# Assignment 6: High Fidelity

*Ankita, Jiangtao*

*10/26/2017*

## Objective

Given a million song dataset, learn the basic of sparks by retrieving the number of distant songs, artists, albums and numerous top 5 attributes from the dataset.

## Dataset

We are using a dataset based on the metadata in the Million Song Database.

## Preparing Data

- Checks done to remove empty entries in the `track_id` and `artist_id` columns.
- Check done to dis-regard non-float entries in `duration` , `tempo` , `song_hotness` , `artist_hotness` and `key-confidence` columns.

## Implementation

- Our solution first takes the two files 'song_info.csv' and 'artist_terms.csv' and converts it into `mapPartitions` that converts each partition of the source RDD into multiple elements of the result.
- After that, for each task like finding the distincts and top'5, we select only the required column from the RDD thus avoiding reading all the columns of the dataset everytime. This improves the execution time as well.

## Assumptions

- We have used Track_Id instead of Title for all our calculation, because each `track_id` is unique to a song while a single `song_id` and `title` can belong to multiple tracks.
- We have also used Artist_Id instead of Artist_Names because in case of collaboration of two artists, a new Artist_Id is not created so we assume that the collaboration work belongs to the Artist_Id it is mapped to.

## Performance and Observation

- Our program takes a total average time of 4.12 minutes for 5 iterations on the big corpus.
- For the subset dataset, the program took an average time of 8 seconds for 5 iterations.
- We observed that for the big corpus, one job -'Top 5 hottest genre' takes approximately 2.25 minutes. The reason for this is, a join operation is being performed between `song_info` and `artist_terms` to find the mean artist hotness in `artist_terms` .
- For the small dataset, the join finished in less then 2 seconds but for large corpus, each stage had a lot tasks, since the tasks are proportional to the total number of partitions the operation need to handle. Hence there is a lot of Shuffle and spill operations thus causing the program to slow down.

## Output

Following are the results when running the program on large dataset.

- Number of Distinct songs :
    - Here we filter the track_id's and do a distinct count on them.

```
1000000
```

- Number of Distinct artists :
    - Here we filter the artist_id's and do a distinct count on them.

```
44745
```

- Number of Distinct albums :
    - Here we filter the artist_id's and album and do a distinct on them.
    - Then we use foldLeft and countByKey(i.e artist_id) all the distinct albums.

```
221753
```

- Top 5 loudest songs :
    - Here we filter the track_id's and loudness and create a map with (K,V) -> (Track_Id, Loudness).
    - Then we do a descending sort on loudness and pick the top 5 loudest songs.

```
(Track Id, Loudness)
(TRDZGER12903CD386D,4.318)
(TRXFHGZ12903CD2C1D,4.3)
(TRZVIPO12903D01BA4,4.231)
(TRONJMK12903CFCCC4,4.166)
(TRXDEFB128F426EA6A,4.15)
```

- Top 5 longest songs :
    - Here we filter the track_id's and duration and create a map with (K,V) -> (Track_Id, Duration).
    - Then we do a descending sort on duration and pick the top 5 longest songs.

```
(Track Id, Length)
(TRDZTTO12903CF1A2E,3034.9058)
(TRVFVTA128F421E809,3033.5996)
(TRSMLIB128F934C0A8,3033.4429)
(TRPIWVS128F4289D7F,3032.7637)
(TRPWIUP128F426B47B,3032.5808)
```

- Top 5 fastest songs :
    - Here we filter the track_id's and tempo and create a map with (K,V) -> (Track_Id, tempo).
    - Then we do a descending sort on tempo and pick the top 5 fastest songs.

```
(Track Id, Tempo)
(TRPPDKE128F930D9C0,302.3)
(TRNPTWJ128F93136D2,296.469)
(TRFWRVO128F425C4EF,285.157)
(TRBHQUV12903CFAFA9,284.208)
(TRLPHPU12903CD8DAA,282.573)
```

- Top 5 most familiar artists :
    - Here we filter the artist_id's and familiarity and create a map with (K,V) -> (Artist_Id, Familiarity).
    - Then we do a descending sort on familiarity and pick the top 5 familiar artists.

```
(Artist Id, Familiarity)
(ARCGJ6U1187FB4D01F,1.0)
(ARNVQR71187FB59D78,0.98993856)
(ARUDYKB11F4C83C269,0.98993856)
(ARUR3Q71187FB594BA,0.97669613)
(ARY7WK51187B9B1941,0.97669613)
```

- Top 5 hottest songs :
    - Here we filter the track_id's and song_hotness and create a map with (K,V) -> (Track_Id, song_hotness).
    - Then we do a descending sort on song_hotness and pick the top 5 hottest songs.

```
(Track Id, Hotness)
(TRFDCPI128F93234B7,1.0)
(TRRMPZP128F426C2A9,1.0)
(TRPNXMB12903D0123B,1.0)
(TRDWTVM128F92EC39C,1.0)
(TREMGWF128F93369FD,1.0)
```

- Top 5 hottest artists :
    - Here we filter the artist_id's and artist_hotness and create a map with (K,V) -> (Artist_Id, artist_hotness).
    - Then we do a descending sort on artist_hotness and pick the top 5 hottest artist.

```
(Artist Id, Hotness)
(ARRH63Y1187FB47783,1.0825026)
(ARF8HTQ1187B9AE693,1.0212556)
(ARF8HTQ1187B9AE693,1.0104903)
(ARTDQRC1187FB4EFD4,1.005942)
(ARRH63Y1187FB47783,1.0052984)
```

- Top 5 hottest genres (mean artists hotness in artist_term) :
    - Here we filter the artist_id's and artist_term from `artist_term.csv` and create (K,V) -> (Artist_Id, Artist_term)
    - Then we join it with (Artist_Id, Artist_Hotness) and using a combineByKey, we get the Artist_term and their mean_artist-Hotness which we then sort by mean_artist-hotness and pick the top 5 hottest genre.

```
(Genre, Hotness)
(christmas songs,0.6084022)
(kotekote,0.60222054)
(female artist,0.5753481)
(girl rockers,0.57378817)
(alternative latin,0.573759)
```

- Top 5 most popular keys (must have confidence > 0.7) :

    - Here we filter the key's and key_confidence>0.7 and create a map with (K,V) -> (Key, Key_Confidence).
    - Then we do a countByKey and sort them based on Key_Confidence and pick the top 5 most popular key.

```
(Key, Key Confidence)
(7,30420)
(0,28333)
(2,25845)
(9,21283)
(4,15214)
```

- Top 5 most prolific artists (include ex-equo items, if any) :
    - Here we filter the artist_id and track_id and create a map with (K,V) -> (Artist_id, Track_id).
    - Then we do a countByKey and get the top 5 prolific artist.

```
(Artist Id, Count of tracks)
(AR6681Y1187FB39B02,208)
(ARXPPEY1187FB51DF4,204)
(ARH861H1187B9B799E,201)
(AR8L6W21187B9AD317,196)
(ARLHO5Z1187FB4C861,194)
```

- Top 5 most common words in song titles (excluding articles, prepositions, conjunctions) :
    - Here from the song title, we filter out the articles, prepositions, conjunctions and empty values and then using countByKey, we calculate the word count of each word and get the top 5 most common words.

```
(Words, Count)
(VERSION,59385)
(ALBUM,35663)
(ME,32901)
(LOVE,28666)
(MY,26204)
```

# Conclusion

- Spark has lots of advantages over Hadoop MapReduce framework in terms of a the speed at which it executes the batch processing jobs because of its in memory computations.
- We could finish all the jobs in ~4 minutes on a local environment which is a quite impossible to achieve using hadoop Map-Reduce.
- But also the same in-memory computations can be an overhead when performing tasks like join.

# Local Execution Environment Specifications:

- Macintosh 2.5Ghz i7 Quad Core
- 16 GB RAM
- macOS Sierra Version 10.12.6
- Java version : 1.8
- Scala version : 2.11.11

- Spark version : 2.2.0