Harshil Shah – SEC01 (NUID 002780887)

Big Data System Engineering with Scala
Spring 2023
Assignment - 7

- GitHub Repo URL - https://github.com/harshilshahneu/CSYE7200-Harshil-Shah

-Kaggle dataset -
https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset (ratings.csv)

- List of Tasks Implemented

ou 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.

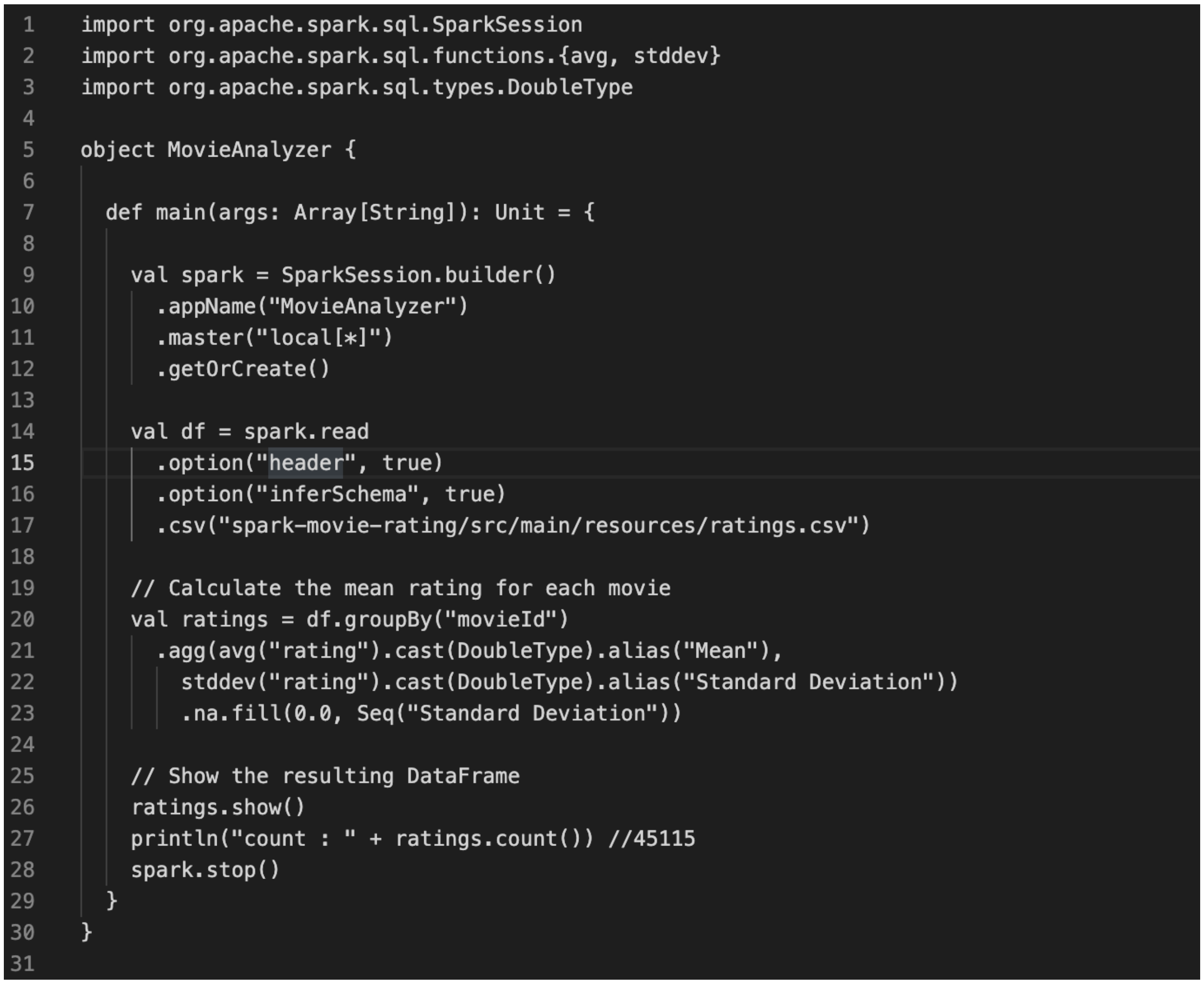
You can refer to *WordCount.scala* file for the basic structure. Notice that you need to use specific Spark version 3.2.1 for Scala 2.12.x support. If you can run it using a version with Scala 2.13, then go ahead.

Note also that there is a module in the CSYE7200 repository called *spark-csv*. If you use that, you will have to edit the *build.sbt* file. You can use that code of course to get started with (show your results if you do this). **However**, you also need to read the CSV file using the Spark utility (see https://spark.apache.org/docs/3.2.1/sql-data-sources-csv.html) and then create a method that accepts a DataFrame and returns the processed DataFrame.

You need to provide your code (in your own repository) together with the mean/std. dev. Don't forget to say where exactly you got the CSV file from.

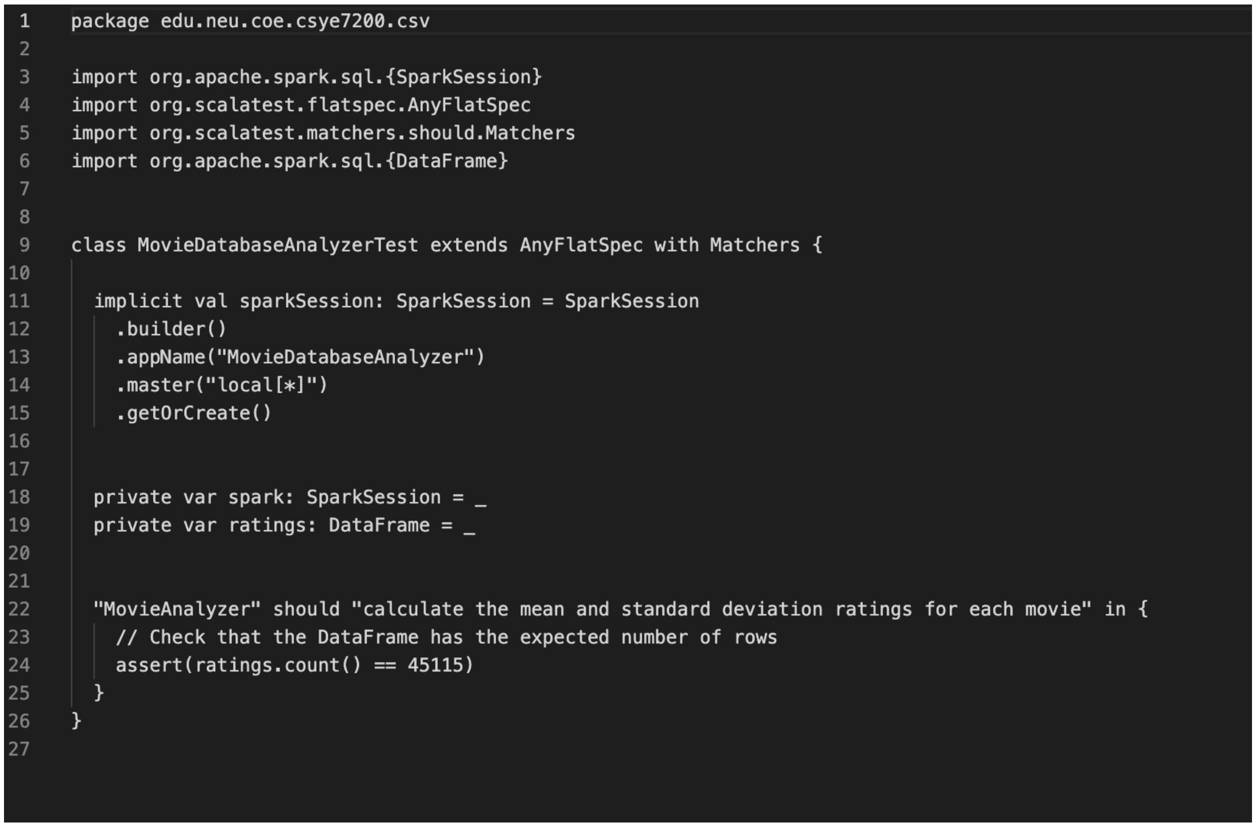
- Code

MovieAnalyzer.scala



```scala
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.{avg, stddev}
import org.apache.spark.sql.types.DoubleType

object MovieAnalyzer {

  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder()
      .appName("MovieAnalyzer")
      .master("local[*]")
      .getOrCreate()

    val df = spark.read
      .option("header", true)
      .option("inferSchema", true)
      .csv("spark-movie-rating/src/main/resources/ratings.csv")

    // Calculate the mean rating for each movie
    val ratings = df.groupBy("movieId")
      .agg(avg("rating").cast(DoubleType).alias("Mean"),
        stddev("rating").cast(DoubleType).alias("Standard Deviation"))
        .na.fill(0.0, Seq("Standard Deviation"))

    // Show the resulting DataFrame
    ratings.show()
    println("count : " + ratings.count()) //45115
    spark.stop()
  }
}

```

MovieDatabaseAnalyzerTest.scala



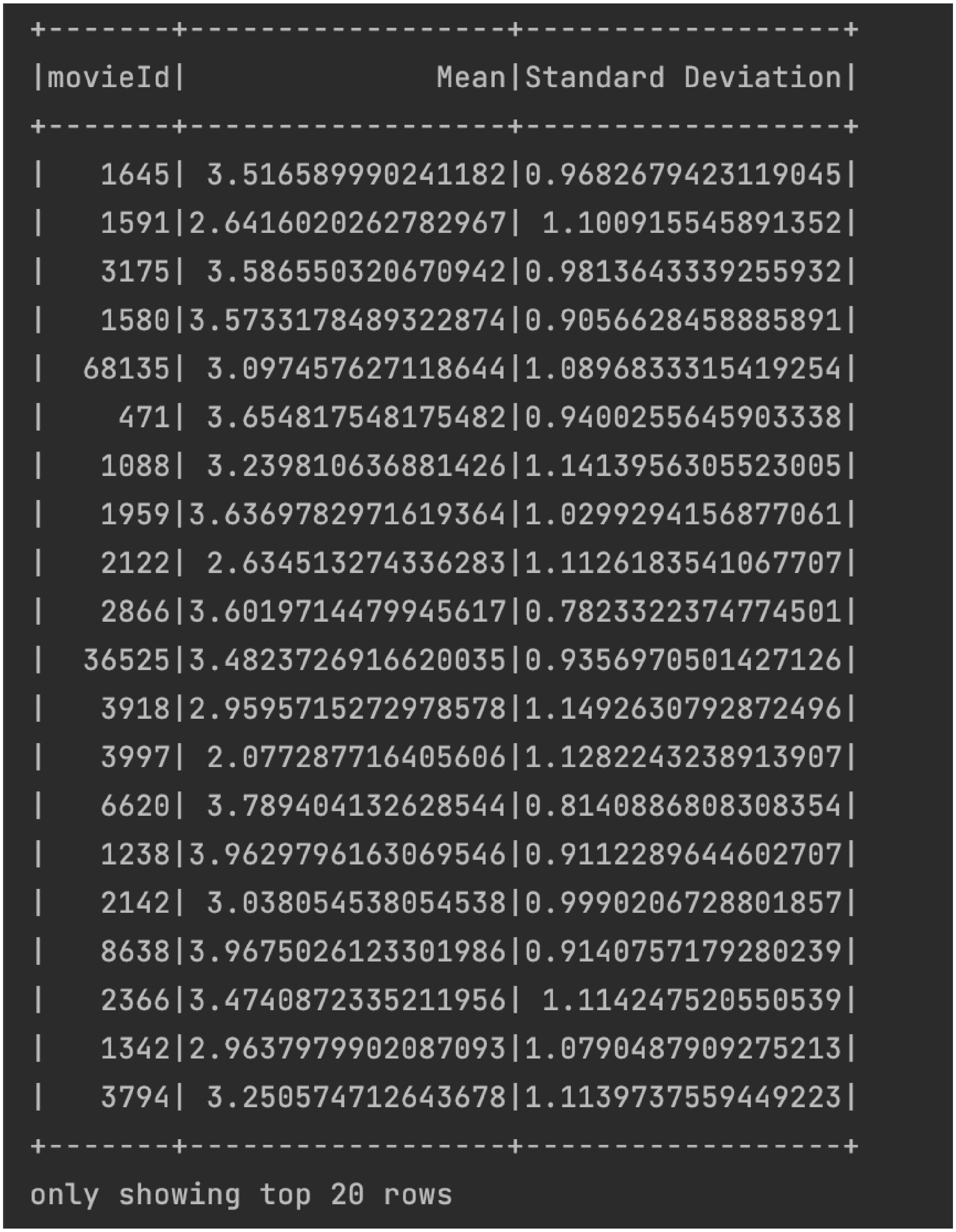
```scala
package edu.neu.coe.csye7200.csv

import org.apache.spark.sql.{SparkSession}
import org.scalatest.flatspec.AnyFlatSpec
import org.scalatest.matchers.should.Matchers
import org.apache.spark.sql.{DataFrame}


class MovieDatabaseAnalyzerTest extends AnyFlatSpec with Matchers {

  implicit val sparkSession: SparkSession = SparkSession
    .builder()
    .appName("MovieDatabaseAnalyzer")
    .master("local[*]")
    .getOrCreate()


  private var spark: SparkSession = _
  private var ratings: DataFrame = _


  "MovieAnalyzer" should "calculate the mean and standard deviation ratings for each movie" in {
    // Check that the DataFrame has the expected number of rows
    assert(ratings.count() == 45115)
  }
}

```

- Results

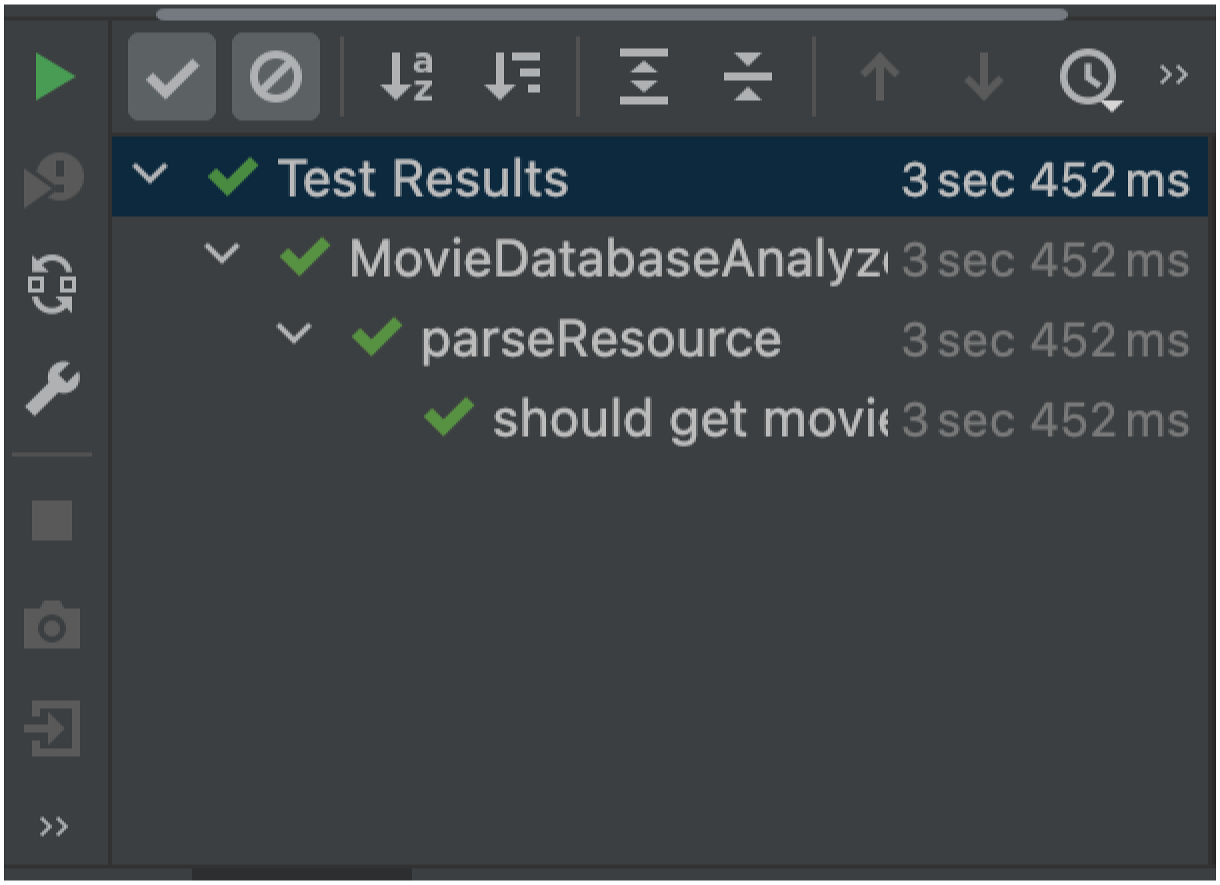
(only 20 rows from the DF)

```
+-------+------------------+------------------+
|movieId|              Mean|Standard Deviation|
+-------+------------------+------------------+
|   1645| 3.516589990241182|0.9682679423119045|
|   1591|2.6416020262782967| 1.100915545891352|
|   3175| 3.586550320670942|0.9813643339255932|
|   1580|3.5733178489322874|0.9056628458885891|
|  68135| 3.097457627118644|1.0896833315419254|
|    471| 3.654817548175482|0.9400255645903338|
|   1088| 3.239810636881426|1.1413956305523005|
|   1959|3.6369782971619364|1.0299294156877061|
|   2122| 2.634513274336283|1.1126183541067707|
|   2866|3.6019714479945617|0.7823322374774501|
|  36525|3.4823726916620035|0.9356970501427126|
|   3918|2.9595715272978578|1.1492630792872496|
|   3997| 2.077287716405606|1.1282243238913907|
|   6620| 3.789404132628544|0.8140886808308354|
|   1238|3.9629796163069546|0.9112289644602707|
|   2142| 3.038054538054538|0.9990206728801857|
|   8638|3.9675026123301986|0.9140757179280239|
|   2366|3.4740872335211956| 1.114247520550539|
|   1342|2.9637979902087093|1.0790487909275213|
|   3794| 3.250574712643678|1.1139737559449223|
+-------+------------------+------------------+
```

Execution Screenshots



```
+-------+------------------+------------------+
|movieId|              Mean|Standard Deviation|
+-------+------------------+------------------+
|   1645| 3.516589990241182|0.9682679423119045|
|   1591|2.6416020262782967| 1.100915545891352|
|   3175| 3.586550320670942|0.9813643339255932|
|   1580|3.5733178489322874|0.9056628458885891|
|  68135| 3.097457627118644|1.0896833315419254|
|    471| 3.654817548175482|0.9400255645903338|
|   1088| 3.239810636881426|1.1413956305523005|
|   1959|3.6369782971619364|1.0299294156877061|
|   2122| 2.634513274336283|1.1126183541067707|
|   2866|3.6019714479945617|0.7823322374774501|
|  36525|3.4823726916620035|0.9356970501427126|
|   3918|2.9595715272978578|1.1492630792872496|
|   3997| 2.077287716405606|1.1282243238913907|
|   6620| 3.789404132628544|0.8140886808308354|
|   1238|3.9629796163069546|0.9112289644602707|
|   2142| 3.038054538054538|0.9990206728801857|
|   8638|3.9675026123301986|0.9140757179280239|
|   2366|3.4740872335211956| 1.114247520550539|
|   1342|2.9637979902087093|1.0790487909275213|
|   3794| 3.250574712643678|1.1139737559449223|
+-------+------------------+------------------+
only showing top 20 rows
```



```
✓ Test Results                        3 sec 452 ms
  ✓ MovieDatabaseAnalyz               3 sec 452 ms
    ✓ parseResource                   3 sec 452 ms
      ✓ should get movi               3 sec 452 ms
```