**MOVIE RATINGS ANALYSIS ON APACHE SPARK**

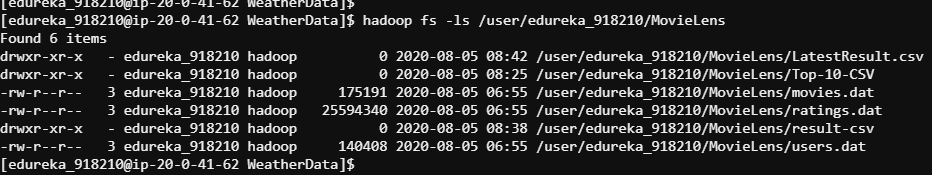
Please find the below detailed steps done/analysis on Movie Rating :

1. Downloaded the Data Files from the MovieLens Site.

Here’s the URL : <https://grouplens.org/datasets/movielens/>

1. Transferring the Data Files in aws – ec2 machine using FTP or AWS-EC2 Machine.
2. Putting the Data in the HDFS – Hadoop File System
3. Analysis the data on Spark running on HDFS ,
4. Creating and Loading of Data into Spark RDD
5. Conversion of Spark RDD into Data Frames
6. Usage of CreateorReplaceTempView and using spark.sql on the same.
7. Carrying the analysis of the structured data once converted to DF

hadoop fs -ls /user/edureka\_918210/MovieLens



**Prepare\_Ratings**

val ratingsRDD=spark.sparkContext.textFile("/user/edureka\_918210/MovieLens/ratings.dat")

import org.apache.spark.sql.Row;

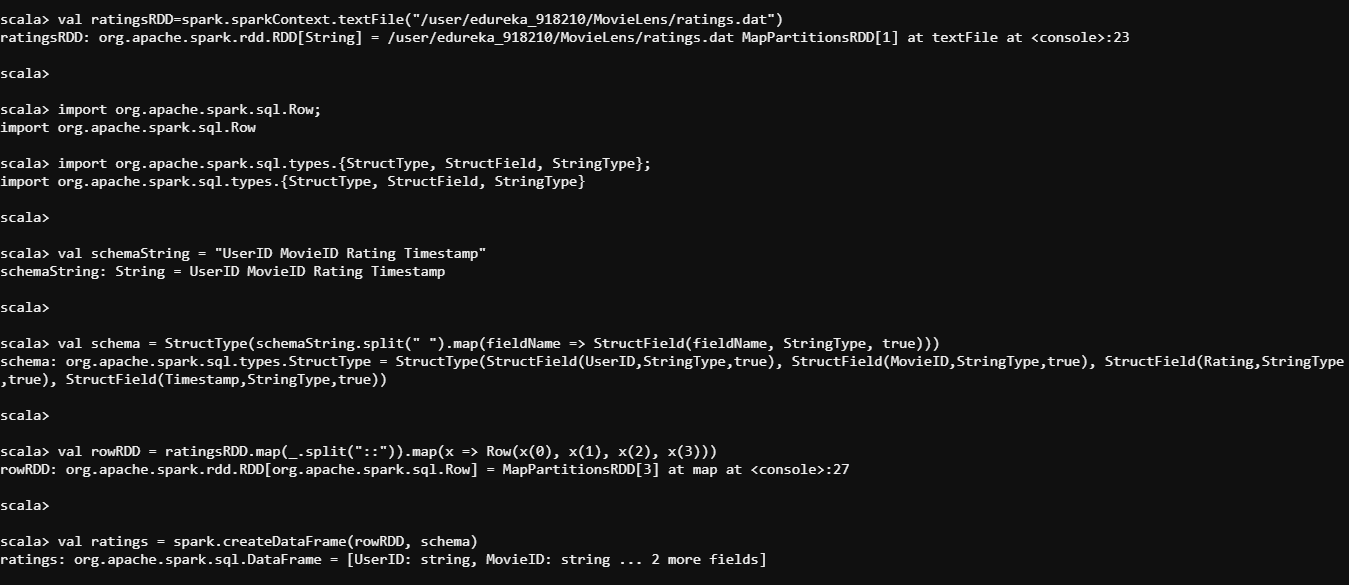
import org.apache.spark.sql.types.{StructType, StructField, StringType};

val schemaString = "UserID MovieID Rating Timestamp"

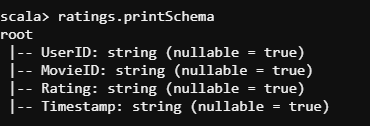
val schema = StructType(schemaString.split(" ").map(fieldName => StructField(fieldName, StringType, true)))

val rowRDD = ratingsRDD.map(\_.split("::")).map(x => Row(x(0), x(1), x(2), x(3)))

val ratings = spark.createDataFrame(rowRDD, schema)



ratings.printSchema



**Prepare\_Users**

val usersRDD=spark.sparkContext.textFile("/user/edureka\_918210/MovieLens/users.dat")

import org.apache.spark.sql.Row;

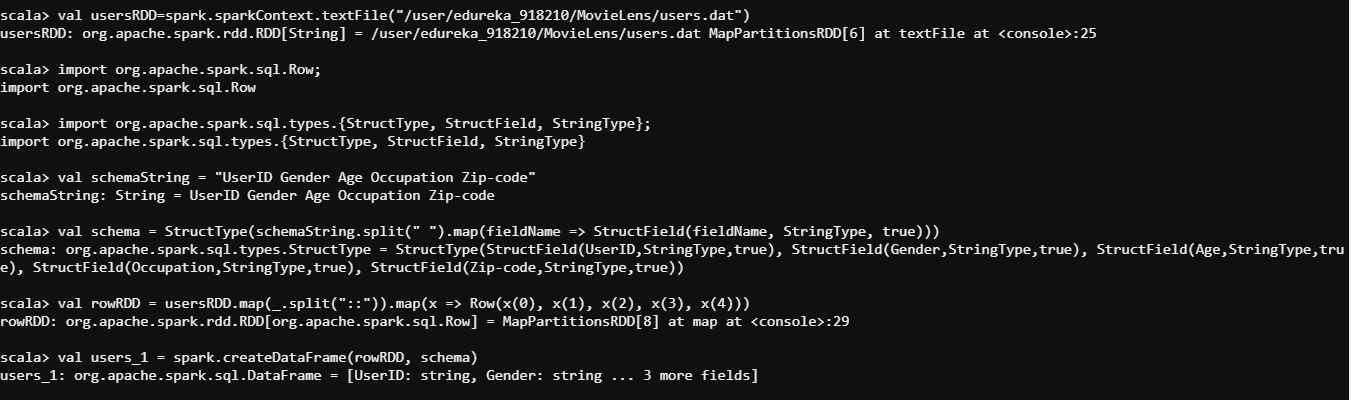
import org.apache.spark.sql.types.{StructType, StructField, StringType};

val schemaString = "UserID Gender Age Occupation Zip-code"

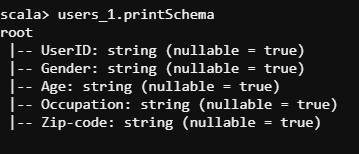
val schema = StructType(schemaString.split(" ").map(fieldName => StructField(fieldName, StringType, true)))

val rowRDD = usersRDD.map(\_.split("::")).map(x => Row(x(0), x(1), x(2), x(3), x(4)))

val users\_1 = spark.createDataFrame(rowRDD, schema)



users\_1.printSchema



**Prepare\_Ratings**

val ratingsRDD=spark.sparkContext.textFile("/user/edureka\_918210/MovieLens/ratings.dat")

import org.apache.spark.sql.Row;

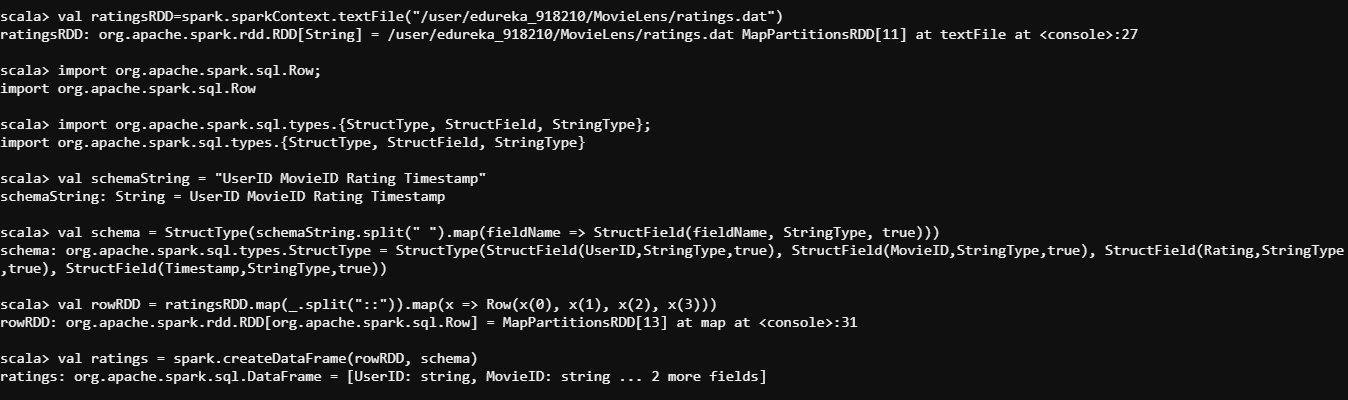
import org.apache.spark.sql.types.{StructType, StructField, StringType};

val schemaString = "UserID MovieID Rating Timestamp"

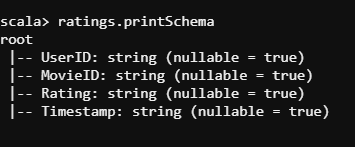
val schema = StructType(schemaString.split(" ").map(fieldName => StructField(fieldName, StringType, true)))

val rowRDD = ratingsRDD.map(\_.split("::")).map(x => Row(x(0), x(1), x(2), x(3)))

val ratings = spark.createDataFrame(rowRDD, schema)



ratings.printSchema

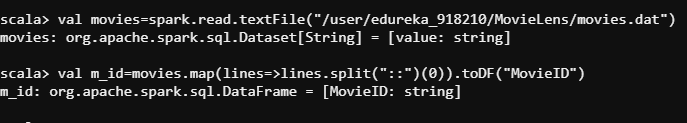


**Prepare\_Movies**

// Clean data into DataFrame

val movies=spark.read.textFile("/user/edureka\_918210/MovieLens/movies.dat")

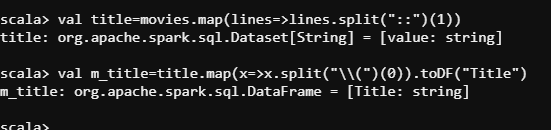
val m\_id=movies.map(lines=>lines.split("::")(0)).toDF("MovieID")



//Extract the title

val title=movies.map(lines=>lines.split("::")(1))

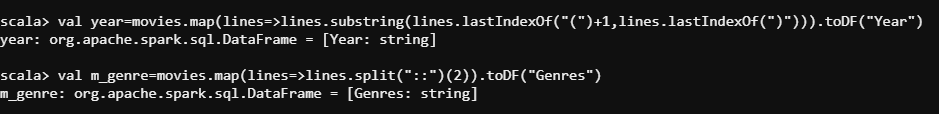
val m\_title=title.map(x=>x.split("\\(")(0)).toDF("Title")



//Extract the year

val year=movies.map(lines=>lines.substring(lines.lastIndexOf("(")+1,lines.lastIndexOf(")"))).toDF("Year")

val m\_genre=movies.map(lines=>lines.split("::")(2)).toDF("Genres")



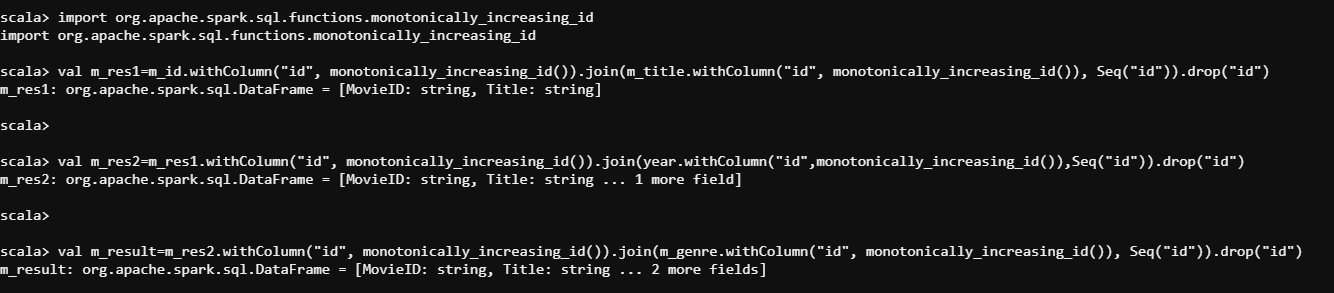
// For appending the dataframes, we need to import monotonically\_increasing\_id

import org.apache.spark.sql.functions.monotonically\_increasing\_id

val m\_res1=m\_id.withColumn("id", monotonically\_increasing\_id()).join(m\_title.withColumn("id", monotonically\_increasing\_id()), Seq("id")).drop("id")

val m\_res2=m\_res1.withColumn("id", monotonically\_increasing\_id()).join(year.withColumn("id",monotonically\_increasing\_id()),Seq("id")).drop("id")

val m\_result=m\_res2.withColumn("id", monotonically\_increasing\_id()).join(m\_genre.withColumn("id", monotonically\_increasing\_id()), Seq("id")).drop("id")



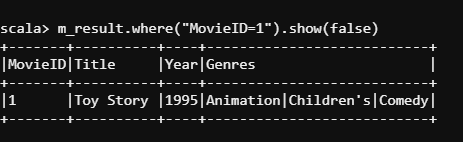
// This will give us the valid data with schema

m\_result.show



**SOME OF THE IMPORTANT EXAMPLES :**

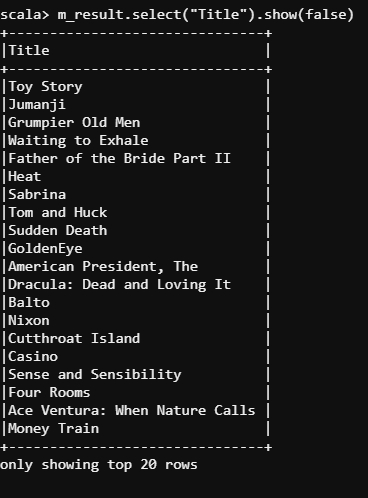
m\_result.where("MovieID=1").show(false)



m\_result.filter("Genres == 'Action'").show(false)



m\_result.select("Title").show(false)



m\_result.filter("Year == '1995'").show(false)



**TOP 10 Viewed Movies**

val ratingsRDD=sc.textFile("/user/edureka\_918210/MovieLens/ratings.dat")

val movies=ratingsRDD.map(line=>line.split("::")(1).toInt)

val movies\_pair=movies.map(mv=>(mv,1))

val movies\_count=movies\_pair.reduceByKey((x,y)=>x+y)

val movies\_sorted=movies\_count.sortBy(x=>x.\_2,false,1)

val mv\_top10List=movies\_sorted.take(10).toList

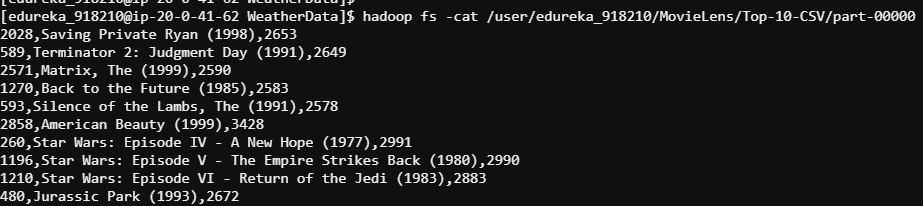
val mv\_top10RDD=sc.parallelize(mv\_top10List)

val mv\_names=sc.textFile("/user/edureka\_918210/MovieLens/movies.dat").map(line=>(line.split("::")(0).toInt,line.split("::")(1)))

val join\_out=mv\_names.join(mv\_top10RDD)

join\_out.sortBy(x=>x.\_2.\_2,false).map(x=> x.\_1+","+x.\_2.\_1+","+x.\_2.\_2).repartition(1).saveAsTextFile("/user/edureka\_918210/MovieLens/Top10-CSV")





**Movies In Each Genre**

val movies\_rdd=sc.textFile("/user/edureka\_918210/MovieLens/movies.dat")

val genre=movies\_rdd.map(lines=>lines.split("::")(2))

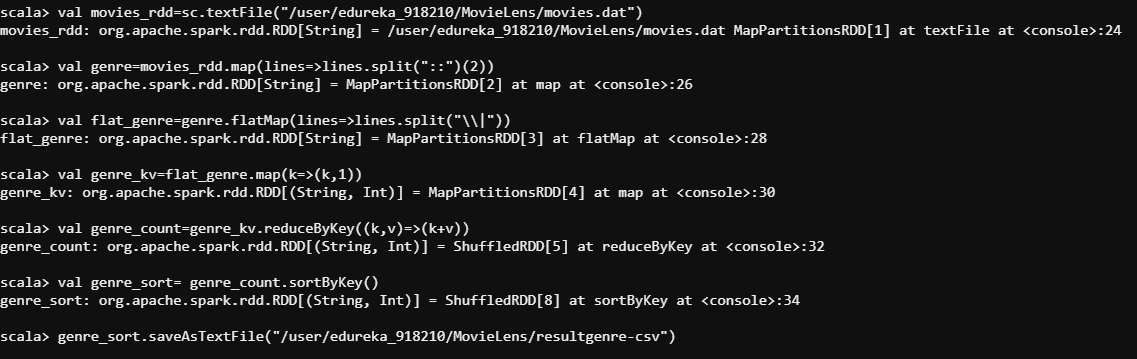
val flat\_genre=genre.flatMap(lines=>lines.split("\\|"))

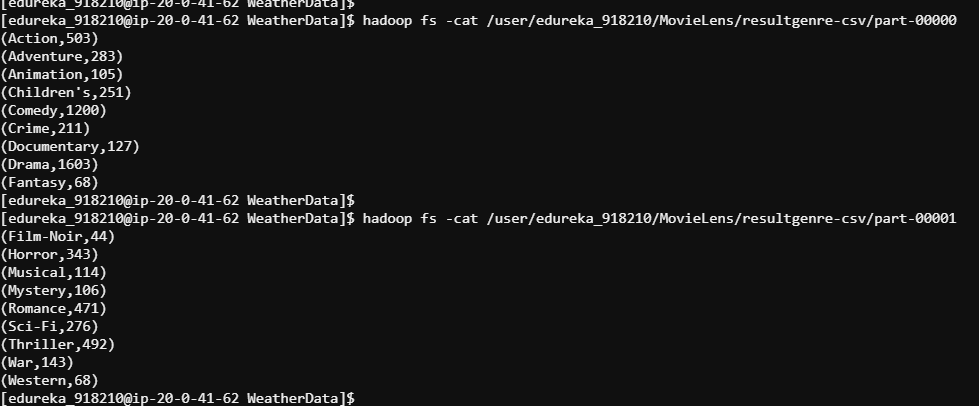
val genre\_kv=flat\_genre.map(k=>(k,1))

val genre\_count=genre\_kv.reduceByKey((k,v)=>(k+v))

val genre\_sort= genre\_count.sortByKey()

genre\_sort.saveAsTextFile("/user/edureka\_918210/MovieLens/resultgenre-csv")





**Latest Movies**

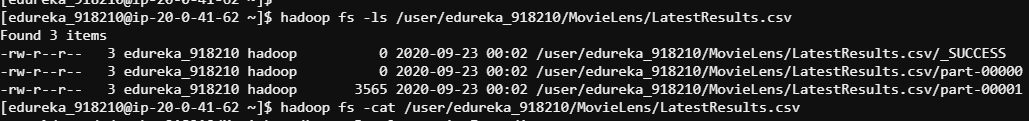
val movies\_rdd=sc.textFile("/user/edureka\_918210/MovieLens/movies.dat")

val movie\_nm=movies\_rdd.map(lines=>lines.split("::")(1))

val year=movie\_nm.map(lines=>lines.substring(lines.lastIndexOf("(")+1,lines.lastIndexOf(")")))

val latest=year.max

val latest\_movies=movie\_nm.filter(lines=>lines.contains("("+latest+")")).saveAsTextFile("/user/edureka\_918210/MovieLens/LatestResults.csv")



SPARK SQL DATALAKE : ALTERNATIVE MOTHOD :

[edureka\_918210@ip-20-0-41-62 ~]$ hadoop fs -ls /user/edureka\_918210/MovieLens

Found 4 items

drwxr-xr-x - edureka\_918210 hadoop 0 2020-08-05 08:25 /user/edureka\_918210/MovieLens/Top-10-CSV

-rw-r--r-- 3 edureka\_918210 hadoop 175191 2020-08-05 06:55 /user/edureka\_918210/MovieLens/movies.dat

-rw-r--r-- 3 edureka\_918210 hadoop 25594340 2020-08-05 06:55 /user/edureka\_918210/MovieLens/ratings.dat

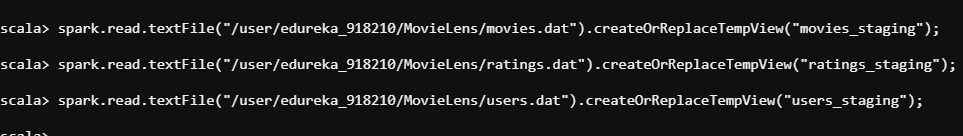
-rw-r--r-- 3 edureka\_918210 hadoop 140408 2020-08-05 06:55 /user/edureka\_918210/MovieLens/users.dat

**// 2nd method is to read the file directly into a dataFrame and create a temp view**

spark.read.textFile("/user/edureka\_918210/MovieLens/movies.dat").createOrReplaceTempView("movies\_staging");

spark.read.textFile("/user/edureka\_918210/MovieLens/ratings.dat").createOrReplaceTempView("ratings\_staging");

spark.read.textFile("/user/edureka\_918210/MovieLens/users.dat").createOrReplaceTempView("users\_staging");



**// Create a database to store the tables**

spark.sql("drop database if exists sparkdatalake cascade")

spark.sql("create database sparkdatalake");

**// Make appropriate schemas for them**

**// movies**

spark.sql(""" Select

split(value,'::')[0] as movieid,

split(value,'::')[1] as title,

substring(split(value,'::')[1],length(split(value,'::')[1])-4,4) as year,

split(value,'::')[2] as genre

from movies\_staging """).write.mode("overwrite").saveAsTable("sparkdatalake.movies");

// users

spark.sql(""" Select

split(value,'::')[0] as userid,

split(value,'::')[1] as gender,

split(value,'::')[2] as age,

split(value,'::')[3] as occupation,

split(value,'::')[4] as zipcode

from users\_staging """).write.mode("overwrite").saveAsTable("sparkdatalake.users");

// ratings

spark.sql(""" Select

split(value,'::')[0] as userid,

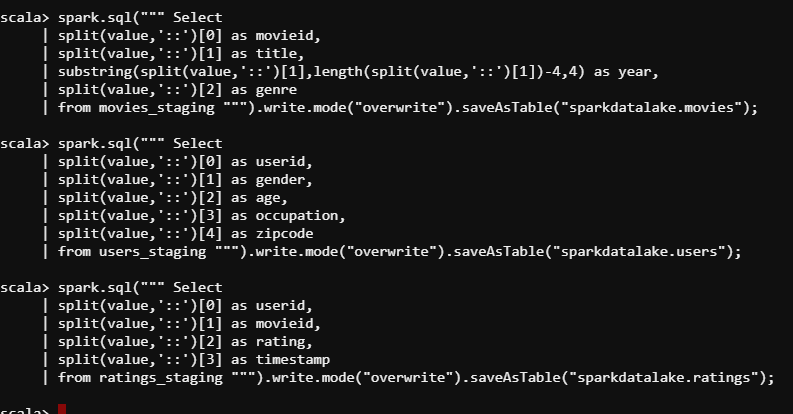
split(value,'::')[1] as movieid,

split(value,'::')[2] as rating,

split(value,'::')[3] as timestamp

from ratings\_staging """).write.mode("overwrite").saveAsTable("sparkdatalake.ratings");

System.exit(0)



TABLES IN SPARK ARE :

//movies

spark.sql(""" Select

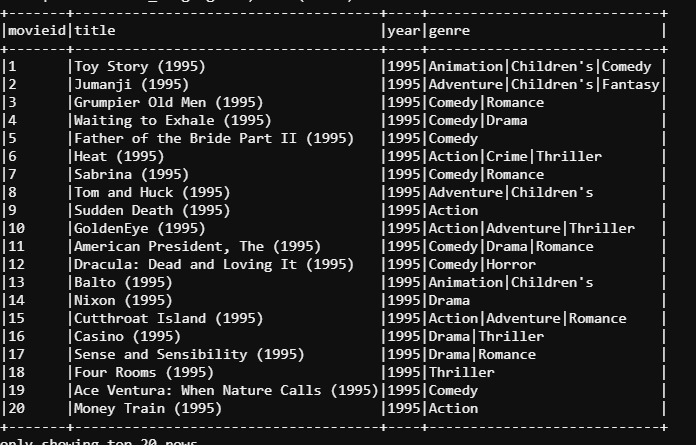
split(value,'::')[0] as movieid,

split(value,'::')[1] as title,

substring(split(value,'::')[1],length(split(value,'::')[1])-4,4) as year,

split(value,'::')[2] as genre

from movies\_staging """).show(false)



// users

spark.sql(""" Select

split(value,'::')[0] as userid,

split(value,'::')[1] as gender,

split(value,'::')[2] as age,

split(value,'::')[3] as occupation,

split(value,'::')[4] as zipcode

from users\_staging """).show(false)

// ratings

==============================================================================================================================================================

spark.sql(""" Select

split(value,'::')[0] as userid,

split(value,'::')[1] as movieid,

split(value,'::')[2] as rating,

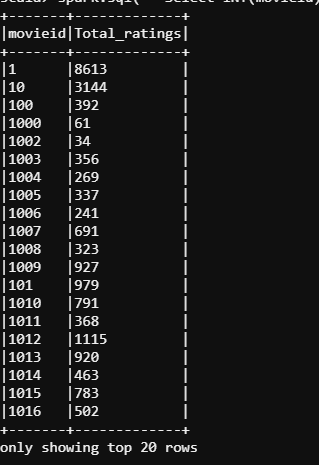
split(value,'::')[3] as timestamp

from ratings\_staging """).show(false)

**//Total Ratings Per Movie**

spark.sql("""Select INT(movieid),sum(INT(rating)) as Total\_ratings from sparkdatalake.ratings group by movieid order by movieid""").show(false)

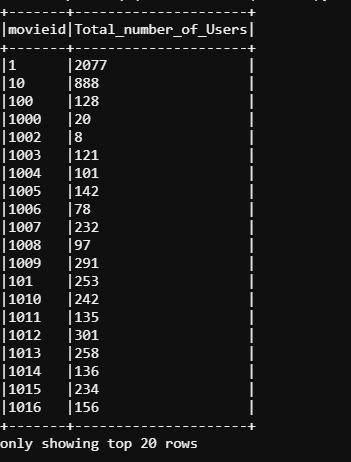
spark.sql("""Select INT(movieid),sum(INT(rating)) as Total\_ratings from sparkdatalake.ratings group by movieid order by movieid""").repartition(1).write.format("csv").option("header","true").save("ResultTotalRatings")



**//Number of Users per movie**

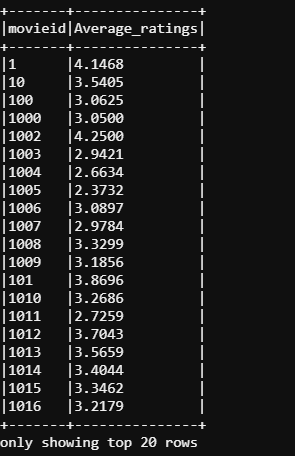
spark.sql("""Select INT(movieid),count(INT(userid)) as Total\_number\_of\_Users from sparkdatalake.ratings group by movieid order by movieid""").show(false)

spark.sql("""Select movieid,count(userid) as Total\_number\_of\_Users from sparkdatalake.ratings group by movieid order by movieid asc""").repartition(1).write.format("csv").option("header","true").save("ratingscount")



**//Average rating per movie**

spark.sql("""Select INT(movieid),avg(decimal(rating)) as Average\_ratings from sparkdatalake.ratings group by movieid order by movieid asc""").show(false)



spark.sql("""Select INT(movieid),avg(decimal(rating)) as Average\_ratings from sparkdatalake.ratings group by movieid order by movieid asc""").repartition(1).write.format("csv").option("header","true").save("resultavgratings")

**//LIST OF OLDEST MOVIES**

val movies\_rdd=sc.textFile("/user/edureka\_918210/MovieLens/movies.dat")

// 1st method, convert existing rdd into DF using toDF function and then make it into a view

val movies\_DF=movies\_rdd.toDF.createOrReplaceTempView("movies\_view")

// To use spark.sql, it should be at least a temporary view or even an table

spark.sql(""" select

split(value,'::')[0] as movieid,

split(value,'::')[1] as moviename,

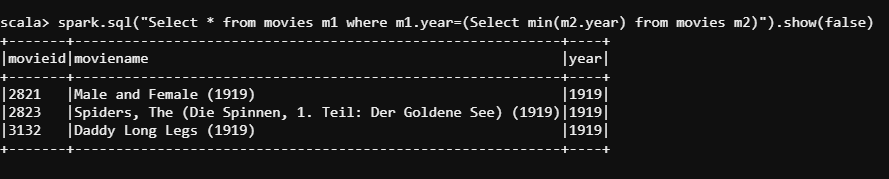
substring(split(value,'::')[1],length(split(value,'::')[1])-4,4) as year

from movies\_view """).createOrReplaceTempView("movies");

// To view the records, use spark.sql("select \* from movies").show()

var result=spark.sql("Select \* from movies m1 where m1.year=(Select min(m2.year) from movies m2)").repartition(1).rdd.saveAsTextFile("result")

spark.sql("Select \* from movies m1 where m1.year=(Select min(m2.year) from movies m2)").show(false)



spark.sql("Select \* from movies m1 where m1.year=(Select min(m2.year) from movies m2)").show(false)

