Module 2

P1

FIND THE COREALTION BETWEEN TWO LIST of numbers implement the formula for person correlation coefficient

object Correlation {

def main(args: Array[String]): Unit = {

val x = List(10.0, 20.0, 30.0, 40.0, 50.0)

val y = List(12.0, 24.0, 33.0, 47.0, 55.0)

val n = x.length

// Check for equal lengths

if (n != y.length) {

println("Lists must be of equal length.")

return

}

val sumX = x.sum

val sumY = y.sum

val sumXY = (x zip y).map { case (a, b) => a \* b }.sum

val sumX2 = x.map(a => a \* a).sum

val sumY2 = y.map(b => b \* b).sum

val numerator = n \* sumXY - sumX \* sumY

val denominator = math.sqrt((n \* sumX2 - sumX \* sumX) \* (n \* sumY2 - sumY \* sumY))

if (denominator != 0)

println(f"Pearson Correlation: ${numerator / denominator}%.4f")

else

println("Correlation undefined (division by zero).")

}

}

P2

Calculate the moving average of a time series data using scala collection

object MovingAverage {

def main(args: Array[String]): Unit = {

val data = List(10.0, 12.0, 14.0, 18.0, 20.0, 24.0, 22.0, 26.0)

val windowSize = 3

// Calculate moving averages using sliding window

val movingAverages = data.sliding(windowSize).map(window => window.sum / window.size).toList

println(s"Original Data: $data")

println(s"$windowSize-point Moving Average: $movingAverages")

}

}

P3

Write a program to compute frequency distribution and cumulative frequency of a dataset

object FrequencyDistribution {

def main(args: Array[String]): Unit = {

val data = List(5, 3, 2, 5, 2, 3, 5, 2, 4, 3, 4, 2, 5)

// Calculate frequency of each value

val freq = data.groupBy(identity)

.view

.mapValues(\_.size)

.toSeq

.sortBy(\_.\_1)

// Calculate cumulative frequency

val cumFreq = freq.scanLeft(0)(\_ + \_.\_2).tail

// Print header

println("Value\tFreq\tCumFreq")

// Print frequency and cumulative frequency rows

freq.zip(cumFreq).foreach { case ((v, f), c) =>

println(s"$v\t$f\t$c")

}

}

}

P4

SORT A DATSET BY A SPEFIC COLUMN AND EXTRACT THE TOP 5 ROWS

case class Person(name: String, age: Int, score: Double)

object SortAndTop5 {

def main(args: Array[String]): Unit = {

val data = List(

Person("Alice", 25, 88.5),

Person("Bob", 22, 91.0),

Person("Charlie", 24, 79.0),

Person("David", 23, 95.5),

Person("Eva", 26, 84.0),

Person("Frank", 27, 89.5),

Person("Grace", 22, 92.0)

)

// Sort by score descending and take top 5

val top5 = data.sortBy(-\_.score).take(5)

// Display results

println("Top 5 by Score:")

top5.foreach(p => println(s"${p.name}\tAge: ${p.age}\tScore: ${p.score}"))

}

}

P11

Perform basic time series analysis in Scala. Generate synthetic time series data (e.g., daily sales over a month).

import scala.util.Random

import java.time.LocalDate

object TimeSeriesAnalysis {

def main(args: Array[String]): Unit = {

val rand = new Random()

val start = LocalDate.of(2025, 7, 1)

// Generate 30 days of sales data: date -> sales (100 to 200)

val data = (0 until 30).map(i => (start.plusDays(i), 100 + rand.nextInt(101)))

println("Date \tSales")

data.foreach { case (d, s) => println(f"$d%-10s\t$s") }

val sales = data.map(\_.\_2)

println(f"\nTotal: ${sales.sum}%,d, Average: ${sales.sum.toDouble / sales.size}%.2f")

println("\n7-Day Moving Average:")

sales.sliding(7).map(\_.sum / 7.0).zipWithIndex.foreach { case (avg, i) =>

println(f"Day ${i + 1}%2d: $avg%.2f")

}

}

}

P9

Set up Apache Spark locally and count the frequency of words in a text

file

val scalaVersionUsed = "2.12.18"

lazy val root = (project in file("."))

.settings(

name := "WordCount",

version := "0.1.0-SNAPSHOT",

scalaVersion := scalaVersionUsed,

libraryDependencies ++= Seq(

"org.apache.spark" %% "spark-core" % "3.5.0",

"org.apache.spark" %% "spark-sql" % "3.5.0"

)

)

sample.txt

Apache Spark is fast.

Spark processes big data fast.

This is a simple word count example using Spark

import org.apache.spark.sql.SparkSession

object WordCount {

def main(args: Array[String]): Unit = {

// Create a SparkSession

val spark = SparkSession.builder()

.appName("WordCountExample")

.master("local[\*]") // Local mode with all cores

.getOrCreate()

// Read a text file (place your file in the project root)

val textFile = spark.sparkContext.textFile("sample.txt")

// Split lines into words and count frequency

val wordCounts = textFile

.flatMap(line => line.split("\\W+"))

.filter(\_.nonEmpty)

.map(word => word.toLowerCase)

.map(word => (word, 1))

.reduceByKey(\_ + \_)

// Collect and display results

println("Word Frequencies:")

wordCounts.collect().foreach { case (word, count) =>

println(s"$word: $count")

}

spark.stop()

}

}

**P10**

Filter rows in a CSV file using Spark DataFrames where a numeric column exceeds a threshold.

val scalaVersionUsed = "2.12.18"

lazy val root = (project in file("."))

.settings(

name := "WordCount",

version := "0.1.0-SNAPSHOT",

scalaVersion := scalaVersionUsed,

libraryDependencies ++= Seq(

"org.apache.spark" %% "spark-core" % "3.5.0",

"org.apache.spark" %% "spark-sql" % "3.5.0"

)

)

people.csv

name, age, score

Alice, 25,88.5

Bob,22,91.0

Charlie, 24,79.0

David, 23,95.5

Eva,26,84.0

Frank,27,89.5

Grace, 22,92.0

import org.apache.spark.sql.SparkSession

import org.apache.spark.sql.functions.\_

object FilterHighScores {

def main(args: Array[String]): Unit = {

// Create SparkSession

val spark = SparkSession.builder()

.appName("CSV Filter Example")

.master("local[\*]")

.getOrCreate()

// Read the CSV into a DataFrame

val df = spark.read

.option("header", "true")

.option("inferSchema", "true") // Automatically detects column types

.csv("people.csv")

// Filter rows where score > 90

val highScores = df.filter(col("score") > 90)

// Show the filtered results

println("People with Score > 90:")

highScores.show()

spark.stop()

}

}