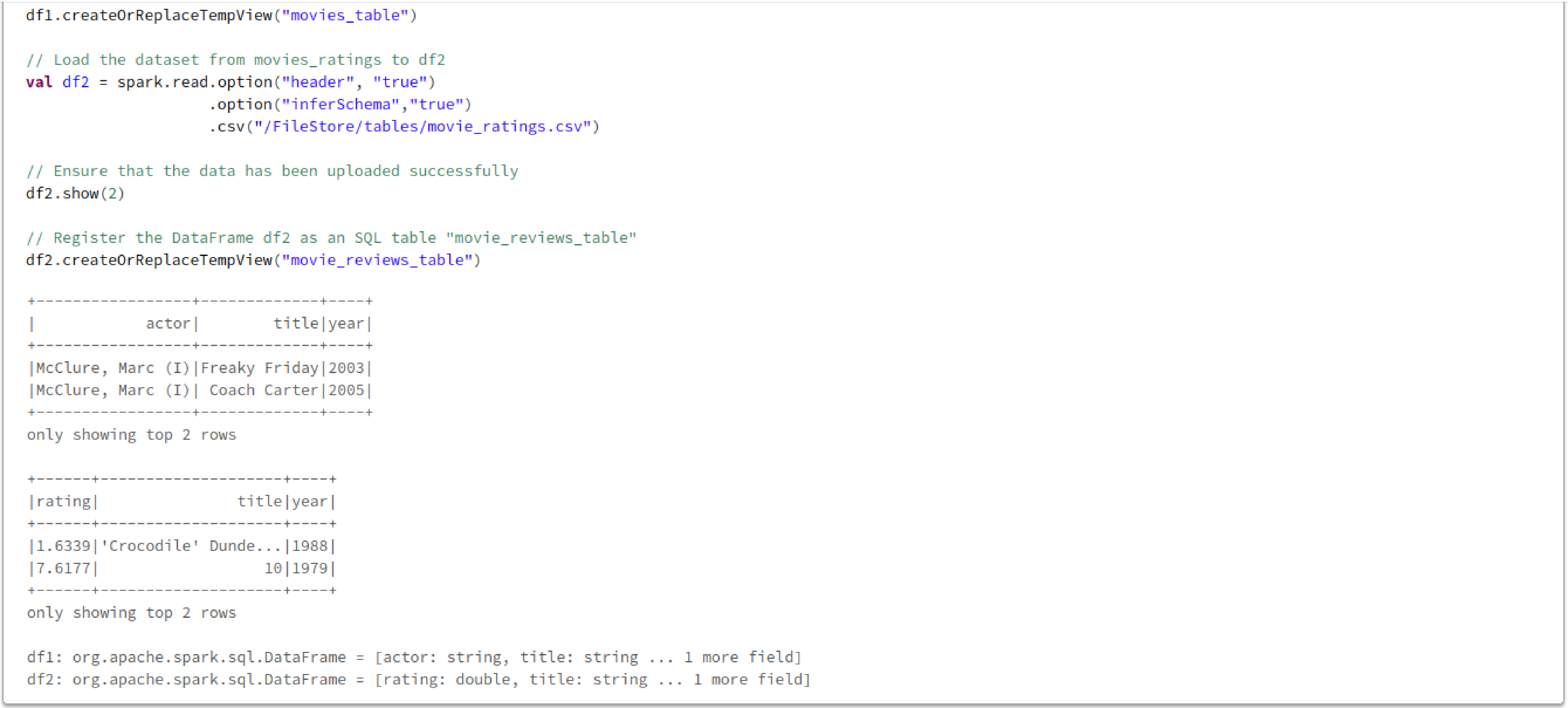

```
df1.createOrReplaceTempView("movies_table")

// Load the dataset from movies_ratings to df2
val df2 = spark.read.option("header", "true")
                    .option("inferSchema","true")
                    .csv("/FileStore/tables/movie_ratings.csv")

// Ensure that the data has been uploaded successfully
df2.show(2)

// Register the DataFrame df2 as an SQL table "movie_reviews_table"
df2.createOrReplaceTempView("movie_reviews_table")
```

```
+-----------------+-------------+----+
|            actor|        title|year|
+-----------------+-------------+----+
|McClure, Marc (I)|Freaky Friday|2003|
|McClure, Marc (I)| Coach Carter|2005|
+-----------------+-------------+----+
only showing top 2 rows

+------+--------------------+----+
|rating|               title|year|
+------+--------------------+----+
|1.6339|'Crocodile' Dunde...|1988|
|7.6177|                  10|1979|
+------+--------------------+----+
only showing top 2 rows

df1: org.apache.spark.sql.DataFrame = [actor: string, title: string ... 1 more field]
df2: org.apache.spark.sql.DataFrame = [rating: double, title: string ... 1 more field]
```

```
/*
Write DataFrame-based Spark code to find the number of distinct movies in the file movies.csv
*/

import org.apache.spark.sql.functions.countDistinct

// Count distinct movie titles with renaming
val distinctMoviesCountDataFrameWay = df1.select(countDistinct("title").as("Distinct Movies Count"))

distinctMoviesCountDataFrameWay.show
```

```
+---------------------+
|Distinct Movies Count|
+---------------------+
|                 1409|
+---------------------+

import org.apache.spark.sql.functions.countDistinct
distinctMoviesCountDataFrameWay: org.apache.spark.sql.DataFrame = [Distinct Movies Count: bigint]
```

```
/*
Write DataFrame-based Spark code to find the titles of the movies that appear in
the file movies.csv but do not have a rating in the file movie_ratings.csv. Remark: the answer
could be empty.
*/

// Expression to join both data frame based on similar title
var joinExpression = df1.col("title") === df2.col("title")

/*
* Steps:
* 1. Left anti join: Keeps rows from movies.csv (df1) which does not match join expression
* 2. Select only title column
* 3. Drop duplicate title values if any
*/
val moviesOnlyInMoviesCsvDataFrameWay = df1.join(df2, joinExpression, "left_anti")
                                            .select("title")
                                            .dropDuplicates("title")

moviesOnlyInMoviesCsvDataFrameWay.show
```

```
+-----+
|title|
+-----+
+-----+

joinExpression: org.apache.spark.sql.Column = (title = title)
moviesOnlyInMoviesCsvDataFrameWay: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [title: string]
```

```
/*
Write DataFrame-based Spark code to find the number of movies that appear in the
ratings file (i.e., movie_ratings.csv) but not in the movies file (i.e., movies.csv).
*/

import org.apache.spark.sql.functions.countDistinct

// Expression to join both data frame based on similar title
var joinExpression = df1.col("title") === df2.col("title")

/*
* Steps:
* 1. Left anti join: Keeps rows from movie_ratings.csv (df2) which does not match join expression
* 2. Count distinct values of title column
*/
var moviesOnlyInMovieRatingsCsvDataFrameWay = df2.join(df1, joinExpression, "left_anti")
                                                   .select(countDistinct("title"))

moviesOnlyInMovieRatingsCsvDataFrameWay.show
```

```
+---------------------+
|count(DISTINCT title)|
+---------------------+
|                 2127|
+---------------------+

import org.apache.spark.sql.functions.countDistinct
joinExpression: org.apache.spark.sql.Column = (title = title)
moviesOnlyInMovieRatingsCsvDataFrameWay: org.apache.spark.sql.DataFrame = [count(DISTINCT title): bigint]
```

```
/*
Write DataFrame-based Spark code to find the total number of distinct movies that
appear in either movies.csv, or movie_ratings.csv, or both.
*/

import org.apache.spark.sql.functions.countDistinct

/*
* Steps:
* 1. Union only title column from both movies.csv and movie_ratings.csv
* 2. Count distinct values from the union-ed title column
*/
val totalMoviesInBothTableDrameWay = df1.select("title")
                                        .union(df2.select("title"))
                                        .select(countDistinct("title"))

totalMoviesInBothTableDrameWay.show
```

```
+---------------------+
|count(DISTINCT title)|
+---------------------+
|                 3536|
+---------------------+

import org.apache.spark.sql.functions.countDistinct
totalMoviesInBothTableDrameWay: org.apache.spark.sql.DataFrame = [count(DISTINCT title): bigint]
```

```
* 2. Filtering out movies that were remade more than 1 year
* 3. Select the asked columns, title and year with ascending order by title
*/
val remadeMoviesDataFrameWay = df2.withColumn("count", count("year").over(windowSpec))
                                   .where("count > 1")
                                   .select("title", "year")
                                   .orderBy('title.asc)

remadeMoviesDataFrameWay.show(false)
```

```
+-------------------------+----+
|title                    |year|
+-------------------------+----+
|A Nightmare on Elm Street|1984|
|A Nightmare on Elm Street|2010|
|Casino Royale            |1967|
|Casino Royale            |2006|
|Conan the Barbarian      |2011|
|Conan the Barbarian      |1982|
|Death at a Funeral       |2007|
|Death at a Funeral       |2010|
|Dracula                  |1979|
|Dracula                  |1992|
|Footloose                |2011|
|Footloose                |1984|
|Fright Night             |1985|
|Fright Night             |2011|
|Hairspray                |1988|
|Hairspray                |2007|
|Halloween                |2007|
|Halloween                |1978|
```

```scala
/*
Write DataFrame-based Spark code to find the rating for every movie that the actor
"Branson, Richard" appeared in. Schema of the output should be (title, year, rating)
*/

// Create a sequence to join on both title and year
var joinSequnce = Seq("title", "year")

/*
* Steps:
* 1. Inner join the two data frames based on both similar title and year to match appearance
* 2. Filter out the actor "Branson, Richard" and select the asked columns
*/
val movieRatingOfBransonDataFrameWay = df1.join(df2, joinSequnce, "inner")
                                          .where("actor = 'Branson, Richard'")
                                          .select("title", "year", "rating")

movieRatingOfBransonDataFrameWay.show
```

```
+--------------------+----+------+
|               title|year|rating|
+--------------------+----+------+
|       Casino Royale|2006|0.2078|
|Around the World ...|2004|1.8631|
|    Superman Returns|2006|0.1889|
+--------------------+----+------+

joinSequnce: Seq[String] = List(title, year)
movieRatingOfBransonDataFrameWay: org.apache.spark.sql.DataFrame = [title: string, year: int ... 1 more field]
```

```
 * 3. Sort the output by year on ascending order
 */
val highestRatedMoviePerYearWithAcotsDataFrameWay = highestRatedMoviePerYearDataFrameWay
                                                     .join(df1, joinSequence, "left_outer")
                                                     .groupBy("year", "title", "rating")
                                                     .agg(collect_list("actor").as("actors"))
                                                     .orderBy('year.asc)

highestRatedMoviePerYearWithAcotsDataFrameWay.show(false)
```

```
+----+-------------------------------+-------+-------------------------------+
|year|title                          |rating |actors                         |
+----+-------------------------------+-------+-------------------------------+
|1937|Snow White and the Seven Dwarfs|2.2207 |[]                             |
|1939|The Wizard of Oz               |7.9215 |[]                             |
|1940|Pinocchio                      |7.8557 |[]                             |
|1942|Bambi                          |1.5053 |[]                             |
|1946|Song of the South              |7.602  |[]                             |
|1950|Cinderella                     |9.4226 |[]                             |
|1953|Peter Pan                      |5.4756 |[]                             |
|1954|Rear Window                    |10.7625|[]                             |
|1955|Lady and the Tramp             |5.1258 |[]                             |
|1956|Around the World in Eighty Days|14.0607|[]                             |
|1959|Sleeping Beauty                |6.3919 |[]                             |
|1960|Psycho                         |10.6375|[]                             |
|1961|One Hundred and One Dalmatians |0.6726 |[Wright, Ben (I), Wickes, Mary]|
|1962|The Longest Day                |12.8866|[]                             |
|1963|It's a Mad Mad Mad Mad World   |6.626  |[]                             |
|1964|My Fair Lady                   |7.587  |[]                             |
|1965|Doctor Zhivago                 |4.9304 |[]                             |
|1966|Who's Afraid of Virginia Woolf?|11.1111|[]                             |
// 1. Join based on the join expression, 2. Group by (actor 1, actor 2) sets and count, and 3. Descending order by count
var actorsWorkedTogetherDataFrameWay = df1ForActor1.join(df1ForActor2, joinExpression)
                                                    .groupBy("actor 1", "actor 2")
                                                    .count()
                                                    .orderBy('count.desc)

actorsWorkedTogetherDataFrameWay.show(false)
```

```
+------------------+-------------------+-----+
|actor 1           |actor 2            |count|
+------------------+-------------------+-----+
|Lynn, Sherry (I)  |McGowan, Mickie    |23   |
|Bergen, Bob (I)   |McGowan, Mickie    |19   |
|Bergen, Bob (I)   |Lynn, Sherry (I)   |19   |
|Angel, Jack (I)   |McGowan, Mickie    |17   |
|Angel, Jack (I)   |Lynn, Sherry (I)   |17   |
|Lynn, Sherry (I)  |Rabson, Jan        |16   |
|McGowan, Mickie   |Rabson, Jan        |16   |
|Darling, Jennifer|McGowan, Mickie    |15   |
|Bergen, Bob (I)   |Rabson, Jan        |14   |
|Bergen, Bob (I)   |Harnell, Jess      |14   |
|Darling, Jennifer|Lynn, Sherry (I)   |14   |
|Farmer, Bill (I)  |McGowan, Mickie    |14   |
|Harnell, Jess     |McGowan, Mickie    |14   |
|Sandler, Adam (I)|Schneider, Rob (I)|14   |
|Angel, Jack (I)   |Bergen, Bob (I)    |13   |
|Bergen, Bob (I)   |Bumpass, Rodger    |13   |
|Farmer, Bill (I)  |Lynn, Sherry (I)   |13   |
|Harnell, Jess     |Lynn, Sherry (I)   |13   |
```