THE MILLION SONG DATASET (MSD)

Name: Pengcheng Wu

Date: 18/05/2022 ~ 03/06/2022

Contents

Foreword

In this assignment we will study a collection of datasets referred to as the Million Song Dataset (MSD), a project initiated by The Echo Nest and LabROSA. The Echo Nest was a research spin-off from the MIT Media Lab established with the goal of understanding the audio and textual content of recorded music, and was acquired by Spotify after 10 years for 50 million Euro. This project aims to explore and study how to set up collaborative filter method.

1 Data Processing

1.1 Exploring the data

1.1.1 Data Structure

Drawing one directory tree showing the overview of the structure of the datasets:

```
/scratch-network/courses/2022/DATA420-22S1/data/msd/
├── audio
│   ├── attributes
│   │   ├── msd-jmir-area-of-moments-all-v1.0.attributes.csv
│   │   ├── msd-jmir-lpc-all-v1.0.attributes.csv
│   │   ├── msd-jmir-methods-of-moments-all-v1.0.attributes.csv
│   │   ├── msd-jmir-mfcc-all-v1.0.attributes.csv
│   │   ├── msd-jmir-spectral-all-all-v1.0.attributes.csv
│   │   ├── msd-jmir-spectral-derivatives-all-all-v1.0.attributes.csv
│   │   ├── msd-marsyas-timbral-v1.0.attributes.csv
│   │   ├── msd-mvd-v1.0.attributes.csv
│   │   ├── msd-rh-v1.0.attributes.csv
│   │   ├── msd-rp-v1.0.attributes.csv
│   │   ├── msd-ssd-v1.0.attributes.csv
│   │   ├── msd-trh-v1.0.attributes.csv
│   │   └── msd-tssd-v1.0.attributes.csv
│   ├── features
│   │   ├── msd-jmir-area-of-moments-all-v1.0.csv
│   │   │   ├── part-00000.csv.gz
│   │   │   ├── part-00001.csv.gz
│   │   │   ├── part-00002.csv.gz
│   │   │   ├── part-00003.csv.gz
│   │   │   ├── part-00004.csv.gz
│   │   │   ├── part-00005.csv.gz
│   │   │   ├── part-00006.csv.gz
│   │   │   └── part-00007.csv.gz
│   │   ├── msd-jmir-lpc-all-v1.0.csv
│   │   │   ├── part-00000.csv.gz
│   │   │   ├── part-00001.csv.gz
│   │   │   ├── part-00002.csv.gz
│   │   │   ├── part-00003.csv.gz
│   │   │   ├── part-00004.csv.gz
│   │   │   ├── part-00005.csv.gz
│   │   │   ├── part-00006.csv.gz
│   │   │   └── part-00007.csv.gz
│   │   ├── msd-jmir-methods-of-moments-all-v1.0.csv
│   │   │   │   ├── part-00000.csv.gz
│   │   │   │   ├── part-00001.csv.gz
│   │   │   │   ├── part-00002.csv.gz
│   │   │   │   ├── part-00003.csv.gz
│   │   │   │   ├── part-00004.csv.gz
│   │   │   │   ├── part-00005.csv.gz
│   │   │   │   ├── part-00006.csv.gz
│   │   │   │   └── part-00007.csv.gz
│   │   │   ├── msd-jmir-mfcc-all-v1.0.csv
│   │   │   │   ├── part-00000.csv.gz
│   │   │   │   ├── part-00001.csv.gz
│   │   │   │   ├── part-00002.csv.gz
│   │   │   │   ├── part-00003.csv.gz
│   │   │   │   ├── part-00004.csv.gz
│   │   │   │   ├── part-00005.csv.gz
│   │   │   │   ├── part-00006.csv.gz
│   │   │   │   └── part-00007.csv.gz
│   │   │   ├── msd-jmir-spectral-all-all-v1.0.csv
│   │   │   │   ├── part-00000.csv.gz
│   │   │   │   ├── part-00001.csv.gz
│   │   │   │   ├── part-00002.csv.gz
│   │   │   │   ├── part-00003.csv.gz
│   │   │   │   ├── part-00004.csv.gz
│   │   │   │   ├── part-00005.csv.gz
│   │   │   │   ├── part-00006.csv.gz
│   │   │   │   └── part-00007.csv.gz
│   │   │   ├── msd-jmir-spectral-derivatives-all-all-v1.0.csv
│   │   │   │   ├── part-00000.csv.gz
│   │   │   │   ├── part-00001.csv.gz
│   │   │   │   ├── part-00002.csv.gz
│   │   │   │   ├── part-00003.csv.gz
│   │   │   │   ├── part-00004.csv.gz
│   │   │   │   ├── part-00005.csv.gz
│   │   │   │   ├── part-00006.csv.gz
│   │   │   │   └── part-00007.csv.gz
│   │   │   ├── msd-marsyas-timbral-v1.0.csv
│   │   │   │   ├── part-00000.csv.gz
│   │   │   │   ├── part-00001.csv.gz
│   │   │   │   ├── part-00002.csv.gz
│   │   │   │   ├── part-00003.csv.gz
│   │   │   │   ├── part-00004.csv.gz
│   │   │   │   ├── part-00005.csv.gz
│   │   │   │   ├── part-00006.csv.gz
│   │   │   │   └── part-00007.csv.gz
│   │   │   ├── msd-mvd-v1.0.csv
│   │   │   │   ├── part-00000.csv.gz
│   │   │   │   ├── part-00001.csv.gz
│   │   │   │   ├── part-00002.csv.gz
│   │   │   │   ├── part-00003.csv.gz
│   │   │   │   ├── part-00004.csv.gz
│   │   │   │   ├── part-00005.csv.gz
│   │   │   │   ├── part-00006.csv.gz
│   │   │   │   └── part-00007.csv.gz
│   │   │   ├── msd-rh-v1.0.csv
│   │   │   │   ├── part-00000.csv.gz
│   │   │   │   ├── part-00001.csv.gz
│   │   │   │   ├── part-00002.csv.gz
│   │   │   ├── part-00003.csv.gz
│   │   │   ├── part-00004.csv.gz
│   │   │   ├── part-00005.csv.gz
│   │   │   ├── part-00006.csv.gz
│   │   │   └── part-00007.csv.gz
│   │   ├── msd-rp-v1.0.csv
│   │   │   ├── part-00000.csv.gz
│   │   │   ├── part-00001.csv.gz
│   │   │   ├── part-00002.csv.gz
│   │   │   ├── part-00003.csv.gz
│   │   │   ├── part-00004.csv.gz
│   │   │   ├── part-00005.csv.gz
│   │   │   ├── part-00006.csv.gz
│   │   │   └── part-00007.csv.gz
│   │   ├── msd-ssd-v1.0.csv
│   │   │   ├── part-00000.csv.gz
│   │   │   ├── part-00001.csv.gz
│   │   │   ├── part-00002.csv.gz
│   │   │   ├── part-00003.csv.gz
│   │   │   ├── part-00004.csv.gz
│   │   │   ├── part-00005.csv.gz
│   │   │   ├── part-00006.csv.gz
│   │   │   └── part-00007.csv.gz
│   │   ├── msd-trh-v1.0.csv
│   │   │   ├── part-00000.csv.gz
│   │   │   ├── part-00001.csv.gz
│   │   │   ├── part-00002.csv.gz
│   │   │   ├── part-00003.csv.gz
│   │   │   ├── part-00004.csv.gz
│   │   │   ├── part-00005.csv.gz
│   │   │   ├── part-00006.csv.gz
│   │   │   └── part-00007.csv.gz
│   │   └── msd-tssd-v1.0.csv
│   │       ├── part-00000.csv.gz
│   │       ├── part-00001.csv.gz
│   │       ├── part-00002.csv.gz
│   │       ├── part-00003.csv.gz
│   │       ├── part-00004.csv.gz
│   │       ├── part-00005.csv.gz
│   │       ├── part-00006.csv.gz
│   │       └── part-00007.csv.gz
│   └── statistics
│       └── sample_properties.csv.gz
├── genre
│   ├── msd-MAGD-genreAssignment.tsv
│   ├── msd-MASD-styleAssignment.tsv
│   └── msd-topMAGD-genreAssignment.tsv
├── main
│   └── summary
│       ├── analysis.csv.gz
│       └── metadata.csv.gz
└── tasteprofile
    ├── mismatches
    │   ├── sid_matches_manually_accepted.txt
    │   └── sid_mismatches.txt
    └── triplets.tsv
        ├── part-00000.tsv.gz
        ├── part-00001.tsv.gz
        ├── part-00002.tsv.gz
        ├── part-00003.tsv.gz
        ├── part-00004.tsv.gz
        ├── part-00005.tsv.gz
        ├── part-00006.tsv.gz
        └── part-00007.tsv.gz

23 directories, 133 files
```

Figure 1.1 tree directory diagram

From the above tree directory, the text documents include txt, csv, and tsv formats corresponding file paths with (tasteprofile/mismatches), (audio/attributes & features), and (genre) separately. The compressed formats contain csv.gz and tsv.gz respectively corresponding file paths with (audio/statistics, main/summary) and (tasteprofile/triplets.tsv), as the Table 1.1 shown.

File Formats	File Path
.txt	tasteprofile/mismatches
.csv	audio/attributes & features
. tsv	genre
csv.gz	audio/statistics, main/summary, audio/features (subfiles in every csv files)
tsv.gz	tasteprofile/triplets.tsv

Table 1.1 data type of the whole of data path

The data size is 12.9 G in total, and it takes up 103.5 G in disk space consumed with all replicas (as the Figure 1.2 shown). There are four items in this dataset: audio, genre, main and tasteprofile. The audio directory in the dataset occupies the largest size which is 12.3G. There are 13 csv files in the attribute's directory of audio directory, and each csv file consists of two columns with the first column being data name and the second being corresponding data type. In the 'audio/features', the main data types in these csv.gz files are numeric, and the file of 'audio/statistics' contains ID (string), song name(string), artist name(string) and numeric data. The genre directory includes three tsv subfiles which all include the first column being track ID (string) and the second one being genre labels (string). The subfile in main directory is csv.gz and the data type of these files consist of string and numeric as data types. In tasteprofile directory, mismatches directory includes txt files with string data type and triplets.tsv includes eight tsv.gz files with string and numeric data types.

```
Total Size:
SIZE    DISK_SPACE_CONSUMED_WITH_ALL_REPLICAS FULL_PATH_NAME
12.9 G  103.5 G                                /data/msd
Each Item Size:
SIZE     DISK_SPACE_CONSUMED_WITH_ALL_REPLICAS FULL_PATH_NAME
12.3 G   98.1 G                                 /data/msd/audio
30.1 M   241.0 M                                /data/msd/genre
```

```
174.4 M  1.4 G                                     /data/msd/main
490.4 M  3.8 G                                     /data/msd/tasteprofile
```

Figure 1.2 data size of the all datasets

In HDFS data is stored in blocks, blocks are the smallest unit of data that the file system stores. HDFS divides files into blocks and stores each block on a DataNode. Multiple DataNodes are linked to the master node (NameNode) in the cluster. The master node distributes replicas of these data blocks across the cluster. The process of data block splitting happens by partitioning the file into smaller and managing data blocks before Name Node stores and manages the data. The default size of a block cannot exceed 128MB. The number of blocks depends on the initial size of the file with the same size, but the last one is what remains of the file. When Spark reads a file from HDFS, it creates one single partition for single input split. Input split is set by the Hadoop Input Format to read a file.

1.1.2 Repartition method in this project

The repartition () method is applied to increase or decrease the number of partitions of an RDD or data frame in Spark cluster. The method happens in a full shuffle of data across all the nodes and creates partitions of more or less equal in size. It is a costly operation. Partitions plays an essential in the degree of parallelism. The number of parallel tasks running in each stage is equal to the number of partitions.

The repartition () method can control parallelism and decide the number of files generated in the output. For example, one ‘audio/features’ subdirectory consists of eight compressed files (without considering the file size) which means there are eight partitions. If the Spark cluster with 8 cores, these partitions can process parallel; under this condition, repartition method would be not necessary. But if a cluster has 12 cores, this method can increase the level of parallelism.

1.1.3 Number of rows in each dataset

Counting the number of rows in each of the datasets (Figure 1.3):

```
# audio/attributes
/data/msd/audio/attributes
/msd-jmir-area-of-moments-all-v1.0.attributes.csv 21
/msd-jmir-lpc-all-v1.0.attributes.csv 21
/msd-jmir-methods-of-moments-all-v1.0.attributes.csv 11
/msd-jmir-mfcc-all-v1.0.attributes.csv 27
/msd-jmir-spectral-all-all-v1.0.attributes.csv 17
/msd-jmir-spectral-derivatives-all-all-v1.0.attributes.csv 17
/msd-marsyas-timbral-v1.0.attributes.csv 125
/msd-mvd-v1.0.attributes.csv 421
/msd-rh-v1.0.attributes.csv 61
```

```
/msd-rp-v1.0.attributes.csv 1441
/msd-ssd-v1.0.attributes.csv 169
/msd-trh-v1.0.attributes.csv 421
/msd-tssd-v1.0.attributes.csv 1177

#audio/features
/data/msd/audio/features/msd-jmir-area-of-moments-all-v1.0.csv 994623
/part-00000.csv.gz 33417
/part-00001.csv.gz 33066
/part-00002.csv.gz 33453
/part-00003.csv.gz 33952
/part-00004.csv.gz 33486
/part-00005.csv.gz 33569
/part-00006.csv.gz 33561
/part-00007.csv.gz 31719

/data/msd/audio/features/msd-jmir-lpc-all-v1.0.csv 994623
/part-00000.csv.gz 25932
/part-00001.csv.gz 25687
/part-00002.csv.gz 25395
/part-00003.csv.gz 25783
/part-00004.csv.gz 25564
/part-00005.csv.gz 25546
/part-00006.csv.gz 25240
/part-00007.csv.gz 24387
/data/msd/audio/features/msd-jmir-methods-of-moments-all-v1.0.csv 994623
/part-00000.csv.gz 19691
/part-00001.csv.gz 19811
/part-00002.csv.gz 19886
/part-00003.csv.gz 19314
/part-00004.csv.gz 19535
/part-00005.csv.gz 19859
/part-00006.csv.gz 19527
/part-00007.csv.gz 19099

/data/msd/audio/features/msd-jmir-mfcc-all-v1.0.csv 994623
/part-00000.csv.gz 32472
/part-00001.csv.gz 32614
/part-00002.csv.gz 32735
/part-00003.csv.gz 32464
/part-00004.csv.gz 32675
/part-00005.csv.gz 32563
/part-00006.csv.gz 32333
/part-00007.csv.gz 30973

/data/msd/audio/features/msd-jmir-spectral-all-all-v1.0.csv 994623
/part-00000.csv.gz 24959
/part-00001.csv.gz 25191
/part-00002.csv.gz 25480
/part-00003.csv.gz 24958
/part-00004.csv.gz 25380
/part-00005.csv.gz 25086
/part-00006.csv.gz 25176
/part-00007.csv.gz 24206

/data/msd/audio/features/msd-jmir-spectral-derivatives-all-all-v1.0.csv
994623
```

```
/part-00000.csv.gz 24959
/part-00001.csv.gz 25191
/part-00002.csv.gz 25480
/part-00003.csv.gz 24958
/part-00004.csv.gz 25380
/part-00005.csv.gz 25086
/part-00006.csv.gz 25176
/part-00007.csv.gz 24206

/data/msd/audio/features/msd-marsyas-timbral-v1.0.csv 995001
/part-00000.csv.gz 211585
/part-00001.csv.gz 211830
/part-00002.csv.gz 212396
/part-00003.csv.gz 210986
/part-00004.csv.gz 211635
/part-00005.csv.gz 211093
/part-00006.csv.gz 211955
/part-00007.csv.gz 203682
/data/msd/audio/features/msd-mvd-v1.0.csv 994188
/part-00000.csv.gz 663899
/part-00001.csv.gz 665675
/part-00002.csv.gz 663516
/part-00003.csv.gz 663391
/part-00004.csv.gz 662462
/part-00005.csv.gz 663993
/part-00006.csv.gz 663818
/part-00007.csv.gz 633908

/data/msd/audio/features/msd-rh-v1.0.csv 994188
/part-00000.csv.gz 113134
/part-00001.csv.gz 112560
/part-00002.csv.gz 112892
/part-00003.csv.gz 113354
/part-00004.csv.gz 113032
/part-00005.csv.gz 112661
/part-00006.csv.gz 113230
/part-00007.csv.gz 108668

/data/msd/audio/features/msd-rp-v1.0.csv 994188
/part-00000.csv.gz 2179745
/part-00001.csv.gz 2181398
/part-00002.csv.gz 2181997
/part-00003.csv.gz 2185657
/part-00004.csv.gz 2184739
/part-00005.csv.gz 2183239
/part-00006.csv.gz 2181977
/part-00007.csv.gz 2081171

/data/msd/audio/features/msd-ssd-v1.0.csv 994188
/part-00000.csv.gz 286349
/part-00001.csv.gz 285764
/part-00002.csv.gz 284379
/part-00003.csv.gz 284436
/part-00004.csv.gz 285610
/part-00005.csv.gz 286054
/part-00006.csv.gz 285284
/part-00007.csv.gz 272151
```

```
/data/msd/audio/features/msd-trh-v1.0.csv 994188
/part-00000.csv.gz 675893
/part-00001.csv.gz 676897
/part-00002.csv.gz 675325
/part-00003.csv.gz 675019
/part-00004.csv.gz 675434
/part-00005.csv.gz 676747
/part-00006.csv.gz 675558
/part-00007.csv.gz 643112

/data/msd/audio/features/msd-tssd-v1.0.csv 994623
/part-00000.csv.gz 1860302
/part-00001.csv.gz 1862402
/part-00002.csv.gz 1862556
/part-00003.csv.gz 1860987
/part-00004.csv.gz 1863542
/part-00005.csv.gz 1862813
/part-00006.csv.gz 1861818
/part-00007.csv.gz 1776530

#audio/statistics
/data/msd/audio/statistics 183262



#genre
/data/msd/genre/msd-MAGD-genreAssignment.tsv #422714
/data/msd/genre/msd-MASD-styleAssignment.tsv #273936
/data/msd/genre/msd-topMAGD-genreAssignment.tsv #406427

#main
/data/msd/main/summary/analysis.csv.gz #239762

#tasteprofile
/data/msd/tasteprofile/mismatches
/sid_matches_manually_accepted.txt 938
/sid_mismatches.txt 19094

/data/msd/tasteprofile/triplets.tsv
/part-00000.tsv.gz 210041
/part-00001.tsv.gz 209199
/part-00002.tsv.gz 208896
/part-00003.tsv.gz 209050
/part-00004.tsv.gz 209315
/part-00005.tsv.gz 210353
/part-00006.tsv.gz 209326
/part-00007.tsv.gz 208540
```

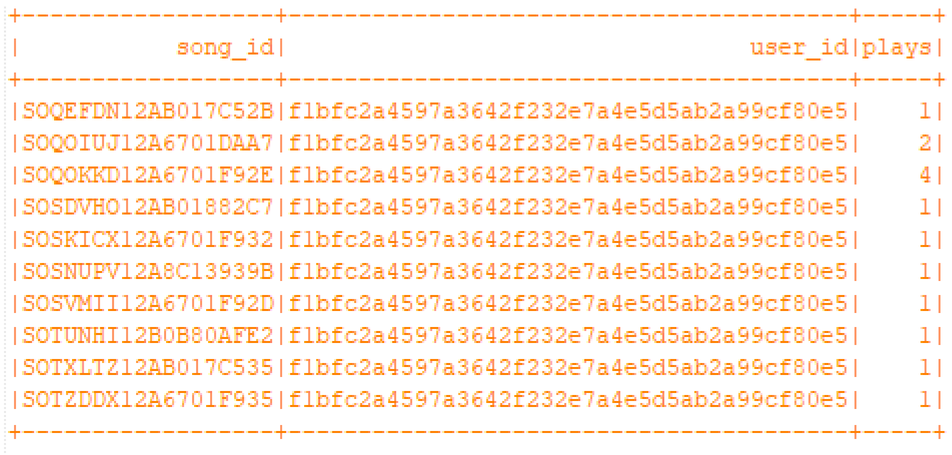
Figure 1.3 the number of rows of files

According to the metadata, there are 994,960 unique songs in the dataset as audio sample. In the ‘audio/features’ subdirectory, each feature dataset includes approximately the same number of rows (nearly 994,623).

1.2 Data Preprocessing

1.2.1 Filtering the Taste Profile dataset

There are two txt files in the mismatch’s directory, including: one file about manually checked matches and other about mismatches without check. There are 19094 records in the mismatched tasteprofile, and 488 matches have been manually accepted in these records. I apply the song_id as join condition and get one new mismatch set that are not manually accepted. After loading and setting schema in triplets file, there are 48,373,586 user-song-play count triplets recorded in this file. Using ‘left_anti’ joining method to remove these ‘mismatches_not_accepted’ items, I get a new clean triplet set (45,795,111 items) are preserved.



```
+------------------+----------------------------------------+-----+
|           song_id|                                 user_id|plays|
+------------------+----------------------------------------+-----+
|SOQEFDN12AB017C52B|f1bfc2a4597a3642f232e7a4e5d5ab2a99cf80e5|    1|
|SOQOIUJ12A6701DAA7|f1bfc2a4597a3642f232e7a4e5d5ab2a99cf80e5|    2|
|SOQOKKD12A6701F92E|f1bfc2a4597a3642f232e7a4e5d5ab2a99cf80e5|    4|
|SOSDVHO12AB01882C7|f1bfc2a4597a3642f232e7a4e5d5ab2a99cf80e5|    1|
|SOSKICX12A6701F932|f1bfc2a4597a3642f232e7a4e5d5ab2a99cf80e5|    1|
|SOSNUPV12A8C13939B|f1bfc2a4597a3642f232e7a4e5d5ab2a99cf80e5|    1|
|SOSVMII12A6701F92D|f1bfc2a4597a3642f232e7a4e5d5ab2a99cf80e5|    1|
|SOTUNHI12B0B80AFE2|f1bfc2a4597a3642f232e7a4e5d5ab2a99cf80e5|    1|
|SOTXLTZ12AB017C535|f1bfc2a4597a3642f232e7a4e5d5ab2a99cf80e5|    1|
|SOTZDDX12A6701F935|f1bfc2a4597a3642f232e7a4e5d5ab2a99cf80e5|    1|
+------------------+----------------------------------------+-----+
```

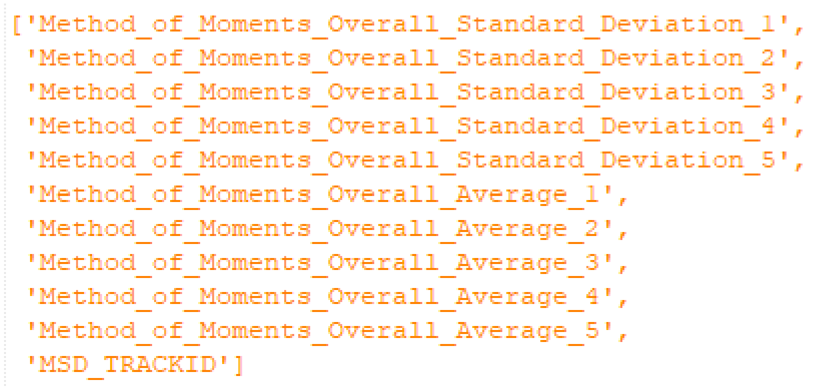
1.2.2 Define schema for the audio feature

In the previous part, we already viewed the data type of each csv file in the feature directory through the corresponding csv file in the attribute’s directory. The attribute files and feature datasets share the same prefix and that the attribute types are named consistently. As the above data exploring, the unique data type of attributes is defined as four types, namely: ‘NUMERIC’, ‘real’, ‘string’ and ‘STRING’. After setting up the mapping and exploring audio dataset file names, the input is the filename in the attribute’s directory. At the first, we can create a empty schemas directory and set the path in order to read ‘{audio_dataset_name}. attributes.csv’ and split line in read lines and save them in different list. Finally, we use StructType() and StructField() to specify the schema to the DataFrame and creating mapping (audio type mapping including ‘NUMERIC’, ‘real’, ‘string’ and ‘STRING’) to generate new schema by mapping the previously segmented items one by one.

2 Audio similarity

2.1 Data Exploratory

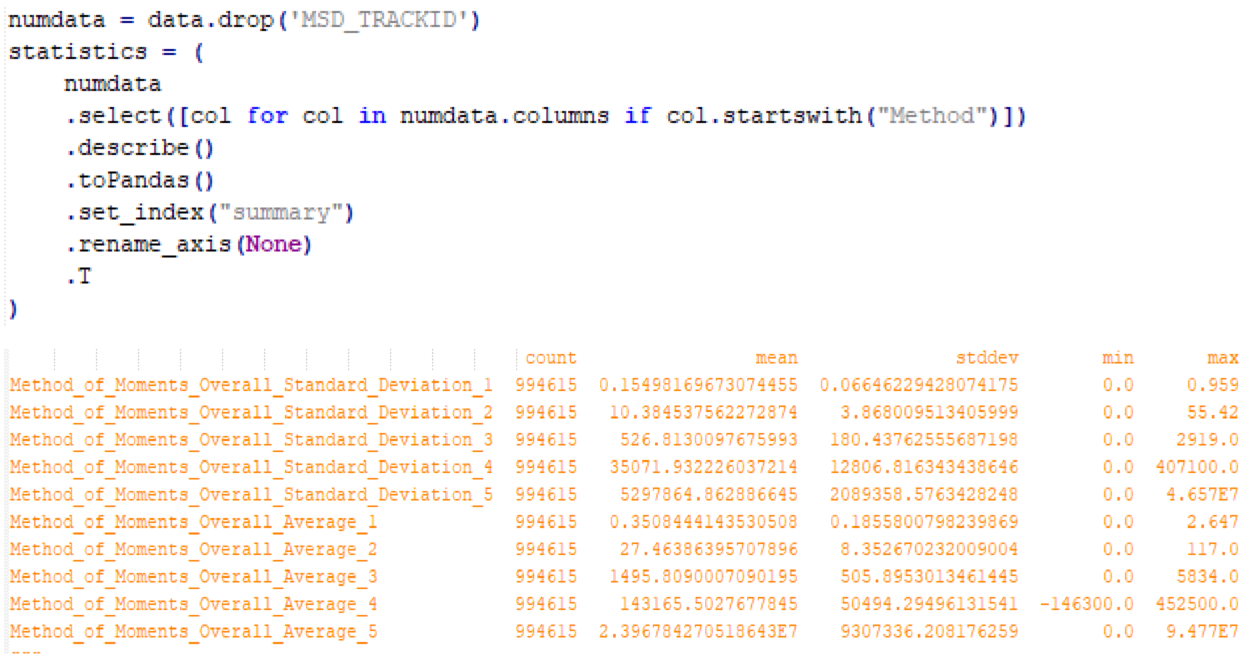
I picked the dataset file ("msd-jmir-methods-of-moments-all-v1.0.csv") with 11 columns in this dataset. The 'MSD_TRACKID' column includes string data type and the remaining columns are numeric as data type, as Figure 2.1 showing:



```
['Method_of_Moments_Overall_Standard_Deviation_1',
 'Method_of_Moments_Overall_Standard_Deviation_2',
 'Method_of_Moments_Overall_Standard_Deviation_3',
 'Method_of_Moments_Overall_Standard_Deviation_4',
 'Method_of_Moments_Overall_Standard_Deviation_5',
 'Method_of_Moments_Overall_Average_1',
 'Method_of_Moments_Overall_Average_2',
 'Method_of_Moments_Overall_Average_3',
 'Method_of_Moments_Overall_Average_4',
 'Method_of_Moments_Overall_Average_5',
 'MSD_TRACKID']
```

Figure 2.1 data items list

After dropping the string column ("MSD_TRACKID"), I uses describe () to generate descriptive statistics (including central tendency, dispersion and shape of the dataset's distribution) and then converts the pyspark data frame into Pandas data frame by using toPandas(). Using set_index(), rename_axis() and .T (transpose) make the data frame easy to read, as the Figure 2.2 shown:

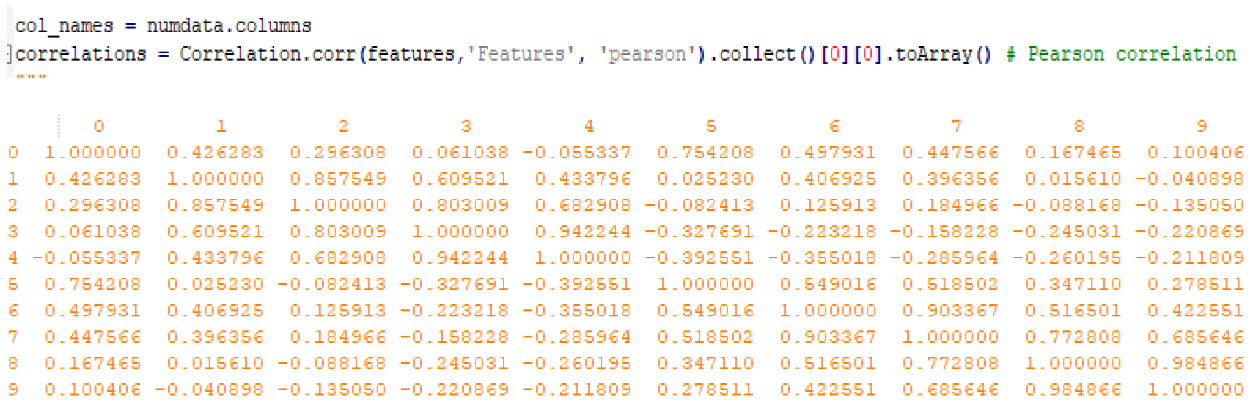


```
numdata = data.drop('MSD_TRACKID')
statistics = (
    numdata
    .select([col for col in numdata.columns if col.startswith("Method")])
    .describe()
    .toPandas()
    .set_index("summary")
    .rename_axis(None)
    .T
)
```

```
                                                count                mean               stddev       min       max
Method_of_Moments_Overall_Standard_Deviation_1  994615  0.15498169673074455  0.06646229428074175       0.0     0.959
Method_of_Moments_Overall_Standard_Deviation_2  994615   10.384537562272874    3.868009513405999       0.0     55.42
Method_of_Moments_Overall_Standard_Deviation_3  994615    526.8130097675993   180.43762555687198       0.0    2919.0
Method_of_Moments_Overall_Standard_Deviation_4  994615   35071.932226037214   12806.816343438646       0.0  407100.0
Method_of_Moments_Overall_Standard_Deviation_5  994615    5297864.862886645   2089358.5763428248       0.0   4.657E7
Method_of_Moments_Overall_Average_1             994615   0.3508444143530508   0.1855800798239869       0.0     2.647
Method_of_Moments_Overall_Average_2             994615    27.46386395707896    8.352670232009004       0.0     117.0
Method_of_Moments_Overall_Average_3             994615   1495.8090007090195    505.8953013461445       0.0    5834.0
Method_of_Moments_Overall_Average_4             994615    143165.5027677845    50494.29496131541 -146300.0  452500.0
Method_of_Moments_Overall_Average_5             994615 2.396784270518643E7    9307336.208176259       0.0   9.477E7
```

Figure 2.2 Each item of statistic result.

I applied Pearson correlation method to compute the matrix correlation by pyspark.ml. stat. Correlation.corr() and finally get correlated ecoefficiencies for each item pairs (Figure 2.3 showing):



```
col_names = numdata.columns
correlations = Correlation.corr(features,'Features', 'pearson').collect()[0][0].toArray() # Pearson correlation
"""
```

```
          0         1         2         3         4         5         6         7         8         9
0  1.000000  0.426283  0.296308  0.061038 -0.055337  0.754208  0.497931  0.447566  0.167465  0.100406
1  0.426283  1.000000  0.857549  0.609521  0.433796  0.025230  0.406925  0.396356  0.015610 -0.040898
2  0.296308  0.857549  1.000000  0.803009  0.682908 -0.082413  0.125913  0.184966 -0.088168 -0.135050
3  0.061038  0.609521  0.803009  1.000000  0.942244 -0.327691 -0.223218 -0.158228 -0.245031 -0.220869
4 -0.055337  0.433796  0.682908  0.942244  1.000000 -0.392551 -0.355018 -0.285964 -0.260195 -0.211809
5  0.754208  0.025230 -0.082413 -0.327691 -0.392551  1.000000  0.549016  0.518502  0.347110  0.278511
6  0.497931  0.406925  0.125913 -0.223218 -0.355018  0.549016  1.000000  0.903367  0.516501  0.422551
7  0.447566  0.396356  0.184966 -0.158228 -0.285964  0.518502  0.903367  1.000000  0.772808  0.685646
8  0.167465  0.015610 -0.088168 -0.245031 -0.260195  0.347110  0.516501  0.772808  1.000000  0.984866
9  0.100406 -0.040898 -0.135050 -0.220869 -0.211809  0.278511  0.422551  0.685646  0.984866  1.000000
```

Figure 2.3 pairs of features correlations

Correlation among the predictors is a problem to be corrected. When the absolute correlation coefficient between two or more independent variables (predictors) is over 0.7 indicates the existence of multicollinearity (Rekha M., 2021), I select these pairs whose coefficients are over 0.8 (as the table 2.1 showing), and find that these items could have collinearity to each other. I will try to solve it later and will manually drop items which have the highest correlating coefficients.

Pairs of correlating items		Correlation Coefficients
Method of Moments Overall Standard Deviation_2	Method of Moments Overall Standard Deviation 3	0.8575494246545305
Method of Moments Overall Standard Deviation 3	Method of Moments Overall Standard Deviation 4	0.8030091088989109
Method of Moments Overall Standard Deviation 4	Method of Moments Overall Standard Deviation 5	0.942244348087381
Method of Moments Overall Average_2	Method of Moments Overall Average 3	0.903367167926527
Method of Moments Overall_Average_4	Method of Moments Overall Average_5	0.9848664625619973

Table 2.1 pairs of features in correlations with coefficient over 0.8

Loading the MSD All Music Genre Dataset (MAGD), there are two columns (‘TRACK_ID’: string and ‘Genre_Name’: string) and 422714 records in this dataset.

Using previous ‘mismatches_not_accepted’ is to filter the MAGD dataset and I finally get 415350 records which are matched song genres. After that, I calculated the counts of each genres and plotted the distribution (Figure 2.3). The Pop_Rock genres account for the most in the dataset, followed by the Electronic and Rap styles respectively in the total 21 different genres.

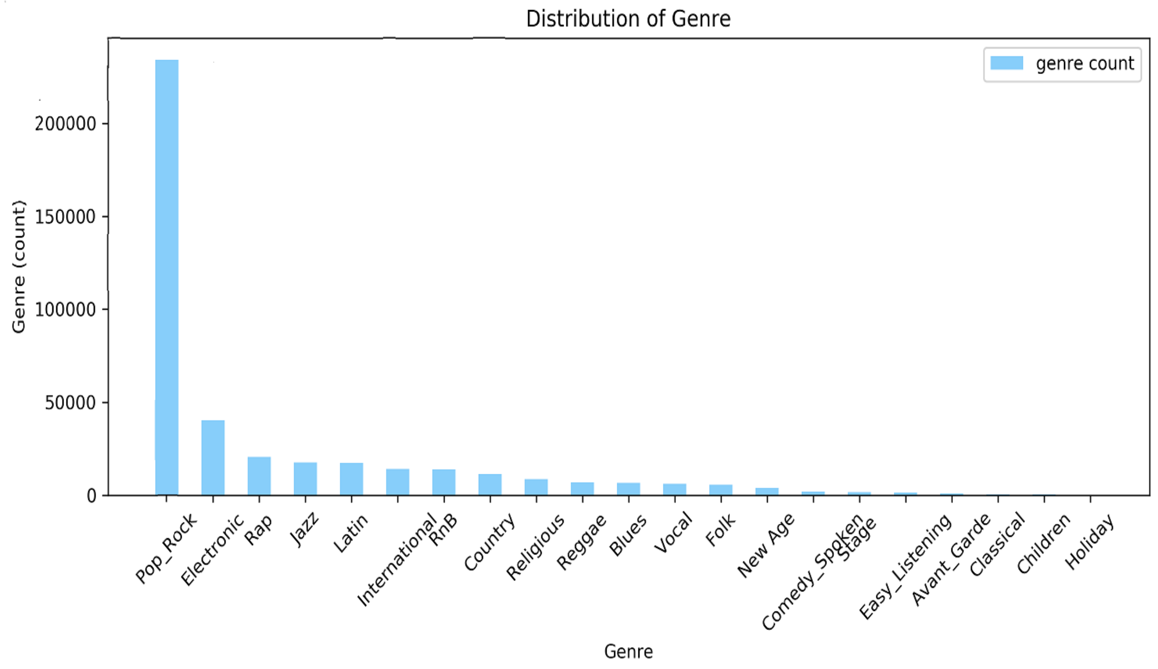


Figure 2.3 Plotting distribution of Genre

Using “TRACK_ID” as a joining column is to merge the genres dataset (MAGD) and audio features dataset and shows 413,289 labeled songs left. Note that the MSD_TRACKID in audio features dataset should be renamed to “TRACK_ID” in order to joining with the MAGD dataset.

2.2 Developing a binary classification model

2.2.1 Exploring classification algorithms

Logistic Regression (lr), Random Forest (rf) and Naïve Bayes (nb) will be implemented in this project as classification algorithms. Before using these algorithms, we will clarify some of their characteristics, such as explain ability, interpretability, predictive accuracy, training speed, hyperparameter tuning, dimensionality, and issues with scaling; especially, we will also explore the applications of these algorithm in Spark.

At the first, let’s show the concepts of these three algorithms:

Logistic Regression is a supervised learning algorithm, mainly used for classification problems. Although the Logistic Regression is called Regression, it is actually a classification model and is often used for dichotomies. it is favored by the industry for its simplicity, parallelization and explicability. In Spark, logistic regression can be used to predict a binary or multi-nominal result.

Random Forest is composed of many decision trees, and there is no correlation between different decision trees. When it is carried out the classification task, new input samples will enter and each decision tree in forest will be allowed to judge and classify separately. Each decision tree will get its own classification result. Which of the classification results of the decision tree has the most classification, then the random forest will take this result as the final result.

Gradient Boosting Trees (GBTs) iteratively trains a sequence of decision trees. the lifting tree uses the addition model and the forward step algorithm to optimize the learning process. When the loss function is square loss and exponential loss function, the optimization of each step is simple, such as square loss function learning residual regression tree. We can find some features about these algorithms in the below table.

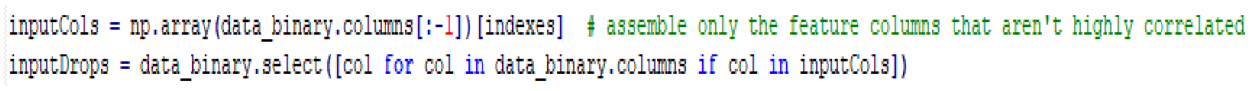
Features	Logistic Regression	Random Forest	Gradient-Boosted Trees (GBTs)
Explain ability & Interpretability	Using binomial logistic regression, or can predict a multiclass outcome by using multinomial logistic regression. Using binary categorical concept to predict the feature result. Easy to interpret	Classification and Regression Combine many decision trees for eliminating overfitting risk. Handling categorical features.	GBTs train one tree at a time, so they can take longer to train than random forests. Random Forests can train multiple trees in parallel. Ensembles of decision trees.
Predictive Accuracy	Correct predictions/total predictions, Confusion Matrix	More trees mean higher accuracy at the cost of slower learning	Better than the other two. Boosting to decision trees.
Training Speed	Slightly fast (1min)	Fast (38s)	Not fast (2min)
Hyperparameter Tuning	No really difference in performance with different solvers	The number of random features to sample at each split point	Regression; Classification; Ranking Careful the parameter
Dimensionality	Tend to overfit the data, particularly in high-dimensional settings	Immunity to the curse of dimensionality Good performance to high dimensionality	Good performance to High Dimensionality
Scaling issues	No need standardization	No need feature scaling	No need feature scaling

Table 2.2.1 Three algorithm features

Because whether or not the datasets have linear integral, The most basic diagnostic of a logistic regression. The faster the training, the better the comprehension. I can use that as a benchmark for comparison. But it is susceptible to extreme values and cannot deal with the high correlation between predictors.

Random forest supports binary and multi-class tags, as well as sequential and categorical features. Random forests also work well when features are at different scales. Random forest is an example of a general approach that always performs well. I want to use a similar classifier. Gradient enhancement tree is a set of decision trees, which minimizes loss function by iteratively training decision tree.

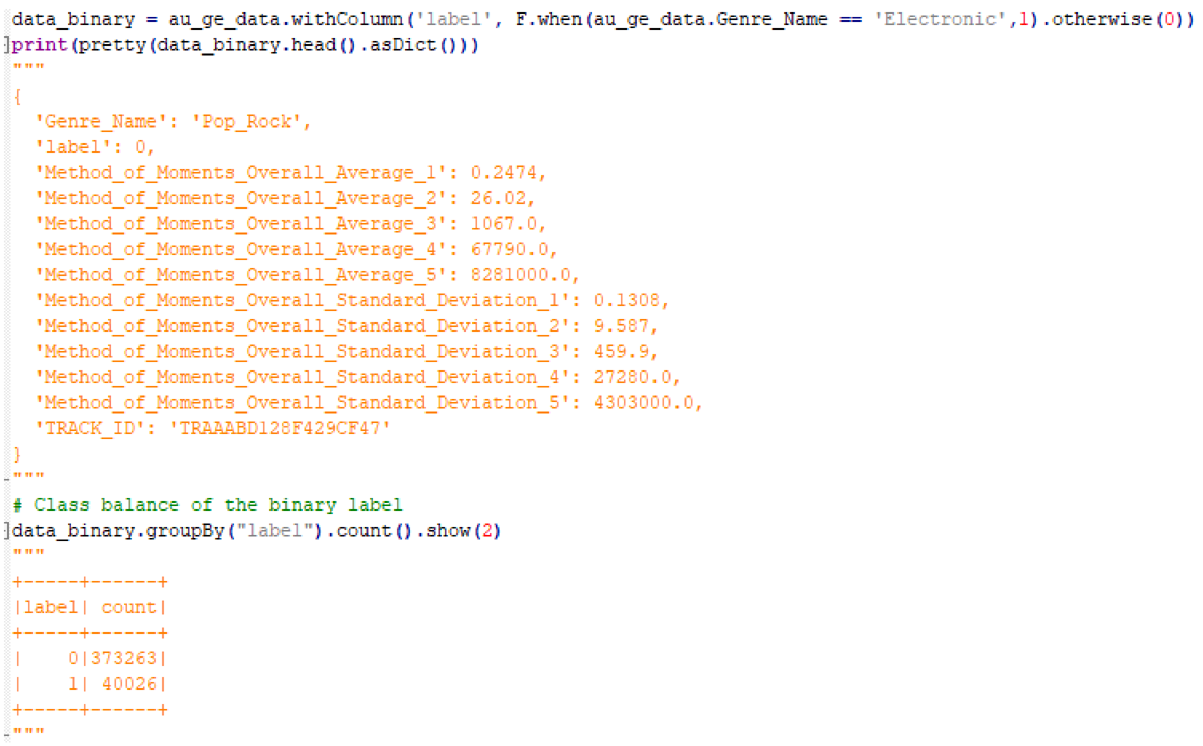
From the above descriptive statistics of audio features, we know that there is strong correlation among the independent variables. Therefore, we tend to remove each of high correlation and assemble vector only from the remaining columns.



```
inputCols = np.array(data_binary.columns[:-1])[indexes]  # assemble only the feature columns that aren't highly correlated
inputDrops = data_binary.select([col for col in data_binary.columns if col in inputCols])
```

2.2.2 Class Balance

In this part, we mainly used ‘label’ as the recording column when selected genre is ‘Electronic’ labelled as 1 and other representing 0. As the final, the genre column is converted into a binary column with one class balance, as the below graph showing:



```
data_binary = au_ge_data.withColumn('label', F.when(au_ge_data.Genre_Name == 'Electronic',1).otherwise(0))
print(pretty(data_binary.head().asDict()))
"""
{
  'Genre_Name': 'Pop_Rock',
  'label': 0,
  'Method_of_Moments_Overall_Average_1': 0.2474,
  'Method_of_Moments_Overall_Average_2': 26.02,
  'Method_of_Moments_Overall_Average_3': 1067.0,
  'Method_of_Moments_Overall_Average_4': 67790.0,
  'Method_of_Moments_Overall_Average_5': 8281000.0,
  'Method_of_Moments_Overall_Standard_Deviation_1': 0.1308,
  'Method_of_Moments_Overall_Standard_Deviation_2': 9.587,
  'Method_of_Moments_Overall_Standard_Deviation_3': 459.9,
  'Method_of_Moments_Overall_Standard_Deviation_4': 27280.0,
  'Method_of_Moments_Overall_Standard_Deviation_5': 4303000.0,
  'TRACK_ID': 'TRAAABD128F429CF47'
}
"""
# Class balance of the binary label
data_binary.groupBy("label").count().show(2)
"""
+-----+------+
|label| count|
+-----+------+
|    0|373263|
|    1| 40026|
+-----+------+
"""
```

We can find that Class_0 = 373263 and Class_1 = 40026. There is a big difference. The imbalanced sample categories will lead to the classification with small sample size containing too few features, and it is difficult to extract rules from them. Even if the classification model is obtained, it is prone to over-dependence and limited data samples, leading to over-fitting problems. When the model is applied to new data, the prediction effect of the model will be poor, and the realization of the concerned evaluation indicators will be not ideal. In the case of imbalanced data categories, adopting the default classification threshold may result in all the output being counter-examples, resulting in false high accuracy and classification failure. Note that downsampling can cause the condition positive unchanged, but the number of condition negative is reduced, meaning FP and TN are reduced

2.2.2 Data Split and Resampling

Since this is an imbalanced classification problem, several approaches, such as sample-based approaches, selecting appropriate indicators, using models based on cost functions, or finding the best thresholds, can mitigate this problem.

In this part, after random splitting the training and test data, I tried three sampling method (stratified sampling, downsampling, and observation weighting), as shown below.

Sampling	Positive Ratio	Negative Ratio	Total Number
Random Split (no sampling)	0.097375	0.902625	330680
Exact Stratification sample	0.096845	0.903155	330630
Downsampling	0.332517	0.667483	96356
Observation weight	0.096602	0.903398	330645

Table 2.2.2 rates of observations in each class in training data

Also, we can find training data after random splitting performs, the ratio of Class_1 to Class_0 is 1:9, and the exact stratification sample and observation weight show similar ratios. However, the downsampling shows a different result than sampling the train data. In the later modelling, we will try the downsampling method for the training set and compare the final model with a basic model.

2.2.3 Train Three Classification Algorithms

I referred the lecture codes to write two functions to custom prediction with the specific threshold and count the true-positive, false-positive, false-negative and true-negative of the confusion matrix so that get precision, accuracy and recall based on the prediction of the model performed on the test dataset. By importing LogisticRegression(), RandomForestClassifier() and GBTClassifier() functions from spark MLlib's classification. During printing binary confusion metrics, the threshold is set up 0.5. I already deal with the intercorrelations issues before.

Model	recall	accuracy	precision
Logistic Regression	0.0034	0.9018	0.18
Random Forest	NA	NA	NA
GBT	0.0594	0.9039	0.5767

Table2.2.3 (1) Basic Model with Logistic Regression, Random Forest & GBT

Model with downsampling training	recall	accuracy	precision	time
Logistic Regression	0.2584	0.8690	0.3013	1min48s
Random Forest	0.8609	0.3576	0.3139	38s
GBT	0.4387	0.8465	0.3028	2min08s

Table2.2.3 (2) Model with Logistic Regression, Random Forest & GBT after Downsampling train data

Through the observation of the above table, I found that when the model uses the training data after downsampling, its recall and precision are significantly improved. Precision (also known as positive predictive value) refers to the proportion of relevant instances among retrieved instances, and recall (also known as sensitivity) refers to the proportion of retrieved relevant instances. Therefore, both precision and recall are based on correlation. It is interesting when we use the unprocessed model to create confusion metrics, since nP, TP, FP are all zero. This phenomenon of overfitting is very likely caused by class imbalance. Therefore, using downsampling to solve the class imbalance problem is important in this project.

Logistic Regression and GBT have the similar better performance but GBT might take longer time. Random Forest is the fastest model but its accuracy is the lowest. It is possible that Random Forest algorithm is based on decision tree, and decision tree is more sensitive to the class balance. Logistic Regression do not need complex tuning hyperparameter and directly model the possibility of classification. However, Logistic Regression is also sensitive to multicollinear issues. It should be considered by us as well.

2.2.4 Tuning the Hyperparameters

Many models have important parameters that cannot be estimated directly from the data. For example, numTrees and maxDepth are the main parameters of a random forest. Adding numTrees will reduce the variance in the prediction, but the training time increases linearly. Adding maxDepth makes the model more expressive and functional. However, deep trees take longer to train. GBT is also derived from the decision tree, so it can be apparently affected in these similar parameters. In general, Logistic Regression is not influenced by the hyperparameters. When I try to normalize it and tune the parameter, I find the better parameter group.

We would try to check the three algorithms after tuning some of hyperparameter, and use 5-fold cross validation to evaluate them. Because we need to created parameter grid for cross validation.

Model	Precision	recall	accuracy	auroc
Logistic Regression	0.3198	0.2042	0.8798	0.6872
Random Forest	0.3149	0.4292	0.8530	0.7721
GBT	0.3131	0.4722	0.8472	0.7845

Hyperparameters optimization improve the performance metrics of these three models, and Random Forest model has the huge changes in accuracy. Logistic Regression has slightly change.

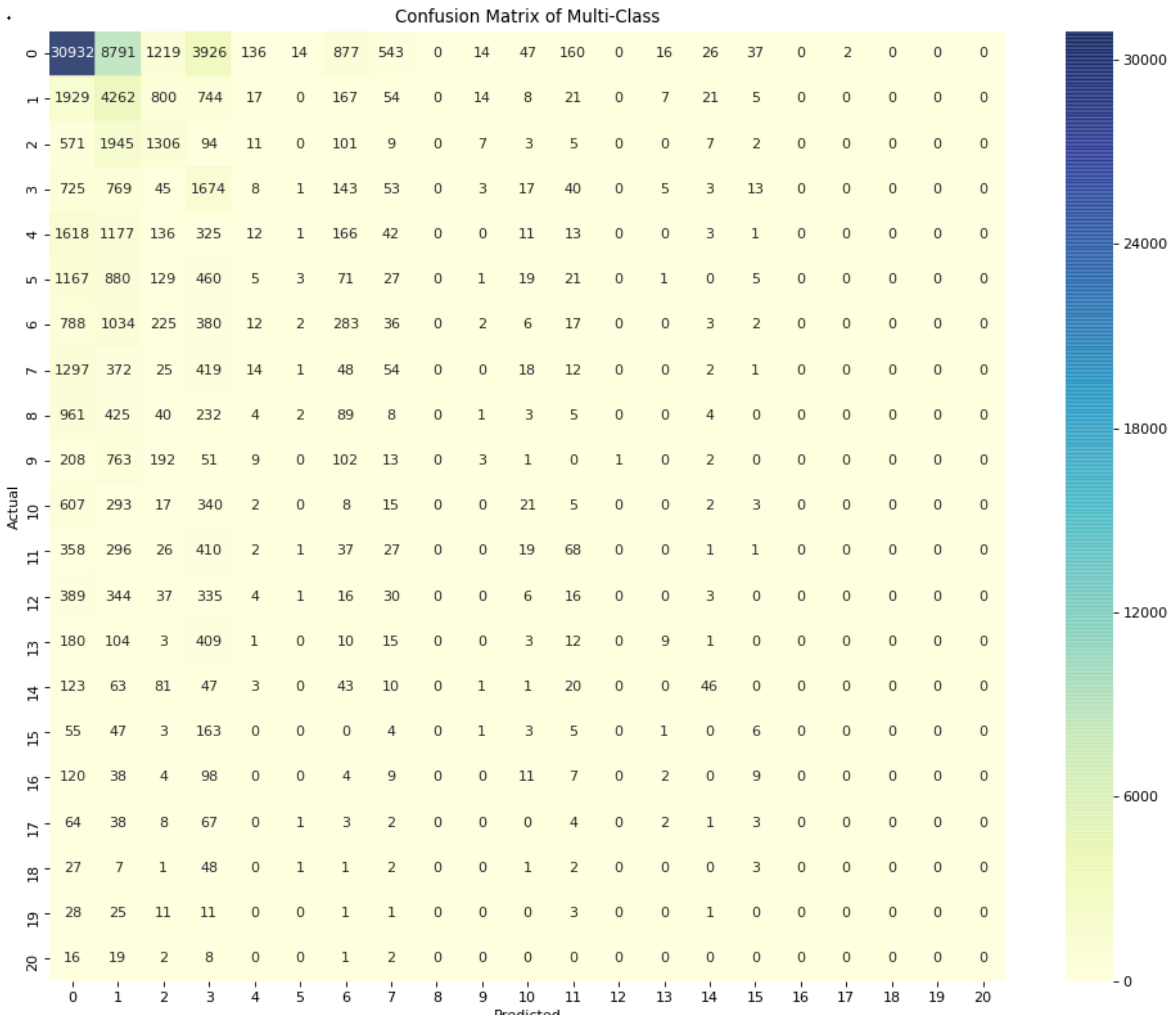
2.2.5 Predict across all genres

We find the best model is logistic regression, but it cannot be directly used to predict multiple class. Multiclass classification is a classification technique that allows us to classify data with more than two class tags. Trained multi-class classifiers are able to predict test labels based on the data in the training data. For example, there are 3 classes in one multi class classification. If we try to use Logistic Regression to predict, we have to train 3 different logistic regression classifiers. When training the classifier for first one, we will treat input data with class 1 label with x ==1 and all other as x==0 or other labels. This will continue for all the classes. For the test input, we should use a trained logistic regression classifier to calculate the "probability" that it belongs to each class. Your pair of all prediction functions will select the class with the highest probability output by the corresponding logistic regression classifier and return the class label (1, 2,... , or K) as an input example for prediction.

To perform multiclass classification, the genre column was converted into an integer index which represents the genre consistently using StringIndexer() function. VectorAssembler() function is to integrate the data for these ‘Method_’ columns and creates a new column called as ‘new_features’. We use random splitting and downsampling to avoid class imbalance. Cross Validation with tuning hyperparameter helps us to evaluate the model. The dataset includes 21 classes. After checking the class balance, we find that the small data has 3925 and the large data has 5701. Thus to study impact of class balancing on multiclass classification, downsample should be used to cut the large data for keeping class balance. After that, we try to use the Logistic Regression to fit train_downsampled and use test data to transform to the model and get actual and prediction data. Finally, we use the confusion matrix function to create and visualization as the below graph.

From the below confusion metrics, we can find the model is not good, even though the model has been working through cross validation and tuning. We can find the left top corner of the metrics in the graph, there are a apparent confusion metrics relations. The possible reason could be the number of data is not enough or the modelling parameters could be tuned more choices.

```
"""
              precision    recall  f1-score   support

        0.0       0.00      0.00      0.00     46740
        1.0       0.13      0.67      0.22      8049
        2.0       0.29      0.34      0.31      4061
        3.0       0.14      0.53      0.23      3499
        4.0       0.05      0.11      0.07      3505
        5.0       0.05      0.06      0.05      2789
        6.0       0.12      0.13      0.13      2790
        7.0       0.05      0.11      0.07      2263
        8.0       0.02      0.02      0.02      1774
        9.0       0.06      0.00      0.00      1345
       10.0       0.09      0.05      0.06      1313
       11.0       0.14      0.06      0.09      1246
       12.0       0.00      0.00      0.00      1181
       13.0       0.16      0.04      0.06       747
       14.0       0.37      0.11      0.17       438
       15.0       0.01      0.00      0.01       288
       16.0       0.00      0.00      0.00       302
       17.0       0.10      0.01      0.01       193
       18.0       0.00      0.00      0.00        93
       19.0       0.00      0.00      0.00        81
       20.0       0.00      0.00      0.00        48

avg / total       0.05      0.12      0.06     82745
```



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	30932	8791	1219	3926	136	14	877	543	0	14	47	160	0	16	26	37	0	2	0	0	0
1	1929	4262	800	744	17	0	167	54	0	14	8	21	0	7	21	5	0	0	0	0	0
2	571	1945	1306	94	11	0	101	9	0	7	3	5	0	0	7	2	0	0	0	0	0
3	725	769	45	1674	8	1	143	53	0	3	17	40	0	5	3	13	0	0	0	0	0
4	1618	1177	136	325	12	1	166	42	0	0	11	13	0	0	3	1	0	0	0	0	0
5	1167	880	129	460	5	3	71	27	0	1	19	21	0	1	0	5	0	0	0	0	0
6	788	1034	225	380	12	2	283	36	0	2	6	17	0	0	3	2	0	0	0	0	0
7	1297	372	25	419	14	1	48	54	0	0	18	12	0	0	2	1	0	0	0	0	0
8	961	425	40	232	4	2	89	8	0	1	3	5	0	0	4	0	0	0	0	0	0
9	208	763	192	51	9	0	102	13	0	3	1	0	1	0	2	0	0	0	0	0	0
10	607	293	17	340	2	0	8	15	0	0	21	5	0	0	2	3	0	0	0	0	0
11	358	296	26	410	2	1	37	27	0	0	19	68	0	0	1	1	0	0	0	0	0
12	389	344	37	335	4	1	16	30	0	0	6	16	0	0	3	0	0	0	0	0	0
13	180	104	3	409	1	0	10	15	0	0	3	12	0	9	1	0	0	0	0	0	0
14	123	63	81	47	3	0	43	10	0	1	1	20	0	0	46	0	0	0	0	0	0
15	55	47	3	163	0	0	0	4	0	1	3	5	0	1	0	6	0	0	0	0	0
16	120	38	4	98	0	0	4	9	0	0	11	7	0	2	0	9	0	0	0	0	0
17	64	38	8	67	0	1	3	2	0	0	0	4	0	2	1	3	0	0	0	0	0
18	27	7	1	48	0	1	1	2	0	0	1	2	0	0	0	3	0	0	0	0	0
19	28	25	11	11	0	0	1	1	0	0	0	3	0	0	1	0	0	0	0	0	0
20	16	19	2	8	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 2.2.5 Confusion Matrix

3. Song Recommendations

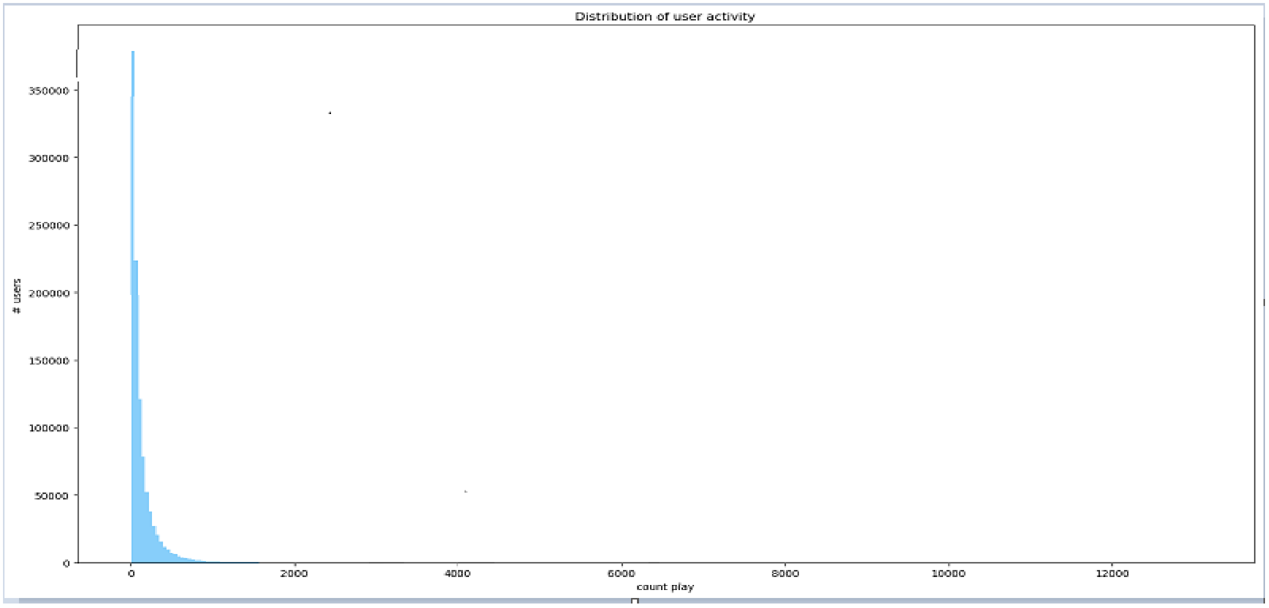
3.1 Preparation for Data

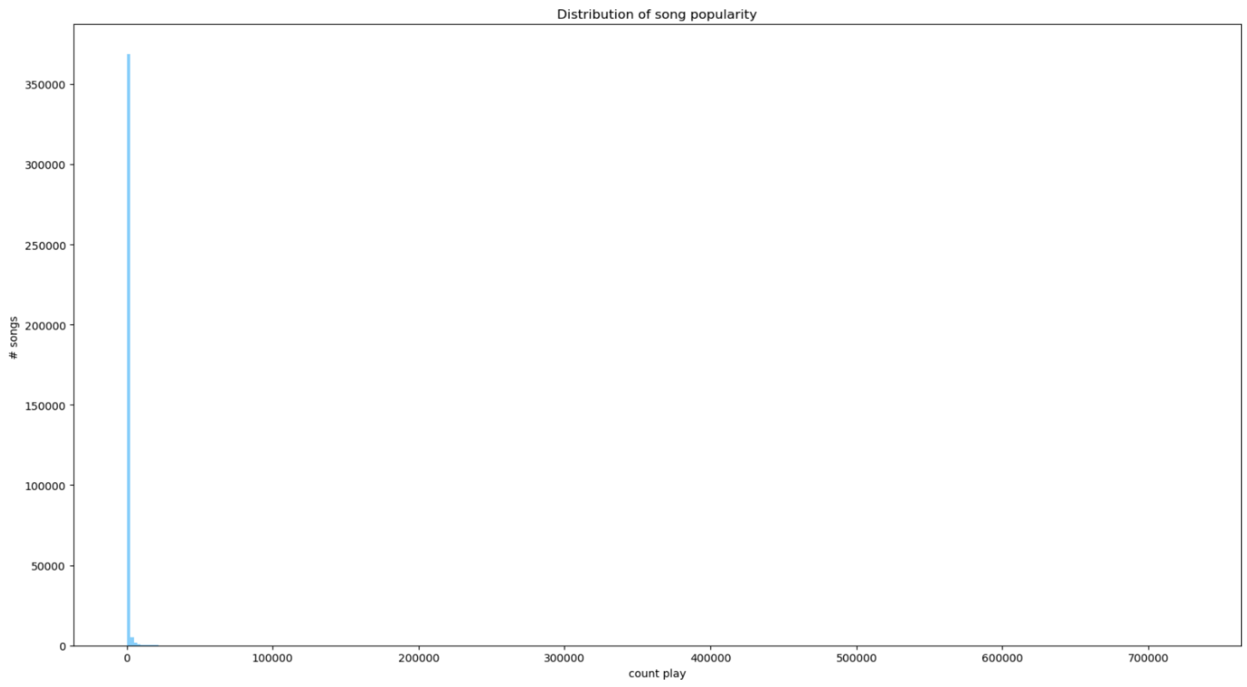
There are 378310 unique songs and 1019318 users in Taste Profile dataset. I used the triplets_not_mismatched dataset which is get from the DataProcessing second part. I selected each of the user ID column and song ID column.

The most active user played 195 unique songs with only occupying the total number of the unique songs (0.52%). I grouped the user ID and the played count to get the descending order of the total number of songs that each user played. The first one was the most active user who had 13074 total plays. I filter the song played times by the most active user by the user ID. We get the top ten of the count of songs that each user is shown below.

```
+----------------------------------------+----------+----------
|user_id                                 |song_count|play_count
+----------------------------------------+----------+----------
|093cb74eb3c517c5179ae24caf0ebec51b24d2a2|195       |13074
|119b7c88d58d0c6eb051365c103da5caf817bea6|1362      |9104
|3fa44653315697f42410a30cb766a4eb102080bb|146       |8025
|a2679496cd0af9779a92a13ff7c6af5c81ea8c7b|518       |6506
|d7d2d888ae04d16e994d6964214a1de81392ee04|1257      |6190
|4ae01afa8f2430ea0704d502bc7b57fb52164882|453       |6153
|b7c24f770be6b802805ac0e2106624a517643c17|1364      |5827
|113255a012b2affeab62607563d03fbdf31b08e7|1096      |5471
|99ac3d883681e21ea68071019dba828ce76fe94d|939       |5385
|6d625c6557df84b60d90426c0116138b617b9449|1307      |5362
+----------------------------------------+----------+----------
```

I group song ID and sum the played count to get the total number of each song are played. I import the matplotlib to plot the distribution of song popularity and the distribution of the user activity.

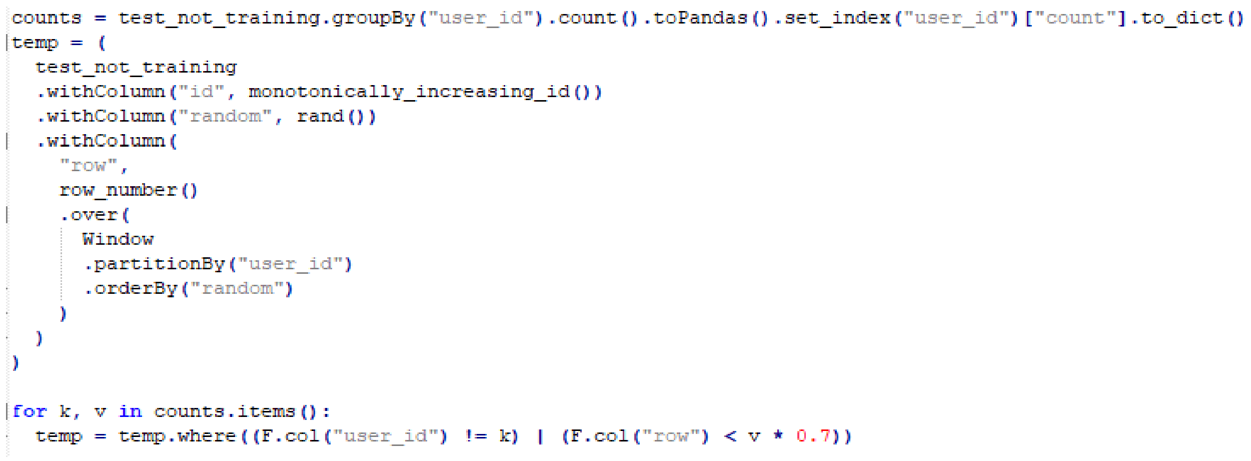




The song popularity distribution map shows that the song popularity distribution is skewed to the right, which is the resulting activity with low data set and user boundaries.

I transferred the data frame of song popularity and user activity into a panda data frame and used the Pandas Quantile() method to get the first quantile of user activity (N=32) and the same method to get the first quantile of song popularity (M=8). I filtered users and songs less than 32 and 8 using the first quantile. 254,739 users played songs less than 32, and 90,750 songs played less than 8. Left-anti join is used to remove user IDs with fewer than 32 plays and song IDs with fewer than 8 plays in the Taste Profile dataset. The filtered dataset has 41978747 rows. The dataset should remove songs which have been played less than N times and users who have listened to fewer than M songs in total. We can use inner join method for the triplets_not_mismatched grouped by user_id and song_id which only is larger than the threshold.

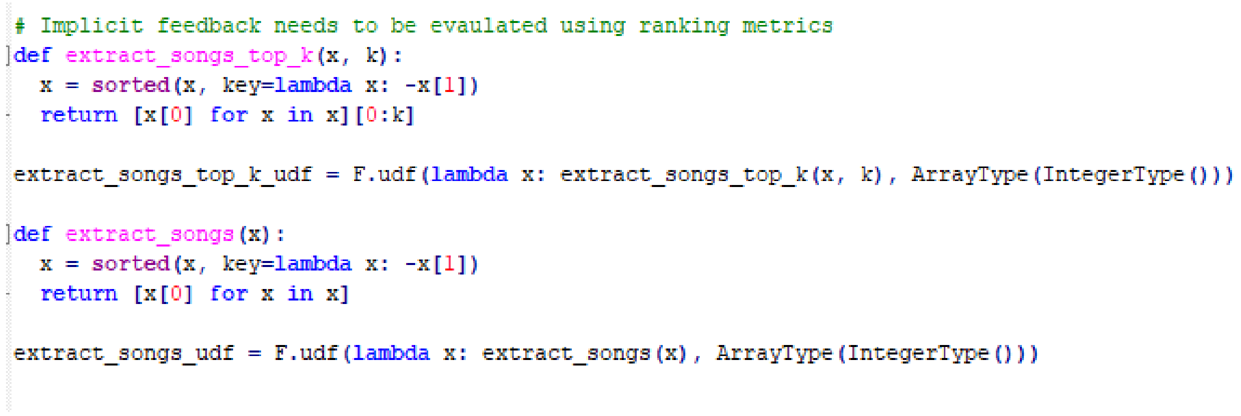
After encoding the user_id and song_is as string index, I randomly split the dataset to get 70% train data and 30% test data. I need to ensure that every uer in the test set has some user-song plays in the training set as well. I grouped by the user_id and converted it to panda data frame with setting index as count. I used the for loop to run the whole data to check whether or not there are test data do not keep in training data. It needs to confirm all the test should be in the training set. It is one cold start issue. The project cold start problem is when a project is added to a catalog with no or very little interaction. This is primarily a problem with collaborative filtering algorithms, as they rely on item interactions to make recommendations.(Yusp,2021)



```
counts = test_not_training.groupBy("user_id").count().toPandas().set_index("user_id")["count"].to_dict()
temp = (
  test_not_training
  .withColumn("id", monotonically_increasing_id())
  .withColumn("random", rand())
  .withColumn(
    "row",
    row_number()
    .over(
      Window
      .partitionBy("user_id")
      .orderBy("random")
    )
  )
)

for k, v in counts.items():
  temp = temp.where((F.col("user_id") != k) | (F.col("row") < v * 0.7))
```

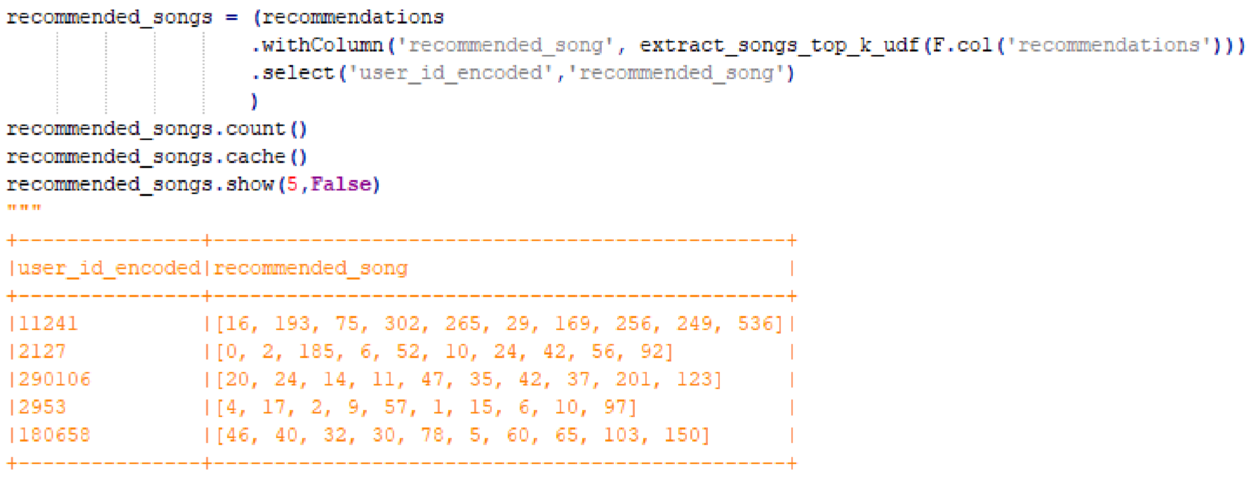
3.2 Collaborative Filtering Model

Using the Alternating Least Squares (ALS) to train implicit matrix factorization which is a class of collaborative filtering algorithm which can be used for recommender system. rank is the number of features to use (also referred to as the number of latent factors). iterations is the number of iterations of ALS to run. ALS recommender is a matrix factorization algorithm that uses Alternating Least Squares with Weighted-Lamda-Regularization (ALS-WR). It factors the user to item matrix A into the user-to-feature matrix U and the item-to-feature matrix M: It runs the ALS algorithm in a parallel fashion. For easier using, ‘play’ column is renamed as ‘rating’ in train and test datasets. Finally, I used RMSE to evaluate the feedback. UDF (user define function) was used to extract recommended songs from the array.



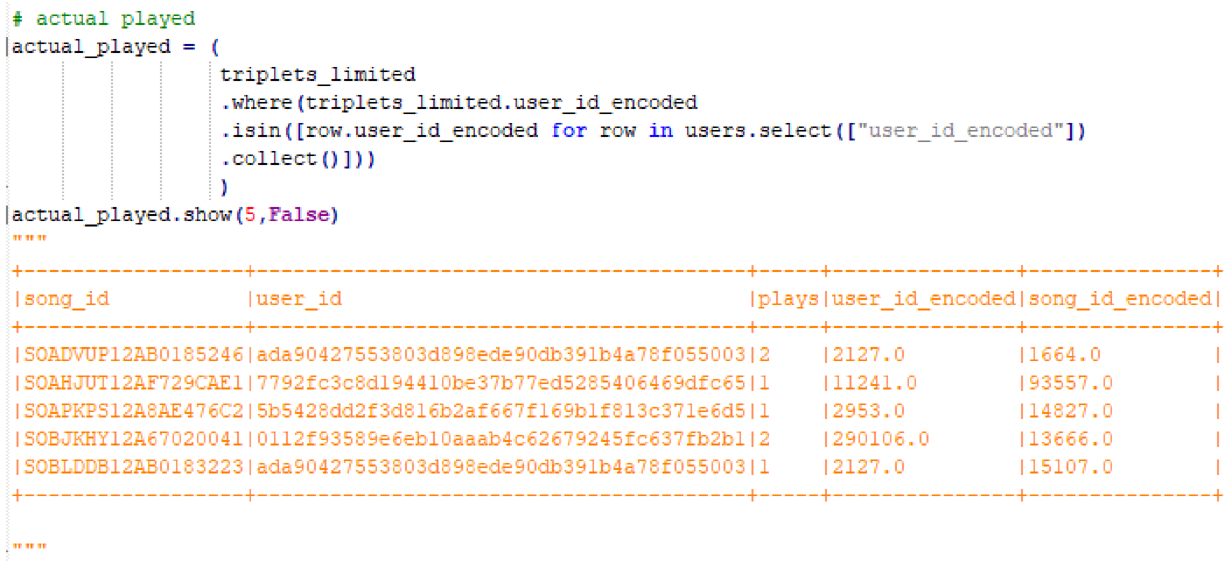
```
# Implicit feedback needs to be evaulated using ranking metrics
def extract_songs_top_k(x, k):
  x = sorted(x, key=lambda x: -x[1])
  return [x[0] for x in x][0:k]

extract_songs_top_k_udf = F.udf(lambda x: extract_songs_top_k(x, k), ArrayType(IntegerType()))

def extract_songs(x):
  x = sorted(x, key=lambda x: -x[1])
  return [x[0] for x in x]

extract_songs_udf = F.udf(lambda x: extract_songs(x), ArrayType(IntegerType()))
```

Using the above UDF function we can get the below result.



```
recommended_songs = (recommendations
                     .withColumn('recommended_song', extract_songs_top_k_udf(F.col('recommendations')))
                     .select('user_id_encoded','recommended_song')
                     )
recommended_songs.count()
recommended_songs.cache()
recommended_songs.show(5,False)
"""
+---------------+--------------------------------------------+
|user_id_encoded|recommended_song                            |
+---------------+--------------------------------------------+
|11241          |[16, 193, 75, 302, 265, 29, 169, 256, 249, 536]|
|2127           |[0, 2, 185, 6, 52, 10, 24, 42, 56, 92]      |
|290106         |[20, 24, 14, 11, 47, 35, 42, 37, 201, 123]  |
|2953           |[4, 17, 2, 9, 57, 1, 15, 6, 10, 97]         |
|180658         |[46, 40, 32, 30, 78, 5, 60, 65, 103, 150]   |
+---------------+--------------------------------------------+
```

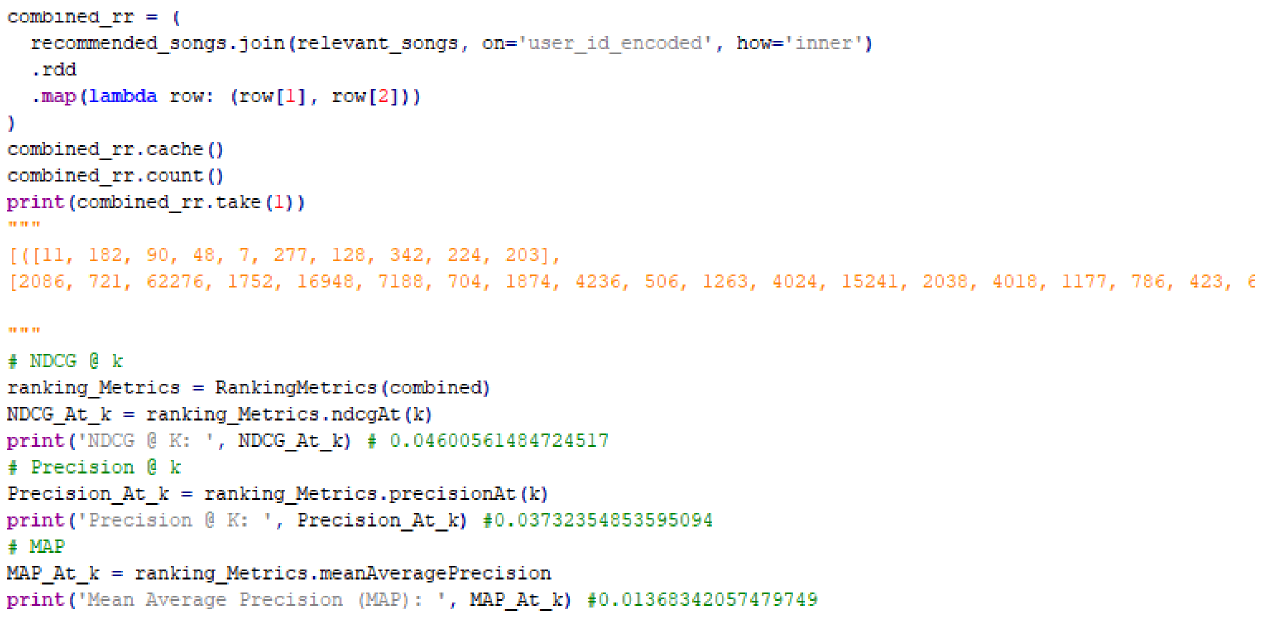
Actual played songs for each user are collected from the test using isin() and collect().



```
# actual played
actual_played = (
                triplets_limited
                .where(triplets_limited.user_id_encoded
                .isin([row.user_id_encoded for row in users.select(["user_id_encoded"])
                .collect()]))
                )
actual_played.show(5,False)
"""
+------------------+----------------------------------------+-----+---------------+---------------+
|song_id           |user_id                                 |plays|user_id_encoded|song_id_encoded|
+------------------+----------------------------------------+-----+---------------+---------------+
|SOADVUP12AB0185246|ada90427553803d898ede90db391b4a78f055003|2    |2127.0         |1664.0         |
|SOAHJUT12AF729CAE1|7792fc3c8d194410be37b77ed5285406469dfc65|1    |11241.0        |93557.0        |
|SOAPKPS12A8AE476C2|5b5428dd2f3d816b2af667f169b1f813c371e6d5|1    |2953.0         |14827.0        |
|SOBJKHY12A67020041|0112f93589e6eb10aaab4c62679245fc637fb2b1|2    |290106.0       |13666.0        |
|SOBLDDB12AB0183223|ada90427553803d898ede90db391b4a78f055003|1    |2127.0         |15107.0        |
+------------------+----------------------------------------+-----+---------------+---------------+

"""
```

We can measure the effectiveness of the recommendation system by the precision of the recommender system. Firstly, we can select user_id_encoded , song_id_encoded and ratings and aggressive the user and preference with grouping the user encoded. Secondly, we can filter the relevant songs for selected users. We can get one table including user_id_encoded and relevant songs. We choose one user (user encoded 290106) and extract the relevant songs list. We can compare the recommended list and relevant list. The result is not satisfactory.

I try to use the metadata to explore the matching songs and information with joining and filtering recommended list. User the test set user_song plays and recommendations from the collaborative filtering model to compute result. The metadata provided in the MSD summary can provide more detailed preferences and drill down to the most sensitive findings of the song collection. I noticed that there is a lot of research on the relationship between artist popularity and song popularity, these are very sensitive methods, more explicit information more relationships can be considered, new models can be implemented



```
combined_rr = (
  recommended_songs.join(relevant_songs, on='user_id_encoded', how='inner')
  .rdd
  .map(lambda row: (row[1], row[2]))
)
combined_rr.cache()
combined_rr.count()
print(combined_rr.take(1))
"""
[([11, 182, 90, 48, 7, 277, 128, 342, 224, 203],
[2086, 721, 62276, 1752, 16948, 7188, 704, 1874, 4236, 506, 1263, 4024, 15241, 2038, 4018, 1177, 786, 423, 6

"""
# NDCG @ k
ranking_Metrics = RankingMetrics(combined)
NDCG_At_k = ranking_Metrics.ndcgAt(k)
print('NDCG @ K: ', NDCG_At_k) # 0.04600561484724517
# Precision @ k
Precision_At_k = ranking_Metrics.precisionAt(k)
print('Precision @ K: ', Precision_At_k) #0.03732354853595094
# MAP
MAP_At_k = ranking_Metrics.meanAveragePrecision
print('Mean Average Precision (MAP): ', MAP_At_k) #0.01368342057479749
```

Precision@K: The metrics shines for binary ratings, not fit for fine-grained numeric ratings. In recommendations list, relevant items are rewarded the same no matter their positions. Ideally, relevant items that are towards the bottom should be rewarded less. And non-relevant items should be penalized. MAP: The metrics shines for binary ratings, not fit for fine-grained numeric ratings. The penalty of MAP is linear, it's too hard.

NDCG: We needs to manually handle the relevance scores. When we have a new user that we have no information about him at all, it is an issue. And NDCG has some issues with partial feedback. When we don't have complete ratings, we need to decide how to impute the missing ratings. In online way, we have users to generate feedback in real time. Going through A/B test, and we could use metrics like click-through rate(CTR), customer lifetime value(CLTV).

In real world, A/B test is the only robust way to assess the performance of a ranking model. then we can evaluate any business metric that is associated with the system(product) using our model. For example, if we generate five ranked recommendations to put in a marketing email, we could look at the change in sales revenue from those items. these offline metrics, where Di is based on a finite set of observations about ui will near be very robust; however, they are well defined and consistent and will enable you to compare models to choose the best candidates in real world. In today's recommender systems, both implicit and explicit ratings are considered to recommend items or content. When dealing with collaborative filtering, it might be a normalization approach, focusing on some unusual patterns (unusual taste leads to harmful recommendations), finding out what is causing them, and removing them from the model. Improvement suggestions includes requirements, features and development for every system or users' personal issues could be critical leading to inaccuracy problems. Study based recommendation is based on demand and preference of users and can automatically record and calculate what users do need.

Reference

Rekha M. (2021, December 10). Correlation analysis and Collinearity | Data science | Multicollinearity | Clairvoyant Blog. Medium. https://blog.clairvoyantsoft.com/correlation-and-collinearity-how-they-can-make-or-break-a-model-9135fbe6936a

Yusp. (2022, May 12).How recommendation systems tackle the Cold Start Problem - YUSP Blog. Retrieved June 10, 2022, from https://www.yusp.com/blog-posts/cold-start-problem/

Data420 Lecture example codes.