Task3

```
//task3
// Write DataFrame-based Spark code to find the number of distinct movies in the file movies.csv
df1.select("title").distinct().count() //selecting the distinct movie title and counting them
```

```
(3) Spark Jobs
res24: Long = 1409
Command took 2.05 seconds -- by sc26s@login.missouristate.edu at 10/20/2023, 12:46:48 PM on HW
```

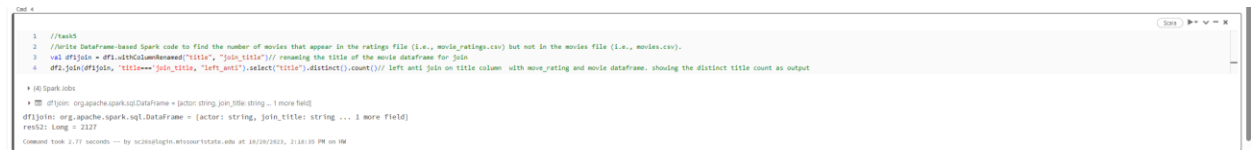
Task 4

```
//task4
//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.
val df2join = df2.withColumnRenamed("title", "join_title")// renaming the title in movie_rating dataframe
df1.join(df2join, "title==='join_title,"left_anti" ).select("title").show(false) // left anti join on title column  with movie and move_rating dataframe. showing the title as output
```

```
(2) Spark Jobs
df2join:  org.apache.spark.sql.DataFrame = [rating: double, join_title: string ... 1 more field]
+------+
|title|
+------+
+------+

df2join: org.apache.spark.sql.DataFrame = [rating: double, join_title: string ... 1 more field]
Command took 3.25 seconds -- by sc26s@login.missouristate.edu at 10/20/2023, 12:49:26 PM on HW
```

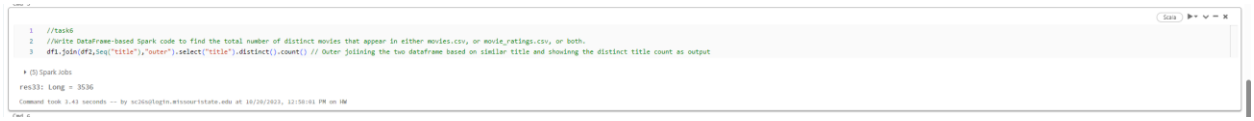
Task 5



```
//task5
//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).
val df1join = df1.withColumnRenamed("title", "join_title")// renaming the title of the movie dataframe for join
df2.join(df1join, 'title==='join_title, "left_anti").select("title").distinct().count()// left anti join on title column  with move_rating and movie dataframe. showing the distinct title count as output
```

```
(4) Spark Jobs
df1join:  org.apache.spark.sql.DataFrame = [actor: string, join_title: string ... 1 more field]
df1join: org.apache.spark.sql.DataFrame = [actor: string, join_title: string ... 1 more field]
res52: Long = 2127
Command took 2.77 seconds -- by sc26s@login.missouristate.edu at 10/20/2023, 2:18:35 PM on HW
```

Task 6



```
//task6
//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.
df1.join(df2,Seq("title"),"outer").select("title").distinct().count() // Outer joining the two dataframe based on similar title and showing the distinct title count as output
```

```
(5) Spark Jobs
res33: Long = 3536
Command took 3.43 seconds -- by sc26s@login.missouristate.edu at 10/20/2023, 12:58:01 PM on HW
```

Task 7

```
//task7
//Write DataFrame-based Spark code to find the title and year for movies that were remade. These movies appear more than once in the ratings file with the same title but different years. Sort the output by title.
val df2join=df2.groupBy("title").count().where("count>1").withColumnRenamed("title","join_title")// Grouping the moview _rating dataframe with title and counting them. Filtering where count>1 to get the remake, renaming the column title of the movie_rating dataframe for join
df2.join(df2join,'title==='join_title).select("title","year").sort(asc("title")).show(false)// inner join with movew _rating dataframe , selecting year and title showing the output sorted ascending order with title
```

```
(3) Spark Jobs
df2join:  org.apache.spark.sql.DataFrame = [join_title: string, count: long]
+-------------------------+-----+
|title                    |year|
+-------------------------+-----+
|A Nightmare on Elm Street|2010|
|A Nightmare on Elm Street|1984|
|Casino Royale            |2006|
|Casino Royale            |1967|
|Conan the Barbarian      |1982|
|Conan the Barbarian      |2011|
|Death at a Funeral       |2010|
|Death at a Funeral       |2007|
|Dracula                  |1992|
|Dracula                  |1979|
|Footloose                |1984|
|Footloose                |2011|
|Fright Night             |2011|
|Fright Night             |1985|
|Hairspray                |2007|
|Hairspray                |1988|
|Halloween                |1978|
|Halloween                |2007|
Command took 4.65 seconds -- by sc26s@login.missouristate.edu at 10/19/2023, 10:14:51 PM on HW
```

Task 8

```scala
//task8
//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
val df1join = df1.withColumnRenamed("title", "join_title").withColumnRenamed("year", "join_year").select("join_title","actor", "join_year") //renaming the column title,year of the movie_rating dataframe for join
df2.join(df1join,'title===`join_title`) // joining the dataframes based on similar title
.where(col("actor").equalTo("Branson, Richard"))// filtering actor equal "Branson, Richard"
.where(df1join.col("join_year")===df2.col("year"))// filtering the remake actor did not worked
.select("title","year","rating").show(false)// selecting title, year, rating as output
```

▸ (2) Spark Jobs

▸ df1join: org.apache.spark.sql.DataFrame = [join_title: string, actor: string ... 1 more field]

```
+----------------------------+----+------+
|title                       |year|rating|
+----------------------------+----+------+
|Casino Royale               |2006|0.2078|
|Around the World in 80 Days|2004|1.8631|
|Superman Returns            |2006|0.1889|
+----------------------------+----+------+

df1join: org.apache.spark.sql.DataFrame = [join_title: string, actor: string ... 1 more field]
```

Command took 3.98 seconds -- by sc26s@login.mtsouristate.edu at 10/20/2023, 12:55:04 PM on HW

Task 9

```scala
//task9
// Write DataFrame-based Spark code to find the highest-rated movie per year and include all the actors in that movie. The output should have only one movie per year, and it should contain four columns: year, movie title, rating, and a list of actor names. Sort the output by year.
val windowSpec = Window.partitionBy("year").orderBy(col("rating").desc)// creating window partition by year and descending order by rating to get the top rated movies
val rankedmovie=df2.withColumn("rank", rank().over(windowSpec)).where("rank === 1) // using rank function and filtering based on rank =1 to get the top movie of each year
rankedmovie.join(df1,Seq("title","year"),"left_outer")// left outer join two dataframes based on year and title
.drop(df1.col("year"))// dropping duplicate column year
.groupBy("year","title","rating")// grouping the dataframe based on "year","title","rating"
.agg(collect_list("actor").as("actor_list"))// aggreating the actors as a list and named the column as actor_list
.select("year","title","rating","actor_list").show(false)// selecting showing the "year","title","rating","actor_list" as output
```

▸ (3) Spark Jobs

▸ rankedmovie: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [rating: double, title: string ... 2 more fields]

```
+----+--------------------------------+------+--------------------------------+
|year|title                           |rating|actor_list                      |
+----+--------------------------------+------+--------------------------------+
|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|4.9607|[]                              |
|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|[]                             |
```

Command took 3.39 seconds -- by sc26s@login.mtsouristate.edu at 10/20/2023, 1:14:26 PM on HW

Task 10

```scala
//task10
//Write DataFrame-based Spark code to determine which pair of actors worked together most. Working together is defined as appearing in the same movie. The output should have three columns: actor 1, actor 2, and count. The output should be sorted by the count in descending order.
val windowSpec2 = Window.orderBy(col("actor").asc)// creating window ascending order by rating to get the top rated movies
val rankedmovie=df1.withColumn("rank", rank().over(windowSpec2)).withColumnRenamed("actor", "actor 1")// using rank function to rank all the actor and renaming the actor column as actor 1
val newrankedmovie= rankedmovie.withColumnRenamed("rank","rank_new").withColumnRenamed("title","new_title").withColumnRenamed("actor 1","actor 2").withColumnRenamed("year","year_new") // creating another dataframe with renaming rank as rank_new, title as new_title, actor 1 as actor 2 and year as year_new
val n=rankedmovie.join(newrankedmovie).where(rankedmovie("title")===newrankedmovie("new_title")).where(rankedmovie("rank")<newrankedmovie("rank_new"))//joining the two dataframe based on similar title and actor 1 rank greater than actor 2 rank to avoid similar actor.
.groupBy("actor 1","actor 2")//Group by the dataframe with actor 1 and actor 2
n.count().sort(desc("count")).show()// counting the movies based on actor pair and showing the output as descending order
```

▸ (3) Spark Jobs

▸ rankedmovie: org.apache.spark.sql.DataFrame = [actor 1: string, title: string ... 2 more fields]

▸ newrankedmovie: org.apache.spark.sql.DataFrame = [actor 2: string, new_title: string ... 2 more fields]

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

Command took 5.66 seconds -- by sc26s@login.mtsouristate.edu at 10/20/2023, 1:26:59 PM on HW