Вариант

In [1]:
```
surname = "марчук" #Ваша фамилия

alp = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'
w = [1, 42, 21, 21, 34,  6, 44, 26, 18, 44, 38, 26, 14, 43,  4, 49, 45,
        7, 42, 29,  4,  9, 36, 34, 31, 29,  5, 30,  4, 19, 28, 25, 33]

d = dict(zip(alp, w))
variant =  sum([d[el] for el in surname.lower()]) % 40 + 1

print("Задача № 1: ", variant % 3 + 1)
print("Задача № 2: ", variant % 2 + 1 )
```

```
Задача № 1:  1
Задача № 2:  2
```

Датасеты

hadoop fs -mkdir /dataset/hw4/small

hdfs dfs -put /home/ubuntu/_practice/hw4/small/links.csv /dataset/hw4/small/links.csv hdfs dfs -put /home/ubuntu/_practice/hw4/small/movies.csv /dataset/hw4/small/movies.csv hdfs dfs -put /home/ubuntu/_practice/hw4/small/ratings.csv /dataset/hw4/small/ratings.csv hdfs dfs -put /home/ubuntu/_practice/hw4/small/tags.csv /dataset/hw4/small/tags.csv

In [2]:
```
# зависимости из дз1
import os

# пути к Java и Spark
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/home/ubuntu/_practice/spark-3.5.4-bin-hadoop3"
os.environ["PATH"] += os.pathsep + os.path.join(os.environ["SPARK_HOME"], "bin")

import findspark
findspark.init()

import pyspark
print(pyspark.__version__)

from pyspark import SparkContext, SparkConf
```

```
3.5.4
```

Задание 1. Анализ датасета

Вариант 1. Animation, Romance, Documentary

⚠️ Замечание: Один фильм может принадлежать разным жанрам

1. Выведите данные, сопоставляющие жанры и количество фильмов
2. Выведите первые 10 фильмов с наибольшим количеством рейтингов для каждого жанра в соответствии с вариантом
3. Выведите первые 10 фильмов с наименьшим количеством рейтингов (но больше 10) для каждого жанра в соответствии с вариантом
4. Выведите первые 10 фильмов с наибольшим средним рейтингом при количестве рейтингов больше 10 для каждого жанра в соответствии с вариантом
5. Выведите первые 10 фильмов с наименьшим средним рейтингом при количестве рейтингов больше 10 для каждого жанра в соответствии с вариантом

In [3]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, explode, split, count, mean, desc, asc
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number

# Инициализация SparkSession
spark = SparkSession.builder.appName("Movies Marchuk").getOrCreate()

# Пути к файлам в HDFS
movies_path = "/dataset/hw4/small/movies.csv"
ratings_path = "/dataset/hw4/small/ratings.csv"

# Загрузка данных
movies_df = spark.read.csv(movies_path, header=True, inferSchema=True)
ratings_df = spark.read.csv(ratings_path, header=True, inferSchema=True)

# Разделение жанров на отдельные строки
movies_with_genres = movies_df.withColumn("genre", explode(split(col("genres"), "\\|")))

# Фильтрация по целевым жанрам
target_genres = ["Animation", "Romance", "Documentary"]
movies_target_genres = movies_with_genres.filter(col("genre").isin(target_genres))
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/01/13 23:11:04 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

In [4]:
```
# 1. Сопоставление жанров и количества фильмов
genre_counts = movies_target_genres.groupBy("genre").agg(count("*").alias("movie_count"))
genre_counts.show()

# Присоединение рейтингов
movies_ratings = movies_target_genres.join(ratings_df, "movieId")
```

```
+-----------+-----------+
|      genre|movie_count|
+-----------+-----------+
|    Romance|       1596|
|Documentary|        440|
|  Animation|        611|
+-----------+-----------+
```

In [5]:
```python
# 2. Топ-10 фильмов с наибольшим количеством рейтингов для каждого жанра

# Создаем окно для нумерации фильмов в каждом жанре
window_spec = Window.partitionBy("genre").orderBy(col("rating_count").desc())

# Добавляем колонку с порядковым номером фильма в жанре
ranked_movies = (
    movies_ratings.groupBy("genre", "movieId", "title")
    .agg(count("rating").alias("rating_count"))
    .withColumn("rank", row_number().over(window_spec))
)

# Фильтруем, оставляя только топ-10 фильмов в каждом жанре
top_10_rated_per_genre = ranked_movies.filter(col("rank") <= 10)

# Сортируем и отображаем результаты
top_10_rated_per_genre = top_10_rated_per_genre.orderBy("genre", "rank")
top_10_rated_per_genre.show(100, truncate=False)
```

```
+-----------+-------+-------------------------------------------------------+------------+----+
|genre      |movieId|title                                                  |rating_count|rank|
+-----------+-------+-------------------------------------------------------+------------+----+
|Animation  |1      |Toy Story (1995)                                       |215         |1   |
|Animation  |588    |Aladdin (1992)                                         |183         |2   |
|Animation  |364    |Lion King, The (1994)                                  |172         |3   |
|Animation  |4306   |Shrek (2001)                                           |170         |4   |
|Animation  |595    |Beauty and the Beast (1991)                            |146         |5   |
|Animation  |6377   |Finding Nemo (2003)                                    |141         |6   |
|Animation  |4886   |Monsters, Inc. (2001)                                  |132         |7   |
|Animation  |8961   |Incredibles, The (2004)                                |125         |8   |
|Animation  |68954  |Up (2009)                                              |105         |9   |
|Animation  |60069  |WALL·E (2008)                                          |104         |10  |
|Documentary|5669   |Bowling for Columbine (2002)                           |58          |1   |
|Documentary|8464   |Super Size Me (2004)                                   |50          |2   |
|Documentary|8622   |Fahrenheit 9/11 (2004)                                 |37          |3   |
|Documentary|2064   |Roger & Me (1989)                                      |31          |4   |
|Documentary|246    |Hoop Dreams (1994)                                     |29          |5   |
|Documentary|34072  |March of the Penguins (Marche de l'empereur, La) (2005)|18          |6   |
|Documentary|162    |Crumb (1994)                                           |17          |7   |
|Documentary|5785   |Jackass: The Movie (2002)                              |17          |8   |
|Documentary|53894  |Sicko (2007)                                           |14          |9   |
|Documentary|77455  |Exit Through the Gift Shop (2010)                      |13          |10  |
|Romance    |356    |Forrest Gump (1994)                                    |329         |1   |
|Romance    |2858   |American Beauty (1999)                                 |204         |2   |
|Romance    |380    |True Lies (1994)                                       |178         |3   |
|Romance    |377    |Speed (1994)                                           |171         |4   |
|Romance    |4306   |Shrek (2001)                                           |170         |5   |
|Romance    |595    |Beauty and the Beast (1991)                            |146         |6   |
|Romance    |1265   |Groundhog Day (1993)                                   |143         |7   |
|Romance    |1197   |Princess Bride, The (1987)                             |142         |8   |
|Romance    |1704   |Good Will Hunting (1997)                               |141         |9   |
|Romance    |1721   |Titanic (1997)                                         |140         |10  |
+-----------+-------+-------------------------------------------------------+------------+----+
```

In [6]:
```python
# 3. Топ-10 фильмов с наименьшим количеством рейтингов (>10) для каждого жанра
window_spec_rating_count = Window.partitionBy("genre").orderBy(col("rating_count").asc())
least_10_rated_per_genre = (
    movies_ratings.groupBy("genre", "movieId", "title")
    .agg(count("rating").alias("rating_count"))
    .filter(col("rating_count") > 10)  # Условие > 10
    .withColumn("rank", row_number().over(window_spec_rating_count))
    .filter(col("rank") <= 10)  # Топ-10 для каждого жанра
    .orderBy("genre", "rank")
)
least_10_rated_per_genre.show(100, truncate=False)
```

```
+-----------+-------+-------------------------------------------------------------------------+------------+----+
|genre      |movieId|title                                                                    |rating_count|rank|
+-----------+-------+-------------------------------------------------------------------------+------------+----+
|Animation  |55442  |Persepolis (2007)                                                        |11          |1   |
|Animation  |49274  |Happy Feet (2006)                                                        |11          |2   |
|Animation  |97225  |Hotel Transylvania (2012)                                                |11          |3   |
|Animation  |8965   |Polar Express, The (2004)                                                |11          |4   |
|Animation  |709    |Oliver & Company (1988)                                                  |11          |5   |
|Animation  |52435  |How the Grinch Stole Christmas! (1966)                                   |11          |6   |
|Animation  |631    |All Dogs Go to Heaven 2 (1996)                                           |11          |7   |
|Animation  |65261  |Ponyo (Gake no ue no Ponyo) (2008)                                       |11          |8   |
|Animation  |52287  |Meet the Robinsons (2007)                                                |11          |9   |
|Animation  |72737  |Princess and the Frog, The (2009)                                        |12          |10  |
|Documentary|1289   |Koyaanisqatsi (a.k.a. Koyaanisqatsi: Life Out of Balance) (1983)         |11          |1   |
|Documentary|1189   |Thin Blue Line, The (1988)                                               |11          |2   |
|Documentary|54881  |King of Kong, The (2007)                                                 |12          |3   |
|Documentary|80906  |Inside Job (2010)                                                        |12          |4   |
|Documentary|77455  |Exit Through the Gift Shop (2010)                                        |13          |5   |
|Documentary|45950  |Inconvenient Truth, An (2006)                                            |13          |6   |
|Documentary|6331   |Spellbound (2002)                                                        |13          |7   |
|Documentary|7156   |Fog of War: Eleven Lessons from the Life of Robert S. McNamara, The (2003)|13         |8   |
|Documentary|53894  |Sicko (2007)                                                             |14          |9   |
|Documentary|162    |Crumb (1994)                                                             |17          |10  |
|Romance    |48082  |Science of Sleep, The (La science des rêves) (2006)                      |11          |1   |
|Romance    |3259   |Far and Away (1992)                                                      |11          |2   |
|Romance    |1735   |Great Expectations (1998)                                                |11          |3   |
|Romance    |89904  |The Artist (2011)                                                        |11          |4   |
|Romance    |5812   |Far from Heaven (2002)                                                   |11          |5   |
|Romance    |59725  |Sex and the City (2008)                                                  |11          |6   |
|Romance    |3261   |Singles (1992)                                                           |11          |7   |
|Romance    |54190  |Across the Universe (2007)                                               |11          |8   |
|Romance    |31433  |Wedding Date, The (2005)                                                 |11          |9   |
|Romance    |1944   |From Here to Eternity (1953)                                             |11          |10  |
+-----------+-------+-------------------------------------------------------------------------+------------+----+
```

In [7]:

```python
# 4. Топ-10 фильмов с наибольшим средним рейтингом (>10 рейтингов) для каждого жанра
window_spec_avg_rating_desc = Window.partitionBy("genre").orderBy(col("avg_rating").desc())
top_10_avg_rated_per_genre = (
    movies_ratings.groupBy("genre", "movieId", "title")
    .agg(
        count("rating").alias("rating_count"),
        mean("rating").alias("avg_rating")
    )
    .filter(col("rating_count") > 10)  # Условие > 10 рейтингов
    .withColumn("rank", row_number().over(window_spec_avg_rating_desc))
    .filter(col("rank") <= 10)  # Топ-10 для каждого жанра
    .orderBy("genre", "rank")
)
top_10_avg_rated_per_genre.show(100, truncate=False)
```

```
+-----------+-------+-------------------------------------------------------------------------+------------+------------------+----+
|genre      |movieId|title                                                                    |rating_count|avg_rating        |rank|
+-----------+-------+-------------------------------------------------------------------------+------------+------------------+----+
|Animation  |3429   |Creature Comforts (1989)                                                 |12          |4.25              |1   |
|Animation  |55442  |Persepolis (2007)                                                        |11          |4.181818181818182 |2   |
|Animation  |5690   |Grave of the Fireflies (Hotaru no haka) (1988)                           |16          |4.15625           |3   |
|Animation  |5618   |Spirited Away (Sen to Chihiro no kamikakushi) (2001)                     |87          |4.155172413793103 |4   |
|Animation  |741    |Ghost in the Shell (Kôkaku kidôtai) (1995)                               |27          |4.148148148148148 |5   |
|Animation  |3213   |Batman: Mask of the Phantasm (1993)                                      |13          |4.115384615384615 |6   |
|Animation  |78499  |Toy Story 3 (2010)                                                       |55          |4.109090909090909 |7   |
|Animation  |720    |Wallace & Gromit: The Best of Aardman Animation (1996)                   |27          |4.092592592592593 |8   |
|Animation  |1223   |Grand Day Out with Wallace and Gromit, A (1989)                          |28          |4.089285714285714 |9   |
|Animation  |72226  |Fantastic Mr. Fox (2009)                                                 |18          |4.083333333333333 |10  |
|Documentary|7156   |Fog of War: Eleven Lessons from the Life of Robert S. McNamara, The (2003)|13         |4.3076923076923075|1   |
|Documentary|246    |Hoop Dreams (1994)                                                       |29          |4.293103448275862 |2   |
|Documentary|80906  |Inside Job (2010)                                                        |12          |4.291666666666667 |3   |
|Documentary|162    |Crumb (1994)                                                             |17          |4.205882352941177 |4   |
|Documentary|77455  |Exit Through the Gift Shop (2010)                                        |13          |4.038461538461538 |5   |
|Documentary|1189   |Thin Blue Line, The (1988)                                               |11          |4.0               |6   |
|Documentary|6331   |Spellbound (2002)                                                        |13          |3.923076923076923 |7   |
|Documentary|54881  |King of Kong, The (2007)                                                 |12          |3.9166666666666665|8   |
|Documentary|1289   |Koyaanisqatsi (a.k.a. Koyaanisqatsi: Life Out of Balance) (1983)         |11          |3.8636363636363638|9   |
|Documentary|2064   |Roger & Me (1989)                                                        |31          |3.838709677419355 |10  |
|Romance    |951    |His Girl Friday (1940)                                                   |14          |4.392857142857143 |1   |
|Romance    |922    |Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)                            |27          |4.333333333333333 |2   |
|Romance    |905    |It Happened One Night (1934)                                             |14          |4.321428571428571 |3   |
|Romance    |898    |Philadelphia Story, The (1940)                                           |29          |4.310344827586207 |4   |
|Romance    |1235   |Harold and Maude (1971)                                                  |26          |4.288461538461538 |5   |
|Romance    |930    |Notorious (1946)                                                         |20          |4.25              |6   |
|Romance    |912    |Casablanca (1942)                                                        |100         |4.24              |7   |
|Romance    |1197   |Princess Bride, The (1987)                                               |142         |4.232394366197183 |8   |
|Romance    |28     |Persuasion (1995)                                                        |11          |4.2272727272727275|9   |
|Romance    |933    |To Catch a Thief (1955)                                                  |23          |4.217391304347826 |10  |
+-----------+-------+-------------------------------------------------------------------------+------------+------------------+----+
```

In [8]:

```python
# 5. Топ-10 фильмов с наименьшим средним рейтингом (>10 рейтингов) для каждого жанра
window_spec_avg_rating = Window.partitionBy("genre").orderBy(col("avg_rating").asc())
least_10_avg_rated_per_genre = (
    movies_ratings.groupBy("genre", "movieId", "title")
    .agg(
        count("rating").alias("rating_count"),
        mean("rating").alias("avg_rating")
    )
    .filter(col("rating_count") > 10)  # Условие > 10 рейтингов
    .withColumn("rank", row_number().over(window_spec_avg_rating))
    .filter(col("rank") <= 10)  # Топ-10 для каждого жанра
    .orderBy("genre", "rank")
```

```
)
least_10_avg_rated_per_genre.show(100, truncate=False)
```

```
+-----------+-------+-----------------------------------------------------------------+------------+------------------+----+
|genre      |movieId|title                                                            |rating_count|avg_rating        |rank|
+-----------+-------+-----------------------------------------------------------------+------------+------------------+----+
|Animation  |8907   |Shark Tale (2004)                                                |13          |2.3461538461538463|1   |
|Animation  |69644  |Ice Age: Dawn of the Dinosaurs (2009)                            |14          |2.607142857142857 |2   |
|Animation  |49274  |Happy Feet (2006)                                                |11          |2.6818181818181817|3   |
|Animation  |2123   |All Dogs Go to Heaven (1989)                                     |15          |2.7               |4   |
|Animation  |673    |Space Jam (1996)                                                 |53          |2.707547169811321 |5   |
|Animation  |1030   |Pete's Dragon (1977)                                             |15          |2.7666666666666666|6   |
|Animation  |1920   |Small Soldiers (1998)                                            |18          |2.8333333333333335|7   |
|Animation  |1405   |Beavis and Butt-Head Do America (1996)                           |31          |2.935483870967742 |8   |
|Animation  |239    |Goofy Movie, A (1995)                                            |17          |3.0               |9   |
|Animation  |53121  |Shrek the Third (2007)                                           |21          |3.0238095238095237|10  |
|Documentary|8622   |Fahrenheit 9/11 (2004)                                           |37          |3.4864864864864864|1   |
|Documentary|5785   |Jackass: The Movie (2002)                                        |17          |3.5               |2   |
|Documentary|8464   |Super Size Me (2004)                                             |50          |3.51              |3   |
|Documentary|34072  |March of the Penguins (Marche de l'empereur, La) (2005)          |18          |3.5555555555555554|4   |
|Documentary|45950  |Inconvenient Truth, An (2006)                                    |13          |3.576923076923077 |5   |
|Documentary|53894  |Sicko (2007)                                                     |14          |3.7142857142857144|6   |
|Documentary|5669   |Bowling for Columbine (2002)                                     |58          |3.7758620689655173|7   |
|Documentary|2064   |Roger & Me (1989)                                                |31          |3.838709677419355 |8   |
|Documentary|1289   |Koyaanisqatsi (a.k.a. Koyaanisqatsi: Life Out of Balance) (1983)|11          |3.8636363636363638|9   |
|Documentary|54881  |King of Kong, The (2007)                                         |12          |3.9166666666666665|10  |
|Romance    |1556   |Speed 2: Cruise Control (1997)                                   |19          |1.605263157894737 |1   |
|Romance    |1381   |Grease 2 (1982)                                                  |19          |2.0789473684210527|2   |
|Romance    |33836  |Bewitched (2005)                                                 |13          |2.269230769230769 |3   |
|Romance    |502    |Next Karate Kid, The (1994)                                      |15          |2.3666666666666667|4   |
|Romance    |4247   |Joe Dirt (2001)                                                  |21          |2.380952380952381 |5   |
|Romance    |4621   |Look Who's Talking (1989)                                        |18          |2.388888888888889 |6   |
|Romance    |59725  |Sex and the City (2008)                                          |11          |2.409090909090909 |7   |
|Romance    |63992  |Twilight (2008)                                                  |22          |2.409090909090909 |8   |
|Romance    |3824   |Autumn in New York (2000)                                        |11          |2.409090909090909 |9   |
|Romance    |4153   |Down to Earth (2001)                                             |12          |2.4166666666666665|10  |
+-----------+-------+-----------------------------------------------------------------+------------+------------------+----+
```

In [9]:
```
# Остановка SparkSession
spark.stop()
```

Задание 2. Коллаборативная фильтрация

Вариант 2. По схожести объектов

1. Разделите данные с рейтингами на обучающее (train_init - 0.8) и тестовое подмножества (test - 0.2), определите среднее значение рейтинга в обучающем подмножестве и вычислите rmse для тестового подмножества, если для всех значений из test предсказывается среднее значение рейтинга
2. Реализуйте коллаборативную фильтрацию в соответствии с вариантом. Для определения схожести используйте train_init, для расчета rmse - test
3. Определите rmse для тестового подмножества

In [10]:
```
# 1. Подготовка данных. Разделение данных на обучающую и тестовую выборки,
#    вычисление среднего значения рейтинга для обучающей выборки и RMSE для тестовой выборки.

from pyspark.sql import SparkSession
from pyspark.sql.functions import col, explode, split, count, mean, desc, asc, lit, avg, sqrt
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number
from pyspark.ml.evaluation import RegressionEvaluator


# Инициализация SparkSession
spark = SparkSession.builder.appName("Movies2 Marchuk").getOrCreate()

# Загрузка данных из HDFS
ratings = spark.read.csv("/dataset/hw4/small/ratings.csv", header=True, inferSchema=True)

# Разделение на train_init (80%) и test (20%)
train_init, test = ratings.randomSplit([0.8, 0.2], seed=42)

# Среднее значение рейтинга в train_init
mean_rating = train_init.select(avg("rating").alias("mean_rating")).collect()[0]["mean_rating"]

# Предсказание среднего значения для тестового набора
test_with_predictions = test.withColumn("prediction", lit(mean_rating))

# Вычисление RMSE для тестового подмножества
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
rmse_mean = evaluator.evaluate(test_with_predictions)

print(f"Среднее значение рейтинга в train_init: {mean_rating:.2f}")
print(f"RMSE при предсказании среднего рейтинга: {rmse_mean:.4f}")
```

```
Среднее значение рейтинга в train_init: 3.50
RMSE при предсказании среднего рейтинга: 1.0504
```

In [11]:
```
#2. Реализуйте коллаборативную фильтрацию в соответствии с вариантом.
#   Для определения схожести используйте train_init, для расчета rmse - test


from pyspark.sql.functions import col, sqrt, sum as spark_sum
from pyspark.ml.evaluation import RegressionEvaluator

# 1. Создание матрицы рейтингов "userId x movieId" для train_init
ratings_matrix = (
    train_init.groupBy("userId", "movieId")
    .agg(mean("rating").alias("rating"))
)
```

```
# 2. Вычисление косинусного сходства между фильмами
ratings_self_join = ratings_matrix.alias("r1").join(
    ratings_matrix.alias("r2"),
    col("r1.userId") == col("r2.userId")  # Сравнение по одному и тому же пользователю
)

# Подсчет числителя (скалярное произведение) и знаменателя (длины векторов)
movie_similarity = (
    ratings_self_join.groupBy("r1.movieId", "r2.movieId")
    .agg(
        spark_sum(col("r1.rating") * col("r2.rating")).alias("dot_product"),
        sqrt(spark_sum(col("r1.rating")**2)).alias("norm_r1"),
        sqrt(spark_sum(col("r2.rating")**2)).alias("norm_r2"),
    )
    .withColumn("similarity", col("dot_product") / (col("norm_r1") * col("norm_r2")))
    .filter(col("r1.movieId") != col("r2.movieId"))  # Убираем сравнение фильма с самим собой
)
# 3. Генерация предсказаний
# Для каждого фильма из test находим его ближайших соседей в train_init
predictions = (
    test.alias("t").join(
        movie_similarity.select(
            col("r1.movieId").alias("movieId_test"),  # Переименуем столбцы для удобства
            col("r2.movieId").alias("movieId_train"),
            "similarity"
        ).alias("ms"),
        col("t.movieId") == col("ms.movieId_test"),  # Связываем фильмы из тестового множества с похожими
        "left"
    )
    .join(
        train_init.alias("tr"),
        col("ms.movieId_train") == col("tr.movieId"),  # Связываем с рейтингами соседей
        "left"
    )
    .groupBy("t.userId", "t.movieId")
    .agg(
        spark_sum(col("ms.similarity") * col("tr.rating")).alias("weighted_sum"),
        spark_sum(col("ms.similarity")).alias("similarity_sum"),
    )
    .withColumn("prediction", col("weighted_sum") / col("similarity_sum"))
    .select("userId", "movieId", "prediction")
)

# 4. Оценка качества модели (RMSE)
# Объединяем предсказания с реальными рейтингами
test_with_predictions = test.join(predictions, ["userId", "movieId"], "left")

# Заполняем пропущенные значения средним рейтингом (если фильм не имеет похожих)
test_with_predictions = test_with_predictions.fillna(mean_rating, subset=["prediction"])

# Вычисление RMSE
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
rmse_collaborative = evaluator.evaluate(test_with_predictions)

print(f"RMSE для коллаборативной фильтрации: {rmse_collaborative:.4f}")
```

```
25/01/13 23:11:23 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:23 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:23 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:26 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:26 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:28 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:28 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:30 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:30 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:32 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:32 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:34 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:34 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:37 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:37 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:39 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:39 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:41 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:41 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:43 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:11:43 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:03 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:03 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:03 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:03 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:04 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:04 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:04 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:04 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:04 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:04 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:04 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:04 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:04 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:05 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:05 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:05 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
[Stage 21:==============================================>           (4 + 1) / 5]
RMSE для коллаборативной фильтрации: 1.0545
```

In [12]:

```
# 3. Определите rmse для тестового подмножества

from pyspark.ml.evaluation import RegressionEvaluator
```

```
# Оценка качества модели (RMSE)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")

# Вычисление RMSE
rmse_collaborative = evaluator.evaluate(test_with_predictions)

print(f"RMSE для коллаборативной фильтрации: {rmse_collaborative:.4f}")
```

```
25/01/13 23:12:19 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:20 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:21 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:22 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:24 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:24 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:26 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:26 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:28 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:28 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:30 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:30 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:32 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:32 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:34 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:34 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:36 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:36 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:54 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:54 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:54 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:54 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:54 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:54 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:54 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:54 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:54 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:54 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:54 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:54 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:55 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:55 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:55 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
25/01/13 23:12:55 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
[Stage 45:=================================>                       (3 + 2) / 5]
RMSE для коллаборативной фильтрации: 1.0545
```

In [13]:
```
# Остановка SparkSession
spark.stop()
```

Задание 3. Факторизация матрицы

1. Выберите модель ALS по минимальному значению rmse. Для этого используйте кросс-валидацию k-folds с k=4

Параметры:

Количество факторов: [5, 10, 15]

Регуляризация: [0.001, 0.01, 0.1, 1, 10]

⚠️ Замечание: Если какие-то элементы из тестового/валидационного подмножества не встречались в обучающем, то rmse будет NaN

2. Сравните результаты рекомендаций посредством коллаборативной фильтрации и факторизации матрицы рейтингов

In [14]:
```
# 1. Выберите модель ALS по минимальному значению rmse. Для этого используйте кросс-валидацию k-folds с k=4



from pyspark.sql import SparkSession
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql.functions import col
import numpy as np

# Инициализация SparkSession
spark = SparkSession.builder.appName("Movies3  Marchuk").getOrCreate()

# Загрузка данных
ratings = spark.read.csv("/dataset/hw4/small/ratings.csv", header=True, inferSchema=True)

# Параметры для кросс-валидации
k_folds = 4
factors = [5, 10, 15]
reg_params = [0.001, 0.01, 0.1, 1, 10]
seed = 42

# Функция для вычисления RMSE
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")

# Кросс-валидация
min_rmse = float("inf")
best_model_params = None

# Создаем k фолдов
folds = ratings.randomSplit([1.0 / k_folds] * k_folds, seed=seed)

for factor in factors:
    for reg_param in reg_params:
        fold_rmse = []
```

```
        for i in range(k_folds):
            # Определяем обучающую и валидационную выборки
            validation = folds[i]
            train = spark.createDataFrame(
                [row for j, fold in enumerate(folds) if j != i for row in fold.collect()]
            )

            # Инициализация модели ALS
            als = ALS(
                maxIter=10,
                rank=factor,
                regParam=reg_param,
                userCol="userId",
                itemCol="movieId",
                ratingCol="rating",
                coldStartStrategy="drop",  # Убираем NaN предсказания
                seed=seed,
            )

            # Обучение модели
            model = als.fit(train)

            # Предсказания на валидационном наборе
            predictions = model.transform(validation)

            # Вычисление RMSE
            rmse = evaluator.evaluate(predictions)
            fold_rmse.append(rmse)

        # Среднее RMSE для текущих параметров
        avg_rmse = np.mean(fold_rmse)

        print(f"Factor: {factor}, RegParam: {reg_param}, RMSE: {avg_rmse:.4f}")

        # Сохранение лучших параметров
        if avg_rmse < min_rmse:
            min_rmse = avg_rmse
            best_model_params = {"factor": factor, "regParam": reg_param}

print(f"\nЛучшие параметры: {best_model_params}")
print(f"Минимальное RMSE: {min_rmse:.4f}")
```

```
25/01/13 23:13:07 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
25/01/13 23:13:07 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.lapack.JNILAPACK
```

```
Factor: 5, RegParam: 0.001, RMSE: 1.1978
Factor: 5, RegParam: 0.01, RMSE: 1.0453
Factor: 5, RegParam: 0.1, RMSE: 0.8875
Factor: 5, RegParam: 1, RMSE: 1.3214
Factor: 5, RegParam: 10, RMSE: 3.6638
Factor: 10, RegParam: 0.001, RMSE: 1.3454
Factor: 10, RegParam: 0.01, RMSE: 1.1476
Factor: 10, RegParam: 0.1, RMSE: 0.8915
Factor: 10, RegParam: 1, RMSE: 1.3214
Factor: 10, RegParam: 10, RMSE: 3.6638
Factor: 15, RegParam: 0.001, RMSE: 1.4202
Factor: 15, RegParam: 0.01, RMSE: 1.2101
Factor: 15, RegParam: 0.1, RMSE: 0.8918
Factor: 15, RegParam: 1, RMSE: 1.3214
Factor: 15, RegParam: 10, RMSE: 3.6638

Лучшие параметры: {'factor': 5, 'regParam': 0.1}
Минимальное RMSE: 0.8875
```

In [15]:
```
# Остановка SparkSession
spark.stop()
```

2. Сравните результаты рекомендаций посредством коллаборативной фильтрации и факторизации матрицы рейтингов

Факторизация матрицы показала наилучший результат с минимальным RMSE 0.8875, что значительно лучше, чем предсказание среднего рейтинга и коллаборативная фильтрация по схожести объектов. Однако, если доступно мало данных или вычислительные ресурсы ограничены, использование предсказания среднего или коллаборативной фильтрации может быть оправдано. Для больших и сложных наборов данных корее всего больше подойдёт ALS, так как он лучше масштабируется и точнее захватывает скрытые паттерны в данных.

.

.

.

.

================================ БОЛЬШОЙ НАБОР ДАННЫХ ================================

Датасеты

hadoop fs -mkdir /dataset/hw4/big

hdfs dfs -put /home/ubuntu/_practice/hw4/big/links.csv /dataset/hw4/big/links.csv hdfs dfs -put /home/ubuntu/_practice/hw4/big/movies.csv /dataset/hw4/big/movies.csv hdfs dfs -put /home/ubuntu/_practice/hw4/big/ratings.csv /dataset/hw4/big/ratings.csv hdfs dfs -put /home/ubuntu/_practice/hw4/big/tags.csv /dataset/hw4/big/tags.csv

In [16]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, explode, split, count, mean, desc, asc
from pyspark.sql.window import Window
```

```
from pyspark.sql.functions import row_number

# Инициализация SparkSession
spark = SparkSession.builder.appName("Movies Marchuk").getOrCreate()

# Пути к файлам в HDFS
movies_path = "/dataset/hw4/big/movies.csv"
ratings_path = "/dataset/hw4/big/ratings.csv"

# Загрузка данных
movies_df = spark.read.csv(movies_path, header=True, inferSchema=True)
ratings_df = spark.read.csv(ratings_path, header=True, inferSchema=True)

# Разделение жанров на отдельные строки
movies_with_genres = movies_df.withColumn("genre", explode(split(col("genres"), "\\|")))

# Фильтрация по целевым жанрам
target_genres = ["Animation", "Romance", "Documentary"]
movies_target_genres = movies_with_genres.filter(col("genre").isin(target_genres))
```

In [17]:

```
# 1. Сопоставление жанров и количества фильмов
genre_counts = movies_target_genres.groupBy("genre").agg(count("*").alias("movie_count"))
genre_counts.show()

# Присоединение рейтингов
movies_ratings = movies_target_genres.join(ratings_df, "movieId")
```

```
+-----------+-----------+
|      genre|movie_count|
+-----------+-----------+
|    Romance|      10172|
|Documentary|       9283|
|  Animation|       4579|
+-----------+-----------+
```

In [18]:

```
# 2. Топ-10 фильмов с наибольшим количеством рейтингов для каждого жанра

# Создаем окно для нумерации фильмов в каждом жанре
window_spec = Window.partitionBy("genre").orderBy(col("rating_count").desc())

# Добавляем колонку с порядковым номером фильма в жанре
ranked_movies = (
    movies_ratings.groupBy("genre", "movieId", "title")
    .agg(count("rating").alias("rating_count"))
    .withColumn("rank", row_number().over(window_spec))
)

# Фильтруем, оставляя только топ-10 фильмов в каждом жанре
top_10_rated_per_genre = ranked_movies.filter(col("rank") <= 10)

# Сортируем и отображаем результаты
top_10_rated_per_genre = top_10_rated_per_genre.orderBy("genre", "rank")
top_10_rated_per_genre.show(100, truncate=False)
```

```
[Stage 8:========================================>                (5 + 2) / 7]
+-----------+-------+-------------------------------------------------------+------------+----+
|genre      |movieId|title                                                  |rating_count|rank|
+-----------+-------+-------------------------------------------------------+------------+----+
|Animation  |1      |Toy Story (1995)                                       |76813       |1   |
|Animation  |4306   |Shrek (2001)                                           |58529       |2   |
|Animation  |588    |Aladdin (1992)                                         |55791       |3   |
|Animation  |364    |Lion King, The (1994)                                  |53509       |4   |
|Animation  |4886   |Monsters, Inc. (2001)                                  |48441       |5   |
|Animation  |6377   |Finding Nemo (2003)                                    |48124       |6   |
|Animation  |595    |Beauty and the Beast (1991)                            |45404       |7   |
|Animation  |8961   |Incredibles, The (2004)                                |42953       |8   |
|Animation  |60069  |WALL·E (2008)                                          |42033       |9   |
|Animation  |68954  |Up (2009)                                              |38751       |10  |
|Documentary|5669   |Bowling for Columbine (2002)                           |16608       |1   |
|Documentary|8464   |Super Size Me (2004)                                   |14077       |2   |
|Documentary|246    |Hoop Dreams (1994)                                     |11731       |3   |
|Documentary|8622   |Fahrenheit 9/11 (2004)                                 |11553       |4   |
|Documentary|2064   |Roger & Me (1989)                                      |8296        |5   |
|Documentary|162    |Crumb (1994)                                           |6758        |6   |
|Documentary|5785   |Jackass: The Movie (2002)                              |5685        |7   |
|Documentary|34072  |March of the Penguins (Marche de l'empereur, La) (2005)|4542        |8   |
|Documentary|1147   |When We Were Kings (1996)                              |4207        |9   |
|Documentary|45950  |Inconvenient Truth, An (2006)                          |4168        |10  |
|Romance    |356    |Forrest Gump (1994)                                    |113581      |1   |
|Romance    |2858   |American Beauty (1999)                                 |69902       |2   |
|Romance    |4306   |Shrek (2001)                                           |58529       |3   |
|Romance    |1704   |Good Will Hunting (1997)                               |54980       |4   |
|Romance    |380    |True Lies (1994)                                       |52789       |5   |
|Romance    |1197   |Princess Bride, The (1987)                             |50775       |6   |
|Romance    |1721   |Titanic (1997)                                         |50706       |7   |
|Romance    |377    |Speed (1994)                                           |49029       |8   |
|Romance    |1265   |Groundhog Day (1993)                                   |47956       |9   |
|Romance    |7361   |Eternal Sunshine of the Spotless Mind (2004)           |46292       |10  |
+-----------+-------+-------------------------------------------------------+------------+----+
```

In [19]:

```
# 3. Топ-10 фильмов с наименьшим количеством рейтингов (>10) для каждого жанра
window_spec_rating_count = Window.partitionBy("genre").orderBy(col("rating_count").asc())
least_10_rated_per_genre = (
    movies_ratings.groupBy("genre", "movieId", "title")
    .agg(count("rating").alias("rating_count"))
    .filter(col("rating_count") > 10)  # Условие > 10
```

```
    .withColumn("rank", row_number().over(window_spec_rating_count))
    .filter(col("rank") <= 10)  # Топ-10 для каждого жанра
    .orderBy("genre", "rank")
)
least_10_rated_per_genre.show(100, truncate=False)
```

```
[Stage 15:=================================================>        (6 + 1) / 7]
```

genre	movieId	title	rating_count	rank
Animation	182155	Donald's Penguin (1939)	11	1
Animation	182189	The Pied Piper (1933)	11	2
Animation	216819	The Art of Skiing (1941)	11	3
Animation	251630	Maggie Simpson in The Force Awakens from Its Nap (2021)	11	4
Animation	178967	The Lion, the Witch and the Wardrobe (1979)	11	5
Animation	238818	The Games of Angels (1964)	11	6
Animation	229593	Alien Xmas (2020)	11	7
Animation	215413	Away (2019)	11	8
Animation	204632	Technological Threat (1988)	11	9
Animation	163519	Mouse in Manhattan (1945)	11	10
Documentary	70831	Krakatoa: The Last Days (2006)	11	1
Documentary	278170	Untold: The Rise and Fall of AND1 (2022)	11	2
Documentary	162452	Ghosts of Abu Ghraib (2007)	11	3
Documentary	163563	Can We Take a Joke? (2015)	11	4
Documentary	211468	The Rise of Jordan Peterson (2019)	11	5
Documentary	64385	Body of War (2007)	11	6
Documentary	67009	Frontrunners (2008)	11	7
Documentary	199596	Toni Morrison: The Pieces I Am (2019)	11	8
Documentary	48626	Once in a Lifetime: The Extraordinary Story of the New York Cosmos (2006)	11	9
Documentary	133221	The Man Who Skied Down Everest (1975)	11	10
Romance	120290	My Rainy Days (2009)	11	1
Romance	77835	Stage Door Canteen (1943)	11	2
Romance	199071	Under the Eiffel Tower (2019)	11	3
Romance	103528	Shadow Riders, The (1982)	11	4
Romance	210013	Christmas with a Prince (2018)	11	5
Romance	192581	The Matchmaker's Playbook (2018)	11	6
Romance	148640	The American Mall (2008)	11	7
Romance	7441	Thousand Clouds of Peace, A (Mil nubes de paz cercan el cielo, amor, jamás acabarás de ser amor) (2003)	11	8
Romance	33852	Becky Sharp (1935)	11	9
Romance	113630	Man Who Couldn't Say No, The (Mies joka ei osannut sanoa EI) (1975)	11	10

In [20]:

```
# 4. Топ-10 фильмов с наибольшим средним рейтингом (>10 рейтингов) для каждого жанра
window_spec_avg_rating_desc = Window.partitionBy("genre").orderBy(col("avg_rating").desc())
top_10_avg_rated_per_genre = (
    movies_ratings.groupBy("genre", "movieId", "title")
    .agg(
        count("rating").alias("rating_count"),
        mean("rating").alias("avg_rating")
    )
    .filter(col("rating_count") > 10)  # Условие > 10 рейтингов
    .withColumn("rank", row_number().over(window_spec_avg_rating_desc))
    .filter(col("rank") <= 10)  # Топ-10 для каждого жанра
    .orderBy("genre", "rank")
)
top_10_avg_rated_per_genre.show(100, truncate=False)
```

```
[Stage 22:=================================>                        (4 + 3) / 7]
```

genre	movieId	title	rating_count	avg_rating	rank
Animation	163809	Over the Garden Wall (2013)	1430	4.256993006993007	1
Animation	286897	Spider-Man: Across the Spider-Verse (2023)	528	4.252840909090909	2
Animation	256991	Adventure Time: Elements (2017)	12	4.25	3
Animation	5618	Spirited Away (Sen to Chihiro no kamikakushi) (2001)	35375	4.226035335689046	4
Animation	249180	Violet Evergarden: The Movie (2020)	25	4.22	5
Animation	157373	It's Such a Beautiful Day (2011)	328	4.1935975609756095	6
Animation	195159	Spider-Man: Into the Spider-Verse (2018)	10885	4.192053284336242	7
Animation	163134	Your Name. (2016)	3940	4.16751269035533	8
Animation	3000	Princess Mononoke (Mononoke-hime) (1997)	18226	4.166026555470207	9
Animation	5971	My Neighbor Totoro (Tonari no Totoro) (1988)	14010	4.163490364025696	10
Documentary	102672	New York: A Documentary Film (1999)	11	4.5	1
Documentary	171011	Planet Earth II (2016)	2041	4.451739343459089	2
Documentary	159817	Planet Earth (2006)	3015	4.448092868988391	3
Documentary	215615	Pink Floyd: Pulse (1995)	11	4.318181818181818	4
Documentary	179135	Blue Planet II (2017)	1267	4.312943962115233	5
Documentary	142115	The Blue Planet (2001)	1080	4.25	6
Documentary	147124	The Roosevelts: An Intimate History (2014)	46	4.239130434782608	7
Documentary	105250	Century of the Self, The (2002)	397	4.221662468513854	8
Documentary	239316	Can't Get You Out of My Head: An Emotional History of the Modern World (2021)	47	4.212765957446808	9
Documentary	172725	The Secret Life of Chaos (2010)	12	4.208333333333333	10
Romance	203847	Kumbalangi Nights (2019)	18	4.305555555555555	1
Romance	263965	Downton Abbey: Christmas Special 2015 (2015)	16	4.25	2
Romance	249180	Violet Evergarden: The Movie (2020)	25	4.22	3
Romance	122282	Pride and Prejudice (1980)	58	4.206896551724138	4
Romance	44555	Lives of Others, The (Das leben der Anderen) (2006)	12626	4.201409789323618	5
Romance	172719	Notre Dame de Paris (1998)	15	4.2	6
Romance	912	Casablanca (1942)	34813	4.195889466578577	7
Romance	922	Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	9627	4.189934559052665	8
Romance	908	North by Northwest (1959)	21883	4.187337202394553	9
Romance	163134	Your Name. (2016)	3940	4.16751269035533	10

In [21]:

```
# 5. Топ-10 фильмов с наименьшим средним рейтингом (>10 рейтингов) для каждого жанра
window_spec_avg_rating = Window.partitionBy("genre").orderBy(col("avg_rating").asc())
least_10_avg_rated_per_genre = (
```

```
    movies_ratings.groupBy("genre", "movieId", "title")
    .agg(
        count("rating").alias("rating_count"),
        mean("rating").alias("avg_rating")
    )
    .filter(col("rating_count") > 10)  # Условие > 10 рейтингов
    .withColumn("rank", row_number().over(window_spec_avg_rating))
    .filter(col("rank") <= 10)  # Топ-10 для каждого жанра
    .orderBy("genre", "rank")
)
least_10_avg_rated_per_genre.show(100, truncate=False)
```

```
[Stage 29:=========================================>                (5 + 2) / 7]
+-----------+-------+--------------------------------------------------------+------------+------------------+----+
|genre      |movieId|title                                                   |rating_count|avg_rating        |rank|
+-----------+-------+--------------------------------------------------------+------------+------------------+----+
|Animation  |120222 |Foodfight! (2012)                                       |46          |0.9456521739130435|1   |
|Animation  |170903 |The Swan Princess Christmas (2012)                      |11          |1.1363636363636365|2   |
|Animation  |153564 |The Amazing Bulk (2012)                                 |15          |1.1666666666666667|3   |
|Animation  |145096 |Barbie & Her Sisters in the Great Puppy Adventure (2015)|78          |1.1923076923076923|4   |
|Animation  |151313 |Norm of the North (2016)                                |58          |1.5086206896551724|5   |
|Animation  |6371   |Pokémon Heroes (2003)                                   |431         |1.519721577726218 |6   |
|Animation  |5672   |Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)     |614         |1.5358306188925082|7   |
|Animation  |136674 |Maya the Bee Movie (2014)                               |12          |1.5833333333333333|8   |
|Animation  |200802 |Norm of the North: Keys to the Kingdom (2018)           |11          |1.5909090909090908|9   |
|Animation  |179107 |The Legend of the Titanic (1999)                        |11          |1.6363636363636365|10  |
|Documentary|107704 |Justin Bieber's Believe (2013)                          |21          |0.9285714285714286|1   |
|Documentary|193183 |Death of a Nation (2018)                                |14          |1.2142857142857142|2   |
|Documentary|5739   |Faces of Death 6 (1996)                                 |178         |1.2865168539325842|3   |
|Documentary|121103 |Justin Bieber: Never Say Never (2011)                   |63          |1.2936507936507937|4   |
|Documentary|5738   |Faces of Death 5 (1996)                                 |160         |1.365625          |5   |
|Documentary|5740   |Faces of Death: Fact or Fiction? (1999)                 |133         |1.3759398496240602|6   |
|Documentary|158731 |Kony 2012 (2012)                                        |13          |1.3846153846153846|7   |
|Documentary|5737   |Faces of Death 4 (1990)                                 |185         |1.3945945945945946|8   |
|Documentary|5736   |Faces of Death 3 (1985)                                 |207         |1.4951690821256038|9   |
|Documentary|166741 |Electrocuting an Elephant (1903)                        |32          |1.53125           |10  |
|Romance    |171479 |Kidnapping, Caucasian Style (2014)                      |17          |0.9117647058823529|1   |
|Romance    |6483   |From Justin to Kelly (2003)                             |489         |1.0112474437627812|2   |
|Romance    |4775   |Glitter (2001)                                          |788         |1.151015228426396 |3   |
|Romance    |6587   |Gigli (2003)                                            |872         |1.2144495412844036|4   |
|Romance    |103186 |Wedding Trough (Vase de noces) (1975)                   |15          |1.4333333333333333|5   |
|Romance    |171555 |Classmates (2016)                                       |11          |1.5               |6   |
|Romance    |145388 |Forever (2015)                                          |14          |1.5               |7   |
|Romance    |153816 |Tashan (2008)                                           |11          |1.5454545454545454|8   |
|Romance    |3390   |Shanghai Surprise (1986)                                |251         |1.5637450199203187|9   |
|Romance    |43919  |Date Movie (2006)                                       |1130        |1.6176991150442477|10  |
+-----------+-------+--------------------------------------------------------+------------+------------------+----+
```

In [22]:

```
# Остановка SparkSession
spark.stop()
```

Задание 2. Коллаборативная фильтрация

Вариант 2. По схожести объектов

1. Разделите данные с рейтингами на обучающее (train_init - 0.8) и тестовое подмножества (test - 0.2), определите среднее значение рейтинга в обучающем подмножестве и вычислите rmse для тестового подмножества, если для всех значений из test предсказывается среднее значение рейтинга
2. Реализуйте коллаборативную фильтрацию в соответствии с вариантом. Для определения схожести используйте train_init, для расчета rmse - test
3. Определите rmse для тестового подмножества

In [29]:

```
# 1. Подготовка данных. Разделение данных на обучающую и тестовую выборки,
#    вычисление среднего значения рейтинга для обучающей выборки и RMSE для тестовой выборки.

from pyspark.sql import SparkSession
from pyspark.sql.functions import col, explode, split, count, mean, desc, asc, lit, avg, sqrt
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number
from pyspark.ml.evaluation import RegressionEvaluator


# Инициализация SparkSession
spark = SparkSession.builder.appName("Movies2 Marchuk").getOrCreate()

# Загрузка данных из HDFS
ratings = spark.read.csv("/dataset/hw4/big/ratings.csv", header=True, inferSchema=True)

# Разделение на train_init (80%) и test (20%)
train_init, test = ratings.randomSplit([0.8, 0.2], seed=42)

# Среднее значение рейтинга в train_init
mean_rating = train_init.select(avg("rating").alias("mean_rating")).collect()[0]["mean_rating"]

# Предсказание среднего значения для тестового набора
test_with_predictions = test.withColumn("prediction", lit(mean_rating))

# Вычисление RMSE для тестового подмножества
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
rmse_mean = evaluator.evaluate(test_with_predictions)

print(f"Среднее значение рейтинга в train_init: {mean_rating:.2f}")
print(f"RMSE при предсказании среднего рейтинга: {rmse_mean:.4f}")
```

```
[Stage 5:=====================================================>    (6 + 1) / 7]
Среднее значение рейтинга в train_init: 3.54
RMSE при предсказании среднего рейтинга: 1.0640
```

In [ ]:
```
#2. Реализуйте коллаборативную фильтрацию в соответствии с вариантом.
#   Для определения схожести используйте train_init, для расчета rmse - test


from pyspark.sql.functions import col, sqrt, sum as spark_sum
from pyspark.ml.evaluation import RegressionEvaluator

# 1. Создание матрицы рейтингов "userId x movieId" для train_init
ratings_matrix = (
    train_init.groupBy("userId", "movieId")
    .agg(mean("rating").alias("rating"))
)

# 2. Вычисление косинусного сходства между фильмами
ratings_self_join = ratings_matrix.alias("r1").join(
    ratings_matrix.alias("r2"),
    col("r1.userId") == col("r2.userId")  # Сравнение по одному и тому же пользователю
)

# Подсчет числителя (скалярное произведение) и знаменателя (длины векторов)
movie_similarity = (
    ratings_self_join.groupBy("r1.movieId", "r2.movieId")
    .agg(
        spark_sum(col("r1.rating") * col("r2.rating")).alias("dot_product"),
        sqrt(spark_sum(col("r1.rating")**2)).alias("norm_r1"),
        sqrt(spark_sum(col("r2.rating")**2)).alias("norm_r2"),
    )
    .withColumn("similarity", col("dot_product") / (col("norm_r1") * col("norm_r2")))
    .filter(col("r1.movieId") != col("r2.movieId"))  # Убираем сравнение фильма с самим собой
)
# 3. Генерация предсказаний
# Для каждого фильма из test находим его ближайших соседей в train_init
predictions = (
    test.alias("t").join(
        movie_similarity.select(
            col("r1.movieId").alias("movieId_test"),  # Переименуем столбцы для удобства
            col("r2.movieId").alias("movieId_train"),
            "similarity"
        ).alias("ms"),
        col("t.movieId") == col("ms.movieId_test"),  # Связываем фильмы из тестового множества с похожими
        "left"
    )
    .join(
        train_init.alias("tr"),
        col("ms.movieId_train") == col("tr.movieId"),  # Связываем с рейтингами соседей
        "left"
    )
    .groupBy("t.userId", "t.movieId")
    .agg(
        spark_sum(col("ms.similarity") * col("tr.rating")).alias("weighted_sum"),
        spark_sum(col("ms.similarity")).alias("similarity_sum"),
    )
    .withColumn("prediction", col("weighted_sum") / col("similarity_sum"))
    .select("userId", "movieId", "prediction")
)

# 4. Оценка качества модели (RMSE)
# Объединяем предсказания с реальными рейтингами
test_with_predictions = test.join(predictions, ["userId", "movieId"], "left")

# Заполняем пропущенные значения средним рейтингом (если фильм не имеет похожих)
test_with_predictions = test_with_predictions.fillna(mean_rating, subset=["prediction"])

# Вычисление RMSE
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
rmse_collaborative = evaluator.evaluate(test_with_predictions)

print(f"RMSE для коллаборативной фильтрации: {rmse_collaborative:.4f}")
```

Для расчетов не хватает пространства на диске, больше выделить физически не могу :(

In [30]:
```
# Остановка SparkSession
spark.stop()
```

Задание 3. Факторизация матрицы

1. Выберите модель ALS по минимальному значению rmse. Для этого используйте кросс-валидацию k-folds с k=4

Параметры:

Количество факторов: [5, 10, 15]

Регуляризация: [0.001, 0.01, 0.1, 1, 10]

⚠️ Замечание: Если какие-то элементы из тестового/валидационного подмножества не встречались в обучающем, то rmse будет NaN

2. Сравните результаты рекомендаций посредством коллаборативной фильтрации и факторизации матрицы рейтингов

In [31]:
```
# 1. Выберите модель ALS по минимальному значению rmse. Для этого используйте кросс-валидацию k-folds с k=4



from pyspark.sql import SparkSession
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql.functions import col
import numpy as np
```

```python
# Инициализация SparkSession
spark = SparkSession.builder.appName("Movies3 Marchuk").getOrCreate()

# Загрузка данных
ratings = spark.read.csv("/dataset/hw4/big/ratings.csv", header=True, inferSchema=True)

# Параметры для кросс-валидации
k_folds = 4
factors = [5, 10, 15]
reg_params = [0.001, 0.01, 0.1, 1, 10]
seed = 42

# Функция для вычисления RMSE
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")

# Кросс-валидация
min_rmse = float("inf")
best_model_params = None

# Создаем k фолдов
folds = ratings.randomSplit([1.0 / k_folds] * k_folds, seed=seed)

for factor in factors:
    for reg_param in reg_params:
        fold_rmse = []

        for i in range(k_folds):
            # Определяем обучающую и валидационную выборки
            validation = folds[i]
            train = spark.createDataFrame(
                [row for j, fold in enumerate(folds) if j != i for row in fold.collect()]
            )

            # Инициализация модели ALS
            als = ALS(
                maxIter=10,
                rank=factor,
                regParam=reg_param,
                userCol="userId",
                itemCol="movieId",
                ratingCol="rating",
                coldStartStrategy="drop",  # Убираем NaN предсказания
                seed=seed,
            )

            # Обучение модели
            model = als.fit(train)

            # Предсказания на валидационном наборе
            predictions = model.transform(validation)

            # Вычисление RMSE
            rmse = evaluator.evaluate(predictions)
            fold_rmse.append(rmse)

        # Среднее RMSE для текущих параметров
        avg_rmse = np.mean(fold_rmse)

        print(f"Factor: {factor}, RegParam: {reg_param}, RMSE: {avg_rmse:.4f}")

        # Сохранение лучших параметров
        if avg_rmse < min_rmse:
            min_rmse = avg_rmse
            best_model_params = {"factor": factor, "regParam": reg_param}

print(f"\nЛучшие параметры: {best_model_params}")
print(f"Минимальное RMSE: {min_rmse:.4f}")
```

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
Cell In[31], line 41
     37 for i in range(k_folds):
     38     # Определяем обучающую и валидационную выборки
     39     validation = folds[i]
     40     train = spark.createDataFrame(
---> 41         [row for j, fold in enumerate(folds) if j != i for row in fold.collect()]
     42     )
     44     # Инициализация модели ALS
     45     als = ALS(
     46         maxIter=10,
     47         rank=factor,
   (...)
     53         seed=seed,
     54     )

Cell In[31], line 41, in <listcomp>(.0)
     37 for i in range(k_folds):
     38     # Определяем обучающую и валидационную выборки
     39     validation = folds[i]
     40     train = spark.createDataFrame(
---> 41         [row for j, fold in enumerate(folds) if j != i for row in fold.collect()]
     42     )
     44     # Инициализация модели ALS
     45     als = ALS(
     46         maxIter=10,
     47         rank=factor,
   (...)
     53         seed=seed,
     54     )

File ~/_practice/spark-3.5.4-bin-hadoop3/python/pyspark/sql/dataframe.py:1263, in DataFrame.collect(self)
   1243 """Returns all the records as a list of :class:`Row`.
   1244 
   1245 .. versionadded:: 1.3.0
   (...)
   1260 [Row(age=14, name='Tom'), Row(age=23, name='Alice'), Row(age=16, name='Bob')]
   1261 """
   1262 with SCCallSiteSync(self._sc):
-> 1263     sock_info = self._jdf.collectToPython()
   1264 return list(_load_from_socket(sock_info, BatchedSerializer(CPickleSerializer())))

File ~/_practice/spark-3.5.4-bin-hadoop3/python/lib/py4j-0.10.9.7-src.zip/py4j/java_gateway.py:1322, in JavaMember.__call__(self, *args)
   1316 command = proto.CALL_COMMAND_NAME +\
   1317     self.command_header +\
   1318     args_command +\
   1319     proto.END_COMMAND_PART
   1321 answer = self.gateway_client.send_command(command)
-> 1322 return_value = get_return_value(
   1323     answer, self.gateway_client, self.target_id, self.name)
   1325 for temp_arg in temp_args:
   1326     if hasattr(temp_arg, "_detach"):

File ~/_practice/spark-3.5.4-bin-hadoop3/python/pyspark/errors/exceptions/captured.py:179, in capture_sql_exception.<locals>.deco(*a, **kw)
    177 def deco(*a: Any, **kw: Any) -> Any:
    178     try:
--> 179         return f(*a, **kw)
    180     except Py4JJavaError as e:
    181         converted = convert_exception(e.java_exception)

File ~/_practice/spark-3.5.4-bin-hadoop3/python/lib/py4j-0.10.9.7-src.zip/py4j/protocol.py:326, in get_return_value(answer, gateway_client, target
_id, name)
    324 value = OUTPUT_CONVERTER[type](answer[2:], gateway_client)
    325 if answer[1] == REFERENCE_TYPE:
--> 326     raise Py4JJavaError(
    327         "An error occurred while calling {0}{1}{2}.\n".
    328         format(target_id, ".", name), value)
    329 else:
    330     raise Py4JError(
    331         "An error occurred while calling {0}{1}{2}. Trace:\n{3}\n".
    332         format(target_id, ".", name, value))

Py4JJavaError: An error occurred while calling o9309.collectToPython.
: java.lang.OutOfMemoryError: Java heap space
	at scala.collection.mutable.ResizableArray.ensureSize(ResizableArray.scala:106)
	at scala.collection.mutable.ResizableArray.ensureSize$(ResizableArray.scala:96)
	at scala.collection.mutable.ArrayBuffer.ensureSize(ArrayBuffer.scala:49)
	at scala.collection.mutable.ArrayBuffer.$plus$eq(ArrayBuffer.scala:85)
	at org.apache.spark.sql.execution.SparkPlan.$anonfun$executeCollect$2(SparkPlan.scala:449)
	at org.apache.spark.sql.execution.SparkPlan$$Lambda$3541/833111582.apply(Unknown Source)
	at scala.collection.Iterator.foreach(Iterator.scala:943)
	at scala.collection.Iterator.foreach$(Iterator.scala:943)
	at org.apache.spark.util.NextIterator.foreach(NextIterator.scala:21)
	at org.apache.spark.sql.execution.SparkPlan.$anonfun$executeCollect$1(SparkPlan.scala:449)
	at org.apache.spark.sql.execution.SparkPlan.$anonfun$executeCollect$1$adapted(SparkPlan.scala:448)
	at org.apache.spark.sql.execution.SparkPlan$$Lambda$3540/1478847382.apply(Unknown Source)
	at scala.collection.IndexedSeqOptimized.foreach(IndexedSeqOptimized.scala:36)
	at scala.collection.IndexedSeqOptimized.foreach$(IndexedSeqOptimized.scala:33)
	at scala.collection.mutable.ArrayOps$ofRef.foreach(ArrayOps.scala:198)
	at org.apache.spark.sql.execution.SparkPlan.executeCollect(SparkPlan.scala:448)
	at org.apache.spark.sql.Dataset.$anonfun$collectToPython$1(Dataset.scala:4149)
	at org.apache.spark.sql.Dataset$$Lambda$4049/946361721.apply(Unknown Source)
	at org.apache.spark.sql.Dataset.$anonfun$withAction$2(Dataset.scala:4323)
	at org.apache.spark.sql.Dataset$$Lambda$2021/472450626.apply(Unknown Source)
	at org.apache.spark.sql.execution.QueryExecution$.withInternalError(QueryExecution.scala:546)
	at org.apache.spark.sql.Dataset.$anonfun$withAction$1(Dataset.scala:4321)
	at org.apache.spark.sql.Dataset$$Lambda$1673/156838736.apply(Unknown Source)
	at org.apache.spark.sql.execution.SQLExecution$.$anonfun$withNewExecutionId$6(SQLExecution.scala:125)
	at org.apache.spark.sql.execution.SQLExecution$$$Lambda$1684/1386522056.apply(Unknown Source)
```

```
        at org.apache.spark.sql.execution.SQLExecution$.withSQLConfPropagated(SQLExecution.scala:201)
        at org.apache.spark.sql.execution.SQLExecution$.$anonfun$withNewExecutionId$1(SQLExecution.scala:108)
        at org.apache.spark.sql.execution.SQLExecution$$$Lambda$1674/1690301035.apply(Unknown Source)
        at org.apache.spark.sql.SparkSession.withActive(SparkSession.scala:900)
        at org.apache.spark.sql.execution.SQLExecution$.withNewExecutionId(SQLExecution.scala:66)
        at org.apache.spark.sql.Dataset.withAction(Dataset.scala:4321)
        at org.apache.spark.sql.Dataset.collectToPython(Dataset.scala:4146)
```

In [ ]:

```
То же самое, не хватает места(
```

In [32]:

```
# Остановка SparkSession
spark.stop()
```

In [ ]: