CS 5433 - Big Data Management Spring 2022
Spark Programming Assignment -1

4/15/2022
GROUP – 9

Task 1: Calculating the average income by marital status using the SQL 'GROUP' clause.

Spark SQL is a structured data processing Spark module. It offers data frames as a programming abstraction and may also serve as a distributed SQL query engine.

Approach:

- The code for this task is done using Sparksql in Scala. Initially we have imported the *SparkSession* using, '*import.org.apache.spark.sql.SparkSession'*.
- As sparkSession is the entry point for reading data. We created a spark session to send the user commands and data to a spark application. We have used *.getOrCreate()* which would create a new session or get an existing one if it exists already and named application name as 'spark SQL' using *appName()* method. *SparkSession.builder()* method is created for constructing a SpakrSession. The *.config()* is used to set config option which are by default propagated to both 'SparkConf' and 'Spark Session'.
  '*SparkSession.builder().appName*("Spark SQL").*config("spark.some.config.option", "some-value").getOrCreate()*'.
- Next for reading the data from the data sources, we have used data frame reader '*spark.read.format()*' method. Make sure to check if the header was true using *option()* method. For the given file format, we have used the delimiter as "\t" which was specified in the *option()* method and then used load() method to load the path of the data '*/user/kaggle/kaggle_data/marketing_campaign.csv*' to load the data frame in the same format wh the tabular format.
  df=spark.read.format("csv").option("header","true").option("delimiter", "\t").load("/user/kaggle/kaggle_data/marketing_campaign.csv");
- Using the method *createOrReplaceTempView()* to create a local temporary view and to convert the data frame into a temporary view 'people'.
  df.createOrReplaceTempView("people")
- Next, we are creating a new dataframe 'sqlDF' and running our required final sql query using *spark.sql()* method, the query would fetch the 'average_income' grouping the 'marital_status'.
  val sqlDF = spark.sql("Select Marital_Status,avg(Income) as Income from people group by Marital_Status")
- Next, we have used a command to print the output of the program to the console.
  sqlDF.show()
- The final step of the program is to write the output of the program to a file in csv format, the output file should be changed every time as it will not be overwritten. We can run commands in Hadoop to check the output.
  sqlDF.coalesce(1).write.option("header","true").csv("/user/ssamine/spark_task_1.csv ")

Instructions to run code and screenshots for Task 1:

1,1)
Run the command to start spark session. It will open the shell
Command: spark-shell

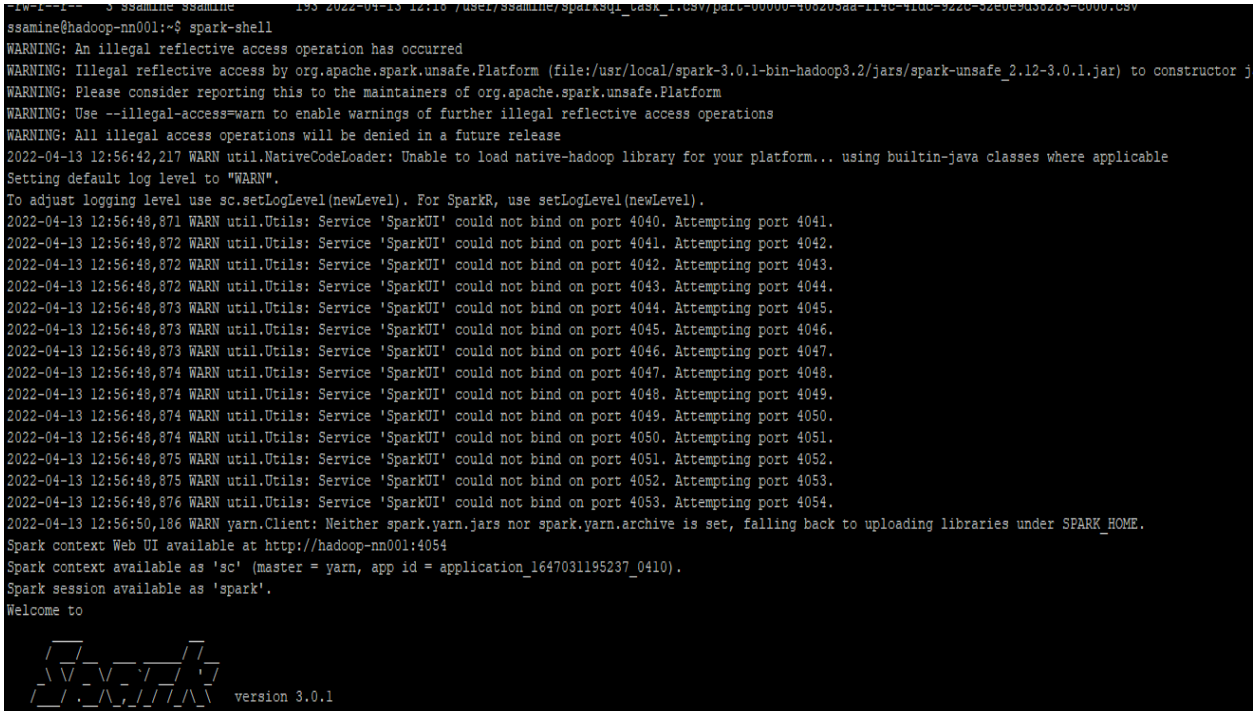

Figure 1,1: Shows the command for starting Spark session

1,2)
Compile and execute the .scala file using the command. Output will be printed on the console and written into file as well.
Command: load filepath
Ex: load /home/ssamine/Group_9_Program_1.scala

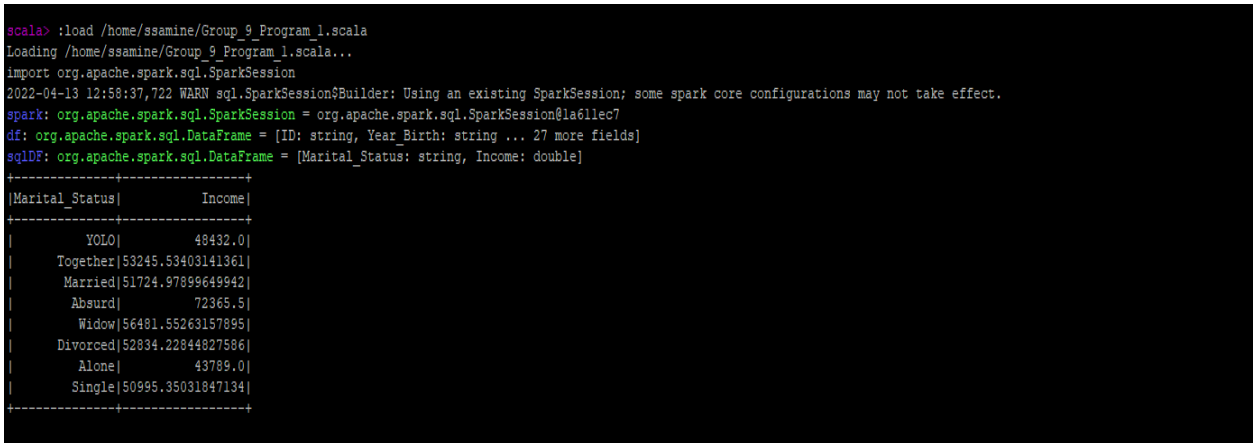

Figure 1,2: Shows the command to compile, run the program and print the output on the console.

1,3)
Checking the output file
Command: hdfs dfs -ls outputpath/outputfilename/part-r-00000
Ex: hdfs dfs -ls /user/ssamine/sparksql_task_1.csv/part-00000-408205aa-ff4c-4fdc-922c-52e0e9d38285-c000.csv

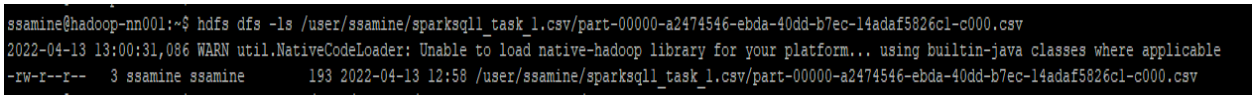

Figure 1,3: Shows the command to view the output file

1,4)
For viewing output
hdfs dfs -cat outputpath/outputfilename/part-r-00000
Ex: hdfs dfs -cat /user/ssamine/sparksql_task_1.csv/part-00000-408205aa-ff4c-4fdc-922c-52e0e9d3828-c000.csv

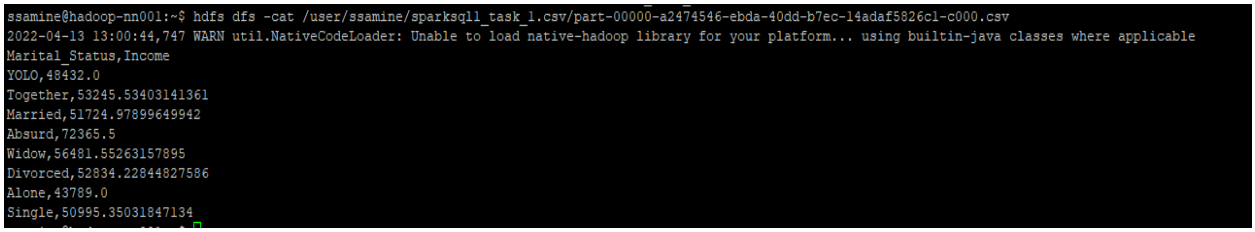

Figure 1,4: Shows the command to view the output written to output file

Sample Output:

Marital_Status,avg_Income
YOLO,48432.0
Together,53245.53403141361
Married,51724.97899649942
Absurd,72365.5
Widow,56481.55263157895
Divorced,52834.22844827586
Alone,43789.0
Single,50995.35031847134

Task 2: Calculating the average income by marital status using Python.

Pyspark is a Python-based connection for Apache Spark. It's a Spark Python API that connects Apache Spark and Python to Resilient Distributed Datasets (RDDs).

Approach:

- For this task 2 we have used pyspark, python. Initially we need to import all the required libraries. We have imported *SparkSession*, *SparkContext* and *SparkConf*. *SparkContect* represents a connection to a spark cluster, Spark Config provides configurations to run a spark application and SparkSession is the entry point for reading data.
  import sys
  from pyspark import SparkContext, SparkConf
  from pyspark.sql.session import SparkSession
- Next, we make sure everything in the program after main is executed when the python interpreter runs the module directly.
  if __name__ == "__main__":
- Then we set the configuration for the spark application using *SparkConf()* method and set the app name as "BDM' using *setAPPName()* method.
  conf = SparkConf().setAppName("BDM")
- Next we create the connection to a spark cluster using *SparkContext()* method .
  sc = SparkContext(conf=conf)
- Next, we created a spark session to send the user commands and data to a spark application. We have used *.getOrCreate()* which would create a new session or get an existing one if it exists already and named application name as 'BDMPyspark' using *appName()* method. *SparkSession.builder* method is created for constructing a SpakrSession.
  spark = SparkSession.builder.appName("BDMPyspark").getOrCreate()
- Next for reading the data from the data sources, we have used data frame reader '*spark.read.csv()*' method. Make sure to check if the header was true. For the given file format, we have used the delimiter as "\t" and then used the path of the data '*/user/kaggle/kaggle_data/marketing_campaign.csv*' to load the data frame in the same format which is the tabular format.
  df = spark.read.csv('/user/kaggle/kaggle_data/marketing_campaign.csv', sep='\t', inferSchema=True, header=True)
- Next created a new data frame 'res', which will give us the average income by the martial status. By using *group()* and *avg()* methods. Here, groupby will group the people who have same marital status with aggerate function average.
  res = df.groupBy("Marital_Status").avg("Income")
- The output is printed to the console.
  res.show()
- The final step of the program is to write the output of the program to a file in csv format.
  res.coalesce(1).write.option("header","true").csv("/user/vgollap/Group_9_Program_2_output.csv")

Instructions to run code and screenshots for Task 2:

2,1)
Run the python file using spark-submit command:
Command: spark-submit pathtoPythonFile/pythonFile.py
Ex: spark-submit Group_9__Program_2.py

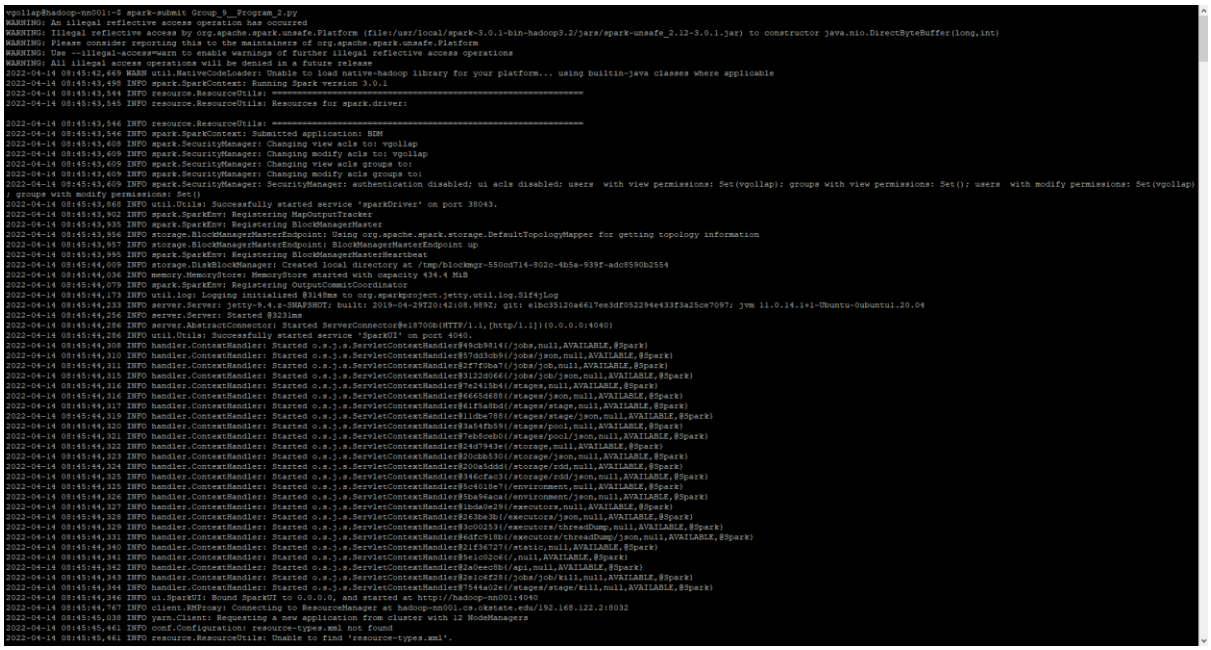

Figure 2,1 Running command in pyspark

2,2) Output will be printed on the console and written into file as well

Figure 2,2 Printed Output on console

2,3)
Checking the output file
Command: hdfs dfs -cat outputpath/outputfilename/part-r-00000
Ex: hdfs dfs -cat /user/vgollap/Group_9_Program_2_output.csv/part-00000-370cc613-7053-4fef-92cc-0cbf782ec4c9-c000.csv

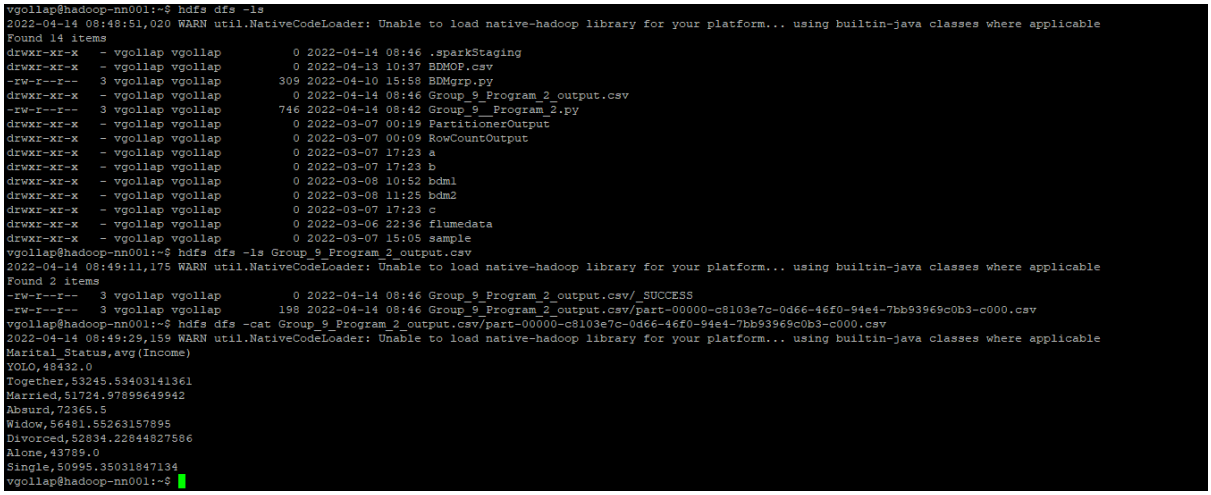

Figure 2,3 To look up the written output file

Sample Output:

Sample-output_Program_2

Marital_Status	avg(Income)
YOLO	48432.0
Together	53245.53403141360
Married	51724.97899649940
Absurd	72365.5
Widow	56481.55263157900
Divorced	52834.22844827590
Alone	43789.0
Single	50995.35031847130