MONASH University

# Week 8

## FIT5202 Big Data Processing

Collaborative Filtering using ALS

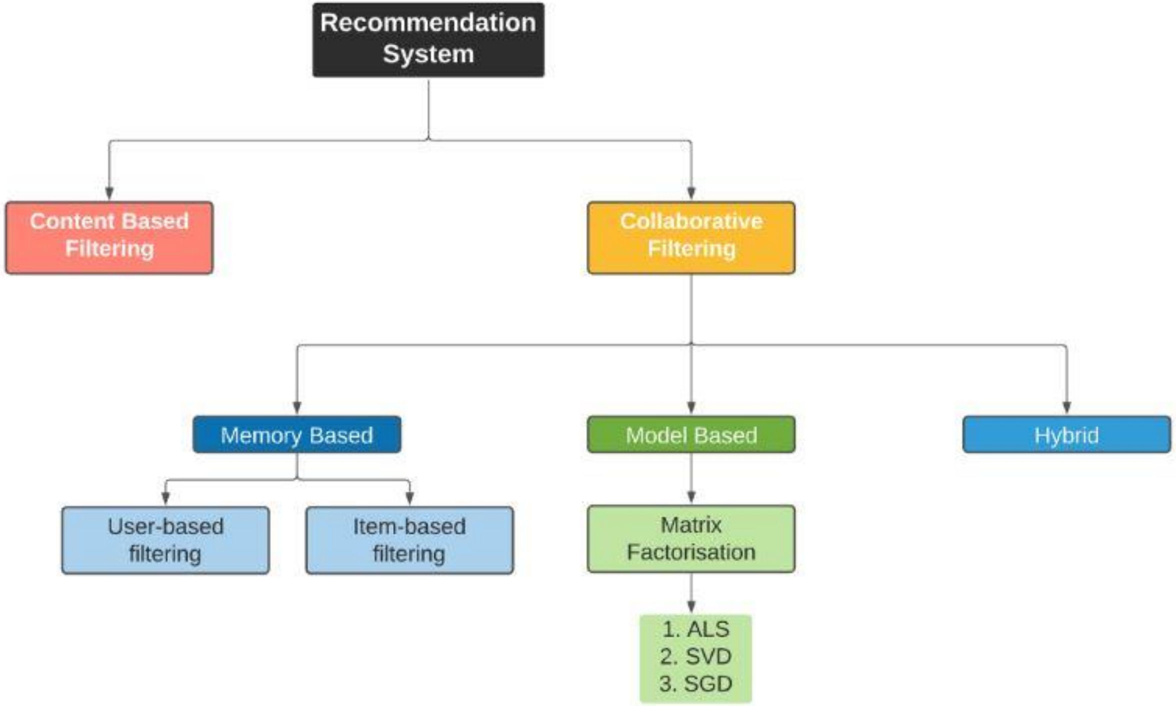
By CM Ting (April 2025)

# Week 8 Agenda

- Week 7 Review
    - K-means clustering
    - Model Selection
    - Model Persistence
- Collaborative Filtering
- Use case : Music Recommendation

SETU FEEDBACK

# Recommender Approaches

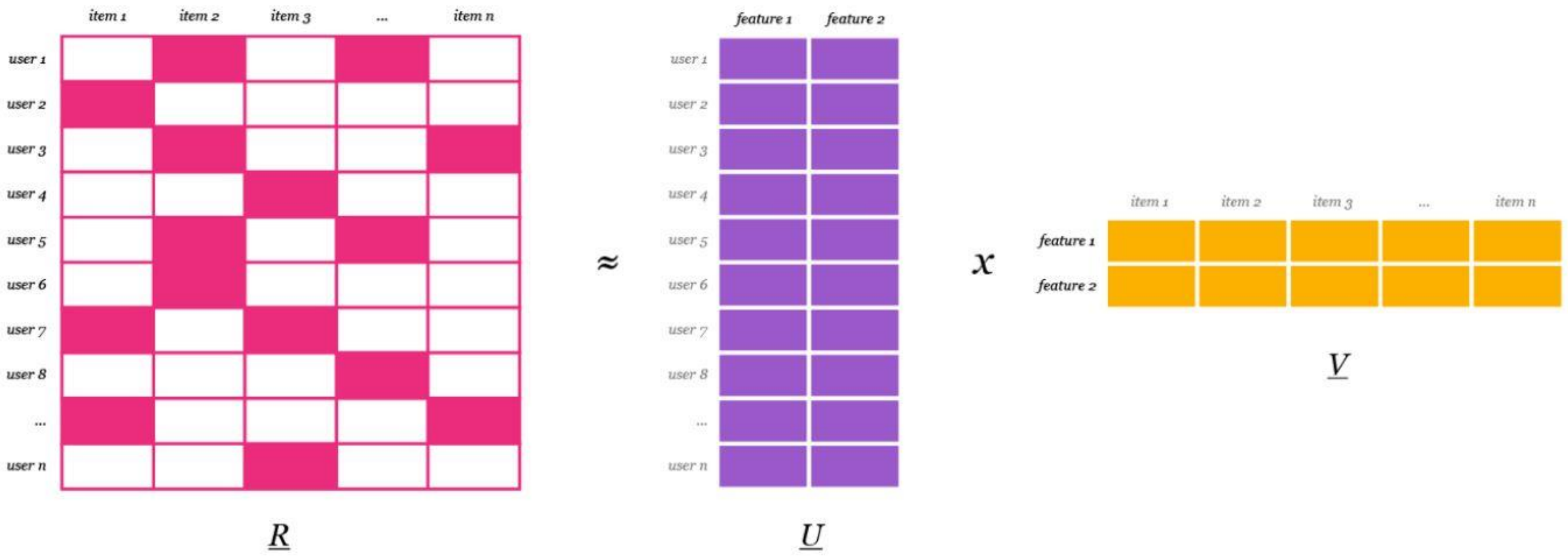



https://towardsdatascience.com/build-recommendation-system-with-pyspark-using-alternating-least-squares-als-matrix-factorisation-ebe1ad2e7679

# Implicit vs Explicit Feedback

- **Explicit** :
  - when we have **some sort of Rating** (i.e. **users provide items' rating explicitly**)
- **Implicit** :
  - data is gathered from user behaviour, e.g. how many times a song is played or a movie is watched.
  - **Advantage** : more data
  - **Disadvantage** : Noisy data, negative preferences are not known

# Matrix Factorization

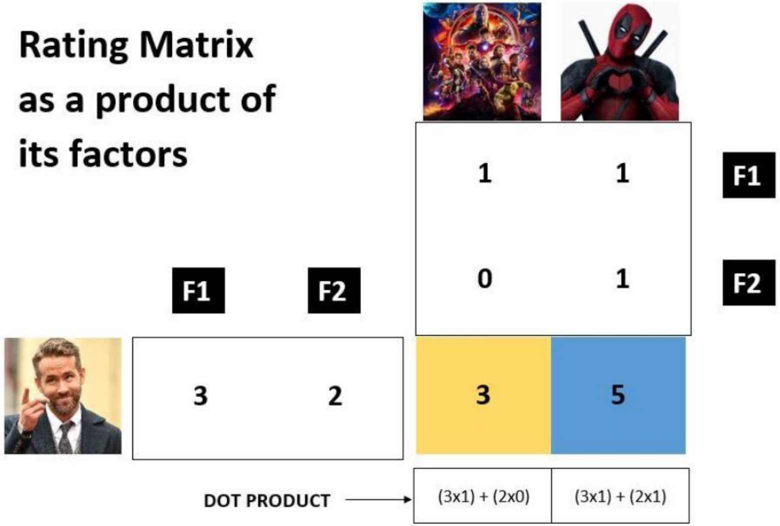



https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe

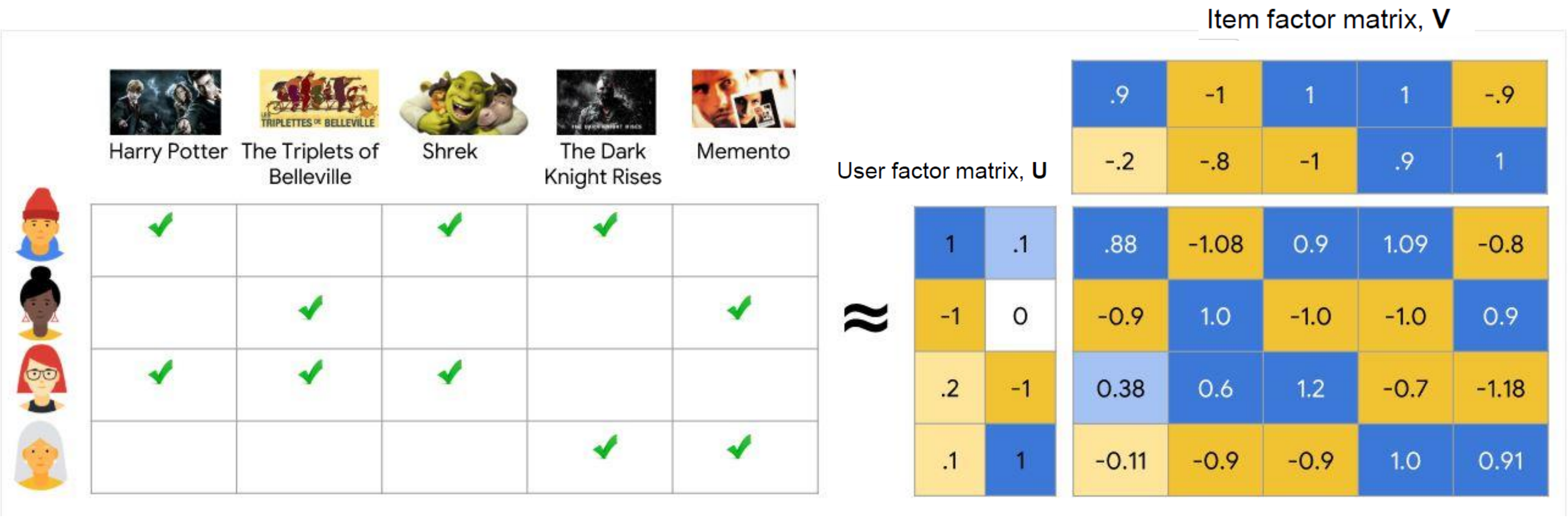
# Matrix Factorization – with Explicit Rating

Item

User

# Matrix Factorization – with Implicit Feedback

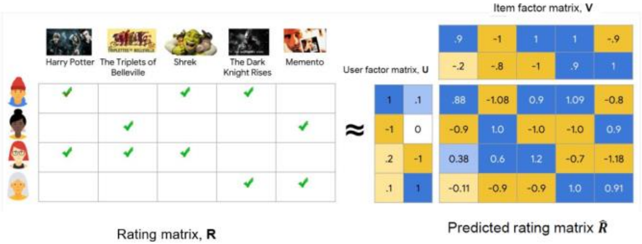



Rating matrix, **R**

Predicted rating matrix $\hat{\boldsymbol{R}}$

# Alternating Least Square (ALS) – Implicit Rating

- **Confidence:** $c_{ui} = 1 + \alpha r_{ui}$
  - Quantify confidence of how much user *u* ***likes*** the item *i* of the user from the implicit rating data ***r*** (e.g., play counts)
- **Alpha** $\alpha$
  - The rate (linear scaling) of confidence increases




Based on paper 'Collaborative Filtering for Implicit Feedback Datasets' by Yifan Hu et al.

- **Optimizing alternately to find U, V**
  - Randomly initialize U and V
  - Iterating the following steps:
    - Fixing U → Optimizing V
    - Fixing V → Optimizing U

$$\min_{x_\star, y_\star} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \left( \sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

λ = regularization parameter (regParam)

*p* = preference of user *u* for an item *i*

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe

# Train/Test Split

item →

user ↓

| | | | | | |
|---|---|---|---|---|---|
| 1 | | 3 | | 2 | 2 |
| | 1 | 2 | 4 | 4 | |
| 1 | | 1 | 1 | | 1 |
| | 3 | | 3 | 4 | 2 |

Origianl Dataset

| | | | | | |
|---|---|---|---|---|---|
| 1 | | X | | X | 2 |
| | 1 | 2 | X | X | |
| 1 | | 1 | 1 | | 1 |
| | 3 | | 3 | X | X |

Training Dataset

| | | | | | |
|---|---|---|---|---|---|
| | | 3 | | 2 | |
| | | | 4 | 4 | |
| | | | | | |
| | | | | 4 | 2 |

Testing Dataset

## Train/Test Split

```
#Write your code here
(train, test) = df_user_artist.randomSplit([0.8, 0.2])
```

https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.recommendation.ALS.html

```
+-------+---------+---------+
|user_id|artist_id|playcount|
+-------+---------+---------+
|1059637|  1000010|      238|
|1059637|  1000049|        1|
|1059637|  1000056|        1|
|1059637|  1000062|       11|
|1059637|  1000094|        1|
|1059637|  1000112|      423|
|1059637|  1000113|        5|
|1059637|  1000114|        2|
|1059637|  1000123|        2|
|1059637|  1000130|    19129|
|1059637|  1000139|        4|
|1059637|  1000241|      188|
|1059637|  1000263|      180|
|1059637|  1000289|        2|
|1059637|  1000305|        1|
|1059637|  1000320|       21|
|1059637|  1000340|        1|
|1059637|  1000427|       20|
|1059637|  1000428|       12|
|1059637|  1000433|       10|
+-------+---------+---------+
only showing top 20 rows
```

df_user_artist

## Model Building and Prediction using ALS (See Demo)

```
als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating",
          coldStartStrategy="drop")
model = als.fit(training)

# Evaluate the model by computing the RMSE on the test data for predicting rating
predictions = model.transform(test)
```

## Evaluate prediction performance based on RMSE:

```
from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator(metricName="rmse", labelCol="playcount",
                                predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))
```

https://towardsdatascience.com/recsys-implementation-on-variants-of-svd-based-recommender-system-a3dc1d059c83

# Evaluation metrics

0% — 100%

Most desirable — Least desirable

Sorted predicted ratings for user *u*

## For explicit feedback

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

*N* - # entries in test set

RMSE may not be appropriate for measuring prediction for implicit feedback

| user_id | artist_id | playcount | prediction |
|---|---|---|---|
| 1001440 | 463 | 2 | -0.6025843 |
| 1046559 | 463 | 782 | 0.6918464 |
| 1059765 | 463 | 793 | -0.045939725 |
| 1024631 | 833 | 5 | 0.8736501 |
| 2010008 | 833 | 185 | 1.0421734 |
| 1029563 | 833 | 3 | 0.38790843 |
| 2010008 | 2366 | 4 | 0.16086206 |
| 2023686 | 3175 | 1 | 0.19943924 |
| 2102019 | 1004021 | 28 | 0.043972284 |
| 1059765 | 1007972 | 21 | 0.46731347 |
| 1024631 | 1012617 | 1 | 0.03206493 |
| 1024631 | 1014191 | 3 | 0.38790256 |
| 2023686 | 1014191 | 3 | 0.16714399 |
| 2023686 | 1014690 | 2 | 0.24097718 |
| 1017610 | 1019303 | 68 | 0.42512476 |
| 1024631 | 1028228 | 1 | 0.1952564 |
| 1059637 | 1048726 | 1 | 0.0038849264 |
| 2069889 | 1048726 | 2 | -0.032158498 |
| 1072684 | 1076507 | 2 | 0.97082245 |
| 2023686 | 1084951 | 1 | -0.027778534 |

only showing top 20 rows

## For implicit feedback

ROEM (Rank Ordering Error Metric)

$$\overline{rank} = \frac{\sum_{u,i} r^t_{ui} rank_{ui}}{\sum_{u,i} r^t_{ui}} \qquad (8)$$

Lower values of $\overline{rank}$ are more desirable, as they indicate ranking actually watched shows closer to the top of the recommendation lists. Notice that for random predictions, the expected value of $rank_{ui}$ is 50% (placing $i$ in the middle of the sorted list). Thus, $\overline{rank} \geqslant 50\%$ indicates an algorithm no better than random.

$r^t_{ui}$ - true rating of user *u* for item *i* in test set

$rank_{ui}$ - percentile-ranking of item *i* within an ordered list of all items for user *u*

| user_id | artist_id | playcount | prediction | percent_rank |
|---|---|---|---|---|
| 1059637 | 1233770 | 613 | 6.226111 | 0.0 |
| 2020513 | 754 | 993 | 4.6966453 | 2.032520325203252E-4 |
| 1072684 | 1330 | 54 | 4.677941 | 4.065040650406504E-4 |
| 1059334 | 228 | 12 | 4.4809046 | 6.097560975609756E-4 |
| 1059637 | 1000130 | 19129 | 3.7980416 | 8.130081300813008E-4 |
| 2069889 | 1000263 | 177 | 3.6094182 | 0.0010162601626016261 |
| 1070641 | 1004294 | 2 | 3.5892177 | 0.0012195121951219512 |
| 1007308 | 393 | 12 | 3.5190763 | 0.0014227642276422765 |
| 2023686 | 1285410 | 3 | 3.4317646 | 0.0016260162601626016 |
| 1047812 | 718 | 10 | 3.3937335 | 0.0018292682926829 27 |
| 1031009 | 4163 | 17 | 3.271598 | 0.0020325203252032522 |
| 1055449 | 407 | 39 | 3.1267304 | 0.0022357723577235773 |
| 1055449 | 1194 | 119 | 3.109819 | 0.0024390243902439024 |
| 2023686 | 2884 | 1 | 3.0556219 | 0.0026422764227642275 |
| 1058890 | 1233770 | 38 | 3.0266361 | 0.002845528455284553 |
| 2023686 | 1270 | 26 | 3.0264745 | 0.003048780487804878 |
| 2005710 | 1001412 | 1575 | 2.981335 | 0.0032520325203252032 |
| 2062243 | 1000323 | 241 | 2.9571671 | 0.0034552845528455283 |
| 1059637 | 1000926 | 1 | 2.9293735 | 0.003658536585365854 |
| 2023686 | 1002262 | 39 | 2.9215589 | 0.003861788617886179 |

only showing top 20 rows

From paper 'Collaborative Filtering for Implicit Feedback Datasets'

# Cold-Start Problem

- ❑ Cold-start: New users will have no to little information about them to be compared with other users.

- ❑ Cold starts occur when we attempt to predict a rating for users and/or items in the test dataset that were not present during training the model

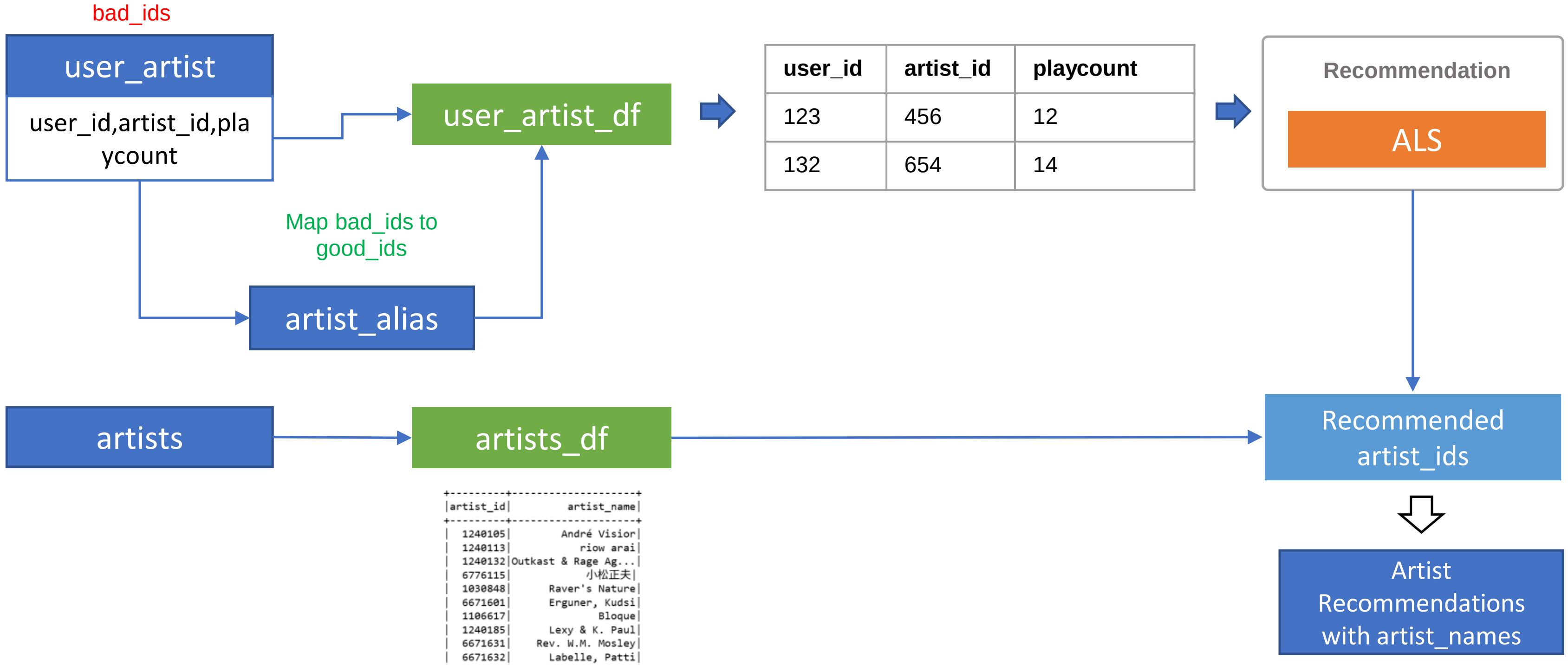
**Two strategies for handling this problem:**

"NaN" - return an empty variable.

- Spark assigns NaN predictions during ALSModel.transform when a user and/or item factor is not present in the model.
- In development however, this result prevents us from calculating a performance metric to evaluate the system.

"drop" - this option simply removes the row/column from the predictions that contain NaN values. Our result will therefore only contain valid numbers that can be used for evaluation.

# Use Case : Music Recommendation




| user_id | artist_id | playcount |
|---|---|---|
| 123 | 456 | 12 |
| 132 | 654 | 14 |

| artist_id | artist_name |
|---|---|
| 1240105 | André Visior |
| 1240113 | riow arai |
| 1240132 | Outkast & Rage Ag... |
| 6776115 | 小松正夫 |
| 1030848 | Raver's Nature |
| 6671601 | Erguner, Kudsi |
| 1106617 | Bloque |
| 1240185 | Lexy & K. Paul |
| 6671631 | Rev. W.M. Mosley |
| 6671632 | Labelle, Patti |
| 1240238 | the Chinese Stars |
| 1240262 | The Gufs |
| 6718605 | Bali Music |
| 6828988 | Southern Conferen... |
| 1240415 | Paul & Paula |
| 1009439 | Cinnamon |
| 1018275 | School Of Fish |
| 6671680 | Armstrong, Louis ... |
| 1240508 | The Ozark Mountai... |
| 1240510 | The Mercury Program |

# Thank You!

See you next week.