Practical 1. Write a simple Scala program that prints a welcome message for data scientists.

```
object WelcomeDataScientists {
def main(args: Array[String]): Unit = {
println(" 👩‍🔬 👨‍🔬 Welcome to the world of Data Science with Scala! 🚀 ")
println("Let's explore data, build models, and uncover insights together.")
}
}
```

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

```
object BasicStats {
  def mean(xs: Seq[Double]): Double = xs.sum / xs.size

  def median(xs: Seq[Double]): Double = {
    val sorted = xs.sorted
    val n = sorted.size
    if (n % 2 == 1)
      sorted(n / 2)
    else
      (sorted(n / 2 - 1) + sorted(n / 2)) / 2.0
  }

  def mode[T](xs: Seq[T]): Seq[T] = {
    val freqs = xs.groupBy(identity).view.mapValues(_.size)
    val maxFreq = freqs.values.max
    freqs.collect { case (v, f) if f == maxFreq => v }.toSeq
  }

  def main(args: Array[String]): Unit = {
    val data = List(4.0, 2.0, 5.0, 2.0, 3.0, 4.0)
    println(s"Data   : $data")
    println(f"Mean   : ${mean(data)}%.2f")
    println(s"Median : ${median(data)}")
    println(s"Mode   : ${mode(data)}")
  }
}
```

Practical 3:Generate a random dataset of 10 numbers and calculate its variance and standard deviation.

```
import scala.util.Random
import scala.math.sqrt

object StatsExample {
  def main(args: Array[String]): Unit = {
    // Generate 10 random integers between 1 and 100
    val data: Array[Double] = Array.fill(10)(Random.nextInt(100) + 1).map(_.toDouble)

    println("Generated Data: " + data.mkString(", "))

    // Mean
    val mean = data.sum / data.length

    // Variance (sample variance or population variance? -> Here we take population variance)
    val variance = data.map(x => math.pow(x - mean, 2)).sum / data.length

    // Standard deviation
    val stdDev = sqrt(variance)

    println(f"Mean: $mean%.2f")
    println(f"Variance: $variance%.2f")
    println(f"Standard Deviation: $stdDev%.2f")
  }
}
```

Practical 4:Create a dense vector using Breeze and calculate its sum, mean, and dot product with another vector.

```
import breeze.linalg._

object VectorStats {
  def main(args: Array[String]): Unit = {
    // Create two dense vectors
    val v1 = DenseVector(1.0, 2.0, 3.0, 4.0, 5.0)
    val v2 = DenseVector(5.0, 4.0, 3.0, 2.0, 1.0)

    // Calculate sum and mean of v1
    val sum = breeze.linalg.sum(v1)
    val mean = sum / v1.length

    // Calculate dot product
    val dotProduct = v1 dot v2

    // Output
    println(s"Vector v1: $v1")
    println(s"Vector v2: $v2")
    println(s"Sum of v1: $sum")
    println(f"Mean of v1: $mean%.2f")
    println(s"Dot Product of v1 and v2: $dotProduct")
  }
}
```

Practical 5 Generate a random matrix using Breeze and compute its transpose and determinant.•

```
import breeze.linalg._
import breeze.stats.distributions.Rand

object BreezeMatrixExample {
  def main(args: Array[String]): Unit = {
    // Generate a 3x3 random matrix with values from uniform(0,1)
    val mat: DenseMatrix[Double] = DenseMatrix.rand(3, 3, Rand.uniform)

    println("Original Matrix:")
    println(mat)

    // Transpose
    val transposed = mat.t
    println("\nTranspose of Matrix:")
    println(transposed)

    // Determinant
    val det = det(mat)
    println(s"\nDeterminant of Matrix: $det")
  }
}
```

Practical No. 6: Slice a Breeze matrix to extract a sub-matrix and calculate its row and column sums.

```
import breeze.linalg._

object BreezeMatrixSlicing {
  def main(args: Array[String]): Unit = {
    // Create a 4x4 matrix
    val mat = DenseMatrix(
      (1.0, 2.0, 3.0, 4.0),
      (5.0, 6.0, 7.0, 8.0),
      (9.0, 10.0, 11.0, 12.0),
      (13.0, 14.0, 15.0, 16.0)
    )

    println("Original Matrix:")
    println(mat)

    // Slice: take a 2x3 sub-matrix (rows 1 to 2, cols 0 to 2)
    val subMat = mat(1 to 2, 0 to 2)
    println("\nSub-Matrix (rows 1-2, cols 0-2):")
    println(subMat)

    // Row sums
    val rowSums = sum(subMat(*, ::))  // sum across columns for each row
    println("\nRow Sums:")
    println(rowSums)

    // Column sums
    val colSums = sum(subMat(::, *))  // sum across rows for each column
    println("\nColumn Sums:")
    println(colSums)
  }
}
```

Practical 7 Write a program to perform element-wise addition, subtraction, multiplication, and division of two Breeze matrices.

```
import breeze.linalg._

object BreezeMatrixOps {
 def main(args: Array[String]): Unit = {
   // Define two 3x3 matrices
   val A = DenseMatrix(
     (1.0, 2.0, 3.0),
     (4.0, 5.0, 6.0),
     (7.0, 8.0, 9.0)
   )

   val B = DenseMatrix(
     (9.0, 8.0, 7.0),
     (6.0, 5.0, 4.0),
     (3.0, 2.0, 1.0)
   )

   println("Matrix A:\n" + A)
   println("\nMatrix B:\n" + B)

   // Element-wise addition
   val add = A + B
   println("\nElement-wise Addition (A + B):\n" + add)

   // Element-wise subtraction
   val sub = A - B
   println("\nElement-wise Subtraction (A - B):\n" + sub)

   // Element-wise multiplication (Hadamard product)
   val mul = A *:* B
   println("\nElement-wise Multiplication (A *:* B):\n" + mul)

   // Element-wise division
   val div = A /:/ B
   println("\nElement-wise Division (A /:/ B):\n" + div)
 }
}
```

Practical 8

```
import com.github.tototoshi.csv._
import breeze.stats._
import breeze.linalg._

import java.io.File

object ReadCSVStats {
 def main(args: Array[String]): Unit = {
   // Path to your CSV file (example: data.csv in project folder)
   val file = new File("C:\\Users\\Chandrashekhar\\IdeaProjects\\Practical8\\data.csv")

   val reader = CSVReader.open(file)

   // Read all rows (excluding header)
   val allRows = reader.allWithHeaders()
   reader.close()

   println("CSV Data:")
   println(allRows.take(5)) // print first 5 rows

   // Convert each column to numeric values
   val headers = allRows.head.keys.toList

   headers.foreach { col =>
     val values = allRows.flatMap(row => row.get(col).flatMap(v => v.toDoubleOption))

     if (values.nonEmpty) {
       val vec = DenseVector(values.toArray)
       println(s"\nStatistics for column: $col")
       println(s"Count: ${values.length}")
       println(s"Min: ${min(vec)}")
       println(s"Max: ${max(vec)}")
       println(s"Mean: ${mean(vec)}")
       println(s"Variance: ${variance(vec)}")
       println(s"Std Dev: ${stddev(vec)}")
     }
   }
 }
}
```

Practical 9 Handle missing values in a dataset. Replace missing values with the column mean.

```scala
import com.github.tototoshi.csv._
import breeze.stats._
import breeze.linalg._
import java.io.File

object HandleMissingValues {
  def main(args: Array[String]): Unit = {
    // Load CSV file
    val reader = CSVReader.open(new File("data.csv"))
    val allRows = reader.allWithHeaders()
    reader.close()

    println("Original Data:")
    allRows.foreach(println)

    if (allRows.nonEmpty) {
      val headers = allRows.head.keys.toList

      headers.foreach { col =>
        // Extract numeric values (ignore blanks and NA)
        val values = allRows.flatMap(row => row.get(col).flatMap(v => v.toDoubleOption))

        if (values.nonEmpty) {
          val colMean = mean(DenseVector(values.toArray))

          // Replace missing with mean
          val filledValues = allRows.map { row =>
            row.get(col) match {
              case Some(v) if v.trim.isEmpty || v.equalsIgnoreCase("NA") =>
                colMean
              case Some(v) if v.toDoubleOption.isDefined =>
                v.toDouble
              case _ =>
                colMean
            }
          }

          val vec = DenseVector(filledValues.toArray)
          println(s"\nStatistics for column: $col (missing handled)")
          println(s"Count: ${filledValues.length}")
          println(s"Min: ${min(vec)}")
          println(s"Max: ${max(vec)}")
          println(s"Mean: ${mean(vec)}")
          println(s"Variance: ${variance(vec)}")
          println(s"Std Dev: ${stddev(vec)}")
        }
      }
    }
  }
}
```

Practical 10:.Filter rows in a dataset where a specific column value exceeds a threshold.

```
import com.github.tototoshi.csv._
import java.io.File

object FilterRows {
  def main(args: Array[String]): Unit = {
    // Load CSV file
    val reader = CSVReader.open(new File("data.csv"))
    val allRows = reader.allWithHeaders()
    reader.close()

    println("Original Data:")
    allRows.foreach(println)

    // Example: Filter rows where Marks > 80
    val threshold = 80
    val filteredRows = allRows.filter { row =>
      row.get("Marks").flatMap(_.toDoubleOption).exists(_ > threshold)
    }

    println(s"\nFiltered Rows where Marks > $threshold:")
    filteredRows.foreach(println)
  }
}
```

On Scastie

```
//> using scala "2.13.12"
//> using lib "com.github.tototoshi::scala-csv:1.3.10"

import com.github.tototoshi.csv._
import java.io.StringReader

object FilterRows extends App {
  // Example CSV data (embedded as a string)
  val csvData =
    """Name,Age,Marks,Height
      |Alice,23,88,165
      |Bob,25,72,170
      |Charlie,22,95,180
      |David,24,68,175
      |Eve,26,85,160
      |""".stripMargin

  // Read CSV from string
  val reader = CSVReader.open(new StringReader(csvData))
  val allRows = reader.allWithHeaders()
  reader.close()

  println("Original Data:")
  allRows.foreach(println)

  // Filter condition: Marks > 80
  val threshold = 80
  val filteredRows = allRows.filter { row =>
    row.get("Marks").flatMap(_.toDoubleOption).exists(_ > threshold)
  }

  println(s"\nFiltered Rows where Marks > $threshold:")
  filteredRows.foreach(println)
}
```

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

```
import scala.io.Source

object WordFrequencyCounter {
 def main(args: Array[String]): Unit = {
   // Change the path to your file
   val filePath = "C:\\Users\\Chandrashekhar\\IdeaProjects\\Practical8\\data.txt"

   try {
     // Read the file
     val text = Source.fromFile(filePath).getLines().mkString(" ")

     // Tokenize (split by non-word characters)
     val tokens = text.toLowerCase.split("\\W+").filter(_.nonEmpty)

     // Count word frequencies
     val wordCounts = tokens.groupBy(identity).mapValues(_.length)

     // Print results
     println("Word Frequencies:")
     wordCounts.toSeq.sortBy(-_._2).foreach { case (word, count) =>
       println(s"$word -> $count")
     }

   } catch {
     case e: Exception =>
       println(s"Error reading file: ${e.getMessage}")
```

```
    }
  }
}
```

Practical 12.Create a scatter plot of random data using Breeze-viz. Label the axes and customize the color of points.

```
ThisBuild / scalaVersion := "2.13.12"


lazy val root = (project in file("."))
.settings(
name := "ScatterPlotBreeze",
version := "0.1.0-SNAPSHOT",
libraryDependencies ++= Seq(
"org.scalanlp" %% "breeze" % "2.1.0",
"org.scalanlp" %% "breeze-viz" % "2.1.0"
)
)
```

Practical 13Create a scatter plot of random data using Breeze-viz. Label the axes and customize the color of points.

```
import breeze.linalg._
import breeze.plot._

object ScatterPlotExample {
 def main(args: Array[String]): Unit = {
   // Generate 100 random values between 0 and 10
   val x = DenseVector.rand(100) * 10.0
   val y = DenseVector.rand(100) * 10.0

   val fig = Figure("Scatter Plot Example")
   val plt = fig.subplot(0)

   // Scatter plot with blue points
   plt += plot(x, y, '.', colorcode = "blue")

   // Labels and title
   plt.xlabel = "X Axis"
   plt.ylabel = "Y Axis"
   plt.title = "Random Data Scatter Plot"
   fig.refresh()
 }
}
```

Practical 13 :Plot a line graph for a dataset showing a trend over time.

```
ThisBuild / scalaVersion := "2.13.12"

lazy val root = (project in file("."))
 .settings(
   name := "ScatterPlotBreeze",
   version := "0.1.0-SNAPSHOT",
   libraryDependencies ++= Seq(
     "org.scalanlp" %% "breeze" % "2.1.0",
     "org.scalanlp" %% "breeze-viz" % "2.1.0"
   )
 )
```

Practical 13 :Plot a line graph for a dataset showing a trend over time.

```
import breeze.linalg._
import breeze.plot._

object LinePlotExample {
 def main(args: Array[String]): Unit = {
   // Simulated dataset: time in months (1 to 12)
   val time = DenseVector.rangeD(1, 13, 1)  // 1 to 12
   val sales = DenseVector(10.0, 12.5, 13.0, 15.0, 18.0, 20.0,
     19.0, 22.5, 25.0, 27.0, 30.0, 32.0)

   // Create figure
   val fig = Figure("Line Graph Example")
   val plt = fig.subplot(0)

   // Line plot with green color
   plt += plot(time, sales, colorcode = "green")

   // Labels and title
   plt.xlabel = "Time (Months)"
   plt.ylabel = "Sales (in Units)"
   plt.title = "Sales Trend Over Time"

   fig.refresh()
 }
}
```

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

```
ThisBuild / scalaVersion := "2.13.12"

lazy val root = (project in file("."))
 .settings(
   name := "ScatterPlotBreeze",
   version := "0.1.0-SNAPSHOT",
   libraryDependencies ++= Seq(
     "org.scalanlp" %% "breeze" % "2.1.0",
     "org.scalanlp" %% "breeze-viz" % "2.1.0"
   )
 )
```

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

```
import breeze.linalg._
import breeze.plot._

object CombinedPlotExample {
 def main(args: Array[String]): Unit = {
   // Generate data
   val x = DenseVector.rangeD(0.0, 10.0, 0.5)   // X values
   val yLine = x.map(v => 2.0 * v + 1.0)        // Line: y = 2x + 1
   val yScatter = yLine + DenseVector.rand(x.length) * 5.0 // Scatter: noisy data

   // Create figure
   val fig = Figure("Combined Scatter and Line Plot")
   val plt = fig.subplot(0)

   // Line plot (red)
   plt += plot(x, yLine, colorcode = "red")

   // Scatter plot (blue)
   plt += plot(x, yScatter, '.', colorcode = "blue")

   // Labels and title
   plt.xlabel = "X Axis"
   plt.ylabel = "Y Axis"
   plt.title = "Line + Scatter Combined"

   fig.refresh()
 }
}
```