

Anirudha Joshi – SEC01 (NUID 002991365)

Big Data System Engineering with Scala

Spring 2023

Assignment No. 7 (Movie Analysis)



GitHub - https://github.com/anirudhajoshi2808/Anirudha_Joshi_CSYE7200

- Questions:

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.

- Code

```
localhost:8888/notebooks/Documents/NEU_Work/NEU_Spring2023_Work/Big_Data_Scala/Assignment7/Movie-Analysis.ipynb
jupyter Movie-Analysis Last Checkpoint: Yesterday at 13:20 (autosaved) Logout
File Edit View Insert Cell Kernel Widgets Help Not Trusted spylon-kernel
In [1]: import org.apache.spark.sql.SparkSession

val spark = SparkSession
    .builder()
    .appName("Assignment7")
    .getOrCreate()

Initializing 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-1680021394693)
SparkSession available as 'spark'

23/03/28 12:36:37 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.

Out[1]: import org.apache.spark.sql.SparkSession
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession#5f30e7a1

Define Spark Context

In [2]: val sc = spark.sparkContext

Out[2]: sc: org.apache.spark.SparkContext = org.apache.spark.SparkContext#46e24902

Ingest Data, Read from CSV
CSV file for movie downloaded from Kaggle https://www.kaggle.com/datasets/db55ac3df0098a0cf96dd542807f9253a16587f233e06baef372bccfd09942

In [6]: val df = spark.read.option("header", "true").csv("Top_10000_Movies.csv")
df.show()

+-----+-----+-----+-----+-----+-----+-----+-----+
| id|original_language|overview|revenue|original_title|popularity|release_date|vote_average|vote_count|
| genre|tagline|
+-----+-----+-----+-----+-----+-----+-----+-----+
|580489|en|Venom: Let There ...|5401.308|9/30/21|6.8|1736|['Sci
noe Fiction...|After finding a h...|424000000|97|11/3/21|7.1|622|['Acti
|524434|en|Eternals|3365.535|157|In the beginning...|
on', 'Adven...|The Eternals are ...|165000000|
```

```
localhost:8888/notebooks/Documents/NEU_Work/NEU_Spring2023_Work/Big_Data_Scala/Assignment7/Movie-Analysis.ipynb
jupyter Movie-Analysis Last Checkpoint: Yesterday at 13:20 (autosaved)
File Edit View Insert Cell Kernel Widgets Help Not Trusted spylon-kernel

Calculate Mean for vote_average column that signifies rating
Used select method to use avg function available in Spark function

In [10]: df.select(avg($"vote_average").alias("Mean")).show()
+-----+
|      Mean      |
+-----+
|6.29874999999994|
+-----+

Calculate Standard Deviation for vote_average column that signifies rating
Used select method to use stddev function available in Spark function

In [11]: df.select(stddev($"vote_average").alias("Standard Deviation")).show()
+-----+
|Standard Deviation|
+-----+
|1.4342597908380366|
+-----+
```

- Unit tests

```
localhost:8888/notebooks/Documents/NEU_Work/NEU_Spring2023_Work/Big_Data_Scala/Assignment7/Movie-Analysis.ipynb
jupyter Movie-Analysis Last Checkpoint: Yesterday at 13:20 (autosaved)
File Edit View Insert Cell Kernel Widgets Help Not Trusted spylon-kernel

Test cases to check for Mean and Standard Deviation functions

1. Check if avg and stddev functions work for Empty Dataframe
2. Check if avg and stddev functions work for Negative values in Dataframe
3. Check if avg and stddev functions work for Multiple Rows in Dataframe
4. Check if avg and stddev functions work for Decimal Values in Dataframe

In [31]: // Test case for an empty DataFrame
val df_empty = Seq.empty[(Int, Double)].toDF("id", "value")
val mean_empty = df_empty.select(avg("value")).first().get(0)
val stdev_empty = df_empty.select(stddev("value")).first().get(0)
assert(mean_empty == null && stdev_empty == null)

// Test case for a DataFrame with negative numbers
val df_negative = Seq((1, -3.0), (2, -2.0), (3, -1.0), (4, 0.0), (5, 1.0), (6, 2.0), (7, 3.0)).toDF("id", "value")
val mean_negative = df_negative.select(avg("value")).first().getAs[Double](0)
val stdev_negative = df_negative.select(stddev("value")).first().getAs[Double](0)
assert(mean_negative == 0.0 && stdev_negative == 2.160246899469287)

// Test case for a DataFrame with multiple rows
val df_multiple = Seq((1, 2.0), (2, 4.0), (3, 6.0), (4, 8.0), (5, 10.0)).toDF("id", "value")
val mean_multiple = df_multiple.select(avg("value")).first().getAs[Double](0)
val stdev_multiple = df_multiple.select(stddev("value")).first().getAs[Double](0)
assert(mean_multiple == 6.0 && stdev_multiple == 3.1622776601683795)

// Test case for a DataFrame with decimal numbers
val df_decimal = Seq((1, 1.2), (2, 2.5), (3, 3.7), (4, 4.9), (5, 5.1)).toDF("id", "value")
val mean_decimal = df_decimal.select(avg("value")).first().getAs[Double](0)
val stdev_decimal = df_decimal.select(stddev("value")).first().getAs[Double](0)
assert(mean_decimal == 3.4799999999999995 && stdev_decimal == 1.646815108019112)

Out[31]: df_empty: org.apache.spark.sql.DataFrame = [id: int, value: double]
mean_empty: Any = null
stdev_empty: Any = null
df_negative: org.apache.spark.sql.DataFrame = [id: int, value: double]
mean_negative: Double = 0.0
stdev_negative: Double = 2.160246899469287
df_multiple: org.apache.spark.sql.DataFrame = [id: int, value: double]
mean_multiple: Double = 6.0
stdev_multiple: Double = 3.1622776601683795
df_decimal: org.apache.spark.sql.DataFrame = [id: int, value: double]
mean_decimal: Double = 3.4799999999999995
stdev_decimal: Double = 1.646815108019112
```

