Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

J Component report

Programme : M.Tech. (Integrated) - Computer Science with Specialization in Business Analytics

Course Title : Big Data Framework

Course Code : CSE3120

Slot : F1

Title: Music Recommendation System using Spark

Team Members: Tharani Kumar|19MIA1033
Parvathy AJ|19MIA1048
Madasu Deepika|19MIA1066
Harinisri G|19MIA1069

Faculty: Suganeshwari G

Sign:

Date:

INDEX

Abstract:

Music is consumed by millions of people, everyday, all over the world. In recent years, the music industry has shifted more and more towards digital distribution through online music stores and streaming services such as iTunes and Spotify. With this, a wider variety of music has been made available to the users and there is now a need for a recommender system to suggest music to users, which they haven’t heard before and which they are likely to enjoy. In this paper, collaborative filtering was applied on data regarding users’ listening activity with respect to the artists they listen to, in order to recommend new artists. The Alternating Least Squares (ALS) algorithm and Stochastic Gradient Descent (SGD) algorithm were used for developing the system and the data volume was increased to yield better results. The model trained on the bigger dataset was able to make recommendations faster, within 0.638 seconds and the better accuracy was achieved with the smaller dataset, with a model score of approximately 0.09949 at rank 10. Ratings of the artists were generated by means of SGD.

Introduction:

Music is one of the most popular forms of media and it plays a part in the day-to-day life of every individual. For all kinds of people and events all over the world, there is music for everyone and for everything. Today, almost a billion songs are currently in existence in the world, spanning an incredibly wide variety of genres, artists and languages. Thousands of songs continue to be made everyday, and it is impossible for anyone to listen to all these songs. People are always looking for new music to listen to and explore, and yet, the majority of new releases that are being put out go unnoticed. Searching for new releases manually is also a difficult task, as there is a seemingly endless amount of lists and websites which some might find difficult to navigate. With the overwhelming amount of options, it is easy for a person to feel confused on what songs they would like to listen to.

A recommender system would prove greatly helpful in this situation. A recommender system is a tool that helps in predicting whether a user may or may not like something among a list of given items. A system which is designed to analyze a user’s musical preferences, identify other similar songs or other users with similar preferences and suggest new music to the user based on this data, would serve to not only introduce the user to a lot of new music which they are more likely to enjoy, but also help musical artists gain more recognition based on the lyrical and musical content of their songs, rather than just the popularity.

Literature Survey:

Ahemd Ibrahim, Dina Nawara, Mahsa Ebrahimian, Rasha Kashef and Zeshan Fayyaz, in their paper, "Recommendation Systems: Algorithms, Challenges, Metrics and Business Opportunities", gave an overview of the current situation of research in recommender systems and aimed to identify the potential directions for the different applications in the field, such as media, healthcare, e-commerce etc. They also provided discussions on the qualitative metrics to evaluate and assess the quality of a recommendation system. Recall and Precision, Accuracy, ROC Curve and F-Measure were the metrics recorded in the paper. Collaborative filtering, content-based, demographic-based, utility-based, knowledge-based and hybrid-based were the type of recommender systems for which a detailed survey was performed. Four challenges were mainly identified which pose the biggest threat to the performance of a recommendation system, which are cold-start, data sparsity, scalability and diversity. The study found that recommender systems now have a wilder application due to the arrival of more robust recommendation algorithms and that the usage of recommender systems in everyday life has been facilitated with the advancement in technology and the emergence of smartphones.

D.H.Manjaiah and Mohammed Fadhel Aljunid, in their paper, "Movie Recommender System Based on Collaborative Filtering using Apache Spark", developed a system to suggest movies using the Collaborative Filtering algorithm on Apache Spark. They implemented the Alternating Least Squares (ALS) algorithm for matrix factorization. Various different parameters for the ALS algorithm were selected and tested to find the ones that worked most efficiently for the recommender system. Execution time, root mean squared error of rating prediction and rank were the metrics chosen for evaluating the model.

Ni Zhan and Yilin Yang, in their paper, "Music Recommender System" aimed to create an effective recommendation system, which took the user's previous musical preferences into account in order to make novel recommendations. They worked on the Million Song Dataset and used simple recommendations baselines and Collaborative Filtering as the method to make the recommendations. They used Apache Spark for their implementation, using weighted matrix factorization CF. Dimensionality reduction was tested using song meta-data as input, which generated the songs' latent features, which were clustered using k-means method among others in order to perform item-based recommendation. The quantitative evaluation of the system was done using the metric, mean average precision and the qualitative evaluation was done by means of visualization and human inspection.

Avick Kumar Dey, Pijush Kanti Dutta Pramanik, Pradeep Kumar Singh and Prasenjit Choudhury, in their paper, "Recommender Systems: An Overview, Research Trends and Future Directions" provide a complete comprehensive study on recommender systems, focusing on information retrieval, the techniques and approaches for it and the issues associated with it. The paper focused mainly on spotting the research trend in the field of recommender systems. The future of recommender systems was also discussed in the paper with the intention of opening up new firections of research in the domain. It was found that the research in the field primarily focussed on collaborative filtering and knowledge-based approach. China was found to be the top contribuer in the field and the highest number of relatd papers were published by IEEE. It was observed that the research in the field of recommender systems was at its highest in the time period of 2013-2014, and then witnessed a gradual decline, which the study attributes to saturation.

Dataset Description:

The dataset used is collected from the publicly available song data from Audioscrobbler and the original dataset contains information about 141K users, and 1.6 million artists. It lists the users, the artists that each user listens to, and a counter indicating how many times each user played each artist. In this project, a trimmed version of the original dataset containing information about the 50 most active users is used.

There are three files within the dataset,

- user_artist_data.txt: It contains data regarding the listener/user and the artists they listen to. It contains 3 columns, userid, artistid and playcount.
- artist_data.txt: It contains data regarding the artists with 2 columns, artistid and artist_name.
- artist_alias.txt: It is the data regarding misspelled artist names and it contains 2 columns, badid and goodid.

The column, userid, contains the values of the user’s identification numbers. The “artistid” column shows the identification number of each artist and the “artist_name” column shows the corresponding names of the artists. The “playcount” column shows the number of times that the specific user has played a song by the specific artist. The “badid” column shows the identification number of an artist name that is commonly misspelled or written incorrectly and the “goodid” column shows the identification number corresponding to the correct name of the artist.

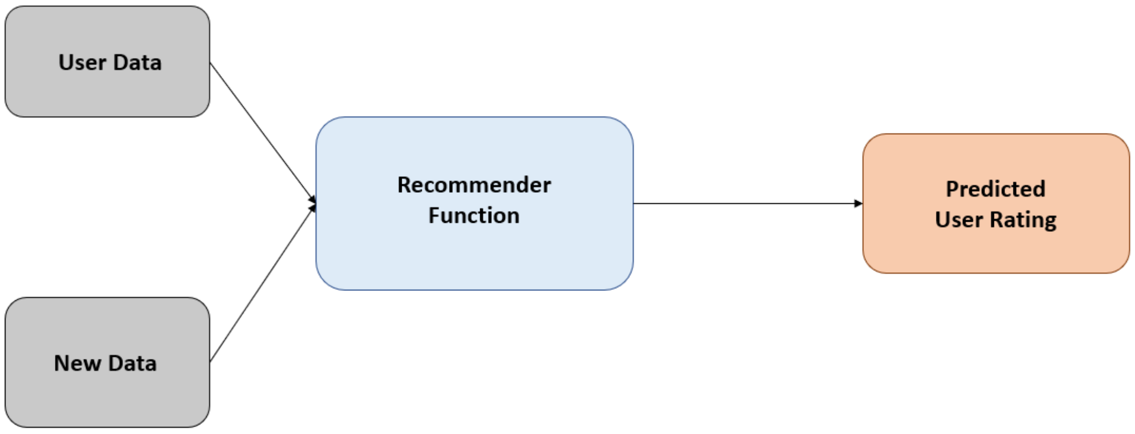
Problem Statement:

The goal of this project is to develop a music recommendation system to recommend new musical artists to users based on their listening history.

Proposed System:

A recommendation system is proposed to pinpoint accurate user preferences. The system uses ALS (Alternating Least Square) algorithm where a given matrix R is factorized into two factors U and V such that R≈UTV. If there are any unknown row dimension, it is given as parameter to the algorithm and called latent features. The model also uses SGD (Stochastic Gradient Descent) Algorithm. It uses convex loss functions to fit linear classifiers and regressors.

The recommendation system finds patterns in the collected consumer behaviour data and operates on this principle. The recommendation system proposed in this project uses collaborative filtering where the system predicts what a user will like based on their similarity with other users. Collaborative filtering picks items to recommend based on what they know about the user and doesn't need to analyze or understand the content.

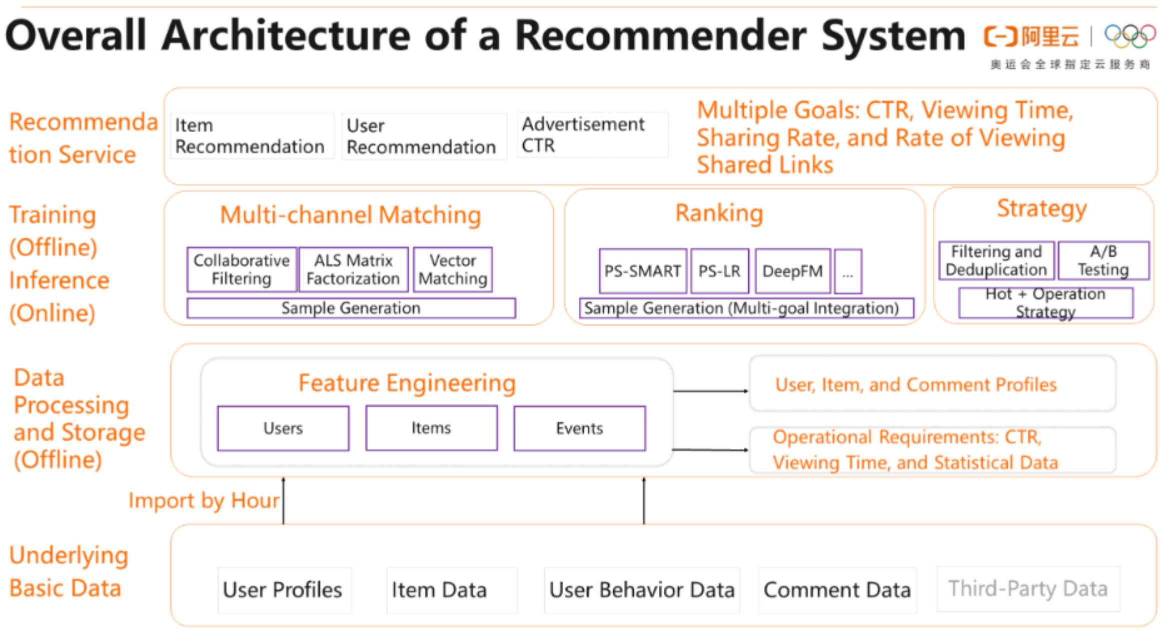


Architecture:

The architecture diagram shows the underlying basic data layer. It contains information like user profile data, artist data, behaviour data, etc. The user profile data consists of their User ID, artist they listen to and their play count. The artist profile data consists of Artist ID and their name. Behaviour data consists of the data collected from the interaction between the users and artists. These interactions are formed when a user listens to songs made by a particular artist.

When the user data, artist data and behaviour data are ready, comes the layer where training of the dataset takes place and inferences are gathered. This is followed by the data processing and storage layer. In this layer, all the data collected are processed like identifying user features. After these features are identified, the models are made.

The recommendation process consists of two modules, namely matching and ranking. It is necessary to filter and perform A/B tests on the recommendation results and try operational strategies before the system provides accurate song recommendations to users. The top layer of the architecture is the recommendation service which recommends songs to its users.



Modules:

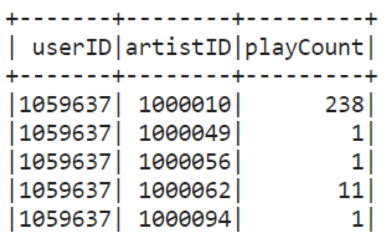
- **Data Preprocessing:** The data is checked for null values and the missing values are handled using the suitable imputation methods. The incorrect artist names are corrected based on the artist_alias.txt file.
- **Visualizations:** The data is visualized using graphs, plots etc. in order to better interpret and analyze the data. The methods of visualization include word cloud, bar graphs, tree maps etc.

- **Information extraction:** Extraction of information about users like artist play counts, user-mean play count, etc. by filtering, manipulating and transforming the RDDs. The RDDs are further sorted to find the most active user and extract various information like artists play count or user mean play count.
- **Splitting of data:** Data is split into training, validation and test set in the ratio of 4:4:2.
- **Model Selection:** For recommendation systems, models are chosen based on Collaborative Filtering. Collaborative Filtering is the method that makes recommendations by using the data of users who have similar music activity.
- **Model Building and Evaluation:** The accuracy of the model can be evaluated based on the validation set and a suitable error function.
- **Prediction and Recommendation:** Artist recommendations are made based on the predictions made by the model.
- **Recommendation System on Python:** Recommendation system is built on python. Various data like top ten artists are visualized and songs are recommended to the users.

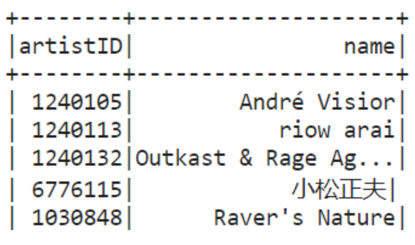
Implementation and Results:

1. Data Pre-processing:

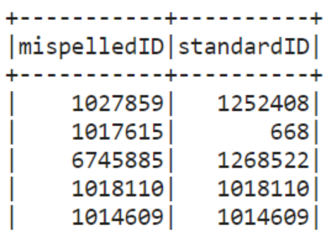
User data, Artist data and Artist Alias Data are displayed below respectively:



```
+-------+--------+---------+
| userID|artistID|playCount|
+-------+--------+---------+
|1059637| 1000010|      238|
|1059637| 1000049|        1|
|1059637| 1000056|        1|
|1059637| 1000062|       11|
|1059637| 1000094|        1|
```



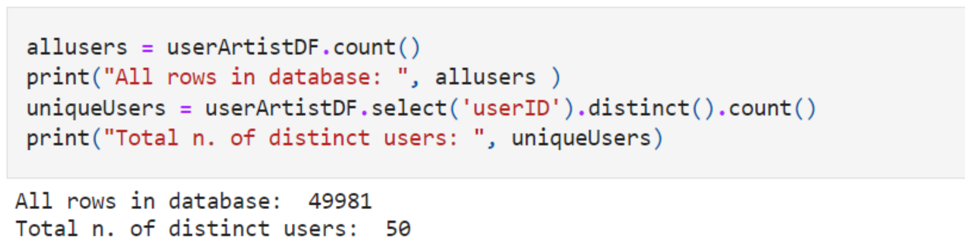
```
+--------+--------------------+
|artistID|                name|
+--------+--------------------+
| 1240105|        André Visior|
| 1240113|           riow arai|
| 1240132|Outkast & Rage Ag...|
| 6776115|                小松正夫|
| 1030848|       Raver's Nature|
```



```
+-----------+----------+
|mispelledID|standardID|
+-----------+----------+
|    1027859|   1252408|
|    1017615|       668|
|    6745885|   1268522|
|    1018110|   1018110|
|    1014609|   1014609|
```

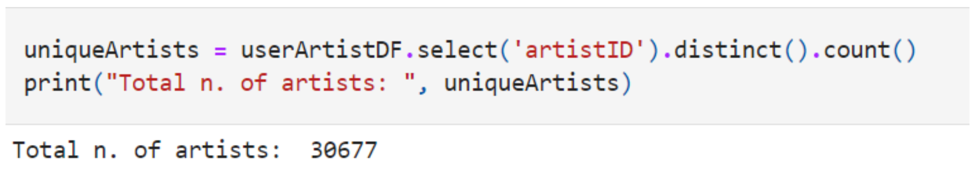
USER DATASET

There are two datasets: a large dataset and a small dataset. The total number of distinct users and total number of artists are displayed below. From the output, there are total of 50 distinct users and a total of 30,677 artists in the small dataset and 148111 distinct users and 1631028 in the large dataset.



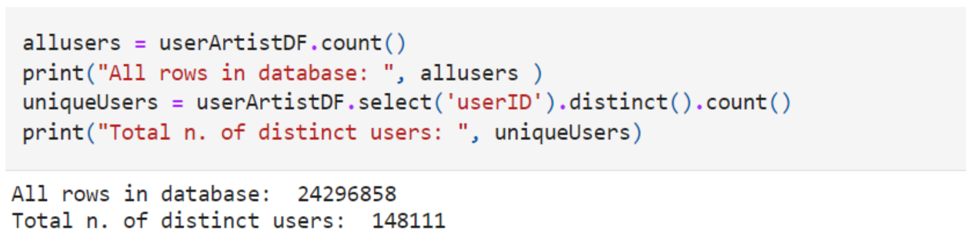
```
allusers = userArtistDF.count()
print("All rows in database: ", allusers )
uniqueUsers = userArtistDF.select('userID').distinct().count()
print("Total n. of distinct users: ", uniqueUsers)
```

```
All rows in database:  49981
Total n. of distinct users:  50
```



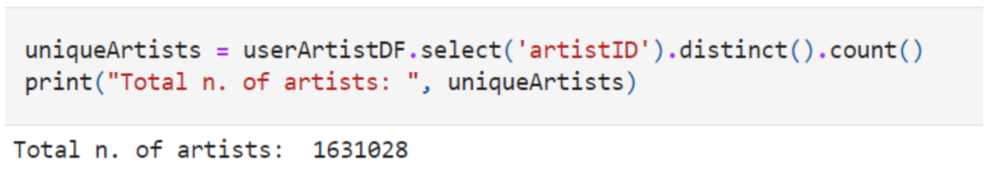
```
uniqueArtists = userArtistDF.select('artistID').distinct().count()
print("Total n. of artists: ", uniqueArtists)
```

```
Total n. of artists:  30677
```



```
allusers = userArtistDF.count()
print("All rows in database: ", allusers )
uniqueUsers = userArtistDF.select('userID').distinct().count()
print("Total n. of distinct users: ", uniqueUsers)
```

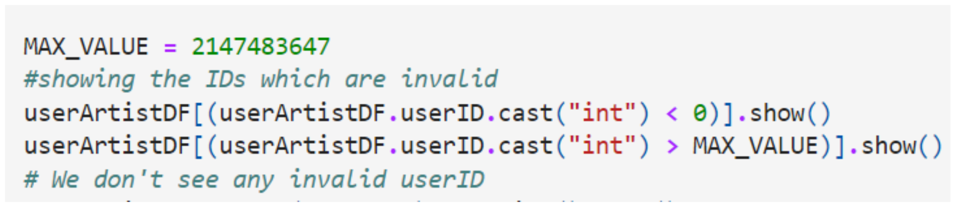
```
All rows in database:  24296858
Total n. of distinct users:  148111
```



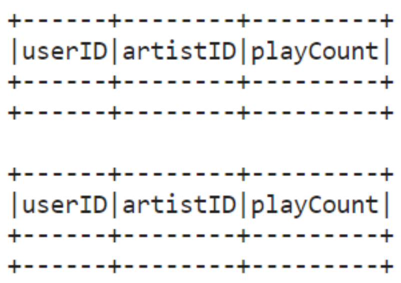
```
uniqueArtists = userArtistDF.select('artistID').distinct().count()
print("Total n. of artists: ", uniqueArtists)
```

```
Total n. of artists:  1631028
```

The validity of the User ID is checked. There are 49,981 and 24296858 users in the dataset out of which 50 and 148111are distinct, as found above. Invalid User IDs are lesser than 0 or greater than 2147483647. There is no invalid User IDs in the dataset.

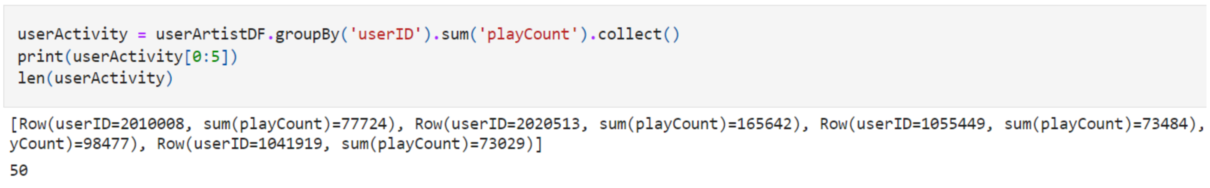


```
MAX_VALUE = 2147483647
#showing the IDs which are invalid
userArtistDF[(userArtistDF.userID.cast("int") < 0)].show()
userArtistDF[(userArtistDF.userID.cast("int") > MAX_VALUE)].show()
# We don't see any invalid userID
```



```
+------+--------+---------+
|userID|artistID|playCount|
+------+--------+---------+
+------+--------+---------+

+------+--------+---------+
|userID|artistID|playCount|
+------+--------+---------+
+------+--------+---------+
```

The data is grouped by User ID and sum of play count is calculated:



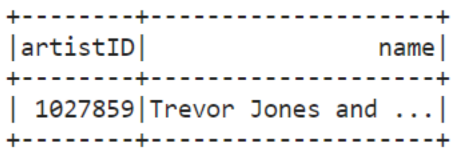
```
userActivity = userArtistDF.groupBy('userID').sum('playCount').collect()
print(userActivity[0:5])
len(userActivity)
```

```
[Row(userID=2010008, sum(playCount)=77724), Row(userID=2020513, sum(playCount)=165642), Row(userID=1055449, sum(playCount)=73484),
yCount)=98477), Row(userID=1041919, sum(playCount)=73029)]
50
```

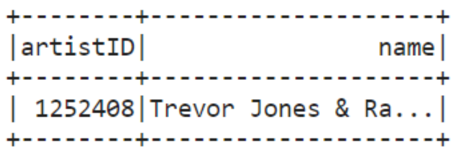
ARTIST DATASET and ARTIST ALIAS DATASET:

Due to artist ambiguation in the dataset, there is a chance that the artist IDs mentioned in the dataset is not unique. Analysis of the artist_alias dataset is done to figure out the unique IDs of artist and replace any other IDs with this ID.

An example of this is the artist Trevor Jones and Randy Edelman. Locating them in the dataset results gives two results for example. There is an ID 1027859 whose name is 'Trevor Jones and Randy Edelman' and 1252408 whose name is also 'Trevor Jones & Randy Edelman'. There's a chance that both these artists are the same.

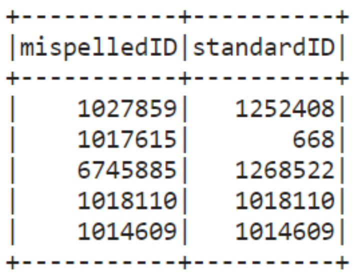


```
+--------+--------------------+
|artistID|                name|
+--------+--------------------+
| 1027859|Trevor Jones and ...|
+--------+--------------------+
```



```
+--------+--------------------+
|artistID|                name|
+--------+--------------------+
| 1252408|Trevor Jones & Ra...|
+--------+--------------------+
```

Another dataframe is created with just misspelled ID and Original ID.



```
+-----------+----------+
|mispelledID|standardID|
+-----------+----------+
|    1027859|   1252408|
|    1017615|       668|
|    6745885|   1268522|
|    1018110|   1018110|
|    1014609|   1014609|
+-----------+----------+
```

After verification, it can be concluded that ID 1027859 and ID 1252408 represent the same artist ‘Trevor Jones & Randy Edelman’.

```
artistAliasDF[artistAliasDF.mispelledID == "1027859" ].show()
artistAliasDF[artistAliasDF.mispelledID == "1252408" ].show()
```

```
+-----------+----------+
|mispelledID|standardID|
+-----------+----------+
|    1027859|   1252408|
|    1027859|   1252408|
+-----------+----------+

+-----------+----------+
|mispelledID|standardID|
+-----------+----------+
+-----------+----------+
```

This is performed on the entire dataset to remove duplicate IDs.

```
from time import time

bArtistAlias = sc.broadcast(artistAlias)

def replaceMispelledIDs(fields):
    finalID = bArtistAlias.value.get(fields[1] ,fields[1])
    return (fields[0], finalID, fields[2])

t0 = time()

newUserArtistDF = sqlContext.createDataFrame(
    userArtistDF.rdd.map(replaceMispelledIDs),
    userArtistDataSchema
)
newUserArtistDF.show(5)
t1 = time()

print('The script takes %f seconds' %(t1-t0))
newUserArtistDF = newUserArtistDF.cache()
```

```
+-------+--------+---------+
| userID|artistID|playCount|
+-------+--------+---------+
|1059637| 1000010|      238|
|1059637| 1000049|        1|
|1059637| 1000056|        1|
|1059637| 1000062|       11|
|1059637| 1000094|        1|
+-------+--------+---------+
only showing top 5 rows

The script takes 1.059970 seconds
```

After performing this operation on all artist IDs, the count of artists drops from 30,677 to 30,100 and 1631028. This means that 577 and 62,902 artist IDs were duplicate of the same artist.

```
uniqueArtists = newUserArtistDF.select('artistID').distinct().count()
print("Total n. of artists: ", uniqueArtists)
```

```
Total n. of artists:  30100
```

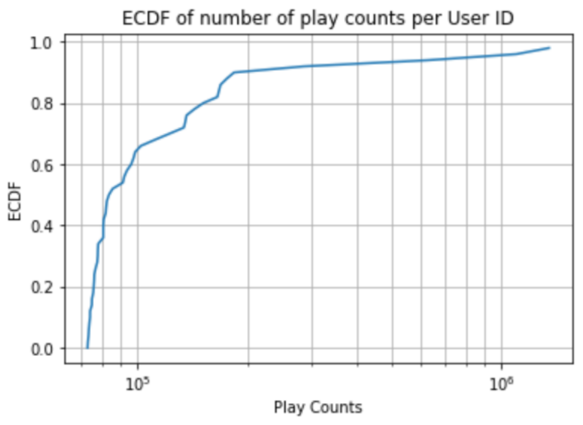
```
uniqueArtists = newUserArtistDF.select('artistID').distinct().count()
print("Total n. of artists: ", uniqueArtists)
```

```
Total n. of artists:  1568126
```

2. Visualization:

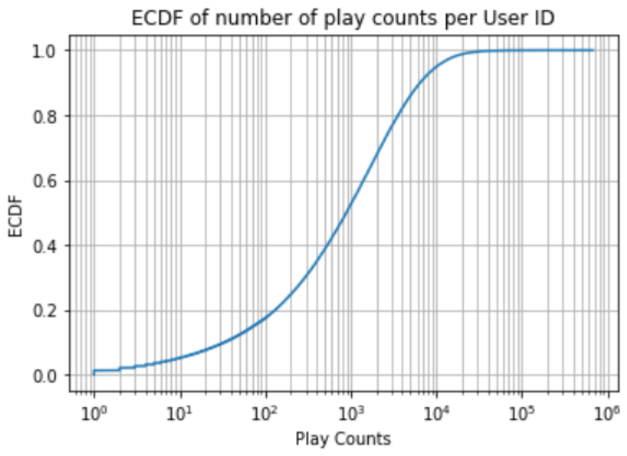
The following is the Empirical Distribution Function of number of play counts per user ID in the smaller dataset. From the graph it can be interpreted that a little more than 60% of the users have a user play count of 10,000.

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49]
```

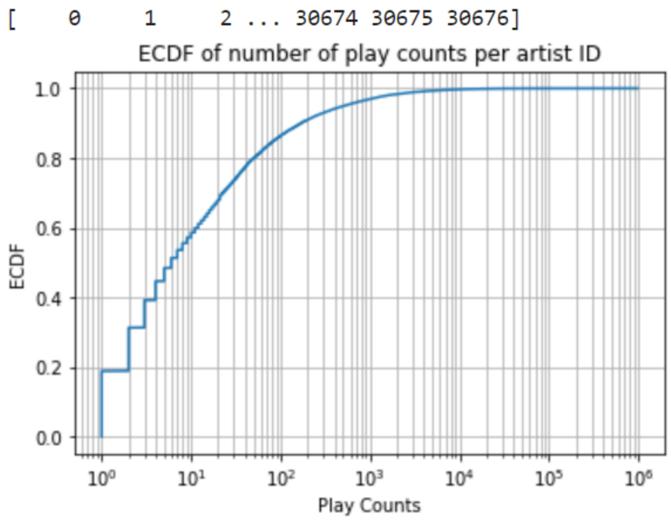


The following is the Empirical Distribution Function of number of play counts per user ID in the bigger dataset. From the graph it can be interpreted that a little more than 80% of the users have a user play count of 100,000.

```
[     0      1      2 ... 148108 148109 148110]
```

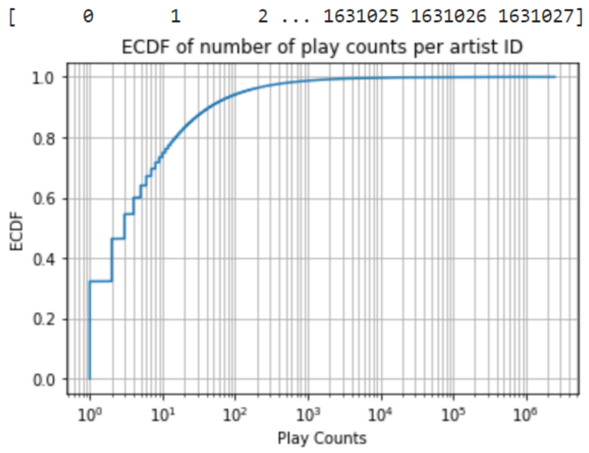


Similarly, the following is the Empirical Distribution Function of number of play counts per artist ID for the smaller dataset. From the graph it can be interpreted that only 10-20% of artists have a playcount close to 100,000.



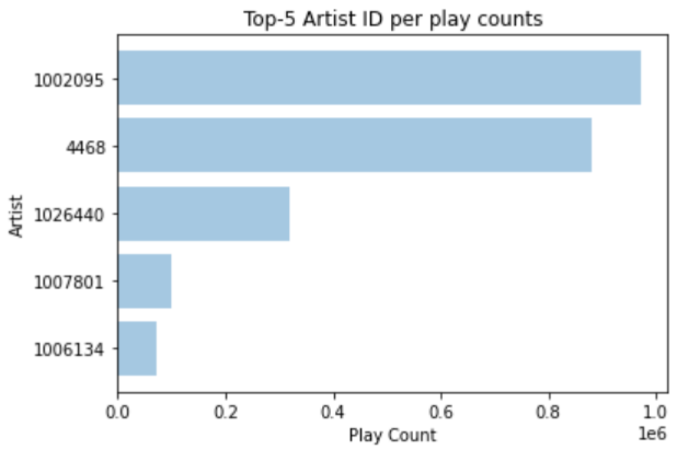
```
[     0      1      2 ... 30674 30675 30676]
```

The following is the Empirical Distribution Function of number of play counts per artist ID for the bigger dataset. From the graph it can be interpreted that only 80% artists have playcount less than 100.

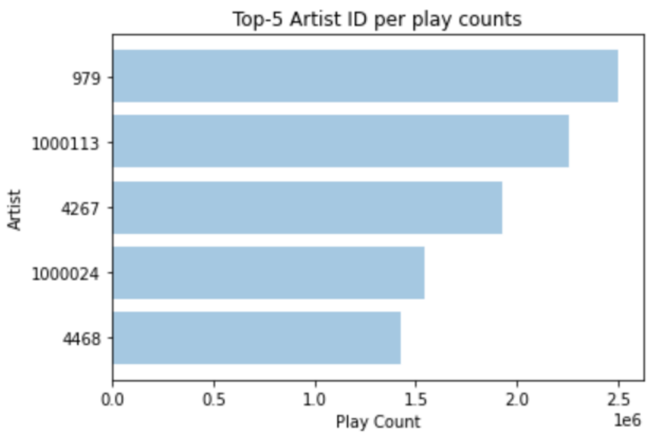


```
[       0       1       2 ... 1631025 1631026 1631027]
```

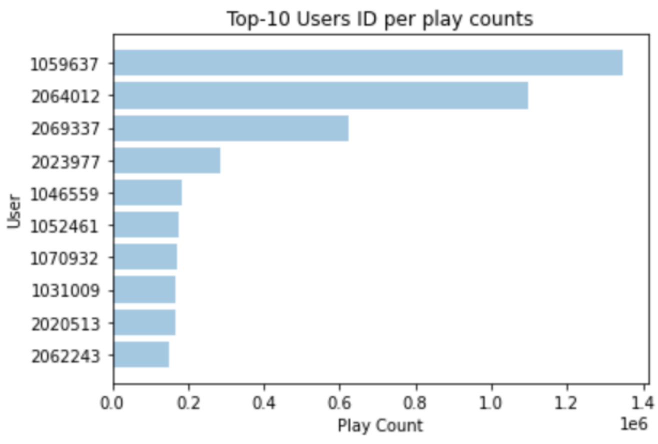
The following graph visualizes the count of Artist ID per Play Counts for the smaller dataset. Artist ID 1002095 has the highest playcount. The artist's name is 'Something Corporate'. This is followed by 'System of a Down', 'My Chemical Romance' and 'Sage Francis'.



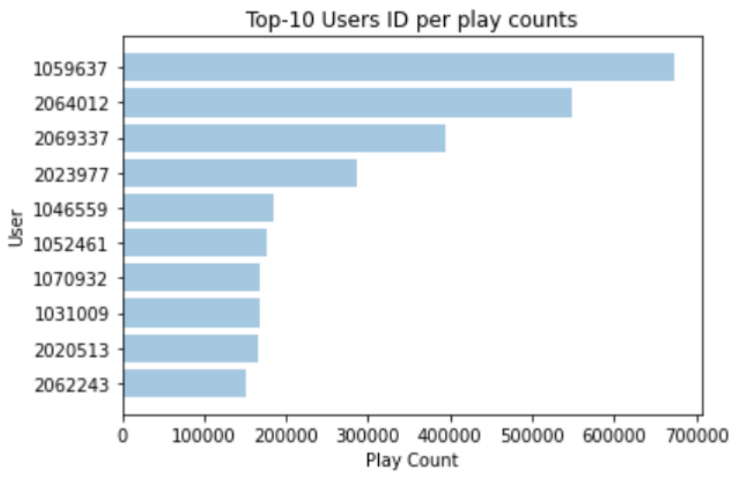
The following graph visualizes the count of Artist ID per Play Counts for the bigger dataset. Artist ID 979 has the highest playcount. The artist’s name is ‘Radiohead’. This is followed by ‘The Beatles’, ‘Green Day’ and ‘Metallica’.



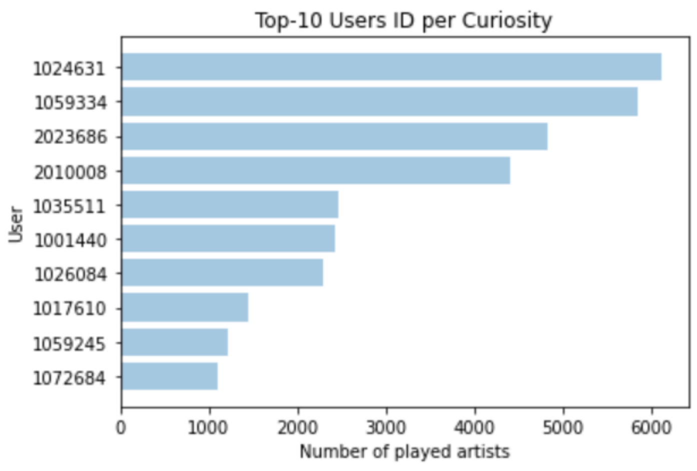
The following graph represents the count of User IDs per play count for the smaller dataset. User ID 1059637 has the highest count of play counts followed by 2064012, 2069337, etc.



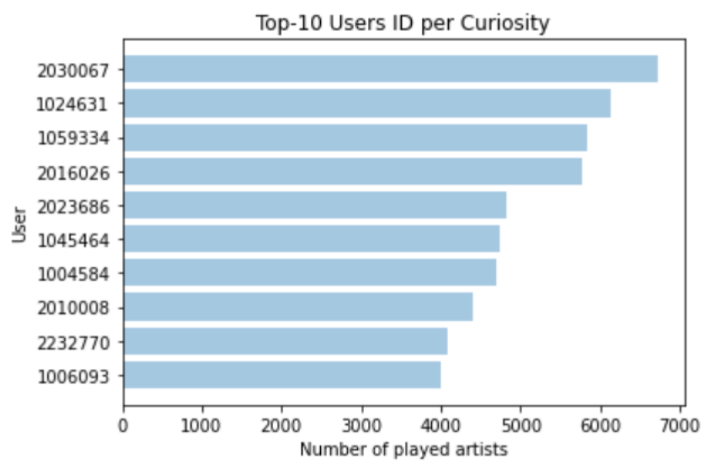
The following graph represents the count of User IDs per play count for the bigger dataset. User ID 1059637 has the highest count of play counts followed by 2064012, 2069337, etc.



The following graph represents the top 10 users based on curiosity on the smaller dataset. Curiosity refers to the number of diverse artists a user has listened to. User ID 1024631 is the most curious user with a count of 6000 diverse artists.

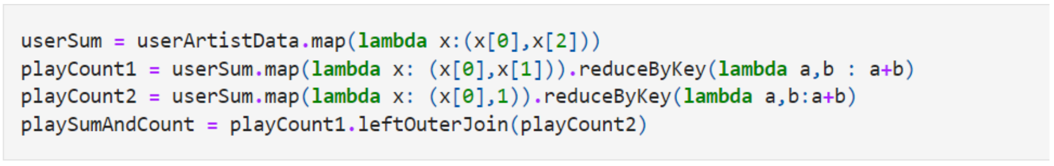


The following graph represents the top 10 users based on curiosity on the bigger dataset. User ID 2030067 is the most curious user with a count of 6000 diverse artists.

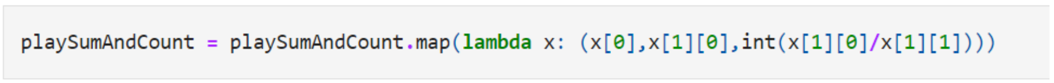


3. Information Extraction:

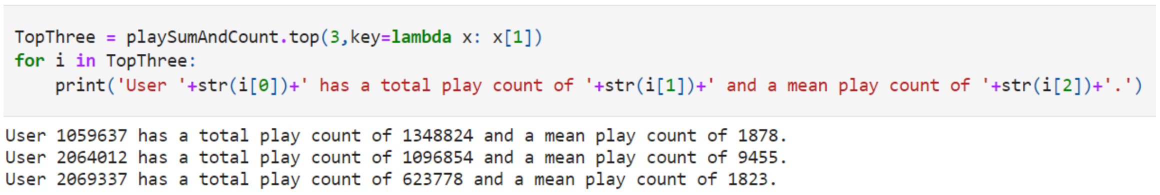
Mean User play count and total user play count for top 3 user is calculated and found. The top 3 users are User 1059637, 2064012 and 2069337. Their mean playcount and total playcount are displayed below:



```
userSum = userArtistData.map(lambda x:(x[0],x[2]))
playCount1 = userSum.map(lambda x: (x[0],x[1])).reduceByKey(lambda a,b : a+b)
playCount2 = userSum.map(lambda x: (x[0],1)).reduceByKey(lambda a,b:a+b)
playSumAndCount = playCount1.leftOuterJoin(playCount2)
```



```
playSumAndCount = playSumAndCount.map(lambda x: (x[0],x[1][0],int(x[1][0]/x[1][1])))
```

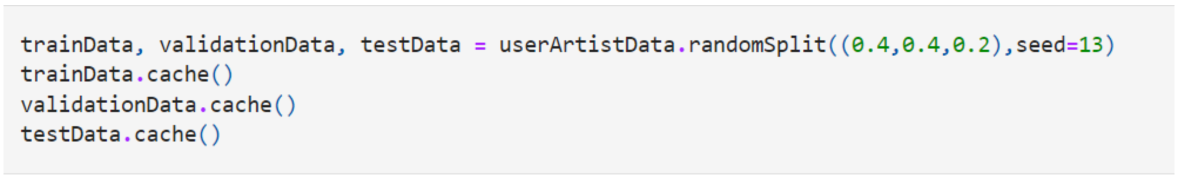


```
TopThree = playSumAndCount.top(3,key=lambda x: x[1])
for i in TopThree:
    print('User '+str(i[0])+' has a total play count of '+str(i[1])+' and a mean play count of '+str(i[2])+'.')
```

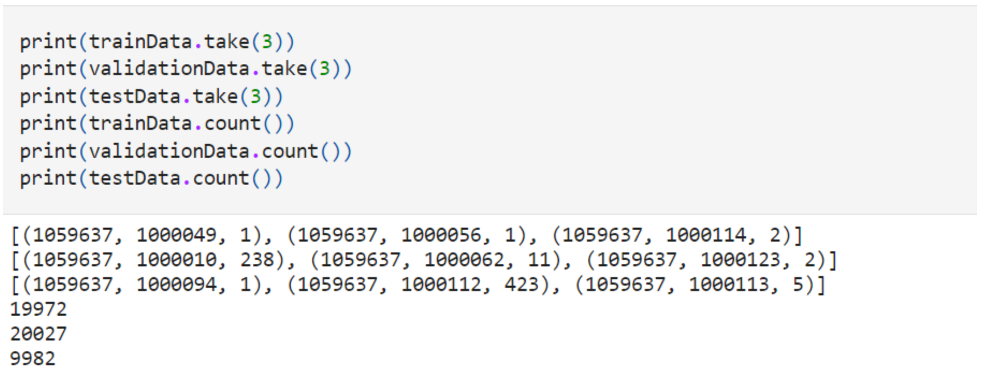
```
User 1059637 has a total play count of 1348824 and a mean play count of 1878.
User 2064012 has a total play count of 1096854 and a mean play count of 9455.
User 2069337 has a total play count of 623778 and a mean play count of 1823.
```

4. Splitting of Data:

The dataset is split into train, validation and test set in the ratio 4:4:2.



```
trainData, validationData, testData = userArtistData.randomSplit((0.4,0.4,0.2),seed=13)
trainData.cache()
validationData.cache()
testData.cache()
```

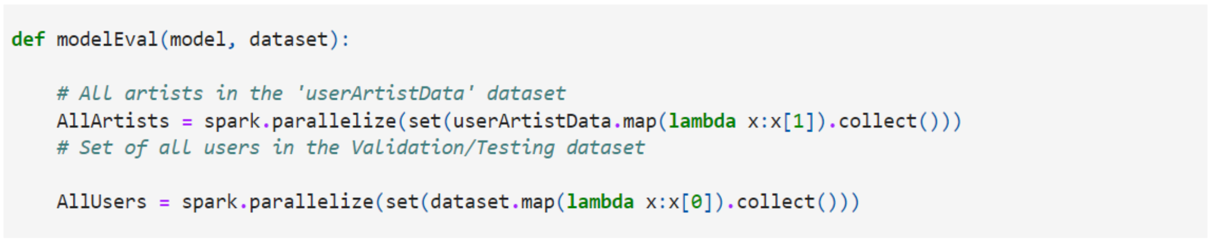


```
print(trainData.take(3))
print(validationData.take(3))
print(testData.take(3))
print(trainData.count())
print(validationData.count())
print(testData.count())
```

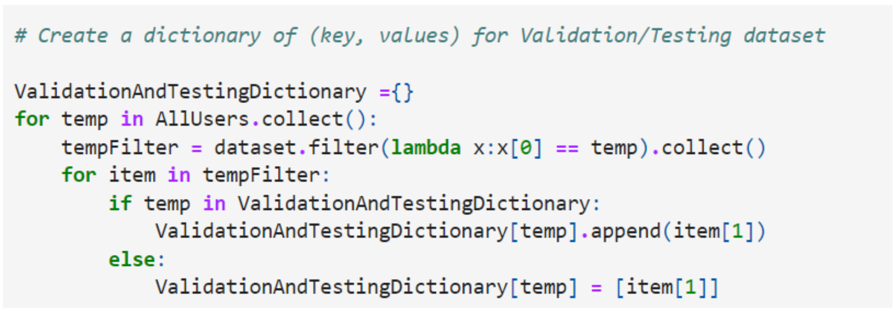
```
[(1059637, 1000049, 1), (1059637, 1000056, 1), (1059637, 1000114, 2)]
[(1059637, 1000010, 238), (1059637, 1000062, 11), (1059637, 1000123, 2)]
[(1059637, 1000094, 1), (1059637, 1000112, 423), (1059637, 1000113, 5)]
19972
20027
9982
```

5. Model Building and Evaluation:

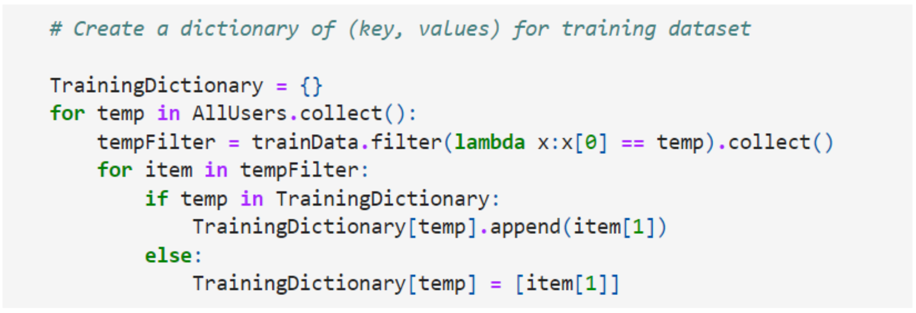
All the artists in ‘userArtistData’ dataset and all users in the validation and test set are parallelized and their values are collected. A dictionary of key-value pairs is created for the validation/test set. Similarly, another dictionary for key-value pair is created for the training set. The prediction score is calculated for each user and the average score of the model for all users for specified rank is also displayed.



```
def modelEval(model, dataset):

    # All artists in the 'userArtistData' dataset
    AllArtists = spark.parallelize(set(userArtistData.map(lambda x:x[1]).collect()))
    # Set of all users in the Validation/Testing dataset

    AllUsers = spark.parallelize(set(dataset.map(lambda x:x[0]).collect()))
```



```
# Create a dictionary of (key, values) for Validation/Testing dataset

ValidationAndTestingDictionary ={}
for temp in AllUsers.collect():
    tempFilter = dataset.filter(lambda x:x[0] == temp).collect()
    for item in tempFilter:
        if temp in ValidationAndTestingDictionary:
            ValidationAndTestingDictionary[temp].append(item[1])
        else:
            ValidationAndTestingDictionary[temp] = [item[1]]
```



```
# Create a dictionary of (key, values) for training dataset

TrainingDictionary = {}
for temp in AllUsers.collect():
    tempFilter = trainData.filter(lambda x:x[0] == temp).collect()
    for item in tempFilter:
        if temp in TrainingDictionary:
            TrainingDictionary[temp].append(item[1])
        else:
            TrainingDictionary[temp] = [item[1]]
```

```
# Calculate the prediction score for each user

PredictionScore = 0.00
for temp in AllUsers.collect():
    ArtistPrediction =  AllArtists.map(lambda x:(temp,x))
    ModelPrediction = model.predictAll(ArtistPrediction)
    tempFilter = ModelPrediction.filter(lambda x :not x[1] in TrainingDictionary[x[0]])
    topPredictions = tempFilter.top(len(ValidationAndTestingDictionary[temp]),key=lambda x:x[2])
    l=[]
    for i in topPredictions:
        l.append(i[1])
    PredictionScore+=len(set(l).intersection(ValidationAndTestingDictionary[temp]))/len(ValidationAndTestingDictionary[temp])


# Print average score of the model for all users for the specified rank

print("The model score for rank "+str(model.rank)+" is ~"+str(PredictionScore/len(ValidationAndTestingDictionary)))
```

6. Prediction and Recommendation:

The model score for specific ranks is evaluated and displayed:

Smaller dataset:

```
rankList = [2,10,20]
for rank in rankList:
     model = ALS.trainImplicit(trainData, rank , seed=345)
     modelEval(model,validationData)
```

```
The model score for rank 2 is ~0.07478965804909071
The model score for rank 10 is ~0.09949125799664481
The model score for rank 20 is ~0.09020351262501673
```

```
bestModel = ALS.trainImplicit(trainData, rank=10, seed=345)
modelEval(bestModel, testData)
```

```
The model score for rank 10 is ~0.06814203309788236
```

Bigger Dataset:

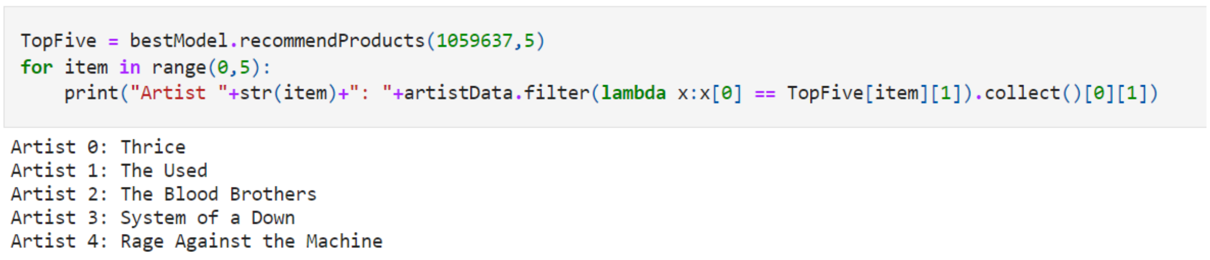
```
rankList = [2,10,20]
for rank in rankList:
     model = ALS.trainImplicit(trainData, rank , seed=345)
     modelEval(model,validationData)
```

```
The model score for rank 2 is ~0.07535599799693796
The model score for rank 10 is ~0.09765418032407164
The model score for rank 20 is ~0.08667584027612182
```

```
bestModel = ALS.trainImplicit(trainData, rank=10, seed=345)
modelEval(bestModel, testData)
```

```
The model score for rank 10 is ~0.06445912100969664
```

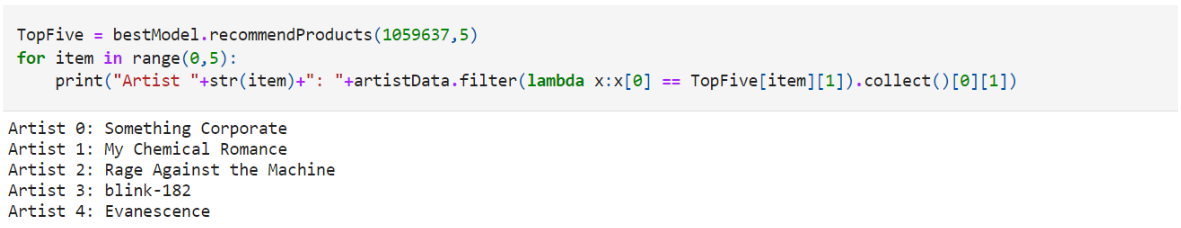
The system then recommends the top 5 artists for particular user. In the following example, the recommendation system recommends Thrice, The Used, The Blood Brothers and so on for User with ID 1059637 on the smaller dataset:



```
TopFive = bestModel.recommendProducts(1059637,5)
for item in range(0,5):
    print("Artist "+str(item)+": "+artistData.filter(lambda x:x[0] == TopFive[item][1]).collect()[0][1])
```

```
Artist 0: Thrice
Artist 1: The Used
Artist 2: The Blood Brothers
Artist 3: System of a Down
Artist 4: Rage Against the Machine
```

In the following example, the recommendation system recommends Something Corporate, My Chemical Romance and so on for User with ID 1059637 on the bigger dataset:

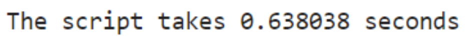


```
TopFive = bestModel.recommendProducts(1059637,5)
for item in range(0,5):
    print("Artist "+str(item)+": "+artistData.filter(lambda x:x[0] == TopFive[item][1]).collect()[0][1])
```

```
Artist 0: Something Corporate
Artist 1: My Chemical Romance
Artist 2: Rage Against the Machine
Artist 3: blink-182
Artist 4: Evanescence
```

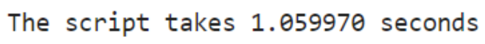
7. **Comparing the script time of the recommendation system on smaller and bigger dataset:**

Script Time for Recommendation System on Smaller dataset:



```
The script takes 0.638038 seconds
```

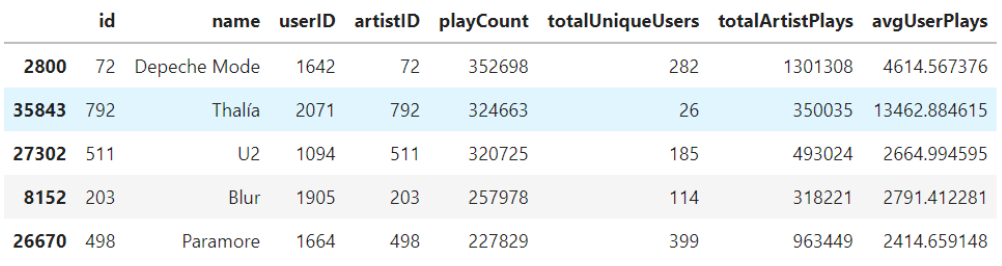
Script Time for Recommendation System on Bigger dataset:



```
The script takes 1.059970 seconds
```

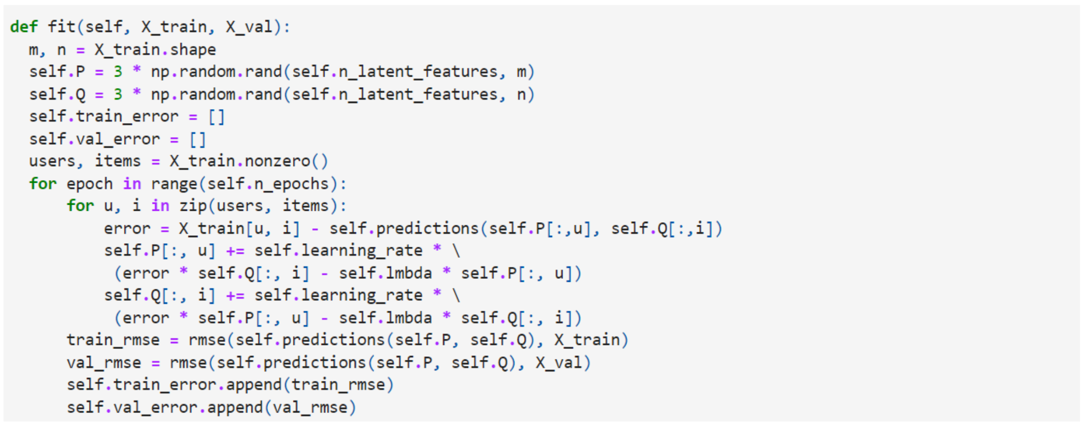
8. Python Code for Music Recommendation:

The datasets are merged together to form a single dataframe that contains information like ID, Name, User ID, Artist ID, etc.



	id	name	userID	artistID	playCount	totalUniqueUsers	totalArtistPlays	avgUserPlays
2800	72	Depeche Mode	1642	72	352698	282	1301308	4614.567376
35843	792	Thalía	2071	792	324663	26	350035	13462.884615
27302	511	U2	1094	511	320725	185	493024	2664.994595
8152	203	Blur	1905	203	257978	114	318221	2791.412281
26670	498	Paramore	1664	498	227829	399	963449	2414.659148

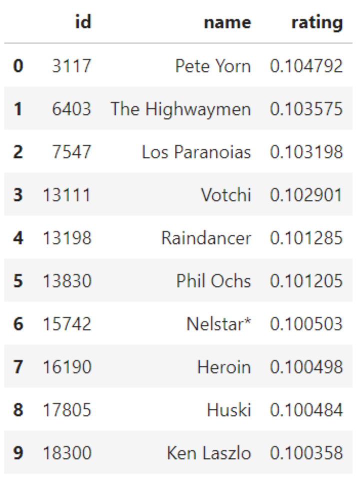
SGD model is built to train the recommendation model on:



```
def fit(self, X_train, X_val):
  m, n = X_train.shape
  self.P = 3 * np.random.rand(self.n_latent_features, m)
  self.Q = 3 * np.random.rand(self.n_latent_features, n)
  self.train_error = []
  self.val_error = []
  users, items = X_train.nonzero()
  for epoch in range(self.n_epochs):
      for u, i in zip(users, items):
          error = X_train[u, i] - self.predictions(self.P[:,u], self.Q[:,i])
          self.P[:, u] += self.learning_rate * \
           (error * self.Q[:, i] - self.lmbda * self.P[:, u])
          self.Q[:, i] += self.learning_rate * \
           (error * self.P[:, u] - self.lmbda * self.Q[:, i])
      train_rmse = rmse(self.predictions(self.P, self.Q), X_train)
      val_rmse = rmse(self.predictions(self.P, self.Q), X_val)
      self.train_error.append(train_rmse)
      self.val_error.append(val_rmse)
```

2 matrices are created for latent features of the users and ratings. For each user-artist pair, error is calculated. PP and QQ are then updated using gradient descent.

Finally, a dataframe is created where ID, Artist Name and the predicted Rating is displayed.



	id	name	rating
0	3117	Pete Yorn	0.104792
1	6403	The Highwaymen	0.103575
2	7547	Los Paranoias	0.103198
3	13111	Votchi	0.102901
4	13198	Raindancer	0.101285
5	13830	Phil Ochs	0.101205
6	15742	Nelstar*	0.100503
7	16190	Heroin	0.100498
8	17805	Huski	0.100484
9	18300	Ken Laszlo	0.100358

Conclusion:

On performing the implementation and testing, it was found that the script takes a time period of 1.05 seconds to run on the smaller dataset taken initially and it takes 0.638 seconds to run on the bigger dataset which was used later. The bigger dataset has a larger amount of data to work with and thus the training of the model on this dataset was better and more efficient in comparison with the smaller dataset. This resulted in a shorter time period for the running of the script in the case of the model using the bigger dataset. Both the models showed a better model score at rank 10, with values of 0.09949 for the smaller dataset and 0.09765 for the bigger dataset. On average, the model with the smaller dataset returned a better model score. The ratings for each artist based on the users' preference were also generated using the SGD algorithm.

Future Enhancement:

The recommender system can be improved and worked on in the future in several ways to increase its performance and functionality. The system can be expanded to include music from other languages and generate recommendations based on a user's region. The system could also be modified to recommend not only artists but also specific songs and albums, with suitable data. Other, more complex factors such as region, time of the day while listening, amount of time listened etc. could be introduced and incorporated into the system to make highly personalized recommendations. The system could be modified with the use of sentiment analysis to identify the mood of the user based on the music they're listening to, and recommend similar music at that time. An application could be developed for the system, which connects to the user's music platform of choice (such as Spotify, Google Play Music etc.), where it can create a playlist for the user comprising the recommended music.

References:

- Liao, K. (2018, November 19). Prototyping a recommender system step by Step Part 1: Knn Item-based collaborative filtering. Medium. Retrieved April 27, 2022, from https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea?source=rss-1a2cfc6139e2------3
- What is a recommendation engine and how does it work? Appier. (n.d.). Retrieved April 27, 2022, from https://www.appier.com/blog/what-is-a-recommendation-engine-and-how-does-it-work#:~:text=A%20recommendation%20engine%20is%20a,be%20collected%20implicitly%20or%20explicitly.
- *Basic concepts and architecture of a recommender system*. Alibaba Cloud Community. (n.d.). Retrieved April 27, 2022, from https://www.alibabacloud.com/blog/basic-concepts-and-architecture-of-a-recommender-system_596642#:~:text=A%20typical%20recommender%20system%20based,items%20user%20A%20may%20like.
- *ML: Stochastic gradient descent (SGD)*. GeeksforGeeks. (2021, September 13). Retrieved April 27, 2022, from https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/
- Aljunid, M. F., &amp; Manjaiah, D. H. (2018). Movie Recommender System based on collaborative filtering using Apache Spark. Data Management, Analytics and Innovation, 283–295. https://doi.org/10.1007/978-981-13-1274-8_22
- Fayyaz, Z., Ebrahimian, M., Nawara, D., Ibrahim, A., &amp; Kashef, R. (2020). Recommendation systems: Algorithms, challenges, metrics, and business opportunities. Applied Sciences, 10(21), 7748. https://doi.org/10.3390/app10217748
- Celma, Ò. (2010). The Long Tail in recommender systems. Music Recommendation and Discovery, 87–107. https://doi.org/10.1007/978-3-642-13287-2_4
- Singh, P. K., Choudhury, P., Dey, A. K., &amp; Pramanik, P. K. (2021). Recommender Systems: An overview, research trends, and Future Directions. International Journal of Business and Systems Research, 15(1), 14. https://doi.org/10.1504/ijbsr.2021.10033303
- Music Recommender Systems. (2019). The SAGE International Encyclopedia of Music and Culture. https://doi.org/10.4135/9781483317731.n497