Komal Suryan : NUID 002747707

Big Data System Engineering with Scala
Spring 2023
Assignment No. 7 (Spark)

You are required to analyze a movie rating dataset. The data is stored in a CSV file (either use the one in the repository or download the latest from Kaggle). You need to read this file into spark and calculate the mean rating and standard deviation for all movies. There is no test case provided for you, so you need to write your own test cases to ensure that at least your program works well.

```
import org.apache.spark.sql.SparkSession

val spark = SparkSession
  .builder()
  .appName("AssignmentSubmission7")
  .getOrCreate()
```

```
Intitializing Scala interpreter ...
Spark Web UI available at http://10.0.0.18:4040
SparkContext available as 'sc' (version = 3.3.2, master = local[*], app id = local-1680198427513)
SparkSession available as 'spark'
23/03/30 13:47:10 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.
import org.apache.spark.sql.SparkSession
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@5f30e7a1
```

```
[ ] val sc = spark.sparkContext
```

```
sc: org.apache.spark.SparkContext = org.apache.spark.SparkContext@46e24902
```

```
val df = spark.read.option("header", "true").csv("movie_metadata.csv")
df.show()
```

```
23/03/30 13:48:52 WARN package: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.del
+-----+----------------+--------------------+--------+-------------------+------------------+--------------+-------------------+---------
|color|    director_name|num_critic_for_reviews|duration|director_facebook_likes|actor_3_facebook_likes|  actor_2_name|actor_1_facebook_likes|    gross
+-----+----------------+--------------------+--------+-------------------+------------------+--------------+-------------------+---------
|Color|    James Cameron|                 723|     178|                  0|               855|Joel David Moore|               1000|760505847
|Color|   Gore Verbinski|                 302|     169|                563|              1000|  Orlando Bloom|              40000|309404152
|Color|       Sam Mendes|                 602|     148|                  0|               161|   Rory Kinnear|              11000|200074175
|Color|Christopher Nolan|                 813|     164|              22000|             23000| Christian Bale|              27000|448130642
| null|      Doug Walker|                null|    null|                131|              null|    Rob Walker|                131|      nul
|Color|   Andrew Stanton|                 462|     132|                475|               530| Samantha Morton|                640| 73058679
|Color|        Sam Raimi|                 392|     156|                  0|              4000|  James Franco|              24000|336530303
|Color|     Nathan Greno|                 324|     100|                 15|               284|  Donna Murphy|                799|200807262
|Color|      Joss Whedon|                 635|     141|                  0|             19000|Robert Downey Jr.|              26000|458991599
|Color|      David Yates|                 375|     153|                282|             10000| Daniel Radcliffe|              25000|301956980
```

```
val df = spark.read.option("header", "true").csv("movie_metadata.csv")
df.show()
```

```
23/03/30 13:48:52 WARN package: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.del
+-----+----------------+--------------------+--------+-------------------+--------------------+------------------+--------------------+---------+
|color|    director_name|num_critic_for_reviews|duration|director_facebook_likes|actor_3_facebook_likes|        actor_2_name|actor_1_facebook_likes|    gross|
+-----+----------------+--------------------+--------+-------------------+--------------------+------------------+--------------------+---------+
|Color|    James Cameron|                 723|     178|                  0|                 855|    Joel David Moore|                1000|760505847|
|Color|   Gore Verbinski|                 302|     169|                563|                1000|       Orlando Bloom|               40000|309404152|
|Color|       Sam Mendes|                 602|     148|                  0|                 161|        Rory Kinnear|               11000|200074175|
|Color|Christopher Nolan|                 813|     164|              22000|               23000|      Christian Bale|               27000|448130642|
| null|      Doug Walker|                null|    null|                131|                null|         Rob Walker|                 131|      nul|
|Color|   Andrew Stanton|                 462|     132|                475|                 530|     Samantha Morton|                 640| 73058679|
|Color|        Sam Raimi|                 392|     156|                  0|                4000|        James Franco|               24000|336530303|
|Color|     Nathan Greno|                 324|     100|                 15|                 284|        Donna Murphy|                 799|200807262|
|Color|      Joss Whedon|                 635|     141|                  0|               19000|  Robert Downey Jr.|               26000|458991599|
|Color|      David Yates|                 375|     153|                282|               10000|    Daniel Radcliffe|               25000|301956980|
|Color|      Zack Snyder|                 673|     183|                  0|                2000|        Lauren Cohan|               15000|330249062|
|Color|     Bryan Singer|                 434|     169|                  0|                 903|       Marlon Brando|               18000|200069408|
|Color|     Marc Forster|                 403|     106|                395|                 393|     Mathieu Amalric|                 451|168368427|
|Color|   Gore Verbinski|                 313|     151|                563|                1000|       Orlando Bloom|               40000|423032628|
|Color|   Gore Verbinski|                 450|     150|                563|                1000|        Ruth Wilson|               40000| 89289916|
|Color|      Zack Snyder|                 733|     143|                  0|                 748|  Christopher Meloni|               15000|291021565|
|Color|   Andrew Adamson|                 258|     150|                 80|                 201|Pierfrancesco Favino|               22000|141614023|
|Color|      Joss Whedon|                 703|     173|                  0|               19000|   Robert Downey Jr.|               26000|623279547|
|Color|     Rob Marshall|                 448|     136|                252|                1000|         Sam Claflin|               40000|241063875|
|Color| Barry Sonnenfeld|                 451|     106|                188|                 718|   Michael Stuhlbarg|               10000|179020854|
+-----+----------------+--------------------+--------+-------------------+--------------------+------------------+--------------------+---------+
only showing top 20 rows

df: org.apache.spark.sql.DataFrame = [color: string, director_name: string ... 26 more fields]
```

```
import org.apache.spark.sql.functions.avg

val mean = df.agg(avg("imdb_score").alias("Mean")).collect()(0).getAs[Double]("Mean")
println(mean)
```

Unit Tests:

Test case 1: When the column name is valid and contains numerical values, the method should return the correct average of the column.

Test case 2: When the column name is valid but contains non-numerical values, the method should return null.

Test case 3: When the column name is valid and contains numerical values, the method should return the correct standard deviation of the column.

```
6.453200745804848
import org.apache.spark.sql.functions.avg
mean: Double = 6.453200745804848
```

```
[ ] import org.apache.spark.sql.functions.stddev

    val stdDev = df.agg(stddev("imdb_score").alias("Standard Deviation")).collect()(0).getAs[Double]("Standard Deviation")
    println(stdDev)
```

```
0.9988071293753289
import org.apache.spark.sql.functions.stddev
stdDev: Double = 0.9988071293753289
```

```
import org.apache.spark.sql.functions.{avg, stddev}

// Test case 1: When the column name is valid and contains numerical values, the method should return the correct average of the column.
val df1 = Seq((1, 5.5), (2, 4.5), (3, 6.5)).toDF("id", "value")
assert(df1.agg(avg("value")).head().getDouble(0) == 5.5)

// Test case 2: When the column name is valid but contains non-numerical values, the method should return null.
val df2 = Seq((1, "foo"), (2, "bar"), (3, "baz")).toDF("id", "value")
assert(df2.agg(avg("value")).head().isNullAt(0))

// Test case 3: When the column name is valid and contains numerical values, the method should return the correct standard deviation of the column.
val df3 = Seq((1, 5.5), (2, 4.5), (3, 6.5)).toDF("id", "value")
assert(df3.agg(stddev("value")).head().getDouble(0) == 1.0)
```

```
import org.apache.spark.sql.functions.{avg, stddev}
df1: org.apache.spark.sql.DataFrame = [id: int, value: double]
df2: org.apache.spark.sql.DataFrame = [id: int, value: string]
df3: org.apache.spark.sql.DataFrame = [id: int, value: double]
```