna with the forward fill method, song names were propagated forward and to its respective track_ids. Track ids with blank rows were dropped, extrenous number digits in front of the song names were removed, and the index was reset.

This list is the result of finding 131,661 items of 53,471 song objects. This means that 53,471 are unique values to duplicates list dataset. To take this into account, the pandas duplicated method was used to identify unique and duplicated values. Finally, the dataframe was filtered to include only values that were found to be duplicates and remove the ones that are unique.

Part 7. Concatenating Song Metadata

The Song Metadata Dataset, Analysis Metadata Dataset and Musicbrainz datasets were concatenated on the index to crerate one large song metadata dataframe. The resulting dataframe has 1M entries. These columns have missing values: artist_familiarity, artist_hotttnesss, artist_longitude, song_hotttnesss and year.

Part 8. Removing Song Mismatches from Song Metadata

The concatenated song metadata file was merged with the song mismatches dataframe using a left merge approach and on song_id. Due to the merge, new rows and columns were created, including a track_id_y which came from the song mismatches list. The resulting dataframe have ~1.0004M entries.

The track_id_y blanks were replaced with the string ‘not mismatched’. This was done to indicate songs that did not merge with the mismatched items list. The dataframe was filtered by the ‘not mismatched’ string to remove mismatched items. The resulting dataframe has 981,022 entries.

The merge approach was used to filter duplicates rather than comparing the song_id series of each dataframe directly because each series’ length and indexes are different making the comparison challenging and not obvious.

Part 9. Using Duplicates List to Remove Duplicates

The dataframe frame with mismatches removed was merged with the song duplicates dataframe using a left merge approach and on track_id. Due to the merge, new rows and columns were created, including a song_name_y which came from the song duplicates. The resulting dataframe have 981,022 entries.

The song_name_y blanks were replaced with the string 'not duplicate'. This is to indicate songs that did not merged with the duplicates items list. The dataframe was filtered by the 'not duplicate' string to remove mismatched items. The resulting dataframe has 905,712 entries.

The merge approach was used to filter duplicates rather than comparing the track_id series of each dataframe directly because each series' length and indexes are different making the comparison challenging and not obvious.

Part 10. Removing columns created during merges and filling blanks

In this step, columns created by the merged process were dropped. As well blank entries were replaced with 'NaN' using two approaches: fillna and replace. The latter created two blank columns: analyzer version and genre. These blank columns were removed.

Part 11. Converting Datasets into Tidy Format[6]

As part of the wrangling process three main dataframes were created: (1) song metadata, (2) user listening, and (3) artist tags. Song metadata dataframe was melted using song_id as the identifying variable. User listening dataframe was melted using user_id as the identifying variable (the song_id is part of the values). Artist tags and terms dataframe was melted using artist_id as the identifying variable.

[6] More info on tidy data here: https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

Part 12. Output Datasets into CSV files

The song metadata, user listening data and artist tags were exported to a local CSV file using chunksize 500K and the mode append to speed up exporting.

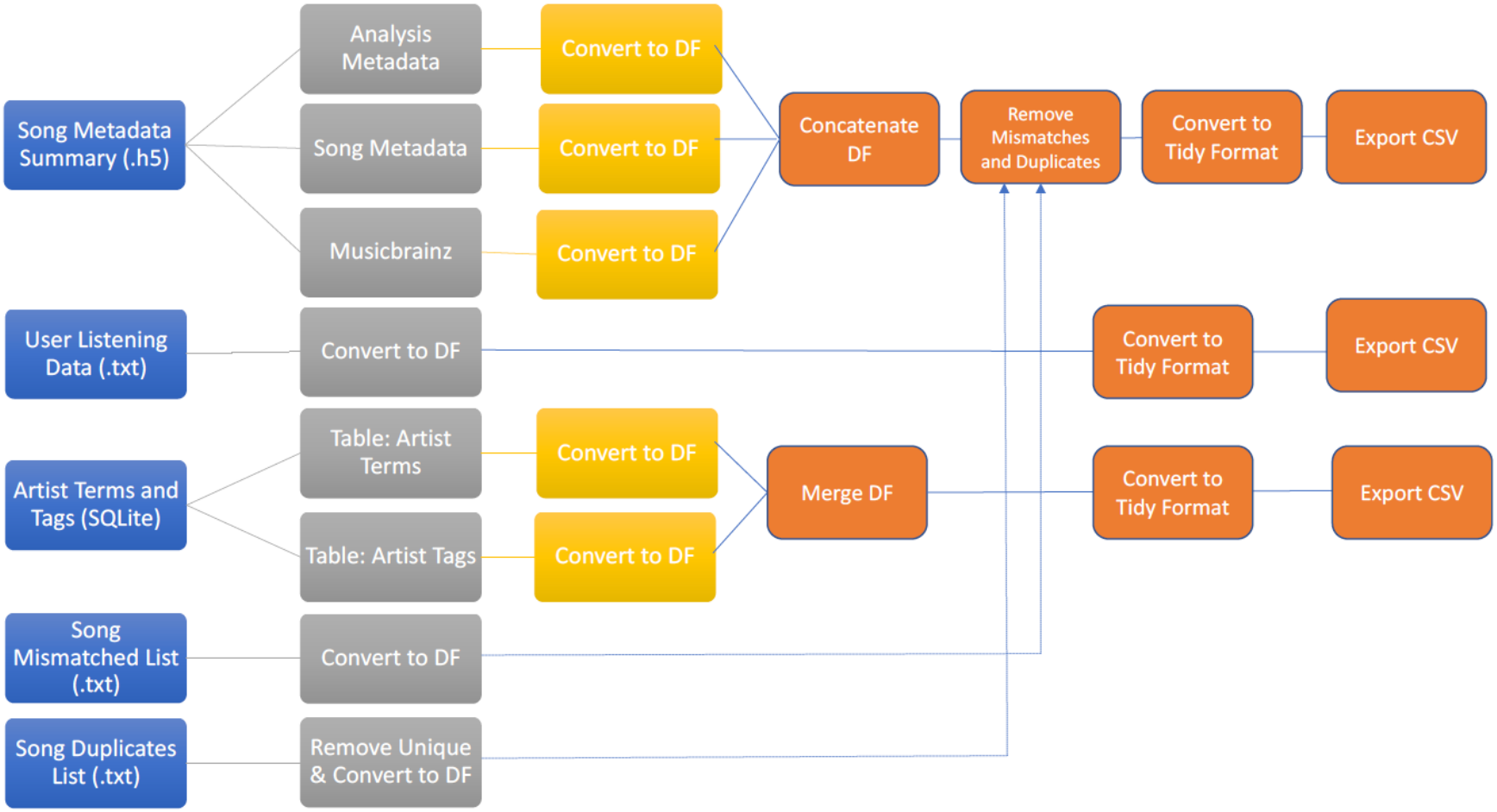


Figure 1. *Summary of Data Wrangling Steps*

Dataset Features[7]

Dataset	Feature Name	Description
User Listen Data	song_id	Echo Nest song ID
	user_id	Echo Nest user ID
	play_count	Number of times user played a song
	play	Whether a user played the song at least once. In recommender systems, it is best practice to treat implicit feedback as 0 or 1 (1 = consumed, 0 = not consumed)
Song Metadata	artist_7digitalid	ID from 7digital.com or -1
	artist_familiarity	algorithmic estimation
	artist_hotttnesss	algorithmic estimation
	artist_id	Echo Nest ID artist ID
	artist_latitude	latitude
	artist_location	longitude
	artist_mbid	ID from musicbrainz.org
	artist_name	artist name
	artist_playmeid	ID from playme.com, or -1
	release	album name
	release_7digitalid	ID from 7digital.com or -1
	song_hotttnesss	algorithmic estimation
	song_id	Echo Nest song ID
	title	song title
	track_7digitalid	ID from 7digital.com or -1
	audio_md5	audio hash code
	danceability	algorithmic estimation
	duration	in seconds
	energy	energy from listener point of view
	key	key the song is in
	loudness	overall loudness in dB
	mode	major or minor
	tempo	estimated tempo in BPM
	time_signature	estimate of number of beats per bar, e.g. 4
	track_id	Echo Nest track ID
	year	song release year from MusicBrainz or 0
Artists Tags	artist_id	Echo Nest artist id
	mbtag	Artists tags from musicbrainz.com
	terms	Artists terms from Echo Nest

[7] https://labrosa.ee.columbia.edu/millionsong/pages/example-track-description

Exploratory Data Analysis and Statistical Inference

Analytic Goals

1- Calculate descriptive statistics of data points/features that will be used in the recommender system.
2- Get a sense of the contents of the song dataset.
3- Gain a strong sense of the distribution of the data points/features that will be used in the recommender system. In particular, user listen counts, song listen counts and artist listen counts.
4- Test hypotheses about the user listen counts, song listen counts and artist listen counts means.
5- Explore relationship between user interactions and basic song features.
6- Cursory view of the top artist tags and tags to gain a sense of top genres represented in the dataset.

Major Findings in Exploratory Data Analysis and Statistical Inference

Song Metadata	**Descriptive Statistics**	905,531 unique songs
	Year of release pattern	Peak year release songs in 2000's.
	Artist song counts pattern	Right skewed with 3 peaks.
Song indicator (y=1, n=0) aggregated by: user, song and artist	**Descriptive Statistics**	1.019M unique users 42% of song the catalog listened 66.5% of artists listened
	Normality Testing: *Graphical and Hypothesis testing*	Highly skewed to the right. Distributions are **not normal**. Some distributions may not satisfy central limit theorem.
	One - sample mean hypothesis testing	High likelihood sample means are close to population means. **User Listen Indicator Means -** *User:* 47 | *Song:* 125 | *Artist:* 1,367
Relationships between song features and play indicator	**Joint plots with Pearson r**	No strong relationship between listens and song features
Artist Tags	**Top 20 artist tags from Echo Nest and MusicBrainz**	Top tags and terms: Rock, electronic and pop.

Part 1. Song Features Exploration

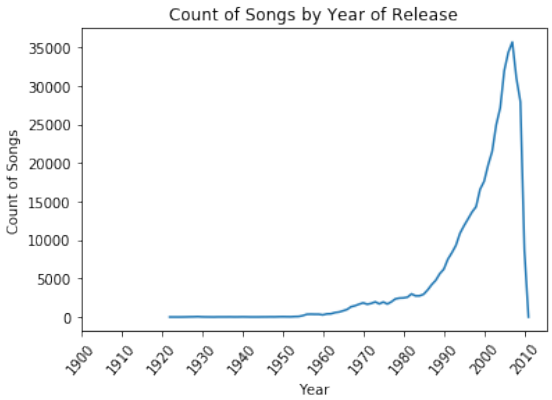
Basic Song Features Descriptive Statistics								
Song Feature	**count**	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
artist_familiarity	905,531	0.55	0.14	0.00	0.48	0.56	0.64	1.00
artist_hotttnesss	905,700	0.38	0.12	0.00	0.33	0.38	0.44	1.08
song_hotttnesss	551,532	0.35	0.23	0.00	0.22	0.38	0.53	1.00
danceability	905,712	0.00	0.00	0.00	0.00	0.00	0.00	0.00
duration	905,712	246.91	124.88	0.31	179.93	227.79	286.41	3034.91
energy	905,712	0.00	0.00	0.00	0.00	0.00	0.00	0.00
key	905,712	5.31	3.59	0.00	2.00	5.00	9.00	11.00
loudness	905,712	-10.16	5.24	-58.18	-12.74	-8.99	-6.39	4.32
mode	905,712	0.67	0.47	0.00	0.00	1.00	1.00	1.00
tempo	905,712	123.84	35.28	0.00	97.73	121.69	144.90	302.30
time_signature	905,712	3.59	1.23	0.00	3.00	4.00	4.00	7.00
year	456,811	1999	10.29	1922	1995	2002	2006	2011

– **Finding #1:** Song Features Descriptive Statistics

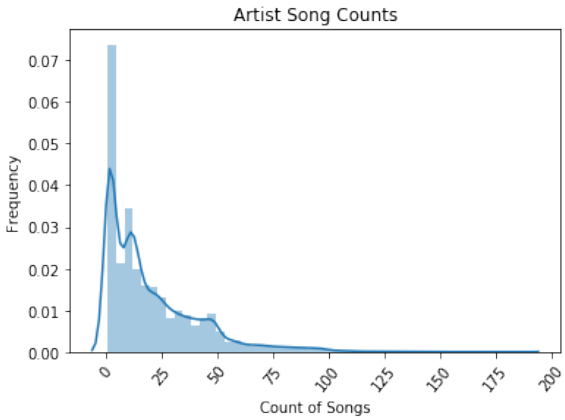
- Looking at the means it seems artist familiarity is 0.55, it falls somewhat in the middle of the possible values of 0 to 1.
- Artist hottness and song hottness mean values skew towards the lower range of hottness.
- In terms of song duration, the mean song is 4.10 minutes (which sounds accurate).
- The mean loudness of the tracks is -10.16, which suggests many of the tracks are not in the extreme of loudness (since this value is far away from max loudness of 4.31)
- The average key is 5.31 (this is around B major or D flat major).
- The mean beats per minute (tempo) is 123. Which means the mean song is in Allegro tempo: fast, quick, and bright.
- The mean mode is 0.66, there is an skew towards songs in the major scale.
- The mean time signature if 3.59 and the 75% percentile is 4. This means most songs are in 4/4 time signature.
- The mean year of release if 1998, with approximately. 25% of songs released between 2006 and 2011.

- **Finding #2:** There are 44,421 unique artists and 905,712 unique songs in the dataset.

- **Finding #3:** Year of release heavily skews to recent releases with a sharp peak in the 2000's. Year data is not complete, I have year data for 456,811 songs or 50% of the song dataset. It is unclear if the missing values are random or systematic, an example of this would be older songs tending to have missing values for year.



- **Finding #4:** Artist song counts are skewed to the right and have some peaks.



Part 2. User Listen Counts

Song Listens by User Descriptive Statistics								
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
play	1,019,318	47.46	57.82	10.00	16.00	27.00	55.00	4,400
play_count	1,019,318	136.05	184.53	10.00	34.00	73.00	163.00	13,132

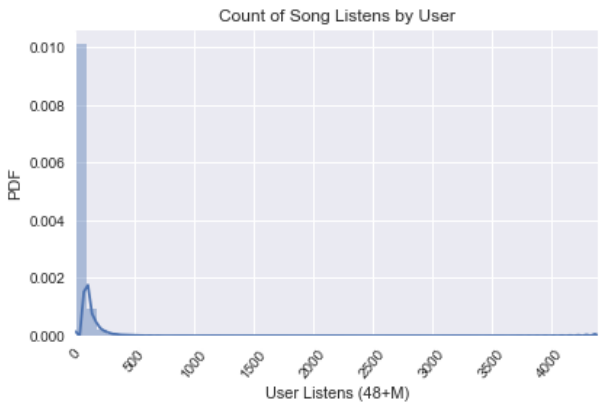
- **Finding #1:** There are 1,019,318 unique users.

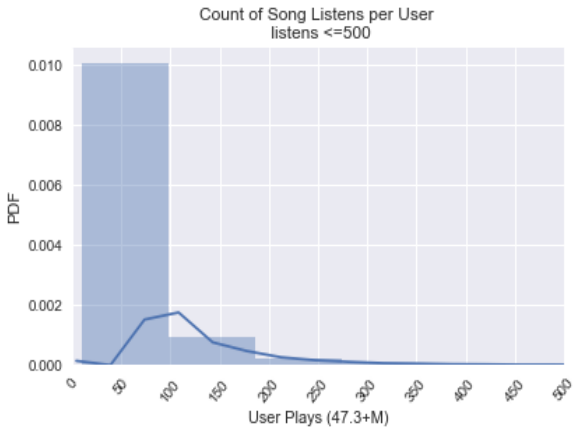
- **Finding #2:** The play indicator has a mean of 47.45 and a standard deviation of 57.82. This means that the average user has listened to 47.45 different songs.

- **Finding #3:** The percentile distribution suggests a long tail somewhere. The 75% percentile is 55 songs and the max song plays is 4,400 songs.

- **Finding #4:** The distribution of user song counts is heavily skewed to the right. The kurtosis and skewedness statistics suggest high levels of positive skewedness and kurtosis.

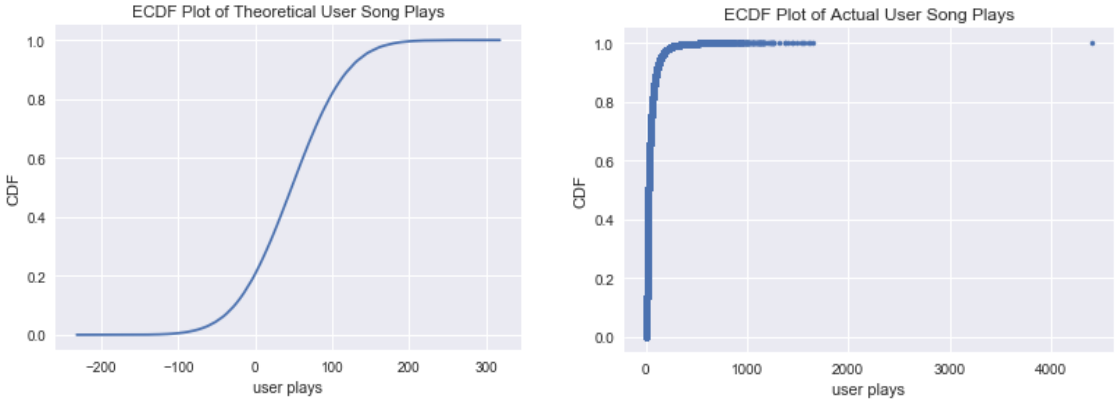
    - Skewness = 4.826
    - Kurtosis = 68.503





- **Finding #5:** Distribution of user listen is **not normally distributed**.

    - Graphical method shows that the user listen counts distribution vary significantly in shape from a theoretically normal distribution of the sample's mean and standard deviation.



    - Hypothesis testing method shows that distribution is **not normal**. The null hypothesis (H_0) is that the distribution is normal. The alternative hypothesis (H_a) is that is not normal. The testing is at the alpha level of 0.05. Since in both tests the p-value is less than 0.05 the null hypothesis is rejected and the alternative hypothesis is accepted that the listens by user distribution is not normal.

        - D'Agostino and Pearson's Normality Test[8]: stat=1059797.83, pvalue=0.0

        - Anderson-Darling Normality Test[9]: stat=105,800.29 > 0.05 critical value of 0.787

- **Finding #6:** The user listen count data satisfies the assumptions of the central limit theorem.

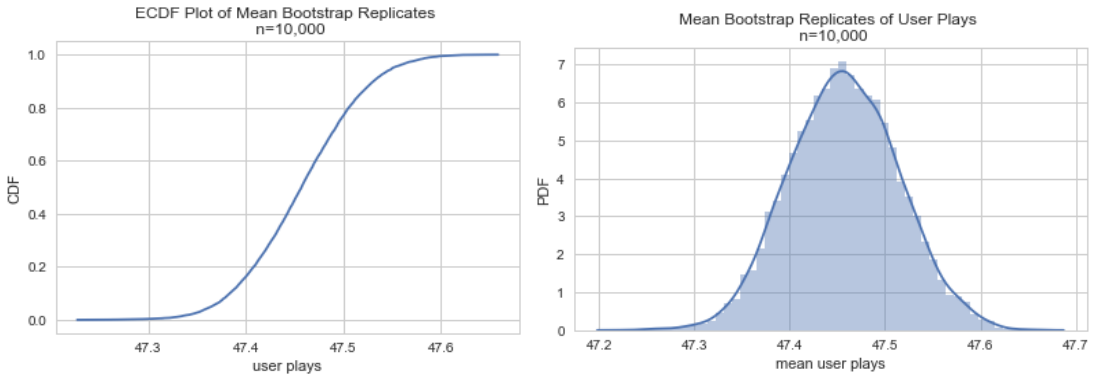
    - *Independence:* To satisfy this condition we will need to assume that the user's listen count is independent of the listen count of another user. This seems to me like a reasonable assumption. Nonetheless, it is unclear if this sample is 10% of the

[8] This function tests the null hypothesis that a sample comes from a normal distribution. It is based on D'Agostino and Pearson's, test that combines skew and kurtosis to produce an omnibus test of normality.
[9] The Anderson-Darling tests the null hypothesis that a sample is drawn from a population that follows a particular distribution. For the Anderson-Darling test, the critical values depend on which distribution is being tested against. This function works for normal, exponential, logistic, or Gumbel (Extreme Value Type I) distributions. This test has been shown to have more statistical power for testing normality.

population. The echo nest claims that this sample is a small subset of their music universe, how small is unclear. Find info here: http://blog.echonest.com/post/11992136676/taste-profiles-get-added-to-the-million-song

- *Randomness:* The Echo Nest randomly selected a sample of users whose play counts matched to the song ID's in the dataset.
- *Sample Size > 30*: The sample size is greater than 30.

- **Finding #8:** There is a 95% chance that the user listens population mean is between 47.34 - 47.57.
  - Using a bootstrap approach with 10,000 trials, the confidence interval for the user listen counts is 47.34 - 47.57. The mean replicates are normally distributed (see figures below).
  - This confidence interval is narrow and contains the sample mean of 47.45.



- **Finding #9:** There is a significant probability that the population mean is 45.47 listens. The null hypothesis (H_0) is that the population mean is 45.47 user song listens. The alternative hypothesis (H_a) is that the population mean is *not* 45.47 user song listens. The alpha level is 0.05. Since in both tests the p-value is greater than 0.05, the null hypothesis that the population mean is 45.47 user song listens is cannot be rejected.
  - Bootstrap hypothesis testing: pvalue = 0.4996
  - One sample t-test: stat=0.119, pvalue=0.905

User Listen Counts/Plays Conclusions

1. The distribution is not normal.
2. According to the bootstrap replicates of the mean and the t-test it is highly likely that the population user listen count mean is around 47.59 unique song listens per user.
3. There are several factors that could be driving these results and creating a heavily right skewed user listen count distribution:
    - The user play dataset is very large.
    - The data was collected in a certain timeframe and procedure and we do not have full user listening history.
    - The song data was extracted in December of 2011 which is holiday season, which could affect the user listen distribution (since only those who matched with the song dataset are included).
    - There are specific characteristics of users with high user-song interactions (such as business customer, several people in the same account etc.).
    - The structure of the music industry, in which a few songs/artists create hits and most working artists do not.

Part 3. User Listen Counts per Song

User Listens by Songs Descriptive Statistics								
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
play	384,546	125.79	799.03	1.00	4.00	13.00	52.00	110,479
play_count	384,546	360.63	3,256.81	1.00	8.00	32.00	133.00	726,885

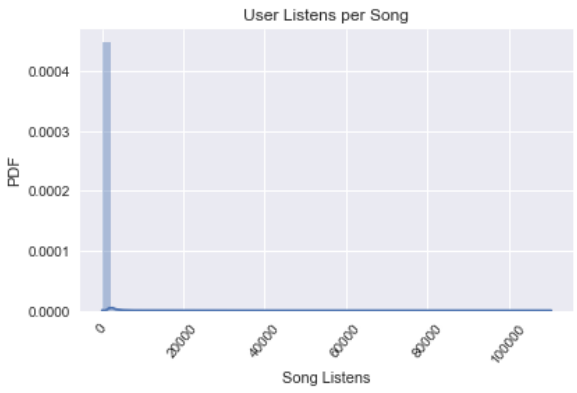
- **Finding #1:** There are 384,546 unique songs that have been listened to. This means that 42% of the song catalog has been listened to and 58% has not been listened to at all.

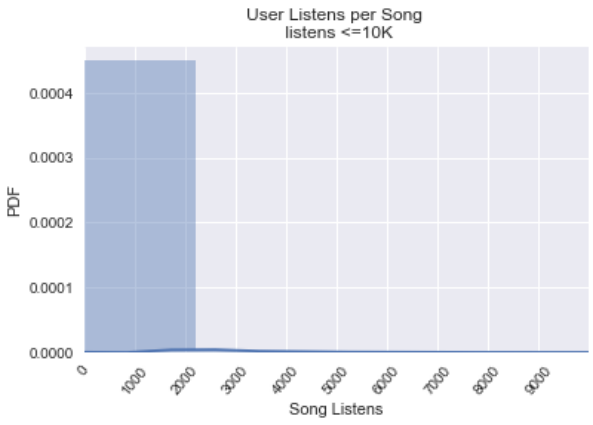
- **Finding #2:** As shown in the previous section, at most a user as listed to 4,400 unique songs, and the average user has listened to 47.45 different songs. We can conclude that the song catalog and user listens are sparsely distributed (most users have listened to a small portion of the available catalog).

- **Finding #3:** The play indicator has a mean of 125.79 and a standard deviation of 799. This means that the average song has 125.79 unique user listens.

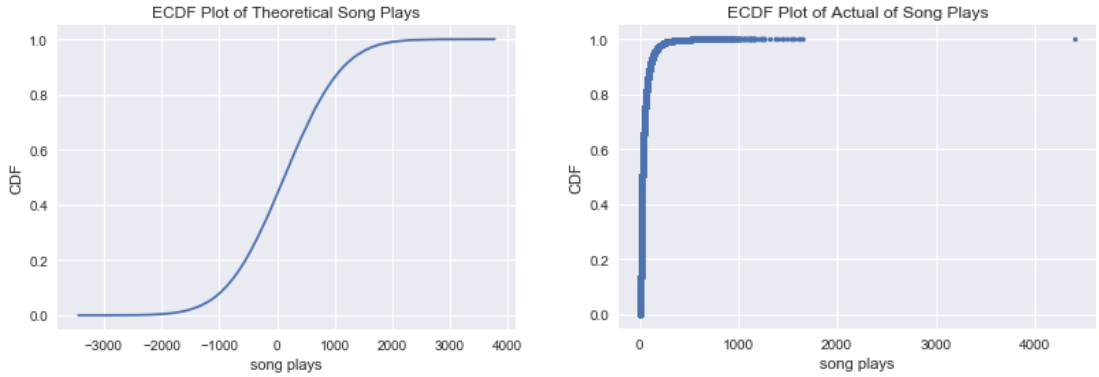
- **Finding #4:** The percentile distribution suggests a long tail somewhere. The 75% percentile is 52 songs and the max song plays is 110,479 unique user listens.

    - Skewness = 46.11
    - Kurtosis = 3780.29





- **Finding #5:** Distribution of user listens per song is **not normally distributed**.



  - Graphical method shows that the user song listens per track distribution vary significantly in shape from a theoretically normal distribution of the sample's mean and standard deviation.

  - Hypothesis testing method shows that distribution is **not normal**. The null hypothesis (H_0) is that the distribution is normal. The alternative hypothesis (H_a) is that is not normal. The testing is at the alpha level of 0.05. Since in both tests the p-value is less than 0.05, the null hypothesis is rejected and the alternative hypothesis is accepted that the listens by user distribution is not normal.

    - D'Agostino and Pearson's Normality Test[10]: stat=1176273.18, pvalue=0.0

    - Anderson-Darling Normality Test[11]: stat= 110035.21 > 0.05 critical value of 0.787

- **Finding #6:** It is suspect that the distribution satisfies the conditions of the central limit theorem.
  - *Independence:* To satisfy this condition we will need to assume that a song play count is independent of the play count of another song. In this case, I think this may not be not a reasonable assumption to make due to popularity, artists similarity,

[10] This function tests the null hypothesis that a sample comes from a normal distribution. It is based on D'Agostino and Pearson's, test that combines skew and kurtosis to produce an omnibus test of normality.
[11] The Anderson-Darling tests the null hypothesis that a sample is drawn from a population that follows a particular distribution. For the Anderson-Darling test, the critical values depend on which distribution is being tested against. This function works for normal, exponential, logistic, or Gumbel (Extreme Value Type I) distributions. This test has been shown to have more statistical power for testing normality.

song similarity and how the echo nest service decided to show the catalog to their users. Also, it is unclear if this sample is 10% of the population.

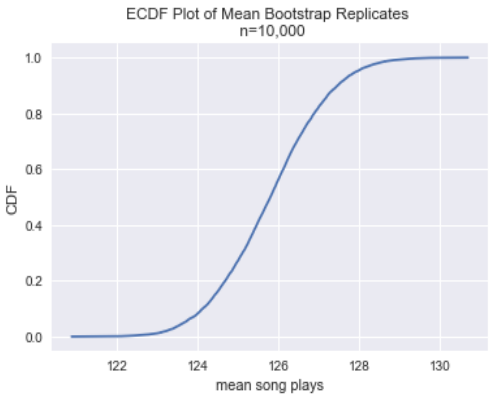
- *Randomness:* The Echo Nest randomly selected a sample of users whose play counts matched to the song ID's in the dataset.

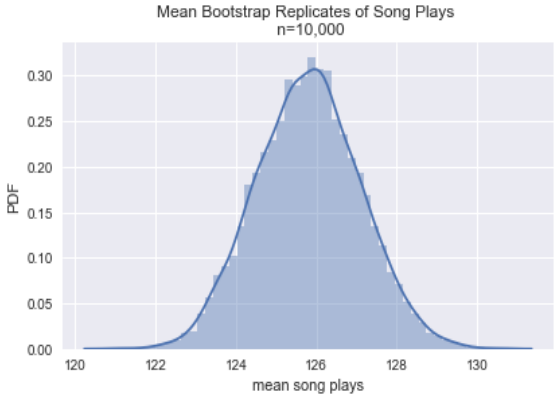
- *Sample Size > 30:* The sample size is greater than 30.

– **Finding #8:** There is a 95% chance that the user listens population mean is between 47.34 - 47.57.

  - Using a bootstrap approach with 10,000 trials, the confidence interval for the user listens per song is 123.31- 128.34. The mean replicates are normally distributed (see figures below).

  - This confidence interval contains the sample mean of 125.79.





– **Finding #9:** There is a significant probability that the population mean is 45.47 listens. The null hypothesis (H_0) is that the population mean is 45.47 user song listens. The alternative hypothesis (H_a) is that the population mean is *not* 45.47 user song listens. The alpha level of 0.05. Since in both tests the p-value is greater than 0.05, the null hypothesis that the population mean is 125.79 user song listens cannot be rejected. The t-test should be taken with some suspicion since this distribution may not satisfy the central limit theorem.

  - Bootstrap hypothesis testing: pvalue = 0.496

  - One sample t-test: stat=0.003, pvalue= 0.997

User Listens/Plays per Song Conclusions

1- The distribution is not normal.
2- According to the bootstrap replicates of the mean and the t-test it is highly likely that the population user listen count by song mean is around 125.79. The t-test should be taken with some suspicion since this distribution may not satisfy the central limit theorem.
3- There are several factors that could be driving these results and creating a heavily right skewed user listen count distribution:
    a. The user play dataset is very large.
    b. The data was in a certain timeframe and procedure and we do not have full user listening history.
    c. The song data was extracted in December of 2011 which is holiday season, which could affect the user song play behavior.
    d. There are specific characteristics of the user with high user play interactions (such as business customer, several people in the same account etc.).
    e. The structure of the music industry, in which a few songs/artists create hits and most working artists do not.

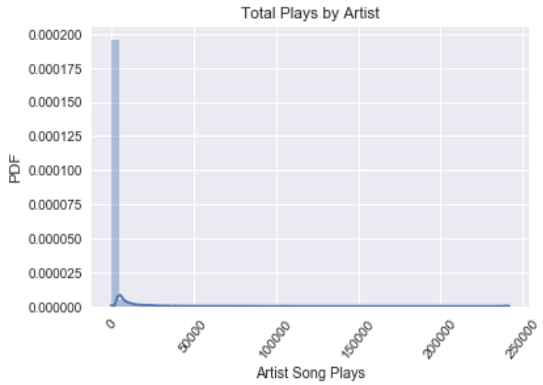
Part 4. User Listen Counts per Artist

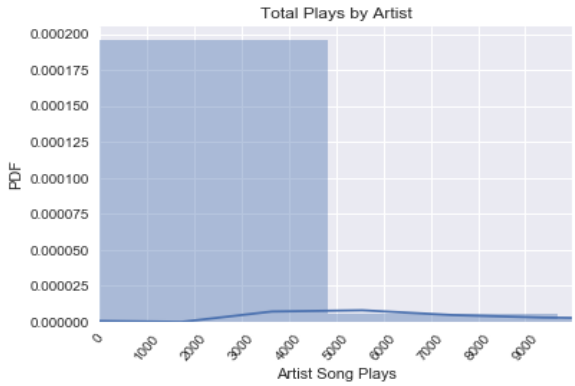
User Listens by Artist Descriptive Statistics								
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
play	29,559.0	1,367.44	6,498.71	1.00	20.00	112.00	569.50	241,823.00
play_count	29,559.0	3,895.82	18,704.71	1.00	50.00	323.00	1,699.00	884,464.00

- **Finding #1:** There are 29,559 unique artists that have been listened to. This means that 66.5% of artists in the song dataset has been listened to, and 33.5% artists have not been listened to at all.

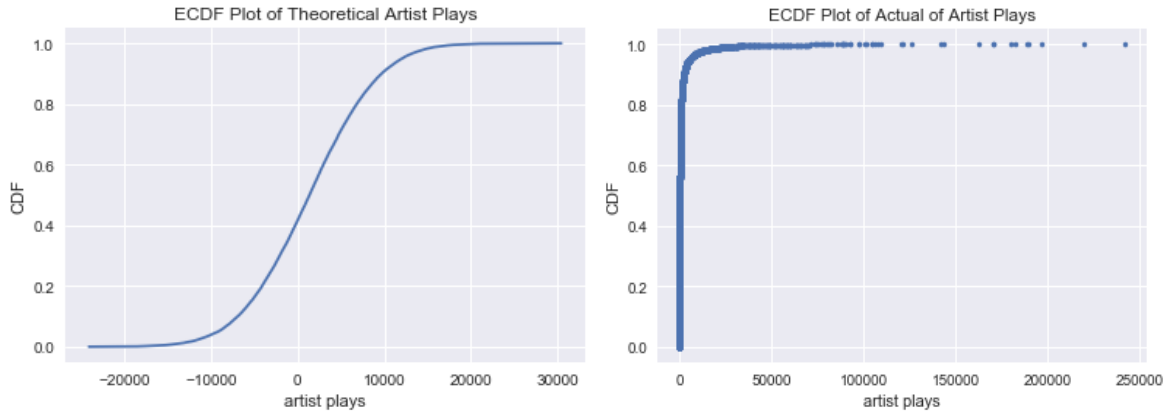
- **Finding #2:** The user listen indicator by artist has a mean of 1,367.44 and a standard deviation of 6,498.71. This means that the average artist has 1,367.44 unique user listens.

- **Finding #3:** The percentile distribution suggests a long tail somewhere. The 75% percentile is 569.50 songs and the max song plays is 241,823 unique user listens.





- **Finding #4:** Distribution of user listens per artist is **not normally distributed**.
    - Skewness = 15.13
    - Kurtosis = 344.37



- Graphical method shows that the user listens by artist distribution vary significantly in shape from a theoretically normal distribution of the sample's mean and standard deviation.

- Hypothesis testing method shows that distribution is **not normal**. The null hypothesis (H_0) is that the distribution is normal. The alternative hypothesis (H_a) is that is not normal. The testing is at the alpha level of 0.05. Since in both tests the p-value is less than 0.05, the null hypothesis is rejected and the alternative hypothesis is accepted that the listens by user distribution is not normal.

  - D'Agostino and Pearson's Normality Test[12]: stat= 56969.63, pvalue=0.0

  - Anderson-Darling Normality Test[13]: stat= 8001.32 > 0.05 critical value of 0.787

– **Finding #6:** It is suspect that the distribution satisfies the conditions of the central limit theorem.
  - *Independence:* To satisfy this condition we will need to assume that an artist play count is independent of the play count of another artist play count. In this case, I think this may not be not a reasonable assumption to make due to popularity effects, artists similarity, song similarity and how the echo nest service decided to show the catalog to their users. However, it may be a reasonable assumption that this sample represents <10% of the artist population.

[12] This function tests the null hypothesis that a sample comes from a normal distribution. It is based on D'Agostino and Pearson's, test that combines skew and kurtosis to produce an omnibus test of normality.
[13] The Anderson-Darling tests the null hypothesis that a sample is drawn from a population that follows a particular distribution. For the Anderson-Darling test, the critical values depend on which distribution is being tested against. This function works for normal, exponential, logistic, or Gumbel (Extreme Value Type I) distributions. This test has been shown to have more statistical power for testing normality.

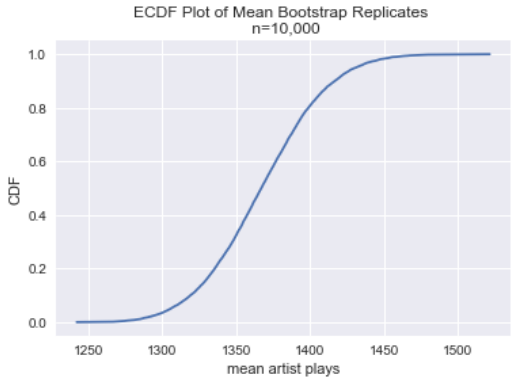
- *Randomness:* The Echo Nest randomly selected a sample of users whose play counts matched to the song ID's in the dataset.

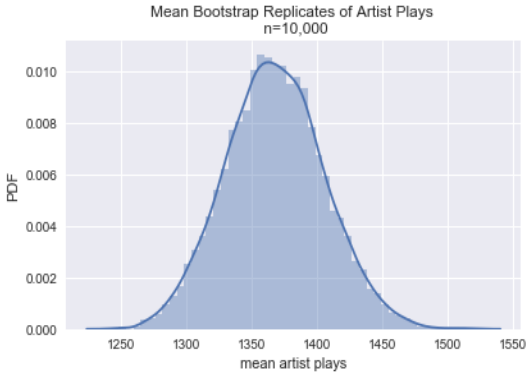
- *Sample Size > 30: The sample size is greater than 30.*

- **Finding #7:** There is a 95% chance that the artists listens population mean is between 1294.93- 1443.78.

    - Using a bootstrap approach with 10,000 trials, the confidence interval for the user listens by artist is 1294.93- 1443.78. The mean replicates are normally distributed (see figures below).

    - This confidence interval contains the sample mean of 1,367.44.





- **Finding #9:** There is a significant probability that the population mean is 1,367.44 user listens per artist. The null hypothesis (H_0) is that the population mean is 1,367.44 user listens per artist. The alternative hypothesis (H_a) is that the population mean is *not* 1,367.44 user listens per artist. The alpha level of 0.05. Since in both tests the p-value is greater 0.05, the null hypothesis that the population mean is not 1,367.44 user listens per artists listens is cannot be rejected. The t-test should be taken with some suspicion since this distribution may not satisfy the central limit theorem.

    - Bootstrap hypothesis testing: pvalue = 0.5033

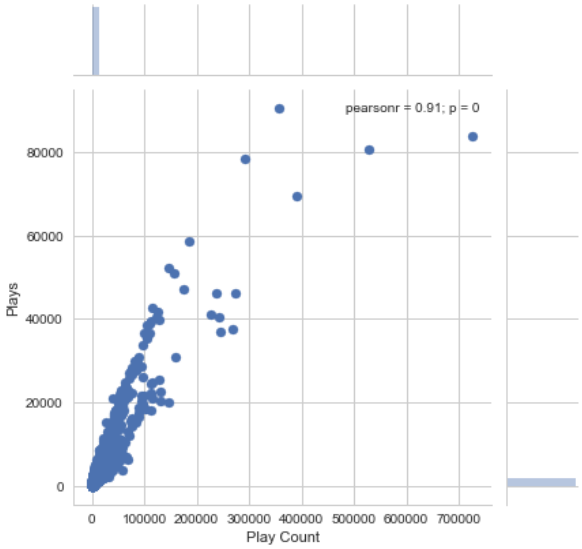
    - One sample t-test: stat= -4.202, pvalue= 0.999

User Listens per Artist Conclusions

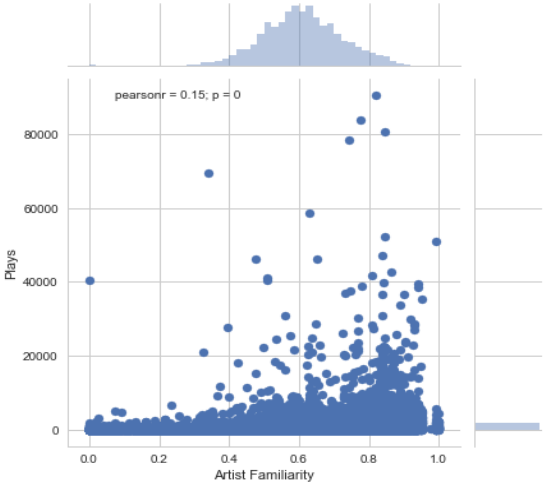
1- The distribution is not normal.
2- According to the bootstrap replicates of the mean and the t-test it is highly likely that the population user listen count per artist mean is around 1,367.44. The t-test should be taken with some suspicion since this distribution may not satisfy the central limit theorem.
3- There are several factors that could be driving these results and creating a heavily right skewed user listen per artist distribution:
    a. The user play dataset is very large.
    b. The data was in a certain timeframe and procedure and we do not have full user listening history.
    c. The song data was extracted in December of 2011 which is holiday season, which could affect the user song play behavior.
    d. There are specific characteristics of the user with high user play interactions (such as business customer, several people in the same account etc.).
    e. The structure of the music industry, in which a few songs/artists create hits and most working artists do not.

Part 5. Relationships between basic song features and song listens

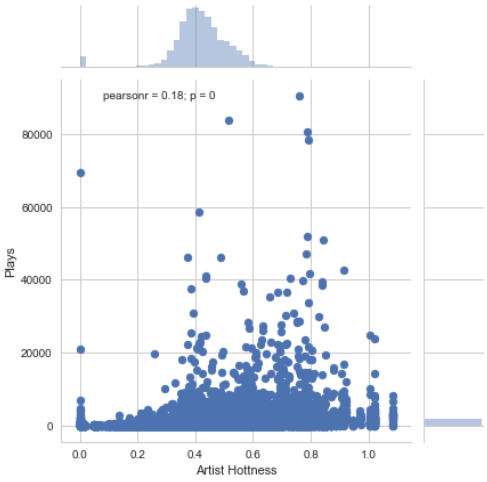
- There does not seem to be a strong relationship between unique song plays/interactions and any of the basic song features.
- The lack of relationship may be due to the sparsity of the data. For example, most users having not listened to that much of the catalog to result in meaningful patterns.
- The lack of relationship may also be due to the fact that many of the songs may be too heterogeneous or different.
- They may be patterns or correlations if features are looked at in combination rather than isolated.
- There seems to be a strong linear relationship between play count and plays. This means that there is a tendency in that the higher the play counts, the higher the number of unique user plays. This makes sense and ought to be expected given that the play indicator is derived from the play count.
- The high person correlation coefficient value of the relationship between user plays and user play counts leads me to think that song/artist popularity maybe an important factor in song/artist/user interactions (i.e. the more people hear about a song/artist the more likely they are to interact with it and have play it more).



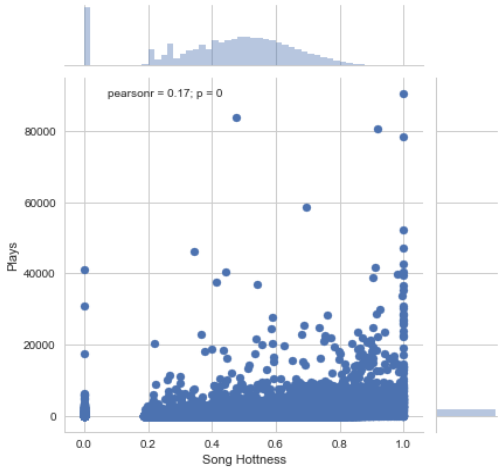
Play Count and Plays



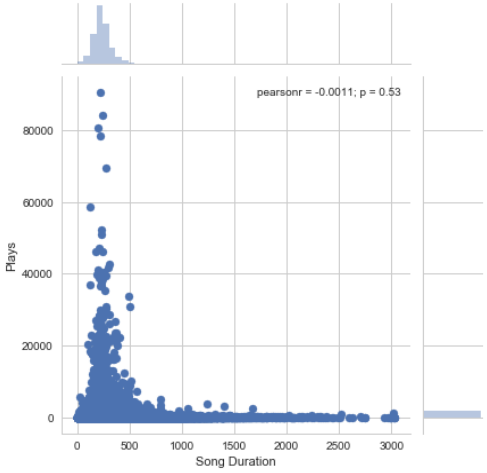
Artist Familiarity



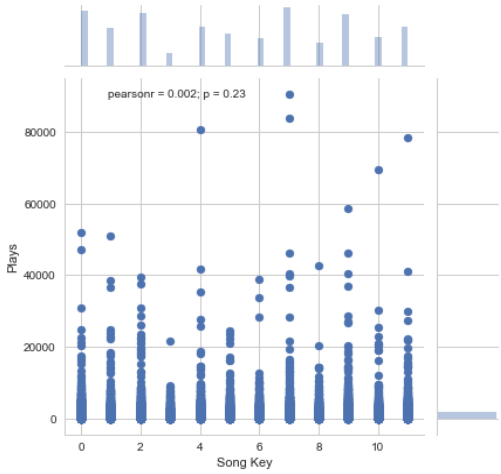
Artist Hotness



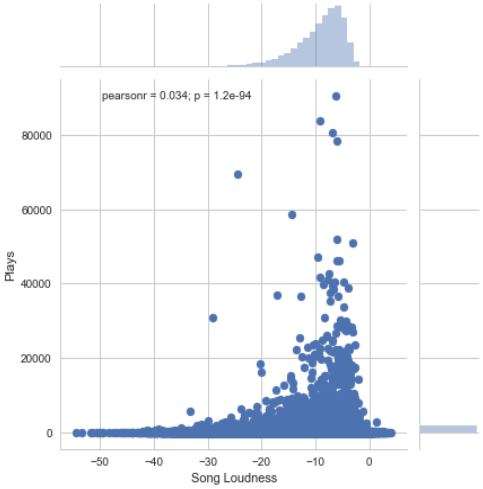
Song Hotness



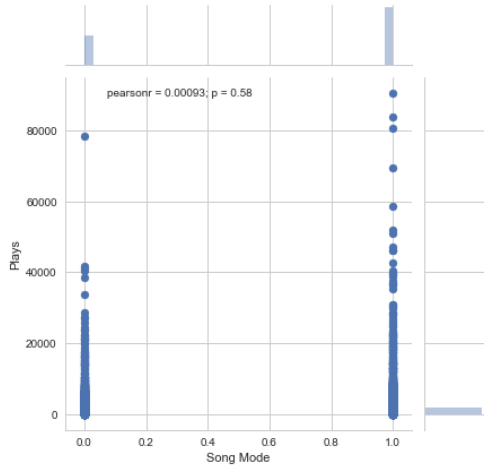
Song Duration



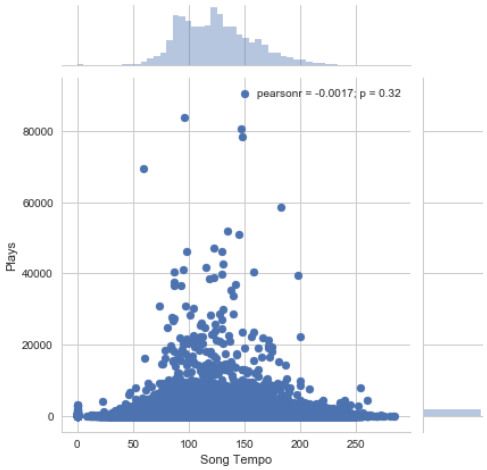
Song Key



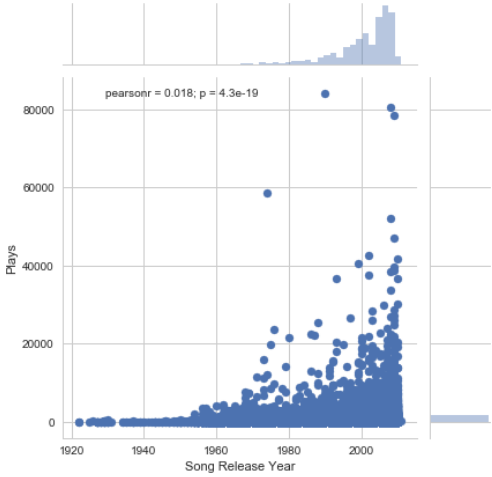
Song Loudness



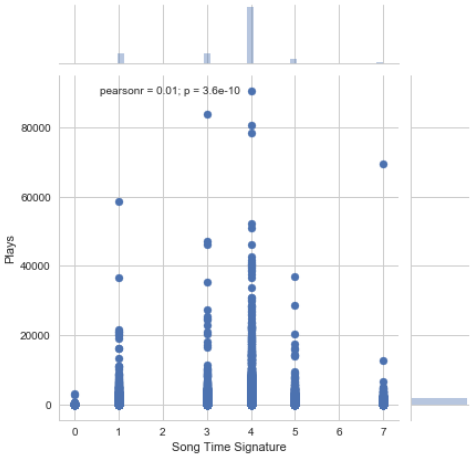
Song Mode



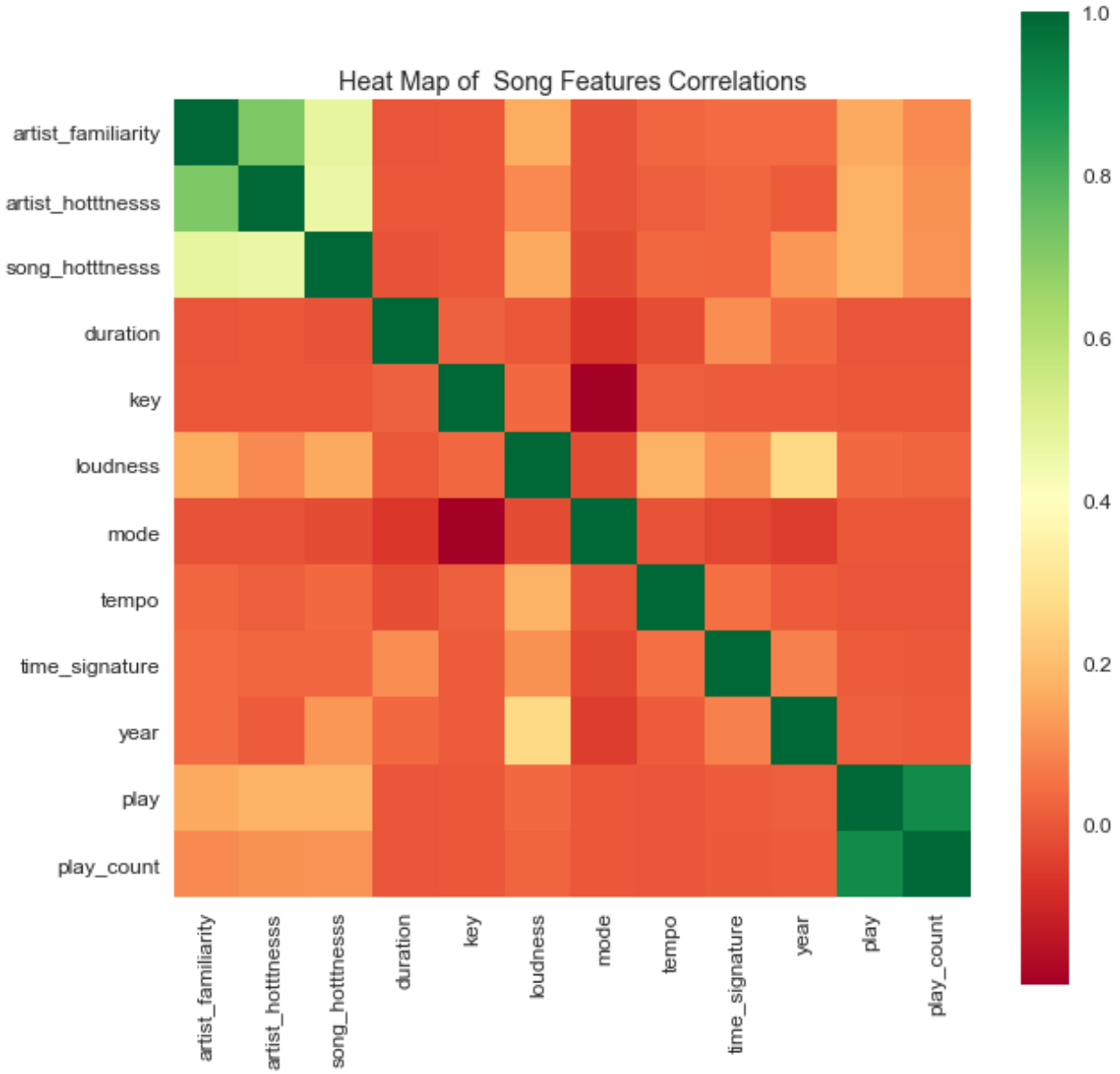
Song Tempo



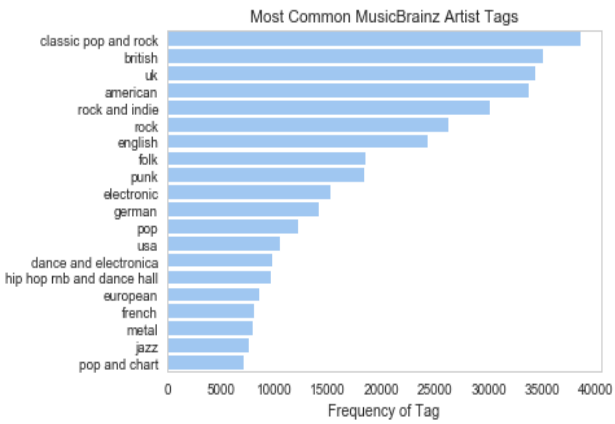
Song Release Year

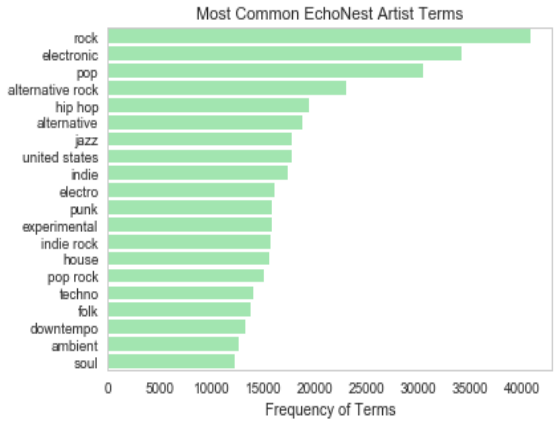


Time Signature


Heat Map of Song Features Correlations
artist_familiarity
artist_hotttnesss
song_hotttnesss
duration
key
loudness
mode
tempo
time_signature
year
play
play_count
1.0
0.8
0.6
0.4
0.2
0.0

Part 6. Artist Tags and Terms





- **MusicBrainz Tags**

    - There is a total of 2,321 unique MusicBrainz tags.
    - The average tag has ~344 unique artists associated with it.
    - There seems to be skewedness, 75% of the tags have 104 artists while the maximum artists associated with a tag is ~38.5K.
    - As discussed previously there are 44,421 unique artists in the song dataset. So this means that 98% of artists in the MSD dataset has one or more MusicBrainz tag.
    - Classic pop, rock, british, uk, american indie, english and folk are the most popular artists tags in the MusicBrainz data.

- **EchoNest Terms**

    - There is a total of 7,643 unique terms.
    - The average tag has ~212 unique artists associated with it.
    - There seems to be some skewedness, 75% of the tags have 32 unique artists while the maximum artists associated with a tag is ~40.8K.
    - As discussed previously, there are 44,421 unique artists in the song dataset. unique artists in the dataset. So this means that 98% of artists in the MSD dataset has one or more EchoNest tag.
    - The Echo Nest has almost 3x more unique tags than the Music Brainz array.
    - Rock, electronic, pop, hip hop, alternative and jazz the most popular artists terms in the Echo Nest data. Rock intersects with the top tags of the MusicBrainz.

Recommendation Algorithms Implementation

This section details the implementation of four recommendation algorithms: (1) song popularity, (2) artist popularity, (3) item-based collaborative filtering and, (4) ranking-matrix factorization. The popularity based recommenders will be used as 'benchmark' or 'point of reference' for the item-based collaborative filtering and ranking matrix factorization. The entirety of the recommendation code is implemented in python.

A. Python Libraries Used

To implement the recommender I used a python library rather than building a recommender algorithm from the ground up. The library used to implement the recommenders was Turi's GraphLab recommendation module.

It is worth noting that this is not the only choice of python libraries for building recommendation models and algorithms. Other options include Surprise built on top of ScikitLearn and ApacheSpark among others. I used GraphLab because of its higher computational efficiency, mainly driven by the fact that the size of the observations of the datasets that will be used is large (songs 905K+, user 31M+).

Complementary libraries used to aid implementation were Pandas, Matlplotlib, Seaborn, Numpy, and SciPy Moreover, the library used to implement the recommendation algorithms is GraphLab.

B. Data Preparation and Preprocessing

Data Upload

- The previously wrangled and cleaned song and user data CSVs are saved in the local machine. These files were uploaded and converted to two pandas dataframes.

Data Preparation

- **User Data:** Computational resources were scarce and it is not possible to use all the users to train and test the recommender algorithms due to lack of sufficient memory resources. For this reason, I filtered out users who: (1) have less than 45 unique song listens and, (2) users who have more than 452 song unique song listens.

  In short, I filtered out users who were at either extreme of the unique song listen distribution. After the filtering process 313,689 out of 1.019M users remained.

Moreover, 65.6% of the available user listening data was preserved for training and testing.

- **Song Data:** To reduce the memory cost of this dataset, I only maintained columns that directly

Data Pre-Processing

- **Conversion to SFrames:** After data preparation the pandas daframes of the user and song listening data were converted to GraphLab's SFrame format. The GraphLab's recommendation library only works with data in SFrames and SArrays formats.

- **Random Shuffle:** It is good practice to randomly shuffle the data before Both the user and song data rows were randomly shuffled using a random seed of 42. This will be the same random seed throughout the implementation.

- **User Data Train and Test Splits:** The train-test split was based on users. This means that the train sample retained some information for all users, and the train sample contains a subset of user information which the trains sample does not contains. This is to test how well the recommender performs in the case it already knows something already about the user.

  If user data rows are split at completely at random a scenario can occur in which we evaluate the recommender on recommendations of users it does no know anything about (i.e. cold start problem). The user songs that are part of the train-test splits are randomly chosen. Two train-test split samples were created: one of 80% train and 20% test, and a second one of 90% train and 10% test.

  Further, for the artist popularity recommender a user dataset that contains the song's artist was created. The same process for train and test splitting was used for this dataset.

C. Recommendation Algorithms

Song Popularity

The song popularity recommender scores songs based on the number of times they are seen in the dataset. The recommender returns the songs that are the most popular to all users and no personalization occurs. This type of recommender can be either used as a baseline or “background” model for new users.

Two recommenders of this type were built, one using the user data 80-20 test-train split and another using the 90-10 train test split. This is to ascertain if significant differences in performance occurs based on the amount of data inputted. Metadata about the songs was incorporated. Training time was small. The training time for the 80% training sample was 0.177s, and for the 90% training sample training time was 0.2021s.
Once trained, the artist popularity recommenders were tested using the test samples described above. Evaluation precision and recall at 18 different cutoff points (number of recommendations made) were calculated. The trained recommender and evaluation results was stored in the local machine.

Artist Popularity

The artist popularity recommender scores artists based on the number of times they were seen in the dataset. The recommender returns the artists that are the most popular to all users and no personalization occurs. This type of recommender can be used as a baseline or “background” model for new users.

Two recommenders of this type were built, one using the user data 80-20 test-train split and another using the 90-10 train test split. This is to ascertain if significant differences in performance occurs based on the amount of data inputted. Metadata about the songs was incorporated. Training time was small. The training time for the 80% training sample was 0.0214s, and for the 90% training sample training time was 0.0175s.

Once trained, the artist popularity recommenders were tested using the test samples described above. Evaluation precision and recall at 18 different cutoff points (number of recommendations made) were calculated. The trained recommender and evaluation results was stored in the local machine.

Item-Based Collaborative Filtering[14]

The item-based collaborative filtering recommender makes recommendations to users by identifying similar songs that were listened by similar users. It recommends similar songs that users have listened to, but that the target user has not necessarily listened to.

More formally, this model first computes the similarity between items using the observations of users who have interacted with both items. Given a similarity between item i and j, S(i,j), it scores an item j for user u using a weighted average of the user's previous observations Iu.[15]

Two recommenders of this type were built, one using the user data 80-20 test-train split and another using the 90-10 train test split. This is to ascertain if significant differences in performance occurs based on the amount of data inputted. Information about the songs was not incorporated. The recommender parameters were as follows:

- Similarity type: Jaccard
- Number of Similar Items[16]: 64
- Nearest neighbor's interaction proportion[17]: 0.001

In this recommender, I used the Jaccard similarity. Generally, Jaccard is a good choice when the data consists of implicit feedback. Other options are cosine and pearson correlation, however these tend to be better suited for explicit feedback situations (where there the items where rated by users).

The Jaccard similarity is computed as follows:

$$JS(i,j) = \frac{|Ui \cap Uj|}{|Ui \cup Uj|}$$

Where U are users who listened to song i.

Training time was substantial. The training time for the 80% training sample was one hour and twenty minutes (1hr 20 min), and for the 90% training sample training time was thirty-seven minutes (37 min).

14

https://turi.com/products/create/docs/generated/graphlab.recommender.item_similarity_recommender.ItemSimilarityRecommender.html#graphlab.recommender.item_similarity_recommender.ItemSimilarityRecommender

15 https://turi.com/products/create/docs/generated/graphlab.recommender.item_similarity_recommender.ItemSimilarityRecommender.html

16 Number of similar items to find and store for each item. Decreasing this decreases the amount of memory required for the model, but may also decrease the accuracy.

17 Any item that has was rated by more than this proportion of users is treated by doing a nearest neighbors search. For frequent items, this is almost always faster, but it is slower for infrequent items. Furthermore, decreasing this causes more items to be processed using the nearest neighbor path, which may decrease memory requirements.

Once trained, the item-based collaborative filtering recommenders were tested using the test samples described above. Evaluation precision and recall at 18 different cutoff points (number of recommendations made) was calculated. The trained recommender and evaluation results were stored in the local machine.

Ranking Matrix Factorization[18]

The ranking matrix factorization algorithm learns latent factors[19] for each user by decomposing the user-song interactions. The decomposed matrix is them used to rank recommended items according to the probability of observing the same user, item pairs.

More formally, this type of model predicts a score for each possible combination of users and items. The internal coefficients of the model are learned from known scores of users and items. Recommendations are then based on these scores. Users and songs are represented by weights and factors. The factor terms model interactions between users and items.

For example, if a user tends to love romance movies and hate action movies, the factor terms attempt to capture that, causing the model to predict lower scores for action movies and higher scores for romance movies. Weights and factors can be controlled by regularization terms.

The predicted score for user *I* on song *j* is given by:

$$score(i,j) = \mu + w_i + w_j + + a^T x_i + b^T y_j + u_i^T v_j$$

where μ is a global bias term, w_i is the weight term for user I, w_j is the weight term for item j, x_i and y_j are respectively the user and item side feature vectors, and a and b are respectively the weight vectors for those side features. The latent factors, which are vectors of length number of factors, are given by u_i and v_j.

Two recommenders of this type were built, one using the user data 80-20 test-train split and another using the 90-10 train test split. This is to ascertain if significant differences in performance occurs based on the amount of data inputted. Information about the songs was not incorporated. The recommender parameters were as follows:

This type of recommender has several parameters to be set and several tailoring options. In this application, I have used the following parameters:

[18]https://turi.com/products/create/docs/generated/graphlab.recommender.ranking_factorization_recommender.RankingFactorizationRecommender.html#graphlab.recommender.ranking_factorization_recommender.RankingFactorizationRecommender

[19] The assumption that there are hidden or unobservable factors that impact similarity between users. The decomposition of the matrix seeks to quantify these 'hidden' factors.

- Regularization = 1e-9
- Linear Regularization = 1e-9: 64
- Number of Latent Factors = 32
- Iterations = 25
- Solver: Implicit Alternating Least Squares

Regularization terms are important for matrix factorization recommenders. The regularization term for interaction terms used here is 1e-9. A typical value is for this parameter is between 1e-12 and 1. A value of zero tends to cause computational issues. The linear regularization term used as 1e-9 as well. The values for this term also tend to range between 1e-12 and 1. Regularization terms are used to control the magnitude of the user-factor and item-factor terms to prevent overfitting.

The solver used to solve the equation above was Implicit Alternating Least Squares.[20] Other options include: Adaptive Gradient Stochastic Gradient Descent and Stochastic Gradient Descent.

Training time was moderate. Training time for the 80% training sample was seventeen (17 min), and for the 90% training sample training time was seventeen minutes (17 min) as well.

Once trained, the item-based collaborative filtering recommenders were tested using the test samples described above. Evaluation precision and recall at 18 different cutoff points (number of recommendations made) was calculated. The trained recommender and evaluation results were stored in the local machine.

[20] For more information on Implicit Alternating Least Squares: http://yifanhu.net/PUB/cf.pdf

D. Evaluation

I will evaluate the recommenders using recall. I am prioritizing recall as the evaluation measure, because as the business problem is framed, the concerned is returning relevant results to the user. Moreover, given that we are working with implicit data and not ratings or label classification precision can take a somewhat ambiguous nature.

Further, given the nature of implicit data and the recommenders used, other metrics such as RMSE do not make much sense in this application. If we return something that the user has 'listened' to does that mean that we are being precise in this application?

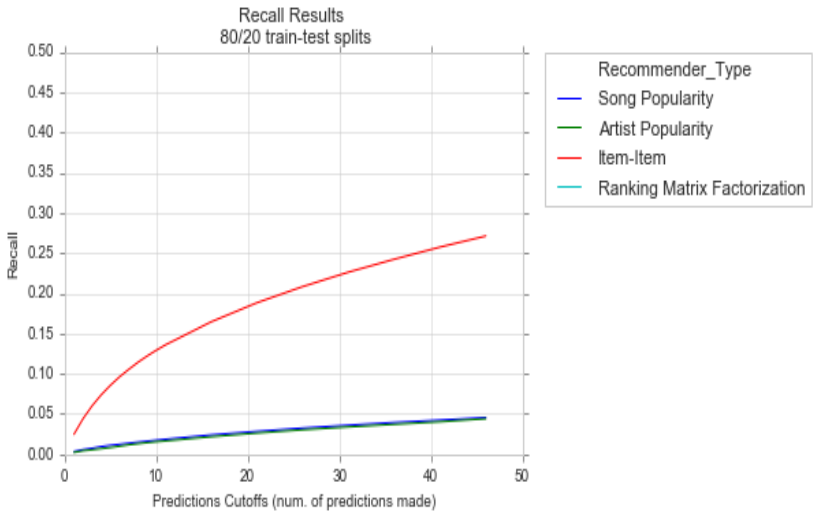
- **Precision (P)** is defined as the number of true positives (TP) over the number of true positives plus the number of false positives (FP). Precision the fraction of relevant instances among the retrieved instances.

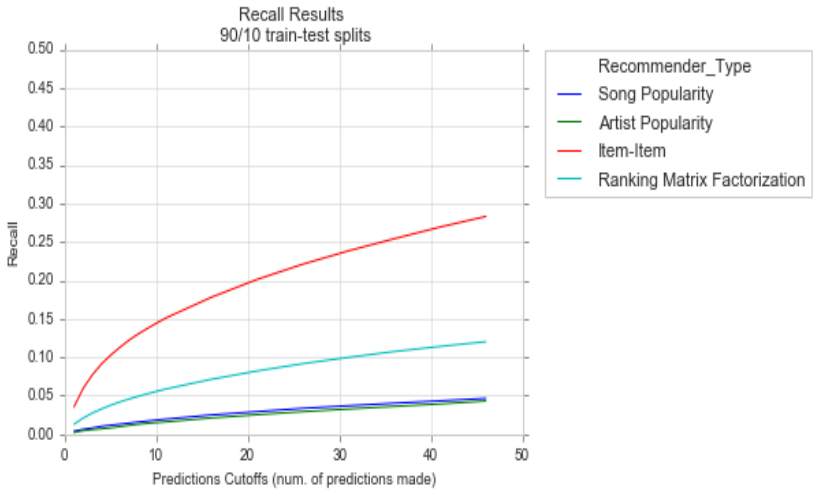
$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

- **Recall (R)** is defined as the number of true positives (TP) over the number of true positives plus the number of false negatives (FN). Fraction of relevant instances that have been retrieved over the total number of relevant instances.

$$Recall = \frac{True\ Positives}{True\ Positives + Negatives}$$

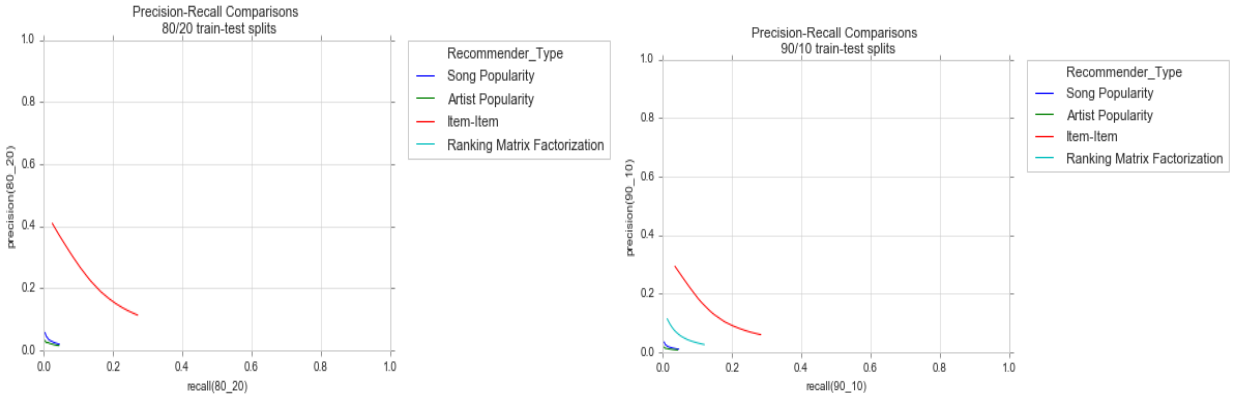
- Both precision and recall are based on an understanding and measure of relevance.
-





Evaluation Findings

- None of the recommenders recall performed better than 0.50. This means, on the whole they are not performing better than random.
- Given the performance results, there are no signs of overfitting.
- Item-based collaborative filtering recall performed significantly better than matrix factorization, song popularity and artist popularity recommenders.
- Both of the personalized recommenders performed better than popularity based recommenders.
- Recall performance was similar for the 80-20 and 90-10 train test samples.
- Recall performance increases with number of predictions made.



E. Limitations

Computational Resources

Computational resources were limited. Resources were not sufficient to implement the following:

(1) Content-Based Recommender: This was attempted and training would have been significant on this dataset even when not all song features were included. Given the fact that we have a lot of data on the songs and much less on the users, this may be a dataset that could benefit from a content-based strategy. The downside of content-based recommender is that they can give recommendations that are ‘too obvious’ or ‘too similar’, thus not encouraging variety in recommendations.

(2) Five-Fold Cross-Validation: Computational resources were not sufficient to implementing the five different times. On the other hand, given the evaluation results there is no reason to be concerned about overfitting at this point.

(3) Hyper-parameter tuning: The matrix factorization recommender and to a lesser extent has several items collaborative filtering has several parameters that can be set or modified to improve performance. In particular, the matrix factorization models are particularly sensitive to the regularization terms. I was unable to run several models with different parameter options and combinations and compare their performance.

Evaluation Limited Usefulness

While the recommender was evaluated on precision and recall, there is several limitations to its usefulness.

(1) Given that the data is implicit, a user listening to a song does not mean they necessarily liked it or that is their preferred genre. Moreover, we have no information on whether the ‘listening’ is skip or not. Thus, if we have users we a lot of skips it means that in reality their matrix is even more sparse than the data suggests and their preferences cannot be learned as easily.

(2) Music listening can be idiosyncratic. Users listenining can be driven by mood or context. For example, I feel like listening something slow or fast. Another example, music that I listen in a party or with friends vs. listening on my own.

Other Options

Not all options for recommendation they may work for the business problem at hand was explored and tested. In the next section, I will elaborate on other strategies that many work and areas for further work.

F. Recommendations and Next Steps

- Explore a content-based strategy. This may be beneficial given the large amounts of information known about the songs.

- The Ranking Matrix Factorization algorithm may have room for improvement in particular using parameter tuning, specially regularization terms and number of latent factors.

- Include song metadata information in the matrix factorization algorithm.

- Instead of ranking matrix factorization, it is possible to use a matrix factorization algorithm that treats the interactions as binary (0,1), and then implements a logistic regression to predict their binary rating.

- Use a mix of methods. In a recommendation context, generally a mix of methods tends to outperform a single recommender. This is because we can exploit the strengths of each recommender and exploit different aspects of the data with each recommender.

- Clustering is another option that maybe worth exploring. However, given the sparsity and heavy skewness of the data, it is a question of whether meaningful and predictive user clusters could be achieved.

- Do more investigation of user listening behavior. As mentioned earlier, it is not necessarily true that if someone has ‘listened’ to a song they necessarily like it. Also, if there if many of the listening are actually ‘skips’ that could have a large impact on the performance of the recommender.

- Benchmark performance. In this implementation, I have used recall as the evaluation metric, however due to the limitations of the data and idiosyncratic nature music listening could take, it may be fruitful to benchmark with other music recommendation systems performance to gain a sense of what is reasonable in this context.

Sources

1- Melville P., Sindhwani V. (2017) Recommender Systems. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning and Data Mining. Springer, Boston, MA

2- Felfernig A., Jeran M., Ninaus G., Reinfrank F., Reiterer S., Stettinger M. (2014) Basic Approaches in Recommendation Systems. In: Robillard M., Maalej W., Walker R., Zimmermann T. (eds) Recommendation Systems in Software Engineering. Springer, Berlin, Heidelberg

3- Aggarwal, Charu C. Recommender Systems. Chaper 1 (pg 1-28) ISBN 978-3-319-29659-3

4- Stanford Info Lab. Chapter 9: Recommendation Systems. http://infolab.stanford.edu/~ullman/mmds/ch9.pdf

5- McFee, Brian, Lanckriet, Gret. LARGE-SCALE MUSIC SIMILARITY SEARCH WITH SPATIAL TREES. https://bmcfee.github.io/papers/ismir2011_sptree.pdf

6- McFee, Brian, Lanckriet, Gret, Ellis, Daniel, Bertin-Mahieux. The Million Somg Dataset Challenge. http://eceweb.ucsd.edu/~gert/papers/msdc.pdf

7- Ellis, Daniel, Thierry Bertin-Mahieux. The Million Somg Dataset Challenge. http://eceweb.ucsd.edu/~gert/papers/msdc.pdf

8- Asanov, Daniar. (2018). Algorithms and Methods in Recommender Systems. https://www.snet.tu-berlin.de/fileadmin/fg220/courses/SS11/snet-project/recommender-systems_asanov.pdf

9- Khusro, Shah & Ali, Zafar & Ullah, Irfan. (2016). Recommender Systems: Issues, Challenges, and Research Opportunities. 1179-1189. 10.1007/978-981-10-0557-2_112.

10- Deep Learning based Recommender System: A Survey and New Perspectives. Zhang Shuai, Yao Lina, Sun Aixin. https://arxiv.org/pdf/1707.07435.pdf

11- B. Akay, O. Kaynar and F. Demirkoparan, "Deep learning based recommender systems," *2017 International Conference on Computer Science and Engineering (UBMK)*, Antalya, 2017, pp. 645-648. doi: 10.1109/UBMK.2017.8093489

12- GraphLab documentation: https://turi.com/products/create/docs/graphlab.toolkits.recommender.html
13- ApacheSpark: https://spark.apache.org/docs/2.3.0/mllib-collaborative-filtering.html
14- Surprise: http://surpriselib.com/
15- https://www.businessinsider.com/netflixs-recommendation-engine-drives-75-of-viewership-2012-4
16- http://rejoiner.com/resources/amazon-recommendations-secret-selling-online/
17- https://labrosa.ee.columbia.edu/millionsong/pages/example-track-description
18- https://labrosa.ee.columbia.edu/millionsong/faq
19- http://blog.echonest.com/post/11992136676/taste-profiles-get-added-to-the-million-song