# WebLog_Processing.scala

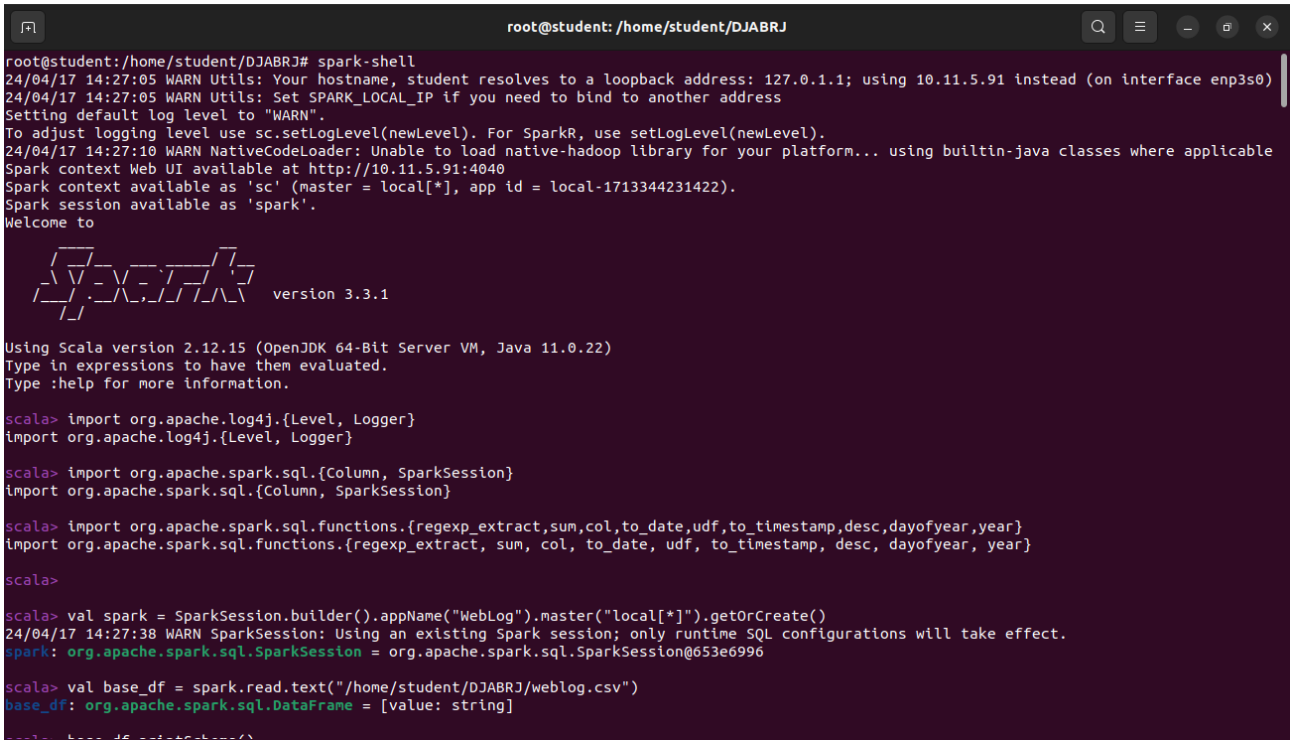
```
import org.apache.log4j.{Level, Logger}
import org.apache.spark.sql.{Column, SparkSession}
import org.apache.spark.sql.functions.
{regexp_extract,sum,col,to_date,udf,to_timestamp,desc,dayofyear,year}

val spark = SparkSession.builder().appName("WebLog").master("local[*]").getOrCreate()
val base_df = spark.read.text("/home/student/DJABRJ/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/student/DJABRJ/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

  // 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()
```

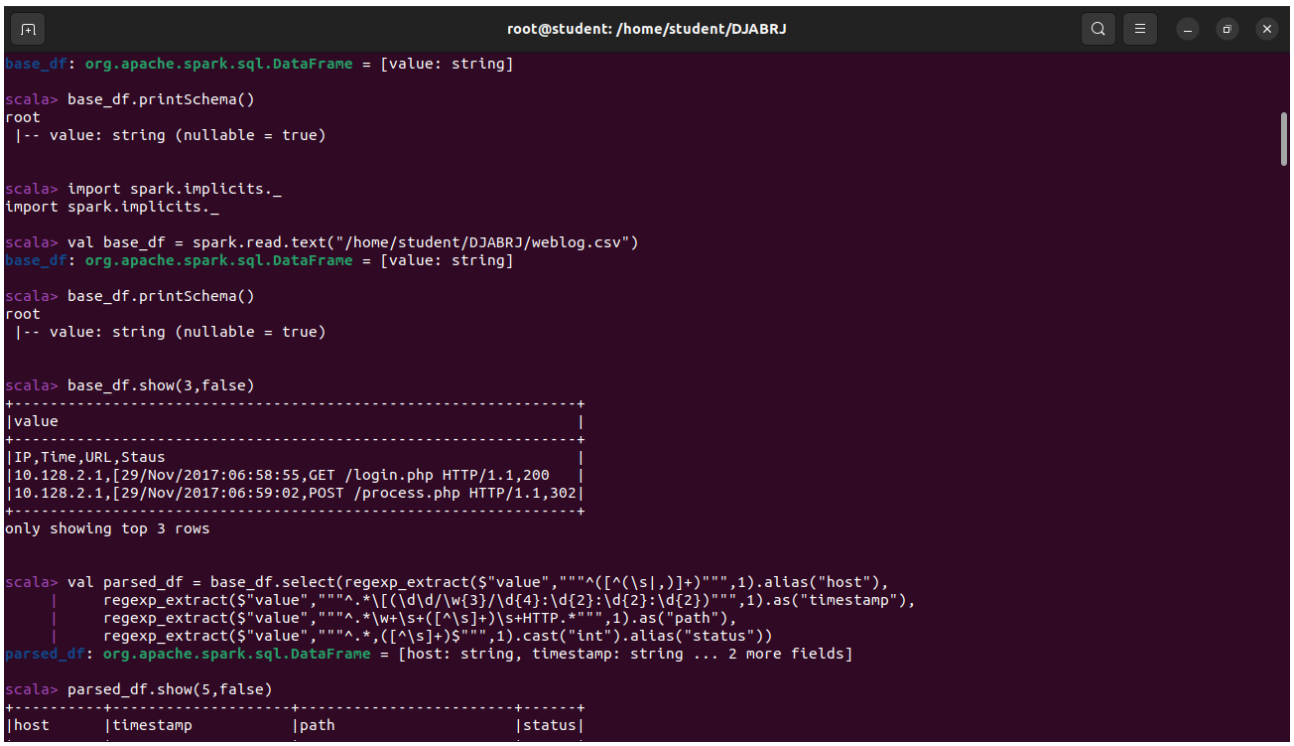
```
  // So all the null values are in status column, let's check what does it contain
  val bad_status_df = base_df.select(regexp_extract($"value","""([^\d]+)
$""",1).as("bad_status")).filter($"bad_status".notEqual(""))
  println("Number of bad rows : " + bad_status_df.count())
  // So the bad content 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
  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
 */
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($"time")).select("host","d
ay","year").distinct().groupBy("day","year").count().sort("year","day").cache()
daily_hosts_df.show(5)
/*
    Average Number of Daily Requests per Host
 */
val total_req_per_day_df = logs_df.withColumn("day", dayofyear($"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") ===
```

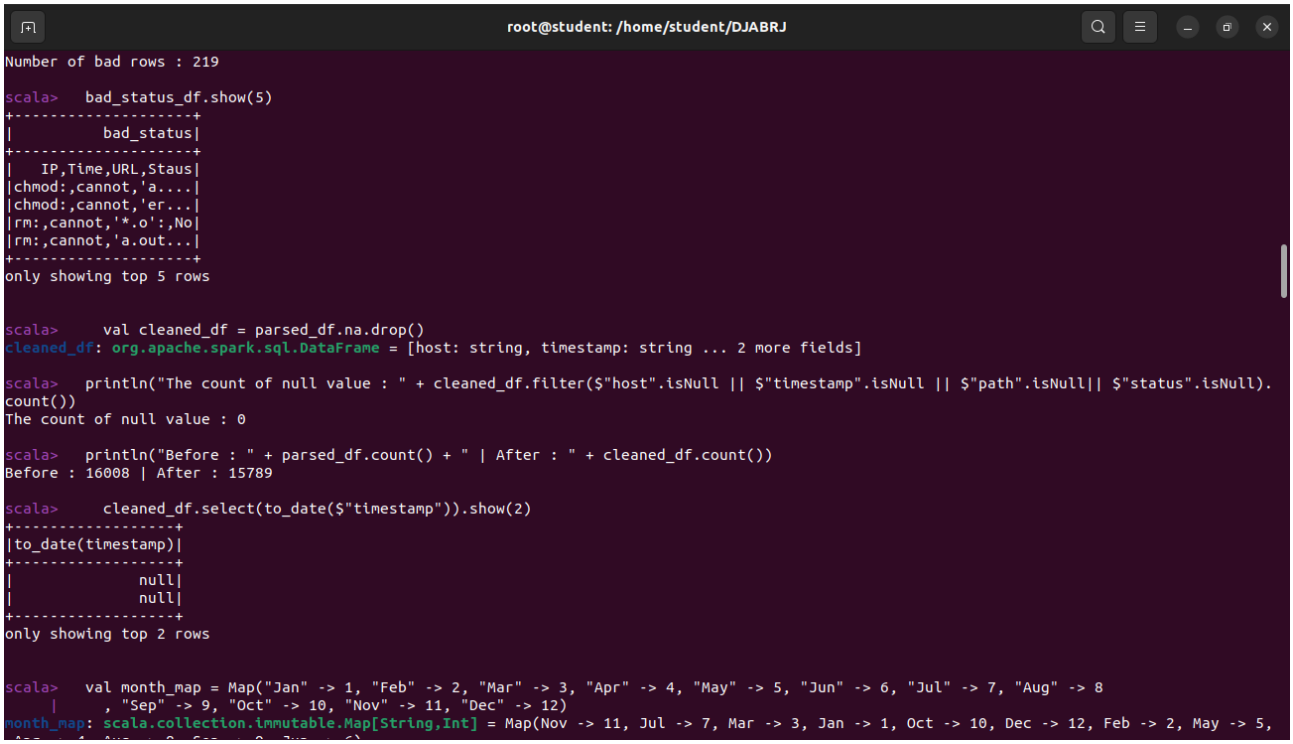
```
daily_hosts_df("year")).select(daily_hosts_df("day"),daily_hosts_df("year"),
(total_req_per_day_df("count")
/daily_hosts_df("count")).alias("avg_req_per_host_per_day")).cache()
  avg_daily_request_per_host_df.show(5)

  //      ====< 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()
  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
*/
```

# Output


root@student: /home/student/DJABRJ
root@student:/home/student/DJABRJ# spark-shell
24/04/17 14:27:05 WARN Utils: Your hostname, student resolves to a loopback address: 127.0.1.1; using 10.11.5.91 instead (on interface enp3s0)
24/04/17 14:27:05 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/04/17 14:27:10 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Spark context Web UI available at http://10.11.5.91:4040
Spark context available as 'sc' (master = local[*], app id = local-1713344231422).
Spark session available as 'spark'.
Welcome to
version 3.3.1
Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 11.0.22)
Type in expressions to have them evaluated.
Type :help for more information.
scala> import org.apache.log4j.{Level, Logger}
import org.apache.log4j.{Level, Logger}
scala> import org.apache.spark.sql.{Column, SparkSession}
import org.apache.spark.sql.{Column, SparkSession}
scala> import org.apache.spark.sql.functions.{regexp_extract,sum,col,to_date,udf,to_timestamp,desc,dayofyear,year}
import org.apache.spark.sql.functions.{regexp_extract, sum, col, to_date, udf, to_timestamp, desc, dayofyear, year}
scala>
scala> val spark = SparkSession.builder().appName("WebLog").master("local[*]").getOrCreate()
24/04/17 14:27:38 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@653e6996
scala> val base_df = spark.read.text("/home/student/DJABRJ/weblog.csv")
base_df: org.apache.spark.sql.DataFrame = [value: string]

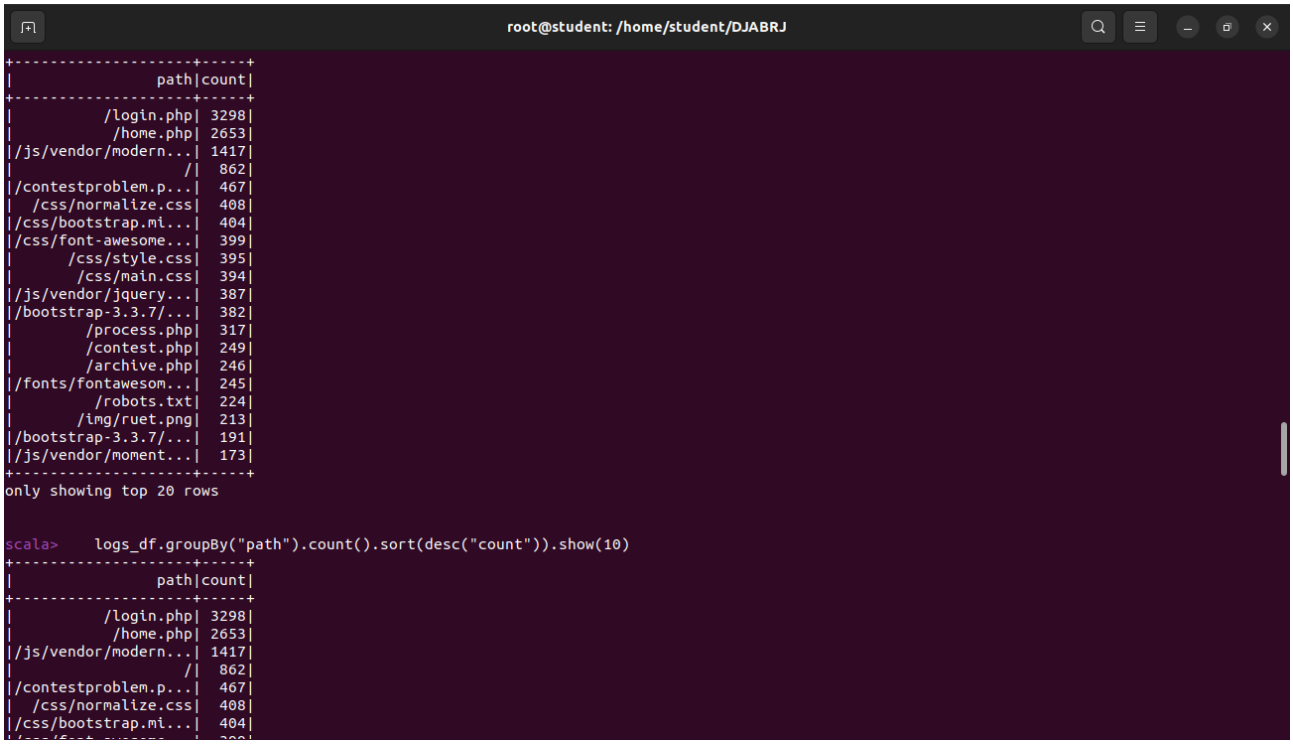

root@student: /home/student/DJABRJ
base_df: org.apache.spark.sql.DataFrame = [value: string]
scala> base_df.printSchema()
root
|-- value: string (nullable = true)
scala> import spark.implicits._
import spark.implicits._
scala> val base_df = spark.read.text("/home/student/DJABRJ/weblog.csv")
base_df: org.apache.spark.sql.DataFrame = [value: string]
scala> base_df.printSchema()
root
|-- value: string (nullable = true)
scala> base_df.show(3,false)
+-----------------------------------------------------------+
|value                                                      |
+-----------------------------------------------------------+
|IP,Time,URL,Staus                                          |
|10.128.2.1,[29/Nov/2017:06:58:55,GET /login.php HTTP/1.1,200   |
|10.128.2.1,[29/Nov/2017:06:59:02,POST /process.php HTTP/1.1,302|
+-----------------------------------------------------------+
only showing top 3 rows
scala> 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: org.apache.spark.sql.DataFrame = [host: string, timestamp: string ... 2 more fields]
scala> parsed_df.show(5,false)
+----------+--------------------+-----------------------+------+
|host      |timestamp           |path                   |status|


root@student: /home/student/DJABRJ
+----------+-------------------+----------------------+------+
only showing top 5 rows
scala> parsed_df.printSchema()
root
|-- host: string (nullable = true)
|-- timestamp: string (nullable = true)
|-- path: string (nullable = true)
|-- status: integer (nullable = true)
scala> println("Number of bad row in the initial dataset : " + base_df.filter($"value".isNull).count())
Number of bad row in the initial dataset : 0
scala> val bad_rows_df = parsed_df.filter($"host".isNull || $"timestamp".isNull || $"path".isNull || $"status".isNull)
bad_rows_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [host: string, timestamp: string ... 2 more fields]
scala> println("Number of bad rows : " + bad_rows_df.count())
Number of bad rows : 219
scala> def count_null(col_name: Column): Column = sum(col_name.isNull.cast("int")).alias(col_name.toString())
count_null: (col_name: org.apache.spark.sql.Column)org.apache.spark.sql.Column
scala> val t = parsed_df.columns.map(col_name => count_null(col(col_name)))
t: Array[org.apache.spark.sql.Column] = Array(sum(CAST((host IS NULL) AS INT)) AS host, sum(CAST((timestamp IS NULL) AS INT)) AS timestamp, su
m(CAST((path IS NULL) AS INT)) AS path, sum(CAST((status IS NULL) AS INT)) AS status)
scala> parsed_df.select(t: _*).show()
+----+---------+----+------+
|host|timestamp|path|status|
+----+---------+----+------+
| 0| 0| 0| 219|
+----+---------+----+------+
scala> val bad_status_df = base_df.select(regexp_extract($"value","""([^\d]+)$""",1).as("bad_status")).filter($"bad_status".notEqual(""))
bad_status_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [bad_status: string]

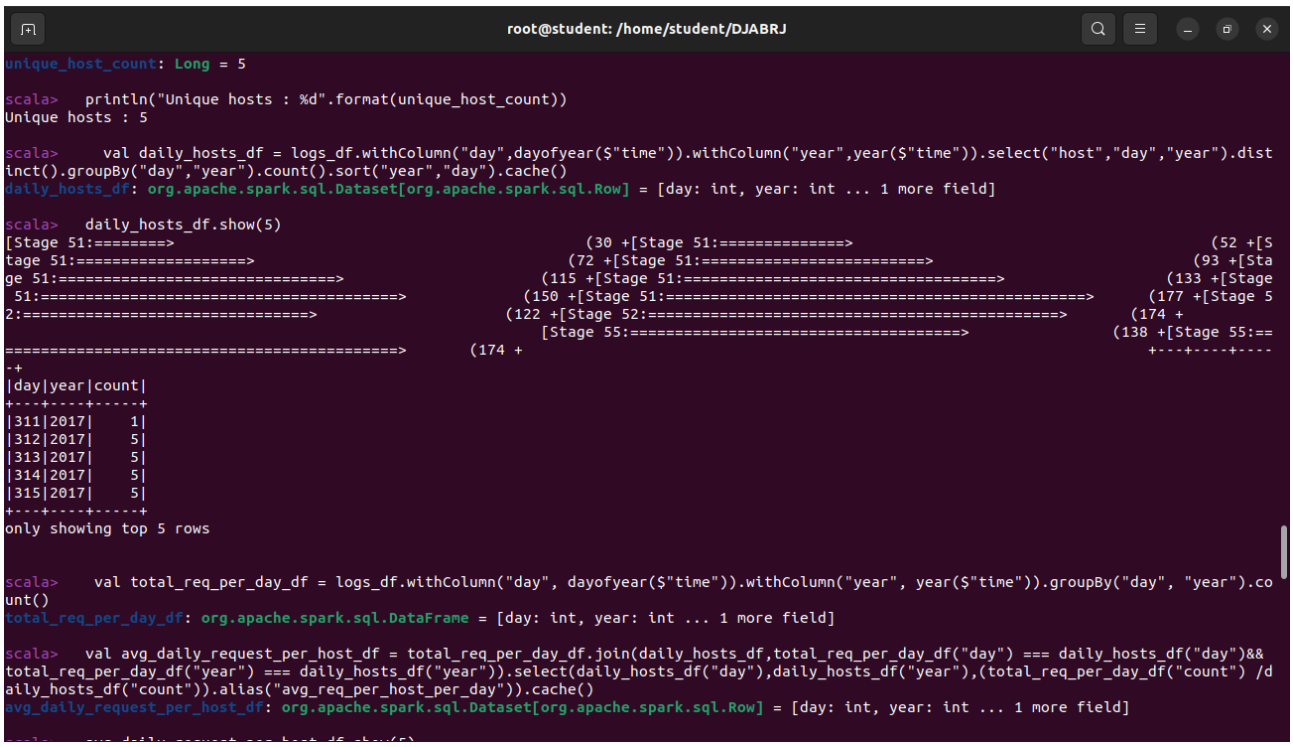

root@student: /home/student/DJABRJ
Number of bad rows : 219
scala> bad_status_df.show(5)
+--------------------+
| bad_status|
+--------------------+
| IP,Time,URL,Staus|
|chmod:,cannot,'a....|
|chmod:,cannot,'er...|
|rm:,cannot,'*.o':,No|
|rm:,cannot,'a.out...|
+--------------------+
only showing top 5 rows
scala> val cleaned_df = parsed_df.na.drop()
cleaned_df: org.apache.spark.sql.DataFrame = [host: string, timestamp: string ... 2 more fields]
scala> println("The count of null value : " + cleaned_df.filter($"host".isNull || $"timestamp".isNull || $"path".isNull|| $"status".isNull).
count())
The count of null value : 0
scala> println("Before : " + parsed_df.count() + " | After : " + cleaned_df.count())
Before : 16008 | After : 15789
scala> cleaned_df.select(to_date($"timestamp")).show(2)
+------------------+
|to_date(timestamp)|
+------------------+
| null|
| null|
+------------------+
only showing top 2 rows
scala> 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)
month_map: scala.collection.immutable.Map[String,Int] = Map(Nov -> 11, Jul -> 7, Mar -> 3, Jan -> 1, Oct -> 10, Dec -> 12, Feb -> 2, May -> 5,

```
root@student: /home/student/DJABRJ

     |      , "Sep" -> 9, "Oct" -> 10, "Nov" -> 11, "Dec" -> 12)
month_map: scala.collection.immutable.Map[String,Int] = Map(Nov -> 11, Jul -> 7, Mar -> 3, Jan -> 1, Oct -> 10, Dec -> 12, Feb -> 2, May -> 5,
 Apr -> 4, Aug -> 8, Sep -> 9, Jun -> 6)

scala>   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))
     |   }
parse_clf_time: (s: String)String

scala>   val toTimestamp = udf[String, String](parse_clf_time(_))
toTimestamp: org.apache.spark.sql.expressions.UserDefinedFunction = SparkUserDefinedFunction($Lambda$4420/0x000000084182f840@1ad08515,StringTy
pe,List(Some(class[value[0]: string])),Some(class[value[0]: string]),None,true,true)

scala>   val logs_df = cleaned_df.select($"*",to_timestamp(toTimestamp($"timestamp")).alias("time")).drop("timestamp")
logs_df: org.apache.spark.sql.DataFrame = [host: string, path: string ... 2 more fields]

scala>   logs_df.printSchema()
root
 |-- host: string (nullable = true)
 |-- path: string (nullable = true)
 |-- status: integer (nullable = true)
 |-- time: timestamp (nullable = true)


scala>   logs_df.show(2)
+----------+-----------+------+-------------------+
|      host|       path|status|               time|
+----------+-----------+------+-------------------+
|10.128.2.1|  /login.php|   200|2017-11-29 06:58:55|
|10.128.2.1|/process.php|   302|2017-11-29 06:59:02|
+----------+-----------+------+-------------------+
only showing top 2 rows


scala>   logs_df.cache()
res15: logs_df.type = [host: string, path: string ... 2 more fields]
```

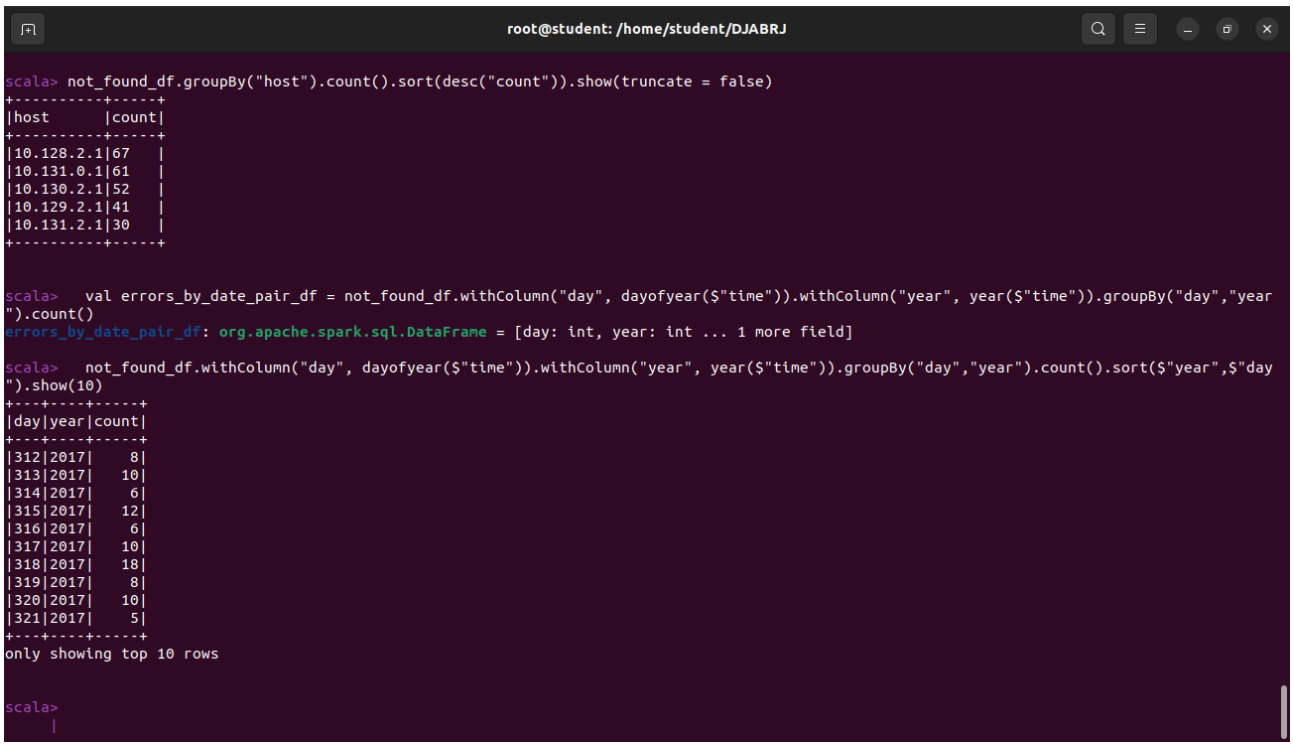
```
root@student: /home/student/DJABRJ

res15: logs_df.type = [host: string, path: string ... 2 more fields]

scala>    logs_df.describe("status").show()
[Stage 26:>                                                         (0                                                                  +-
------+------------------+
|summary|            status|
+-------+------------------+
|  count|             15789|
|   mean|230.19469250744189|
| stddev| 50.05853522906924|
|    min|               200|
|    max|               404|
+-------+------------------+


scala>   logs_df.groupBy("status").count().sort("status").show()
+------+-----+
|status|count|
+------+-----+
|   200|11330|
|   206|   52|
|   302| 3498|
|   304|  658|
|   404|  251|
+------+-----+


scala>  logs_df.groupBy("host").count().filter($"count" > 10).show()
+----------+-----+
|      host|count|
+----------+-----+
|10.131.2.1| 1626|
|10.128.2.1| 4257|
|10.130.2.1| 4056|
|10.131.0.1| 4198|
|10.129.2.1| 1652|
+----------+-----+
```


root@student: /home/student/DJABRJ
+--------------------+-----+
|                path|count|
+--------------------+-----+
|          /login.php| 3298|
|           /home.php| 2653|
|/js/vendor/modern...| 1417|
|                   /|  862|
|/contestproblem.p...|  467|
|  /css/normalize.css|  408|
|/css/bootstrap.mi...|  404|
|/css/font-awesome...|  399|
|      /css/style.css|  395|
|       /css/main.css|  394|
|/js/vendor/jquery...|  387|
|/bootstrap-3.3.7/...|  382|
|        /process.php|  317|
|        /contest.php|  249|
|        /archive.php|  246|
|/fonts/fontawesom...|  245|
|         /robots.txt|  224|
|       /img/ruet.png|  213|
|/bootstrap-3.3.7/...|  191|
|/js/vendor/moment...|  173|
+--------------------+-----+
only showing top 20 rows
scala>    logs_df.groupBy("path").count().sort(desc("count")).show(10)
+--------------------+-----+
|                path|count|
+--------------------+-----+
|          /login.php| 3298|
|           /home.php| 2653|
|/js/vendor/modern...| 1417|
|                   /|  862|
|/contestproblem.p...|  467|
|  /css/normalize.css|  408|
|/css/bootstrap.mi...|  404|


root@student: /home/student/DJABRJ
|  /css/normalize.css|  408|
|/css/bootstrap.mi...|  404|
|/css/font-awesome...|  399|
|      /css/style.css|  395|
|       /css/main.css|  394|
+--------------------+-----+
only showing top 10 rows
scala>  logs_df.filter($"status" =!= 200).groupBy("path").count().sort(desc("count")).show(10)
+--------------------+-----+
|                path|count|
+--------------------+-----+
|           /home.php| 2167|
|                   /|  741|
|        /process.php|  317|
|         /robots.txt|  224|
|         /action.php|   83|
|/contestproblem.p...|   74|
|/js/vendor/jquery...|   73|
|/css/bootstrap.mi...|   72|
|/js/vendor/modern...|   72|
|       /css/main.css|   68|
+--------------------+-----+
only showing top 10 rows
scala>   val unique_host_count = logs_df.select("host").distinct().count()
unique_host_count: Long = 5
scala>   println("Unique hosts : %d".format(unique_host_count))
Unique hosts : 5
scala>     val daily_hosts_df = logs_df.withColumn("day",dayofyear($"time")).withColumn("year",year($"time")).select("host","day","year").dist
inct().groupBy("day","year").count().sort("year","day").cache()
daily_hosts_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [day: int, year: int ... 1 more field]
scala>   daily_hosts_df.show(5)


root@student: /home/student/DJABRJ
unique_host_count: Long = 5
scala> println("Unique hosts : %d".format(unique_host_count))
Unique hosts : 5
scala> val daily_hosts_df = logs_df.withColumn("day",dayofyear($"time")).withColumn("year",year($"time")).select("host","day","year").distinct().groupBy("day","year").count().sort("year","day").cache()
daily_hosts_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [day: int, year: int ... 1 more field]
scala> daily_hosts_df.show(5)
|day|year|count|
|311|2017| 1|
|312|2017| 5|
|313|2017| 5|
|314|2017| 5|
|315|2017| 5|
only showing top 5 rows
scala> val total_req_per_day_df = logs_df.withColumn("day", dayofyear($"time")).withColumn("year", year($"time")).groupBy("day", "year").count()
total_req_per_day_df: org.apache.spark.sql.DataFrame = [day: int, year: int ... 1 more field]
scala> 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") === daily_hosts_df("year")).select(daily_hosts_df("day"),daily_hosts_df("year"),(total_req_per_day_df("count") /daily_hosts_df("count")).alias("avg_req_per_host_per_day")).cache()
avg_daily_request_per_host_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [day: int, year: int ... 1 more field]


root@student: /home/student/DJABRJ
| 60|2018| 10.333333333333334|
|350|2017| 51.666666666666664|
| 46|2018| 6.666666666666667|
only showing top 5 rows
scala> val not_found_df = logs_df.where($"status" === 404).cache()
not_found_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [host: string, path: string ... 2 more fields]
scala> println("found %d 404 Urls".format(not_found_df.count()))
found 251 404 Urls
scala> not_found_df.select("path").distinct().show(40,false)
|path |
|/css/bootstrap.min.css.map |
|/robots.txt |
|/djs/vendor/bootstrap-datetimepicker.js|
|/favicon.ico |
scala> not_found_df.groupBy("path").count().sort("count").show(20,false)
|path |count|
|/css/bootstrap.min.css.map |1 |
|/djs/vendor/bootstrap-datetimepicker.js|7 |
|/favicon.ico |19 |
|/robots.txt |224 |
scala> not_found_df.groupBy("path").agg("host" -> "collect_list","status" -> "count").sort("count(status)").show(20)
| path| collect_list(host)|count(status)|


root@student: /home/student/DJABRJ
scala> not_found_df.groupBy("host").count().sort(desc("count")).show(truncate = false)
+----------+-----+
|host      |count|
+----------+-----+
|10.128.2.1|67   |
|10.131.0.1|61   |
|10.130.2.1|52   |
|10.129.2.1|41   |
|10.131.2.1|30   |
+----------+-----+
scala>   val errors_by_date_pair_df = not_found_df.withColumn("day", dayofyear($"time")).withColumn("year", year($"time")).groupBy("day","year").count()
errors_by_date_pair_df: org.apache.spark.sql.DataFrame = [day: int, year: int ... 1 more field]
scala>   not_found_df.withColumn("day", dayofyear($"time")).withColumn("year", year($"time")).groupBy("day","year").count().sort($"year",$"day").show(10)
+---+----+-----+
|day|year|count|
+---+----+-----+
|312|2017|    8|
|313|2017|   10|
|314|2017|    6|
|315|2017|   12|
|316|2017|    6|
|317|2017|   10|
|318|2017|   18|
|319|2017|    8|
|320|2017|   10|
|321|2017|    5|
+---+----+-----+
only showing top 10 rows
scala>