APACHE SPARK – RDD PROGRAMMING

Contents

[Introduction 3](#_Toc146628904)

[About Data Set 3](#_Toc146628905)

[Procedure 3](#_Toc146628906)

[Queries using Apache Hive 10](#_Toc146628907)

[Using Apache Spark RDD Programming 13](#_Toc146628908)

[Presentation Video 16](#_Toc146628909)

# Introduction

In this assignment, we will be looking into Apache Spark where we are using a higher order function to program Spark RRD. In this assignment, we will be looking into a comparison of Hive where we will be creating a database in hive to run a few queries and in turn we will be making use of Scala for RDD programming in Spark to perform the same operation we have carried out in hive. We are using Hadoop file system to store our dataset and to extract some insights from the data set we have collected from Kaggle.

# About Data Set

The data chosen is related to the Airline industry, where the data holds the “Airline Passenger Satisfaction” based on several factors determining the overall end-user experience. There are various attributes which will help us get various aspects of the Airline industry throughout a passenger's journey. This will help us analyze the factors that contribute to the customer's satisfaction levels. The data set has 76032 rows with data about customer satisfaction based on several factors. The data will be collected and added in 2018. The source of the data set is Kaggle, contributed by John D, and it's licensed publicly. The data set holds the information about passenger satisfaction in the year 2015 for US aviation.

# Procedure

Step 1: We have downloaded the data set from Kaggle by going to this link and download the file.

Source link: <https://www.kaggle.com/datasets/johndddddd/customer-satisfaction>

The excel sheet was in .xlsx format which needed to be saved as a .csv file.

* Creating directory in Hadoop with the command

Command: ***hadoop fs -mkdir -p /BigData/***

***A screenshot of a computer

Description automatically generated***

* Copy file from local to hdfs

Command: ***hadoop fs -copyFromLocal Airline.csv/ BigData/***

A screenshot of a computer

Description automatically generated

We are able to see the file by using the ls command

Command: ***hadoop fs -ls /BigData/***

Creating Database:

Command :  ***hive***

***CREATE DATABASE passenger\_db;***

To list all the databases we used the command ***show databases;***

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Then we will use the command ***use passenger.db*** to use the database

A screenshot of a computer

Description automatically generated

* Add a new table

Based on the columns we have on the database, we will create the table with appropriate datatypes.

Command:

***CREATE EXTERNAL TABLE IF NOT EXISTS passenger\_db (***

***Gender STRING,***

***Type\_of\_Travel STRING,***

***Customer\_Type STRING,***

***Class STRING,***

***Age INT,***

***Flight\_Distance INT,***

***Departure\_Delay INT,***

***Arrival\_Delay INT,***

***Departure\_and\_Arrival\_Time\_Convenience INT,***

***Ease\_of\_Online\_Booking INT,***

***Check\_in\_Service INT,***

***Online\_Boarding INT,***

***Gate\_Location INT,***

***Onboard\_Service INT,***

***Seat\_Comfort INT,***

***Leg\_Room\_Service INT,***

***Cleanliness INT,***

***Food\_and\_Drink INT,***

***Inflight\_Service INT,***

***Inflight\_Wifi\_Service INT,***

***Inflight\_Entertainment INT,***

***Baggage\_Handling INT,***

***Satisfaction STRING***

***)***

***ROW FORMAT DELIMITED***

***FIELDS TERMINATED BY ','***

***TBLPROPERTIES("skip.header.line.count"="1");***

A screenshot of a computer

Description automatically generated

To check if the table has all the columns created properly we use ***Describe passenger\_db;***

A screenshot of a computer

Description automatically generated

* Import file to database

To import the csv file to the database we would use the command.

Command : ***LOAD DATA INPATH '/BigDataAssignment2/hive/satisfaction.csv' INTO TABLE passenger\_db;***

A screenshot of a computer

Description automatically generated

Now that the table is loaded with the data we can run a few test queries to verify.

Test Query 1:

***select \* from passenger\_db where gender = ‘Female’ LIMIT 10;***

A screenshot of a computer

Description automatically generated

Test Query 2:

***select \* from passenger\_db LIMIT 10;***

A screenshot of a computer

Description automatically generated

# Queries using Apache Hive

* Query 1: Maximum distance by type of travel

***SELECT type\_of\_travel,MAX(flight\_distance) AS MaximumFlightDistance***

***FROM passenger\_db***

***GROUP BY type\_of\_travel;***

A screenshot of a computer

Description automatically generated

Based on the different travel types, we will be able to get an insight into the dataset where the distance covered maximum attribute is used. When we investigate the data, we can make out the flight duration for each travel type that is business travelers as well as passengers who are travelling for something personal. We will be able to identify the longest flights when we check the purpose of travel. We will be able to get a basic understanding about the duration and the selection of classes.

With this data we can add further optimization to the routes and identify routes which could be adjusted or maybe we can add additional routes to the fleet. We will be able to study the demand of travel and we can do targeted marketing based on the destinations which the customers are traveling to. We can understand the travel demands of passengers which have specific travel types to make route planning and marketing tactics.

* Query 2:Passengers based on the type of customers***.***

***SELECT customer\_type,COUNT(\*) AS TotalPassengers***

***FROM passenger\_db***

***GROUP BY customer\_type;***

A screenshot of a computer

Description automatically generated

For each of the customer types we have in the data set we will be able to understand the customers type whether they are first-time or those which are returning customers. This query will help us understand the count of first-time vs returning travelers. Additionally, we will be able to get insights into the marketing strategies so that we can build a plan to meet the specific preferences of the passengers. The airline industry can make use of this data for service improvement such as new offering and loyalty programs.

* Query 3: Minimum distance by type of travel

***SELECT type\_of\_travel,MIN(flight\_distance) AS MinimumFlightDistance***

***FROM passenger\_db***

***GROUP BY type\_of\_travel;***

A screenshot of a computer

Description automatically generated

The insights that could be gathered from the query include the minimum distance that is travelled or different types of travel which will give is a comparison of the minimum distance travel vs long distance travel for those passengers who are travelling for difference purposes such as business or personal travel. This will also help understand the route preferences and the destinations to which the passengers are travelling. We can make out the shortest flights that are associated with each of the travel types. When we consider short trips for passengers there are several factors that would impact on customer satisfaction if the service in general or in-flight services are not up to the mark. There can be planning for the operational resource optimization as well.

# Using Apache Spark RDD Programming

RDD program for the hive queries we have defined for the data set that includes the functions such as map , filter and reducebyKey is successfully executed and the comparison is really surprising with the time taken for execution which was instant and there was no waiting when the map reduce is taking place.

1. Apache spark RDD program for the first query we have used for Maximum flight distance based on the type of travel

Apache Spark with Scala shell command used:

***val lines = sc.textFile("hdfs://10.128.0.2:8020/BigData/Airline.csv")***

// The csv file hold the first row as its headers and hence we display it to make sure the headers are selected

***val header = lines.first()***

// To take all the data by applying filter except for the header row in the csv file.

***val data = lines.filter(line => line != header)***

// The data hence filtered is split each line by comma and we will make a key-value pairs with type\_of\_travel as the key and flight\_distance as the value

***val keyValuePairs = data.map(line => {***

***val columns = line.split(",")***

***(columns(1), columns(5).toDouble)*** // column 1 is the type of travel and column 5 is the flight distance

***})***

// The key value pair thus obtained is used along with the function reduceByKey and the max value is found for each of the travel types

***val maxFlightDistanceByType = keyValuePairs.reduceByKey((x, y) => Math.max(x, y))***

// Printing the result

***maxFlightDistanceByType.take(10).foreach(println)***

A screenshot of a computer program

Description automatically generated

1. Apache spark RDD program for the second query query we have used for to find out the count of passengers based on the type of customers.

Apache Spark with Scala shell command used:

As the data is already loaded we can directly run the commands on Spark Shell and the headers are removed.

// Since we have the csv file we will first split each row by comma and we make key-value pairs with customer\_type as the key and 1 as the value

***val keyValuePairs = data.map(line => {***

***val columns = line.split(",")***

***(columns(2), 1)***

***})***

// Making use of the function reducebyey to count the number of passengers and the customer type.

***val totalPassengersByCustomerType = keyValuePairs.reduceByKey(\_ + \_)***

// Printing the result

***totalPassengersByCustomerType.take(10).foreach(println)***

***A screenshot of a computer program

Description automatically generated***

1. Apache spark RDD program for the first query we have used for Minimum flight distance based on the type of travel

Apache Spark with Scala shell command used:

// The data hence filtered is split each line by comma and we will make a key-value pairs with type\_of\_travel as the key and flight\_distance as the value

***val keyValuePairs = data.map(line => {***

***val columns = line.split(",")***

***(columns(1), columns(5).toDouble)*** // column 1 is the type of travel and column 5 is the flight distance

***})***

// The key value pair thus obtained is used along with the function reduceByKey and the min value is found for each of the travel types using the function Math.min()

***val minFlightDistanceByType = keyValuePairs.reduceByKey((x, y) => Math.min(x, y))***

// Printing the result

***minFlightDistanceByType.take(10).foreach(println)***

A screenshot of a computer

Description automatically generated

Reference:

1. <https://stackoverflow.com/questions/54000211/scala-how-to-find-the-minimum-of-more-than-2-elements>
2. <https://stackoverflow.com/questions/48875375/math-max-not-picking-greatest-number>
3. <https://stackoverflow.com/questions/42029053/scala-collect-function>
4. <https://stackoverflow.com/questions/8000903/what-are-all-the-uses-of-an-underscore-in-scala>
5. <https://stackoverflow.com/questions/36304568/calculate-the-numbers-of-occurrences-in-a-csv-file-in-scala-spark-rdd>

# Presentation Video

[](https://www.youtube.com/embed/JaiMJAClMHg?feature=oembed)