Sayeed Ahmed – SEC01 (NUID 002191535)

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

Data source: https://files.grouplens.org/datasets/movielens/ml-latest-small.zip

Github: https://github.com/sayeedahmed01/CSYE7200/tree/Spring2022/Spark-Assignments/Assignment-7

Implementation:

1) Setting schema:

```scala
// schema of the movies dataset
val moviesSchema = StructType(Array(
  StructField("movieId", IntegerType, true),
  StructField("title", StringType, true),
  StructField("genres", StringType, true)))

// schema of the ratings dataset
val ratingsSchema = StructType(Array(
  StructField("userId", IntegerType, true),
  StructField("movieId", IntegerType, true),
  StructField("rating", DoubleType, true),
  StructField("timestamp", LongType, true)))
```

```
moviesSchema: org.apache.spark.sql.types.StructType = StructType(StructField(movieId,IntegerType,true),StructField(title,StringType,true),StructField(genres,StringType,true))
ratingsSchema: org.apache.spark.sql.types.StructType = StructType(StructField(userId,IntegerType,true),StructField(movieId,IntegerType,true),StructField(rating,DoubleType,true),StructField(timestamp,LongType,true))
Command took 1.03 seconds -- by ahmed.say@northeastern.edu at 3/30/2023, 12:52:14 AM on My Cluster
```

2) Loading data into a DF:

```scala
// Read the movies dataset into a Spark DataFrame
val movies = spark.read.format("csv")
  .option("header", "true")
  .schema(moviesSchema)
  .load("dbfs:/FileStore/shared_uploads/ahmed.say@northeastern.edu/movies.csv")

// Read the ratings dataset into a Spark DataFrame
val ratings = spark.read.format("csv")
  .option("header", "true")
  .schema(ratingsSchema)
  .load("dbfs:/FileStore/shared_uploads/ahmed.say@northeastern.edu/ratings.csv")
```

```
▸ movies: org.apache.spark.sql.DataFrame = [movieId: integer, title: string ... 1 more field]
▸ ratings: org.apache.spark.sql.DataFrame = [userId: integer, movieId: integer ... 2 more fields]
movies: org.apache.spark.sql.DataFrame = [movieId: int, title: string ... 1 more field]
ratings: org.apache.spark.sql.DataFrame = [userId: int, movieId: int ... 2 more fields]
Command took 1.75 seconds -- by ahmed.say@northeastern.edu at 3/30/2023, 12:52:14 AM on My Cluster
```

3) Joining the 2 datasets (movies.csv, ratings.csv):

Cmd 4

```scala
// Join the movies and ratings datasets on movieId
val movieRatings = ratings.join(movies, Seq("movieId"), "left_outer")
```

```
▸ movieRatings: org.apache.spark.sql.DataFrame = [movieId: integer, userId: integer ... 4 more fields]
movieRatings: org.apache.spark.sql.DataFrame = [movieId: int, userId: int ... 4 more fields]
Command took 0.80 seconds -- by ahmed.say@northeastern.edu at 3/30/2023, 12:52:14 AM on My Cluster
```

4) Calculating mean and standard deviation for every movie:

```scala
// Calculate the mean and standard deviation of ratings for each movie
val movieStats = movieRatings.groupBy("movieId", "title")
  .agg(avg("rating").alias("mean_rating"), stddev("rating").alias("stddev_rating"))
  .orderBy("movieId")

// Show the results
movieStats.show()
```

5) Result:

movieStats: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [movieId: integer, title: string ... 2 more fields]

```
+-------+--------------------+------------------+------------------+
|movieId|               title|       mean_rating|     stddev_rating|
+-------+--------------------+------------------+------------------+
|      1|    Toy Story (1995)|3.9209302325581397|0.8348591407114045|
|      2|      Jumanji (1995)|3.4318181818181817|0.8817134921476453|
|      3|Grumpier Old Men ...|3.2596153846153846|1.0548226531330254|
|      4|Waiting to Exhale...| 2.357142857142857|0.8521681032463467|
|      5|Father of the Bri...|3.0714285714285716|0.9071475440448852|
|      6|         Heat (1995)| 3.946078431372549|0.8172244221533347|
|      7|      Sabrina (1995)| 3.185185185185185|0.9775609631923282|
|      8| Tom and Huck (1995)|             2.875|1.1259916264596033|
|      9| Sudden Death (1995)|             3.125|0.9746794344808964|
|     10|    GoldenEye (1995)| 3.496212121212121|0.8593806844280277|
|     11|American Presiden...|3.6714285714285713| 0.900425480286211|
|     12|Dracula: Dead and...|2.4210526315789473|1.2501461902817699|
|     13|        Balto (1995)|             3.125|0.6408699444616557|
|     14|        Nixon (1995)|3.8333333333333335|0.7071067811865478|
|     15|Cutthroat Island ...|               3.0|1.2076147288491197|
|     16|       Casino (1995)| 3.926829268292683|0.8858348619181693|
|     17|Sense and Sensibi...|3.7761194029850746|1.1457549518851469|
|     18|   Four Rooms (1995)|               3.7|0.9090191359227177|
```

Command took 3.84 seconds -- by ahmed.say@northeastern.edu at 3/30/2023, 12:52:14 AM on My Cluster

6) Tests:

```
//Test Suit
//Test case 1: Check if the Spark Session is running

// Create a SparTest case 1:k Session for testing
val sparkTest = SparkSession.builder()
  .appName("test")
  .master("local[*]")
  .getOrCreate()

// Verify that the Spark Session is running
assert(sparkTest != null)
assert(sparkTest.version.startsWith("3."))

//Test case 2: Check if the movies DataFrame is loaded correctly.
assert(movies.count() == 9742)

//Test case 3: Check if the ratings DataFrame is loaded correctly.
assert(ratings.count() == 100836)

//Test case 4: Check if the join operation is performed correctly.
assert(movieRatings.count() == 100836)

//Test case 5: Check if the mean and standard deviation of ratings are calculated correctly.

// Select a movie with known mean and standard deviation
val testMovieId = 1
val testMovieStats = movieStats.filter($"movieId" === testMovieId).select("mean_rating", "stddev_rating").head()

// Compare the calculated values with the expected values
assert(Math.abs(testMovieStats.getDouble(0) - 3.9209302325581397) < 0.0001)
assert(Math.abs(testMovieStats.getDouble(1) - 0.8348591407114045) < 0.0001)
```