Use Case 2

Gabriel D Hofer

July 25, 2021

Introduction

The spark-shell was used to complete the following tasks.

Spark Actions & Transformations

Load the matches.csv and deliveries.csv into rdd and rdd2, respectively.

```
scala> val rddFromFile = sc.textFile("/home/gabriel/Data/matches.csv")

scala> val rdd = rddFromFile.map(f=>{ f.split(",") })

scala> val rddFromFile2 = sc.textFile("/home/gabriel/Data/deliveries.csv")

scala> val rdd2 = rddFromFile2.map(f => {f.split(",")})
```

1. Find the number of matches that took place in Hyderabad.

```
scala> rdd.
     | filter(a => a(2).equals("Hyderabad")).
     | count()
res30: Long = 64
```

2. Find the number matches that Sunrisers won in Hyderabad City.

```
scala> rdd.
     | filter(a => a(2).equals("Hyderabad") &&
     | a(10).equals("Sunrisers Hyderabad")).
     | count()
res31: Long = 30
```

3. Find the number of matches Sunrisers won in each season.

```
scala> rdd.
     | filter(a => a(10).equals("Sunrisers Hyderabad")).
     | groupBy(a => a(1)).
     | map{ case(k,v) => (k, v.size) }.
     | collect().
     | sortBy(_._1).
     | foreach(println)
(2013,10)
(2014,6)
(2015,7)
(2016,11)
(2017,8)
(2018,10)
(2019,6)
```

4. In which season did Yuvraj Singh has been awarded maximum Player of the match title.

```
scala> rdd.
     | filter(a => a(13).equals("Yuvraj Singh")).
     | foreach( a => println(a(1)) )
2017
2014
2009
2009
2011
```

5. Which venue hosted maximum matches.

```
scala> rdd.
     | groupBy(a => a(14)).
     | map{ case (k,v) => (k, v.size) }.
     | sortBy(-1 * _._2).
     | take(1).
     | foreach(println)
(Eden Gardens,77)
```

6. Find the average win by runs in Eden Gardens in Season 2017.

```
scala> rdd.
     | filter(a => a(14).equals("Eden Gardens")).
     | filter(a => a(1).equals("2017")).
     | map(_(11).toInt).
     | mean()
res0: Double = 15.428571428571429
```

7. Find out the percentage of each dismissal over all seasons.

```
// Select match id and player_dismissed columns from the RDD
scala> val id_dismissed = rdd2.map( f => { (f(0), if(f.size < 19) "-" else f
    (18) ) } )

// Select match id and the season columns from the RDD
scala> val id_season = rdd.map( f => { (f(0), f(1)) })

// Join the two lists of tuples on the match id
scala> val joinedRdd = id_dismissed.join(id_season)

// Restructure the data - put the season as the first element in the row
scala> val restruct = joinedRdd.map{ case(a,(b,c)) => (c,b,a) }.collect()

// We will use a two mutable maps: dismiss and totals
scala> import scala.collection.mutable.Map

// This maps season year to number of players dismissed in that season
scala> var dismiss= Map[String, Int]()

// This maps season year to number of deliveries
scala> var totals = Map[String, Int]()

// Count number of players dismissed per season and
// also count the total number of deliveries per season
scala> for((a,b,c) <- restruct){
     |  if(!dismiss.contains(a)){ dismiss(a) = 0 }
     |  if(b != "-"){ dismiss(a) += 1 }
     |  if(!totals.contains(a)){ totals(a) = 0 }
     |  totals(a) += 1
     | }

// print the results
scala> for((k,v) <- totals){
     |   println("Season:  "+k+"\tDismissal Rate:  "+dismiss(k).toFloat /
    totals(k).toFloat)
     | }
```

Output:

```
Season:  2016 Dismissal Rate:  0.047247447
Season:  2010 Dismissal Rate:  0.050006896
Season:  2019 Dismissal Rate:  0.04725014
Season:  2013 Dismissal Rate:  0.050173298
Season:  2009 Dismissal Rate:  0.0513009
Season:  2015 Dismissal Rate:  0.050615296
Season:  2018 Dismissal Rate:  0.050314907
Season:  2012 Dismissal Rate:  0.048291776
Season:  2011 Dismissal Rate:  0.047786985
Season:  2014 Dismissal Rate:  0.04713287
Season:  2008 Dismissal Rate:  0.05115279
Season:  2017 Dismissal Rate:  0.051291298
```

8. Which stadium is best suited to bat first.

```
scala> rdd.
     | map(x => (x(14), if(x(11) != "win_by_runs") x(11).toInt else 0)).
     | reduceByKey(_ + _).
     | sortBy(-1 * _._2).
     | take(1).
     | foreach(println)
(M Chinnaswamy Stadium,1310)
```

Checking our answers with the DataFrame API

This section completes the tasks from the previous section again, except this time using the DataFrame API. The purpose of this is to double-check the answers from the previous section.

Load the matches.csv and deliveries.csv into dataframes df and df2, respectively.

```
scala> val df = spark.read.option("header",true).csv("/home/gabriel/Downloads
    /matches.csv")

scala> val df2 = spark.read.option("header",true).csv("/home/gabriel/
    Downloads/deliveries.csv")
```

1. Find the number of matches that took place in Hyderabad.

```
scala> df.
     | groupBy("city").
     | count().
     | filter(col("city") === "Hyderabad").
     | show()
+---------+-----+
|     city|count|
+---------+-----+
|Hyderabad|   64|
+---------+-----+
```

2. Find the number matches that Sunrisers won in Hyderabad City.

```
scala> df.
     | where(col("winner") === "Sunrisers Hyderabad").
     | where(col("city") === "Hyderabad").
     | groupBy("winner").
     | count().
     | show()
+-------------------+-----+
|             winner|count|
+-------------------+-----+
|Sunrisers Hyderabad|   30|
+-------------------+-----+
```

3. Find the number of matches Sunrisers won in each season.

```
scala> df.
     | where(col("winner") === "Sunrisers Hyderabad").
     | groupBy("season").
     | count().
     | show()
+------+-----+
|season|count|
+------+-----+
|  2016|   11|
|  2019|    6|
|  2017|    8|
|  2014|    6|
|  2013|   10|
|  2018|   10|
|  2015|    7|
+------+-----+
```

4. In which season did Yuvraj Singh has been awarded maximum Player of the match title.

```
scala> df.
     | select("season").
     | where(col("player_of_match") === "Yuvraj Singh").
     | show()
+------+
|season|
+------+
|  2017|
|  2009|
|  2009|
|  2011|
|  2014|
+------+
```

5. Which venue hosted maximum matches.

```
scala> df.
     | groupBy("venue").
     | count().
     | orderBy(desc("count")).
     | show(1)
+------------+-----+
|       venue|count|
+------------+-----+
|Eden Gardens|   77|
+------------+-----+
only showing top 1 row
```

6. Find the average win by runs in Eden Gardens in Season 2017.

```
scala> df.
     | where(col("venue") === "Eden Gardens").
     | where(col("season") === "2017").
     | agg(avg($"win_by_runs").as("average_win_by_runs")).
     | show()
+-------------------+
|average_win_by_runs|
+-------------------+
| 15.428571428571429|
+-------------------+
```

7. Find out the percentage of each dismissal over all seasons.

```
scala> df2.
     | join(df, df2("match_id") === df("id"), "left").
     | groupBy("season").
     | agg((sum(when($"player_dismissed".isNull,0).otherwise(1))/count("*")).
     | as("player_dismissed : fraction null")).
     | show()
+------+--------------------------------+
|season|player_dismissed : fraction null|
+------+--------------------------------+
|  2016|             0.04724744608399546|
|  2012|              0.0482917768897394|
|  2019|               0.047250139586823|
|  2017|            0.051291299956716205|
|  2014|            0.047132867132867136|
|  2013|             0.05017329592341971|
|  2009|            0.051300896663236804|
|  2018|            0.050314905528341496|
|  2011|             0.04778698642214777|
|  2008|             0.05115279116316999|
|  2015|             0.05061529446234984|
|  2010|             0.05000689750310388|
+------+--------------------------------+
```

8. Which stadium is best suited to bat first.

```
scala> df.
     | groupBy("venue").
     | agg(sum("win_by_runs").as("win_by_runs_by_venue")).
     | orderBy(desc("win_by_runs_by_venue")).
     | show(1)
+--------------------+--------------------+
|               venue|win_by_runs_by_venue|
+--------------------+--------------------+
|M Chinnaswamy Sta...|              1310.0|
+--------------------+--------------------+
only showing top 1 row
```