databricks Assignment-45.1 - R

```
// Databricks notebook source
// Task 1
/*
Given a list of numbers - List[Int] (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
- find the sum of all numbers
- find the total elements in the list
- calculate the average of the numbers in the list
- find the sum of all the even numbers in the list
- find the total number of elements in the list divisible by both 5 and 3
*/


//Given a list of numbers
val nums: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
//Find the sum of all numbers
val Sum: Float = nums.sum
println("Sum of Elements in List     : " , nums.sum)

//Find the total elements in the list
val Len: Float = nums.length
println("Total Elements in List      : " ,nums.length)

//Calculate the average of the numbers in the list
val Avg: Float = Sum/Len
println("Average of Elements in List : ", Avg)

//Find the sum of all the even numbers in the list
val even = nums.filter(x => x%2 == 0)
println("Sum of Even Numbers in List : ", even.sum)

//Find the total number of elements in the list divisible by both 5 and 3
val div5and3  = nums.filter(x => ((x%5 == 0) && (x%3 == 0)))
print("Number of Elements divisible by 5 & 3 : ", div5and3.length)

(Sum of Elements in List     : ,55)
(Total Elements in List      : ,10)
(Average of Elements in List : ,5.5)
(Sum of Even Numbers in List : ,30)
(Number of Elements divisible by 5 & 3 : ,0)nums: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
Sum: Float = 55.0
Len: Float = 10.0
Avg: Float = 5.5
even: List[Int] = List(2, 4, 6, 8, 10)
div5and3: List[Int] = List()

/*
Task 2
1) Pen down the limitations of MapReduce.
2) What is RDD? Explain few features of RDD?
3) List down few Spark RDD operations and explain each of them.
*/
```

```
/*
1) Pen down the limitations of MapReduce.
Answer :
Key difference between  Spark & Hadoop MapReduce lies in the approach to processing: Spark can do it in-memory, while Hadoop MapReduce has to read from and
write to a disk. As a result, the speed of processing differs significantly - Spark may be up to 100 times faster. However, the volume of data processed also
differs: Hadoop MapReduce is able to work with far larger data sets than Spark.

Tasks Hadoop MapReduce is good for:
Linear processing of huge data sets,
Economical solution, if no immediate results are expected

Tasks Spark is good for:
Fast data processing,
Iterative processing,
Near real-time processing,
Graph processing,
Machine learning,
Joining datasets

2) What is RDD? Explain few features of RDD?
Answer :
It is the primary abstraction in Spark and is the core of Apache Spark.
Immutable and partitioned collection of records, which can only be created by coarse grained operations such as map, filter, group-by, etc.
Can only be created by reading data from a stable storage like HDFS or by transformations on existing RDD's.
One could compare RDDs to collections in Scala, i.e. a RDD is computed on many JVMs while a Scala collection lives on a single JVM.

Features of RDD -
- Resilient, i.e. fault-tolerant with the help of RDD lineage graph and so able to recompute missing or damaged partitions due to node failures
- Distributed with data residing on multiple nodes in a cluster.
- Dataset is a collection of partitioned data with primitive values or values of values, e.g. tuples or other objects
- In-Memory, i.e. data inside RDD is stored in memory as much (size) and long (time) as possible.
- Immutable or Read-Only, i.e. it does not change once created and can only be transformed using transformations to new RDDs.
- Lazy evaluated, i.e. the data inside RDD is not available or transformed until an action is executed that triggers the execution.
- Cacheable, i.e. you can hold all the data in a persistent "storage" like memory (default and the most preferred) or disk (the least preferred due to access
speed).

3) List down few Spark RDD operations and explain each of them.
Answer :

a) Actions - operations that trigger computation and return values

  1. count()
  Action count() returns the number of elements in RDD.

  2. collect()
  The action collect() is the common and simplest operation that returns our entire RDDs content to driver program.

  3. take(n)
  The action take(n) returns n number of elements from RDD.

  4. reduce()
  The reduce() function takes the two elements as input from the RDD and then produces the output of the same type as that of the input elements.

b) Transformations - lazy operations that return another RDD

  1. map(func)
  The map function iterates over every line in RDD and split into new RDD.
  Using map() transformation we take in any function, and that function is applied to every element of RDD.

  2. flatMap()
  With the help of flatMap() function, to each input element, we have many elements in an output RDD.
  The most simple use of flatMap() is to split each input string into words.

  3. filter(func)
  Spark RDD filter() function returns a new RDD, containing only the elements that meet a predicate.

*/
```

```
/*
Task 3
1. Write a program to read a text file and print the number of rows of data in the document.
2. Write a program to read a text file and print the number of words in the document.
3. We have a document where the word separator is -, instead of space. Write a spark code, to obtain the count of the total number of words present in the
document.
Sample document :
This-is-my-first-assignment.
It-will-count-the-number-of-lines-in-this-document.
This-is-the-first-row.
second-row.
third-row
fourth-row.
number-of-lines-in-this-document.
*/


//Read file
val file = sc.textFile("/FileStore/tables/ex45_1-77c4d.txt")
//Number of rows in file
val c = file.count()

file: org.apache.spark.rdd.RDD[String] = /FileStore/tables/ex45_1-77c4d.txt MapPartitionsRDD[22] at textFile at command-98676305355076:2
c: Long = 7

//Number of words in file
val allWords = file.flatMap(_.split("-"))
val d = allWords.count()
```

```
allWords: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[23] at flatMap at command-98676305355077:2
d: Long = 32

//Check for Not null words
val words = allWords.filter(!_.isEmpty)

//Number of Not null words in file
val e = words.count()

words: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[24] at filter at command-98676305355078:2
e: Long = 32

//Frequency count of Words
val pairs = words.map((_,1))
val reducedByKey = pairs.reduceByKey(_ + _)

pairs: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[27] at map at command-98676305355079:2
reducedByKey: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[28] at reduceByKey at command-98676305355079:3

top10words: Array[(String, Int)] = Array((This,2), (this,2), (lines,2), (of,2), (is,2), (the,2), (number,2), (in,2), (document.,2), (row.,2))
```