*30-08-2022*

*Dataframe is a high-level data structure in spark*

*RDD is the fundamental data structure in spark*

*Dataframe is immutable*

cd spark-3.2.1-bin-hadoop3.2/

cd bin/

./spark-shell

*Get dataset from:* [*https://github.com/databricks/Spark-The-Definitive-Guide/blob/master/data/flight-data/json/2015-summary.json*](https://github.com/databricks/Spark-The-Definitive-Guide/blob/master/data/flight-data/json/2015-summary.json)

# Create a dataframe from a data source

val df = spark.read.format("json").load("/path of json file")



# Create a row

import org.apache.spark.sql.Row

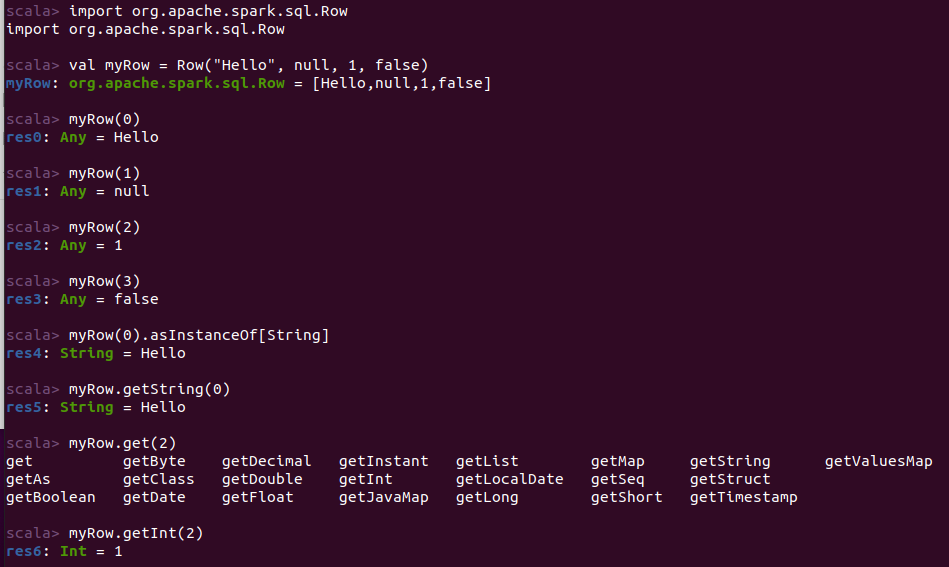
val myRow = Row("Hello", null, 1, false)

myRow(0)

myRow(0).asInstanceOf[String]

myRow.getString(0)

myRow.getInt(2)



# Create a dataframe

import org.apache.spark.sql.Row

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

val myManualSchema = new StructType(Array(new StructField("EmpID", StringType, true), new StructField("EmpName", StringType, true), new StructField("EmpSalary", LongType, true)))

val myRows = Seq(Row("E101", "Arun", 72000L))

val myRDD = spark.sparkContext.parallelize(myRows)

val myDf = spark.createDataFrame(myRDD, myManualSchema)

myDf.show()

Text

Description automatically generated

# Selecting columns from dataframe

df.select("DEST\_COUNTRY\_NAME").show(2)

df.select("DEST\_COUNTRY\_NAME", "ORIGIN\_COUNTRY\_NAME").show(2)

df.select(expr("DEST\_COUNTRY\_NAME")).show(2)

df.select(col(“DEST\_COUNTRY\_NAME”)).show(2)

Text

Description automatically generated

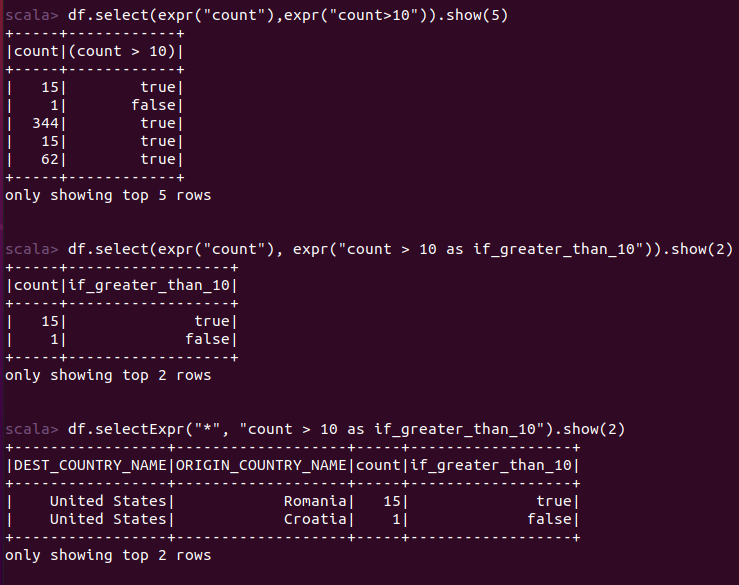
# Operations on column data

df.select(expr("count"),expr("count>10")).show(5)

df.select(expr("count"), expr("count > 10 as if\_greater\_than\_10")).show(2)

df.selectExpr("\*", "count > 10 as if\_greater\_than\_10").show(2)

df.selectExpr("\*","(DEST\_COUNTRY\_NAME == ORIGIN\_COUNTRY\_NAME) as withinCountry").show(100)



A picture containing table

Description automatically generated Table

Description automatically generated with medium confidence

*06-09-2022*

ASSIGNMENT 1

QUESTION 1

Consider the following dataset on salesmen and their profit for three days. Create a data frame by

the name “Salesmen” with the given dataset. Write Spark code to perform the following:

(i) Find the profit average of each day

(ii) Find minimum and maximum profit of each day

(iii) Display the name of all salesmen who has profit less than 1000 on Day3 but profit above 1000 on Day1

(iv) Find unique values from Day2\_Profit

|  |  |  |  |
| --- | --- | --- | --- |
| **Salesman name** | **Day1\_profit** | **Day2\_Profit** | **Day3\_Profit** |
| Kumar | 2000 | 500 | 1200 |
| Raju | 345 | 1775 | 2200 |
| Shiva | 875 | 725 | 1900 |
| Ram | 1025 | 1775 | 645 |

import org.apache.spark.sql.Row

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

val myManualSchema = new StructType(Array(new StructField("Salesman\_namee", StringType, true), new StructField("Day1\_profit", LongType, true), new StructField("Day2\_profit", LongType, true), new StructField("Day3\_profit", LongType, true)))

val myRows = Seq(Row("Kumar",2000L,500L,1200L),Row("Raju",345L,1775L,2200L),Row("Shiva",875L,725L,1900L),Row("Ram",1025L,1775L,645L))

val myRDD = spark.sparkContext.parallelize(myRows)

val salesman = spark.createDataFrame(myRDD, myManualSchema)

salesman.show()

salesman.select(avg("Day1\_profit")).show()

salesman.select(avg("Day2\_profit")).show()

salesman.select(avg("Day3\_profit")).show()

salesman.select(min("Day1\_profit"), max("Day1\_profit")).show()

salesman.select(min("Day2\_profit"), max("Day2\_profit")).show()

salesman.select(min("Day3\_profit"), max("Day3\_profit")).show()

val p1Filter = col("Day3\_profit") < 1000

val p2Filter = col("Day1\_profit") > 1000

salesman.where(p1Filter.and(p2Filter)).show()

salesman.select("Day2\_profit").distinct().count()

QUESTION 2

Consider the following dataset on Airlines and the number of passengers travelled for 3 days. Create a dataframe by the name “Airline” with the given dataset. Write Spark code to perform the following:

(i) Find the airline which has highest number of total passengers. Here, you need not to print the name of the airline. But printing highest of total passengers is sufficient.

(ii) Find and print the maximum number of passengers for each airline

(iii) Display the name of all airlines which has more number of passengers on Day1 than Day2

|  |  |  |  |
| --- | --- | --- | --- |
| **Airline** | **Day1** | **Day2** | **Day3** |
| Indigo | 2250 | 575 | 1500 |
| AisAsia | 275 | 1772 | 2200 |
| Air India | 975 | 725 | 1900 |
| SpiceJet | 1025 | 1250 | 645 |

import org.apache.spark.sql.Row

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

val myManualSchema = new StructType(Array(new StructField("Airline", StringType, true), new StructField("Day1", LongType, true), new StructField("Day2", LongType, true), new StructField("Day3", LongType, true)))

val myRows = Seq(Row("Indigo",2250L,575L,1500L),Row("AisAsia",275L,1772L,2200L),Row("Air India",975L,725L,1900L),Row("SpiceJet",1025L,1250L,645L))

val myRDD = spark.sparkContext.parallelize(myRows)

val airline = spark.createDataFrame(myRDD, myManualSchema)

airline.show()

val airline1 = airline.withColumn("TOTAL", expr("Day1+Day2+Day3"))

airline1.show()

airline1.select(max("TOTAL")).show()

airline.withColumn("maximum", greatest("Day1","Day2","Day3")).show()

val p1Filter = col("Day1") >col("Day2")

airline.where(p1Filter).show()

QUESTION 3

Consider the following dataset on University and the number of admission for three years. Create a data frame by the name “University” with the given dataset. Write Spark code to perform the following:

(i) Find the admission average of each University for three years

(ii) Find the lowest admission of each year

(iii) Display the name of all University which has admission less than 1000 in Year1 but admission above 1000 in Year2

(iv) Find unique values from Year3

|  |  |  |  |
| --- | --- | --- | --- |
| **University** | **Year1** | **Year2** | **Year3** |
| abc | 600 | 750 | 950 |
| xyz | 345 | 1500 | 2100 |
| def | 875 | 725 | 1875 |
| ghi | 1025 | 1772 | 950 |

import org.apache.spark.sql.Row

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

val myManualSchema = new StructType(Array(new StructField("University", StringType, true), new StructField("Year1", LongType, true), new StructField("Year2", LongType, true), new StructField("Year3", LongType, true)))

val myRows = Seq(Row("abc",600L,750L,950L),Row("xyz",345L,1500L,2100L),Row("def",875L,725L,1875L),Row("ghi",1025L,1772L,950L))

val myRDD = spark.sparkContext.parallelize(myRows)

val university = spark.createDataFrame(myRDD, myManualSchema)

university.show()

university.withColumn("AdmisAvg", expr("(Year1+Year2+Year3)/3")).show()

university.select(min("Year1")).show()

university.select(min("Year2")).show()

university.select(min("Year3")).show()

val p1Filter = col("Year1") < 1000

val p2Filter = col("Year2") > 1000

university.where(p1Filter.and(p2Filter)).show()

university.select("Year3").distinct().count()

*13-09-2022*

*In bin folder put the json files*

*create an empty folder in bin folder*

*when you execute last statement in the code, you can add files one by one to that folder that you’ve created*

*as you keep adding the files to the empty folder, your streaming code is already running*

*you keep adding files right, that is equivalent to data coming in streams*

*3rd line of the code, you have to specify the name of the empty folder -> activitydata (name sir choose for the empty folder)*

*4th line, we are giving our query, we know the objective which is grouping*

*5th line, the actual streaming is being stopped. This is where streaming code will be running and we wait for the data to be processed. For format method, it takes one argument, console.*

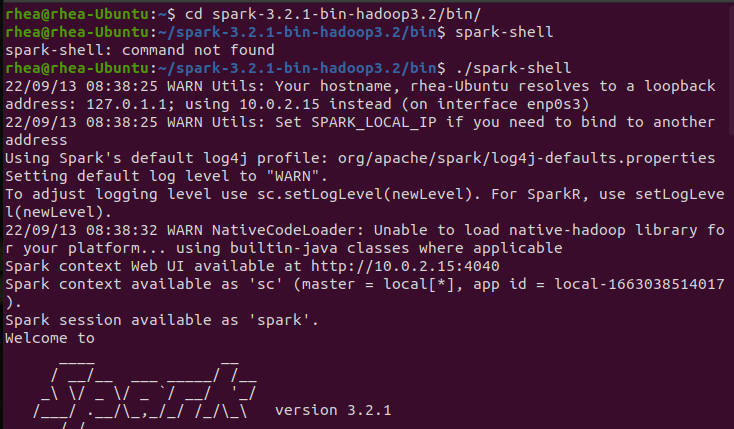
*Apart from the format method there is another method – append , output mode method – complete*

*We will be understanding all the types*

# Put activity1.json in bin folder

# Make a folder called activitydata in bin folder

# Once this is done, open spark shell and execute the following commands



*//reading schema of data file (we can give any one file)*

val static = spark.read.json("activity1.json")

val dataSchema = static.schema

*//activity data is the name of the empty folder*

val streaming = spark.readStream.schema(dataSchema).option("maxFilesPerTrigger", 1).json("activitydata")

*//we give the query -> grouping by activities , gt-> activity field*

val activityCounts = streaming.groupBy("gt").count()

Text

Description automatically generated

*//actual streamming is started*

val activityQuery = activityCounts.writeStream.queryName("activity\_counts100").format("console").outputMode("complete").start

Text

Description automatically generated

Now copy paste the activity 1 and 2 file in activitydata folder 1 by one and observe the terminal, stream processing takes place

When activity1 is put in folder ..

Text

Description automatically generated

When activity 2 is put in folder

A picture containing text

Description automatically generated

QUESTION:

Write a streaming application that groups the streaming data by gender and calculate average weight and height.

You will be provided a set of files having the height and weight of several individuals. Each record will have height, weight, and gender of a person. You’ll have to write a streaming code which groups the record by the gender field. Here you’ll do average as the calculation.

val static = spark.read.option("inferSchema",true).csv("weight-heightaa.csv")

val dataSchema = static.schema

val streaming = spark.readStream.schema(dataSchema).option("maxFilesPerTrigger", 1).csv("height-weight")

val avgheight\_weight = streaming.groupBy("\_c0").avg("\_c1", "\_c2")

val heightWeightQuery = avgheight\_weight.writeStream.queryName("heightWeight100").format("console").outputMode("complete").start

*11-10-2022*