Practical 2

Aim: Calculate mean, median, and mode of a list of numbers. Implement basic statistical calculations using Scala collections.

// File: Statistics.scala object Statistics {

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

val numbers = List(1, 2, 2, 3, 4, 5, 5, 5, 6, 7)

val mean = numbers.sum.toDouble / numbers.length val sorted = numbers.sorted

val median = if (numbers.length % 2 == 0)

(sorted(numbers.length / 2 - 1) + sorted(numbers.length / 2)).toDouble / 2 else

sorted(numbers.length / 2)

val grouped = numbers.groupBy(identity).mapValues(\_.size) val maxFreq = grouped.values.max

val mode = grouped.filter(\_.\_2 == maxFreq).keys println(s"List: $numbers")

println(f"Mean: $mean%.2f") println(s"Median: $median") println(s"Mode: ${mode.mkString(", ")}")

}

}

Practical 3

Aim: Generate a random dataset of 10 numbers and calculate its variance and standard deviation.

// File: RandomStats.scala import scala.util.Random import scala.math.sqrt object RandomStats {

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

val randomData = List.fill(10)(Random.nextInt(100))

val mean = randomData.sum.toDouble / randomData.length

val variance = randomData.map(x => math.pow(x - mean, 2)).sum / randomData.length val stdDev = sqrt(variance

println(s"Random Data: $randomData") println(f"Mean: $mean%.2f") println(f"Variance: $variance%.2f") println(f"Standard Deviation: $stdDev%.2f")

}

}

Practical 4

Aim:- Create a dense vector using Breeze and calculate its sum, mean, and dot product with another vector.

// vector.scala

import breeze.linalg.\_ import breeze.stats.\_ object Main extends App {

val v1 = DenseVector(10.0, 20.0, 30.0)

val v2 = DenseVector(1.0, 2.0, 3.0) println(s"Dense Vector v1: $v1") println(s"Dense Vector v2: $v2") println(s"Dot product: ${v1 dot v2}") println(s"Sum: ${sum(v1)}")

println(s"Mean: ${mean(v1)}")

}

Practical 5B

Aim: Generate a random matrix using Breeze and compute its transpose and determinant

//Build.sbt

name := "MatrixOperations" version := "0.1"

scalaVersion := "2.13.12"

libraryDependencies += "org.scalanlp" %% "breeze" % "2.1.0"

// Operation.scala

// 1. Import required Breeze libraries

import breeze.linalg.\_ // For matrix operations: DenseMatrix, transpose, determinant import breeze.stats.distributions.\_ // For random matrix generation

// 2. Define the program object

object MatrixTransposeDeterminant extends App {

// 3. Generate a random 3x3 matrix with values between 0 and 1 val randomMatrix = DenseMatrix.rand[Double](3, 3) println("Random 3x3 Matrix:")

println(randomMatrix)

// 4. Compute and print the transpose val transposedMatrix = randomMatrix.t println("\nTransposed Matrix:") println(transposedMatrix)

// 5. Compute and print the determinant val determinant = det(randomMatrix)

println(f"\nDeterminant of the matrix: $determinant%.4f")

}

Practical 7

Aim: - Write a program to perform element-wise addition, subtraction, multiplication, and division of two Breeze matrices

Build.sbt

name := "BreezeMatrixDemo" version := "0.1"

scalaVersion := "2.13.13"

libraryDependencies += "org.scalanlp" %% "breeze" % "2.1.0"

MatrixOperations.scala Code

import breeze.linalg.\_

object MatrixOperations extends App {

val mat1 = DenseMatrix((1.0, 2.0), (3.0, 4.0))

val mat2 = DenseMatrix((5.0, 6.0), (7.0, 8.0)) val add = mat1 + mat2

val sub = mat1 - mat2

val mul = mat1.mapPairs { case ((i, j), v) => v \* mat2(i, j) } val div = mat1.mapPairs { case ((i, j), v) => v / mat2(i, j) } println("Matrix 1:\n" + mat1)

println("Matrix 2:\n" + mat2) println("Addition:\n" + add) println("Subtraction:\n" + sub) println("Multiplication:\n" + mul) println("Division:\n" + div)

}

Practical 10

Aim: Filter rows in a dataset where a specific column value exceeds a threshold.

SBT code

name := "FilterRowsProject" version := "0.1"

scalaVersion := "2.13.12"

Code

object FilterRows extends App { val data = List(

("Alice", 75),

("Bob", 45),

("Charlie", 90),

("David", 60)

)

val threshold = 70

val filtered = data.filter { case (\_, score) => score > threshold } println("Students with scores above " + threshold + ":") filtered.foreach { case (name, score) => println(s"$name: $score") }

}

Practical 11

Aim: Write a program to tokenize and count the frequency of words in a text file.

SBT code

name := "WordCountProject" version := "0.1"

scalaVersion := "2.13.12"

Code

import scala.io.Source

object WordCountMini extends App {

val text = Source.fromFile("sample.txt").mkString.toLowerCase

val words = text.replaceAll("[^a-zA-Z ]", " ").split(" ").filter(\_.nonEmpty) val counts = words.groupBy(w => w).mapValues(\_.length) counts.foreach { case (word, count) => println(s"$word: $count") }

}

Practical 12

Aim: Create a scatter plot of random data using Breeze-viz. Label the axes and customize the color of points.

Build.sbt

scalaVersion := "2.12.15" libraryDependencies ++= Seq( "org.scalanlp" %% "breeze" % "2.1.0", "org.scalanlp" %% "breeze-viz" % "2.1.0"

)

Code

import breeze.linalg.\_ import breeze.plot.\_ object ScatterPlot {

def main(args: Array[String]): Unit = { val x = DenseVector.rand(10)

val y = DenseVector.rand(10) val fig = Figure()

val plt = fig.subplot(0)

// First 5 points in blue

plt += plot(x(0 until 5), y(0 until 5), '.', colorcode = "blue")

// Last 5 points in red

plt += plot(x(5 until 10), y(5 until 10), '.', colorcode = "red") plt.xlabel = "X Axis"

plt.ylabel = "Y Axis" fig.saveas("scatter\_plot.png")

println("Scatter plot saved as scatter\_plot.png")

}

}

Practical 13

Aim: Generate a histogram of a dataset using Breeze-viz. Experiment with different bin sizes.

Build.sbt

scalaVersion := "2.12.15" libraryDependencies ++= Seq( "org.scalanlp" %% "breeze" % "2.1.0", "org.scalanlp" %% "breeze-viz" % "2.1.0"

)

Code

import breeze.linalg.\_ import breeze.plot.\_

object HistogramExample {

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

val data = DenseVector.rand(100) // 100 random numbers val fig = Figure()

val plt = fig.subplot(0)

plt += hist(data, 10) // 10 bins plt.title = "Histogram with 10 bins" plt.xlabel = "Value"

plt.ylabel = "Frequency" fig.saveas("histogram.png") // Save to file println("Histogram saved as histogram.png")

}

}

Practical 14

Aim: Plot a line graph for a dataset showing a trend over time.

SBT code

name := "BreezeLinePlot" version := "0.1"

scalaVersion := "2.13.12" libraryDependencies ++= Seq( "org.scalanlp" %% "breeze" % "2.1.0", "org.scalanlp" %% "breeze-viz" % "2.1.0"

)

Code

import breeze.linalg.\_ import breeze.plot.\_

object LinePlotExample extends App { val fig = Figure()

val plt = fig.subplot(0)

val x = linspace(0.0, 10.0, 50) // X-axis: 50 points from 0 to 10 val y = x.map(i => math.sin(i)) // Y-axis: sine values of x

plt += plot(x, y, colorcode = "b") plt.xlabel = "X-axis”

plt.ylabel = "Y-axis"

plt.title = "Simple Line Plot"

// Save the plot as PNG fig.saveas("line\_plot.png")

// Optionally show on screen fig.refresh()

}

Practical 15

Aim: Find the correlation between two lists of numbers. Implement the formula for Pearson correlation coefficient.

Sbt code

name := "PearsonCorrelation" version := "1.0"

scalaVersion := "2.13.12"

Code

import scala.math.\_

object PearsonCorrelation extends App { val x = Seq(1.0, 2.0, 3.0, 4.0, 5.0)

val y = Seq(2.0, 4.0, 5.0, 4.0, 5.0)

val n = x.size

val sumX = x.sum val sumY = y.sum

val sumXY = x.zip(y).map { case (xi, yi) => xi \* yi }.sum val sumX2 = x.map(xi => xi \* xi).sum

val sumY2 = y.map(yi => yi \* yi).sum val r = (n \* sumXY - sumX \* sumY) /

sqrt((n \* sumX2 - pow(sumX, 2)) \* (n \* sumY2 - pow(sumY, 2))) println(f"Pearson correlation: $r%.4f")

}

Practical 16

Aim: Calculate the moving average of a time series data using Scala collections.

Sbt code

name := "MovingAverage" version := "1.0"

scalaVersion := "2.13.12"

Code

object MovingAverage {

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

val data = List(10.0, 20.0, 30.0, 40.0, 50.0) // Example time series val windowSize = 3

val movingAvg = data.sliding(windowSize).map(window => window.sum / window.size).toList println(s"Moving Average: $movingAvg")

}

}

Practical 17

Aim: Write a program to compute frequency distribution and cumulative frequency of a dataset.

Sbt code

name := "FrequencyDistribution" version := "0.1"

scalaVersion := "2.13.12"

Code

object FrequencyDistribution {

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

// Example dataset

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

// Frequency distribution: group numbers and count occurrences

val freq = data.groupBy(x => x).map(x => (x.\_1, x.\_2.size)).toSeq.sortBy(\_.\_1)

// Cumulative frequency calculation var cumulative = 0

println("Value Frequency Cumulative") for ((value, count) <- freq) { cumulative = cumulative + count println(s"$value $count $cumulative")

}

}

}

Practical 18

Aim: Sort a dataset by a specific column and extract the top 5 rows.

SBT code

name := "Top5Students" version := "0.1"

scalaVersion := "2.13.12"

Scala code

object Top5Students {

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

// Example dataset: (Name, Marks) val students = List(

("Alice", 85),

("Bob", 92),

("Charlie", 78),

("David", 88),

("Eva", 95),

("Frank", 67),

("Grace", 90)

)

// Sort students by marks in descending order val sorted = students.sortBy(\_.\_2).reverse

// Take top 5

val top5 = sorted.take(5)

// Print results

println("Top 5 Students by Marks:") top5.foreach { case (name, marks) => println(s"$name -> $marks")

}

}

}

Practical 19

Aim: Combine two plots (e.g., scatter and line plot) in a single visualization using Breeze-viz.

SBT code libraryDependencies ++= Seq(

"org.scalanlp" %% "breeze" % "2.1.0", "org.scalanlp" %% "breeze-viz" % "2.1.0"

)

Code

import breeze.linalg.\_ import breeze.plot.\_ object SavePlot {

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

val fig = Figure()

val plt = fig.subplot(0)

val x = linspace(0, 10, 50)

plt += plot(x, x.map(math.sin), colorcode = "b")

plt += scatter(x, x.map(math.cos), \_ => 0.2, \_ => java.awt.Color.RED)

// Save the image as PNG in the folder fig.saveas("combined\_plot.png")

}

}

Practical 20

Aim: Compute the Euclidean distance between two Breeze vectors. Use it for nearest neighbor classification.

SBT code

name := "BreezeNearestNeighbor" version := "1.0"

scalaVersion := "2.13.12" libraryDependencies ++= Seq( "org.scalanlp" %% "breeze" % "2.1.0",

"org.scalanlp" %% "breeze-natives" % "2.1.0" // optional for faster performance

)

Code

import breeze.linalg.\_ object NearestNeighbor {

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

(DenseVector(1.0, 2.0), "A"),

(DenseVector(2.0, 3.0), "A"),

(DenseVector(3.0, 3.0), "B"),

(DenseVector(6.0, 5.0), "B")

)

val newPoint = DenseVector(2.5, 2.7)

val nearest = data.minBy(p => norm(p.\_1 - newPoint)) println(s"Predicted Class: ${nearest.\_2}")

}

}

Practical 21

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

Step 1: Find the path Cd Desktop

Cd Spark

Spark-shell.cmd

Step 2: Make input.txt using a notepad Apache Spark is fast

Spark is powerful Spark runs everywhere

Step 3: Apache spark Code

val textFile = sc.textFile("input.txt")

val words = textFile.flatMap(line => line.split(" ")) val wordPairs = words.map(word => (word, 1)) val counts = wordPairs.reduceByKey(\_ + \_) counts.collect().foreach(println)

Practical 22

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

Step 1: Find the path Cd Desktop

Cd Spark

Spark-shell.cmd

Step 2: Make data.csv using excel (comma delimited)

Step 3. Data.csv name age salary Alice 25 50000

Bob 35 60000

Charlie 28 45000

David 40 70000 Apache code

val df = spark.read.option("header","true").option("inferSchema","true") .csv("data.csv") df.printSchema()

val filteredDF = df.filter(df("salary") > 55000) filteredDF.show()

val df = spark.read.option("header", "true").option("inferSchema", "true").csv("data.csv") df.printSchema()

val filteredDF = df.filter(df("salary") > 55000) filteredDF.show()