Thejas Bharadwaj – SEC01 (NUID 002727189)

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

-List of Tasks Implemented

1) Created a Github repo - thejas98/Movie-Rating-Spark (github.com)

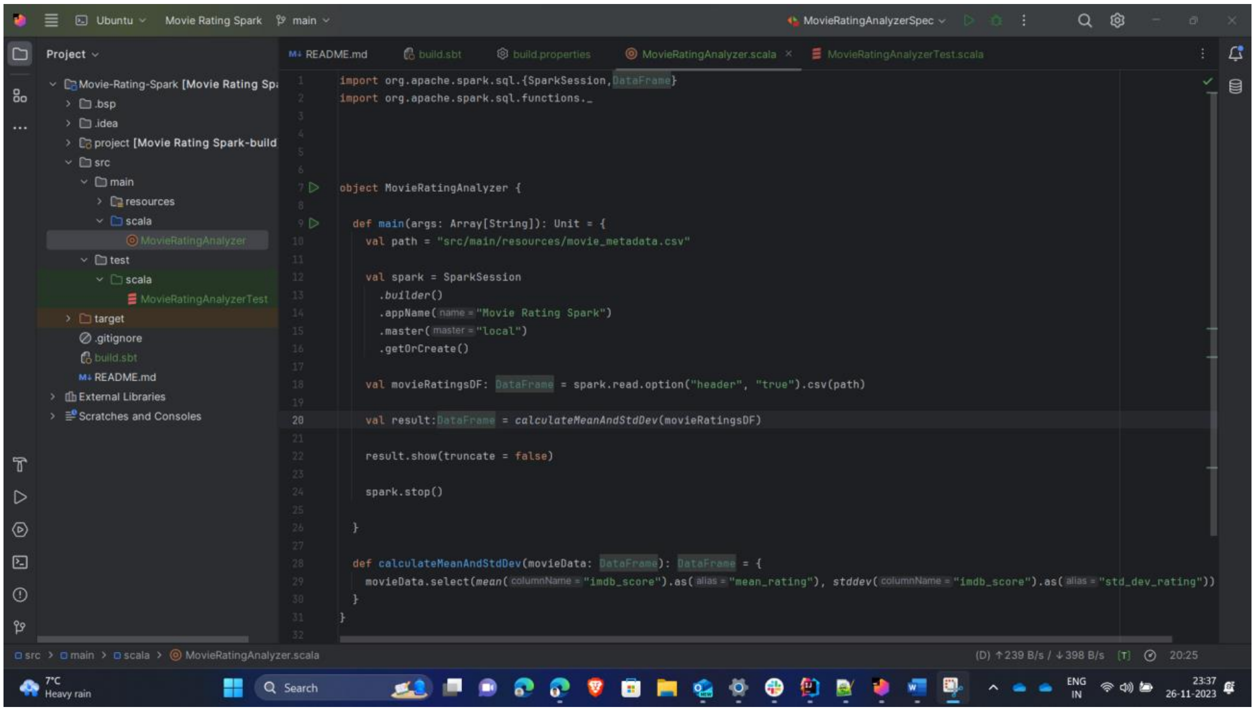
2) Created two code files – main and test

3) The main file has the code to ingest the csv and a function which calculates the mean and the standard deviation of the ‘imdb_score’ column.

4) The test file contains 2 test cases – One creates a sample df and tests if the function from main works correctly

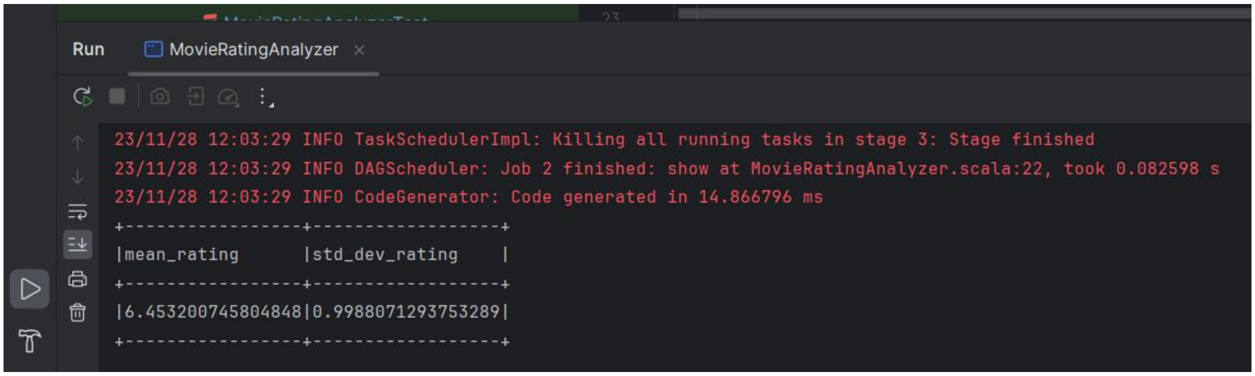
-Code

1) Main file



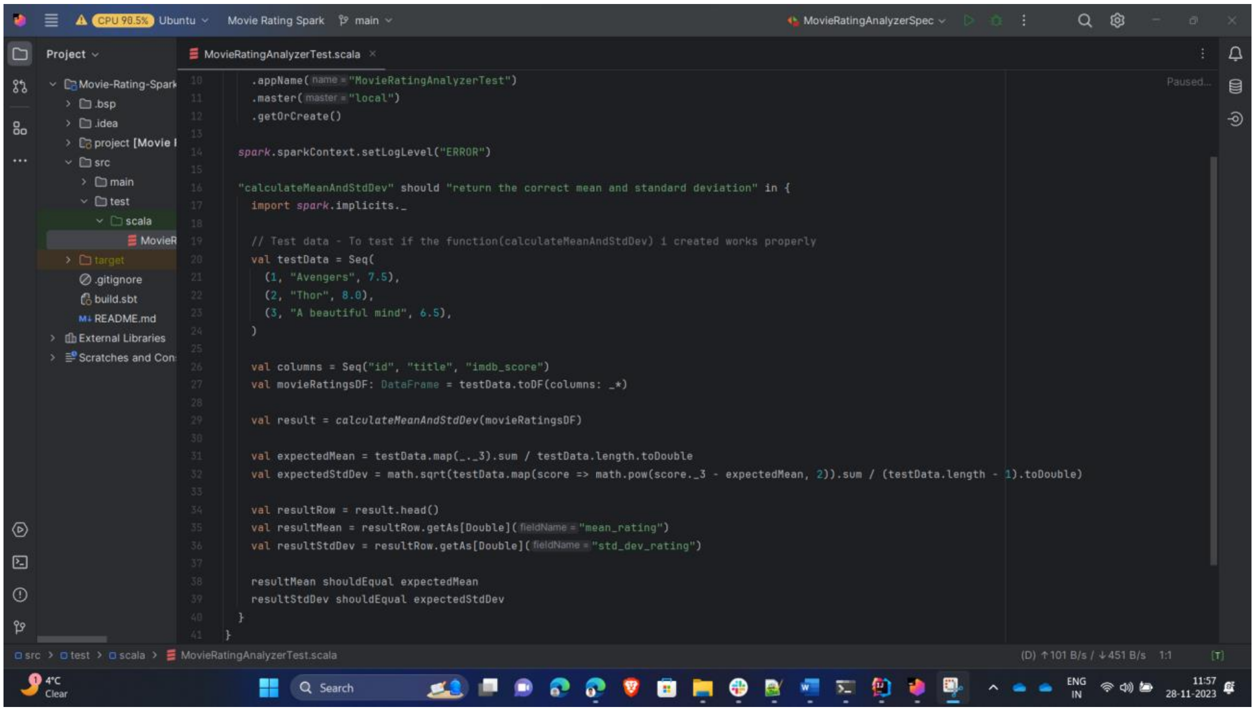
```scala
import org.apache.spark.sql.{SparkSession,DataFrame}
import org.apache.spark.sql.functions._




object MovieRatingAnalyzer {

  def main(args: Array[String]): Unit = {
    val path = "src/main/resources/movie_metadata.csv"

    val spark = SparkSession
      .builder()
      .appName( name = "Movie Rating Spark")
      .master( master = "local")
      .getOrCreate()

    val movieRatingsDF: DataFrame = spark.read.option("header", "true").csv(path)

    val result:DataFrame = calculateMeanAndStdDev(movieRatingsDF)

    result.show(truncate = false)

    spark.stop()

  }

  def calculateMeanAndStdDev(movieData: DataFrame): DataFrame = {
    movieData.select(mean( columnName = "imdb_score").as( alias = "mean_rating"), stddev( columnName = "imdb_score").as( alias = "std_dev_rating"))
  }
}
```

Output –



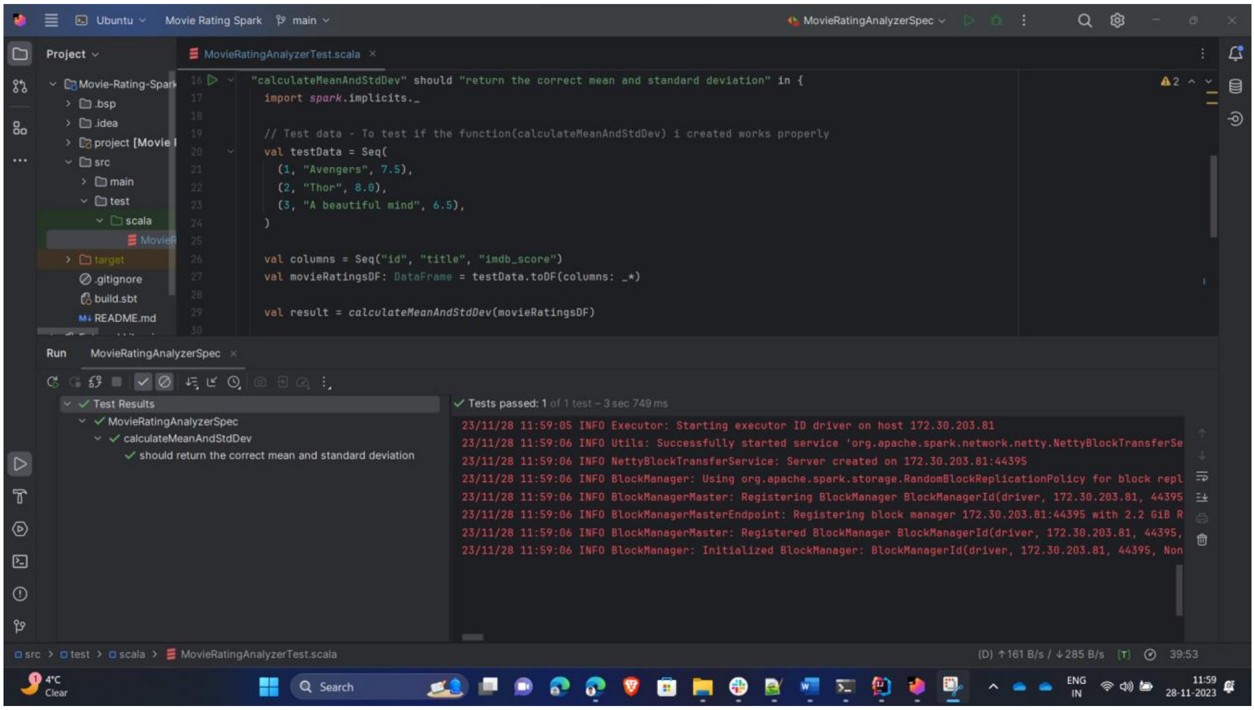
```
23/11/28 12:03:29 INFO TaskSchedulerImpl: Killing all running tasks in stage 3: Stage finished
23/11/28 12:03:29 INFO DAGScheduler: Job 2 finished: show at MovieRatingAnalyzer.scala:22, took 0.082598 s
23/11/28 12:03:29 INFO CodeGenerator: Code generated in 14.866796 ms
+-----------------+------------------+
|mean_rating      |std_dev_rating    |
+-----------------+------------------+
|6.453200745804848|0.9988071293753289|
+-----------------+------------------+
```

Test file



```scala
      .appName(name = "MovieRatingAnalyzerTest")
      .master(master = "local")
      .getOrCreate()

    spark.sparkContext.setLogLevel("ERROR")

    "calculateMeanAndStdDev" should "return the correct mean and standard deviation" in {
      import spark.implicits._

      // Test data - To test if the function(calculateMeanAndStdDev) i created works properly
      val testData = Seq(
        (1, "Avengers", 7.5),
        (2, "Thor", 8.0),
        (3, "A beautiful mind", 6.5),
      )

      val columns = Seq("id", "title", "imdb_score")
      val movieRatingsDF: DataFrame = testData.toDF(columns: _*)

      val result = calculateMeanAndStdDev(movieRatingsDF)

      val expectedMean = testData.map(_._3).sum / testData.length.toDouble
      val expectedStdDev = math.sqrt(testData.map(score => math.pow(score._3 - expectedMean, 2)).sum / (testData.length - 1).toDouble)

      val resultRow = result.head()
      val resultMean = resultRow.getAs[Double](fieldName = "mean_rating")
      val resultStdDev = resultRow.getAs[Double](fieldName = "std_dev_rating")

      resultMean shouldEqual expectedMean
      resultStdDev shouldEqual expectedStdDev
    }
  }
```

-Unit tests



```
"calculateMeanAndStdDev" should "return the correct mean and standard deviation" in {
  import spark.implicits._

  // Test data - To test if the function(calculateMeanAndStdDev) i created works properly
  val testData = Seq(
    (1, "Avengers", 7.5),
    (2, "Thor", 8.0),
    (3, "A beautiful mind", 6.5),
  )

  val columns = Seq("id", "title", "imdb_score")
  val movieRatingsDF: DataFrame = testData.toDF(columns: _*)

  val result = calculateMeanAndStdDev(movieRatingsDF)
```

- ✓ Test Results
  - ✓ MovieRatingAnalyzerSpec
    - ✓ calculateMeanAndStdDev
      - ✓ should return the correct mean and standard deviation

✓ Tests passed: 1 of 1 test – 3 sec 749 ms

```
23/11/28 11:59:05 INFO Executor: Starting executor ID driver on host 172.30.203.81
23/11/28 11:59:06 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferSe
23/11/28 11:59:06 INFO NettyBlockTransferService: Server created on 172.30.203.81:44395
23/11/28 11:59:06 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block repl
23/11/28 11:59:06 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 172.30.203.81, 44395
23/11/28 11:59:06 INFO BlockManagerMasterEndpoint: Registering block manager 172.30.203.81:44395 with 2.2 GiB R
23/11/28 11:59:06 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 172.30.203.81, 44395,
23/11/28 11:59:06 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, 172.30.203.81, 44395, Non
```