WebLog_Processing.scala

C: > Users > abhib > Downloads > WebLog_Processing.scala

```scala
import org.apache.log4j.{Level, Logger}
import org.apache.spark.sql.{Column, SparkSession}
import org.apache.spark.sql.functions.{regexp_extract,sum,col,to_date,udf,to_timestamp,desc,dayofyear,year}

val spark = SparkSession.builder().appName("WebLog").master("local[*]").getOrCreate()
val base_df = spark.read.text("/home/deptii/Web_Log/weblog.csv")
base_df.printSchema()

import spark.implicits._
//this will produce a dataframe with a single column called value
val base_df = spark.read.text("/home/deptii/Web_Log/weblog.csv")
base_df.printSchema()
//let's look at some of the data
base_df.show(3,false)

/*
    Parsing the log file
 */
val parsed_df = base_df.select(regexp_extract($"value","""^([^(\s|,)]+)""",1).alias("host"),
    regexp_extract($"value","""^.*\[(\d\d/\w{3}/\d{4}:\d{2}:\d{2}:\d{2})""",1).as("timestamp"),
    regexp_extract($"value","""^.*\w+\s+([^\s]+)\s+HTTP.*""",1).as("path"),
    regexp_extract($"value","""^.*,([^\s]+)$""",1).cast("int").alias("status"))
parsed_df.show(5,false)
parsed_df.printSchema()

/*
    Data cleaning
 */
 // check if the initial dataset contain any null values
 println("Number of bad row in the initial dataset : " + base_df.filter($"value".isNull).count())

 // lets see our parsed dataset
 val bad_rows_df = parsed_df.filter($"host".isNull || $"timestamp".isNull || $"path".isNull || $"status".isNull)
 println("Number of bad rows : " + bad_rows_df.count())
 // same result as the previous statement but with different syntax
 //val bad_rows_df = parsed_df.filter($"host".isNull.or($"timestamp".isNull).or($"path".isNull)
 // .or($"status".isNull)).count
```

```scala
// lets count number of null values in each column
// we create a function that convert null value to one and then we count the number of one value
def count_null(col_name: Column): Column = sum(col_name.isNull.cast("int")).alias(col_name.toString())
val t = parsed_df.columns.map(col_name => count_null(col(col_name)))
parsed_df.select(t: _*).show()

// So all the null values are in status column, let's check what does it contain
val bad_status_df = base_df.select(regexp_extract($"value","""([^\d]+)$""",1).as("bad_status")).filter($"bad_status".notEqual(""))
println("Number of bad rows : " + bad_status_df.count())
// So the bad content correspond to error result, in our case this is just polluting our logs and our results
bad_status_df.show(5)

/*
    Fix the rows with null status
 */

// we have two option, replace the null value by some other meaningful value, or delete the whole line
// we will go with the other option since those lines are with no value for us
// we will use the na subpackage on a dataframe
val cleaned_df = parsed_df.na.drop()

// let's check that we don't have any null value
println("The count of null value : " + cleaned_df.filter($"host".isNull || $"timestamp".isNull || $"path".isNull|| $"status".isNull).count())
// count before and after
println("Before : " + parsed_df.count() + " | After : " + cleaned_df.count())

/*
    Parsing the timestamp
 */
// let's try to cast the timestamp column to date
// surprised ! we got null value, that's because when spark is not able to convert a date value
// it just return null
cleaned_df.select(to_date($"timestamp")).show(2)

// Let's fix this by converting the timestamp column to the format spark knows
```

```
// Let's fix this by converting the timestamp column to the format spark knows
val month_map = Map("Jan" -> 1, "Feb" -> 2, "Mar" -> 3, "Apr" -> 4, "May" -> 5, "Jun" -> 6, "Jul" -> 7, "Aug" -> 8
  , "Sep" -> 9, "Oct" -> 10, "Nov" -> 11, "Dec" -> 12)
def parse_clf_time(s: String) ={
  "%3$s-%2$s-%1$s %4$s:%5$s:%6$s".format(s.substring(0,2),month_map(s.substring(3,6)),s.substring(7,11)
    ,s.substring(12,14),s.substring(15,17),s.substring(18))
}
val toTimestamp = udf[String, String](parse_clf_time(_))
val logs_df = cleaned_df.select($"*",to_timestamp(toTimestamp($"timestamp")).alias("time")).drop("timestamp")
logs_df.printSchema()
logs_df.show(2)
// We cache the dataset so the next action would be faster
logs_df.cache()

//       ====<  Analysis walk-trough  >====

/*
    status column statistics
 */
logs_df.describe("status").show()
/*
    HTTP status analysis
 */
logs_df.groupBy("status").count().sort("status").show()

/*
    Frequent Hosts
 */
logs_df.groupBy("host").count().filter($"count" > 10).show()
/*
    Visualizing Paths
 */
logs_df.groupBy("path").count().sort(desc("count")).show()
/*
    Top Paths
```

```
/*
    Top Paths
 */
logs_df.groupBy("path").count().sort(desc("count")).show(10)

//       ====< Analyzing Web Server Log File >====

/*
    Top Ten Error Paths
 */
logs_df.filter($"status" =!= 200).groupBy("path").count().sort(desc("count")).show(10)
/*
    Number of unique Hosts
 */
val unique_host_count = logs_df.select("host").distinct().count()
println("Unique hosts : %d".format(unique_host_count))
/*
    Number of Unique Daily Hosts :
 */
val daily_hosts_df = logs_df.withColumn("day",dayofyear($"time")).withColumn("year",year($"time")).select("host","day","year").distinct().groupBy("day","year").count().
daily_hosts_df.show(5)
/*
    Average Number of Daily Requests per Host
 */
val total_req_per_day_df = logs_df.withColumn("day", dayofyear($"time")).withColumn("year", year($"time")).groupBy("day", "year").count()
val avg_daily_request_per_host_df = total_req_per_day_df.join(daily_hosts_df,total_req_per_day_df("day") === daily_hosts_df("day")&& total_req_per_day_df("year") === da
avg_daily_request_per_host_df.show(5)

//      ====< Exploring 404 status codes >====
/*
   Let's drill down and explore the error 404 status records, We've all seen those "404 Not Found" web pages.
   404 errors are returned when the server cannot find the resource (page or object) the browser client requested.
 */

  //   Counting 404 Response Codes
val not_found_df = logs_df.where($"status" === 404).cache()
```

```scala
  //   Counting 404 Response Codes
val not_found_df = logs_df.where($"status" === 404).cache()
println("found %d 404 Urls".format(not_found_df.count()))

  //   Listing 404 Status Code Records
not_found_df.select("path").distinct().show(40,false)

  //   Listing The Top Twenty 404 Response Code Paths :
not_found_df.groupBy("path").count().sort("count").show(20,false)
not_found_df.groupBy("path").agg("host" -> "collect_list","status" -> "count").sort("count(status)").show(20)
not_found_df.groupBy("path").agg("host" -> "collect_set","status" -> "count").sort("count(status)").show(20)

  //   Listing the Top Twenty-five 404 Response Code Hosts
not_found_df.groupBy("host").count().sort(desc("count")).show(truncate = false)

  //   Listing 404 Errors per Day
val errors_by_date_pair_df = not_found_df.withColumn("day", dayofyear($"time")).withColumn("year", year($"time")).groupBy("day","year").count()
not_found_df.withColumn("day", dayofyear($"time")).withColumn("year", year($"time")).groupBy("day","year").count().sort($"year",$"day").show(10)



/* To run the program
scala> :load WebLog_Processing.scala
*/
```