# R PROGRAM

**1.** **a.** Create three different variables, one that is numeric type and other two are vector of characters. Use these to create data frame of student. (USN, Name, Marks)

**b.** Add a new numeric data column to the existing data frame (Age). Provide summary of the data

**c.** Display the list of students whose Age is less than 20 and Marks greater than 25.

```
n <- as.integer(readline(prompt = "Enter no. of students"))

name <- vector(mode = "character", length = n)
usn <- vector(mode = "character", length = n)
marks <- vector(mode = "numeric", length = n)

print("Enter names")

for(i in 1:n)
  name[i] = as.character(readline())

print("Enter usn")

for(i in 1:n)
  usn[i] = as.character(readline())

print("Enter marks")

for(i in 1:n)
  marks[i] = as.numeric(readline())

student <- data.frame(usn,name,marks)

print(student)
```

```
age <- vector(mode = "integer", length = n)

print("Enter ages")

for(i in 1:n)
  age[i] = as.numeric(readline())

student <- cbind(student, age)

print(student)

for(i in 1:n)
if(student[i,3] > 25)
if(student[i,4] < 20)
print(student[i,])
```

**2.** Write a program to create the csv file for storing Employee data, containing the fields

(Emp ID, Emp Name, DOJ, Dept, Desig.)

a. Read the suitable number of employee details from the user.

b. Create a data frame of Employee.
c. Store the data frame in the csv file.
d. Read the data from csv and display the contents.
e. Append a new row into the csv file.

```
n <- as.integer(readline(prompt = "enter no of employees"))

empid <- vector(mode = "character", length = n)
empname <- vector(mode = "character", length = n)
doj <- vector(mode = "character", length = n)
dept <- vector(mode = "character", length = n)
desig <- vector(mode = "character", length = n)

print("enter empid")

for(i in 1:n)
  empid[i] = as.character(readline())

print("enter empname")

for(i in 1:n)
  empname[i] = as.character(readline())

print("enter doj")

for(i in 1:n)
  doj[i] = as.character(readline())
```

```
print("enter dept")
for(i in 1:n)
  dept[i] = as.character(readline())
print("enter desig")
for(i in 1:n)
  desig[i] = as.character(readline())
employee <- data.frame(empid,empname,doj,dept,desig)
print(employee)
write.csv(employee,"emp.csv")
read.csv("emp.csv")
row <- data.frame("031","Zara","21-03-2020","HR","HR")
write.table(row, "emp.csv", append = TRUE, sep = ",", row.names = TRUE, col.names = FALSE, quote = FALSE)
read.csv("emp.csv")
```

**3.** Exploring Dataset

**a.** List the data set available in your system using suitable command.

**b.** Select "MT cars" data set, find and display the number of rows and columns in that data set.

**c.** Find are there more automatic (0) *or* manual (1) transmission-type cars in the dataset? *Hint: 9th column indicates the transmission type*

**d.** Get a scatter plot of 'hp' vs 'weight'.

**e.** Change 'am', 'cyl' and 'vs' to *integer* and store the new dataset as 'newmtc'.

f. Extract the cases where cylinder is less than 5

```
data()
```

```
head(mtcars)
rownum<-nrow(mtcars)
rownum
colnum<-ncol(mtcars)
colnum
x<-data.frame(mtcars)
x
automatic<-0
manual<-0
for(i in 1:rownum)
  ifelse(x[i,9]==1, automatic<-automatic+1, manual<-manual+1)
automatic
manual
if(automatic>manual)
  print("More automatic")
print("more manual")
with(mtcars,scatter.smooth(hp,wt))
am_new<-as.integer(x$am)
am_new
cyl_new<-as.integer(x$cyl)
cyl_new
vs_new<-as.integer(x$vs)
vs_new
newmtc<-data.frame(am_new,cyl_new,vs_new)
newmtc
subset(mtcars,cyl<5)
```

**4.** Consider “Airquality” dataset

**a.** Display the dimension ofthe dataset

**b.** Display the class of each fields in the data set

**c.** Test the missing values

**d.** Recode the missing values, as mean of the column values

**e.** Exclude the missing values

```
df<-airquality
dim(df)
sapply(df,class)
print("the missing values are as follows")
Xcolnames<-colnames(df)
x<-colSums(is.na(df))
print(x)
which(is.na(df))
sum(is.na(df))
df
df1<-as.data.frame(df)
for(i in 1:4)
  df1[,i]<-ifelse(is.na(df[,i]),mean(df[,i],na.rm=TRUE),df[,i])
df1
df2<-na.omit(df)
print(df2)
```

# Scala Program

**1.** Write a program that reads words from a file. Use a mutable map to count how often each word appears.

```
import scala.io.Source

object wordcount{

        def main(args:Array[String]){

        if(args.length!=1){

        System.err.println("Error")

        System.exit(1)

        }

var filename=args(0)

val wordC=scala.collection.mutable.Map[String,Int]()

for(line<-Source.fromFile(filename).getLines)

for(word<-line.split(" "))

wordC(word)=if(wordC.contains(word)) wordC(word)+1 else 1

println(wordC)

for((k,v)<-wordC)

printf("word %s occurs %d times\n",k,v)

}

}
```

**2.** Write a function minmax (values: Array[Int]) that returns a pair containing the smallest and largest values in the array.

```
import scala.io.Source

import scala.io.StdIn

import scala.collection.mutable.ArrayBuffer


object MinMax{
	def main(args: Array[String]):Unit = {
		var numArray = new ArrayBuffer[Int]()
		println("Enter no. of elements")
		val n = scala.io.StdIn.readInt()
		println("Enter elements")
		for(i<- 1 to n)
			numArray+=scala.io.StdIn.readInt()
		println(numArray)
		val t=minmax(numArray)
		println("Max is ",t._1)
		println("Min is ",t._2)
	}
	def minmax(numArray: ArrayBuffer[Int]):(Int,Int)={
		var min:Int=999
		var max:Int=(-999)
		for(value <- numArray){
			if(value>max)
				max = value
			if(value<min)
				min = value
		}
		(max,min)
```

```
        }
}
```

**3.** Write the menu driven program to implement quick sort algorithm using imperative style and functional style.

```
/** Quick sort, functional style */
object FunctionalQuickSort {
  def sort(a: List[Int]): List[Int] = {
    if (a.length < 2)
      a
    else {
      val pivot = a(a.length / 2)
      sort(a.filter(_ < pivot)) :::
          a.filter(_ == pivot) :::
          sort(a.filter(_ > pivot))
    }
  }

  def main(args: Array[String]) {
    val xs = List(6, 2, 8, 5, 1)
    println(xs)
    println(sort(xs))
  }
}
```

```
object ImperativeQuickSort {
def sort(a: Array[Int]) {

  def swap(i: Int, j: Int) {
    val t = a(i); a(i) = a(j); a(j) = t
  }

  def sort1(l: Int, r: Int) {
    val pivot = a((l + r) / 2)
    var i = l
    var j = r
    while (i <= j) {
      while (a(i) < pivot) i += 1
      while (a(j) > pivot) j -= 1
      if (i <= j) {
        swap(i, j)
        i += 1
        j -= 1
      }
    }
    if (l < j) sort1(l, j)
    if (j < r) sort1(i, r)
  }
```

```
 if (a.length > 0)
     sort1(0, a.length - 1)
  }

 def println(ar: Array[Int]) {
    def print1 = {
      def iter(i: Int): String =
        ar(i) + (if (i < ar.length-1) "," + iter(i+1) else "")
      if (ar.length == 0) "" else iter(0)
    }
    Console.println("[" + print1 + "]")
  }

  def main(args: Array[String]) {
    var ar = Array(6,5,2,1,8);
    println(ar)
    sort(ar)
    println(ar)
  }

}
```

**4.** Write the program to illustrate the use of pattern matching in scala, for the following

Matching on case classes. Define two case classes as below:
**abstractclassNotification**
**caseclassEmail**(sender: **String**, title: **String**, body: **String**) **extendsNotification**
**caseclassSMS**(caller: **String**, message: **String**) **extendsNotification**
Define a function showNotification which takes as a parameter the abstract type Notification and matches on the type of Notification (i.e. it figures out whether it's an Email or SMS).
In the case it's an Email(email, title, _) return the string: s"You got an email from $email with title: $title"
In the case it's an SMS return the String: s"You got an SMS from $number! Message: $message"

```
abstract class Notification

case class SMS(mobile: String, msg: String) extends Notification

case class Email(emailAddr: String, subject: String, body: String) extends Notification

object Pattern{


        def showNotification(notification: Notification): String = {
                notification match{
                        case Email(emailAddr, subject, _) =>
                                s"You got an email from $emailAddr with subject: $subject"
                        case SMS(number, message) =>
                                s"You got a SMS from $number! Message: $message"
                }
        }
        def main(args: Array[String]):Unit = {
                val someSMS = SMS("12345","Are you there?")
                val someEmail = Email("xyz@nmit.ac.in","Big data course syllabus", "Intro to
Big data, NoSQL Databases, Spark RDDS, SQL, Streaming")


                println(showNotification(someSMS))
                println(showNotification(someEmail))
        }
}
```

# Spark Programs

**1.** WordCount: Here the goal is to count how many times each word appears in a file and write out a list ofwords whose count is strictly greater than 4.

Use the file log.txt accompanying this assignment to count the words.Save the wordcounts in text form in the "wordcountsDir" using the saveAsTextFile RDD method.Examine the contents ofthe above directory, andthe contents ofthe files ofthe directory.

```
 import org.apache.spark.SparkContext

import org.apache.spark.SparkConf

import org.apache.spark.rdd.RDD



object wordcount{

        def main(args:Array[String]){



        val pathToFile="log.txt"

        val conf = new SparkConf().setAppName("WordCount").setMaster("local[*]")

        val sc = new SparkContext(conf)

        val wordRDD = sc.textFile(pathToFile).flatMap(_.split(" "))

        val wordCountInitRdd = wordRDD.map(word=>(word,1))

        val wordCountRdd = wordCountInitRdd.reduceByKey((v1,v2)=>v1+v2)

        val highfreqwords = wordCountRdd.filter(x=>x._2>0)

        highfreqwords.saveAsTextFile("wordcountsDir")

        }

        }
```

**2.** Tweet Mining: A dataset with the 8198 reduced tweets, reduced-tweets.json will be provided.The data contains reduced tweets as in the sample below:

{"id":"572692378957430785",

"user":"Srkian_nishu ",
"text":"@always_nidhi @YouTube no idnt understand bti loved of this mve is rocking",
"place":"Orissa",
"country":"India"}

Write A function to parse the tweets into an RDD and print the top 10 tweeters.

```
 import org.apache.spark.{SparkContext, SparkConf}

import org.apache.spark.rdd._

object tweetmining {

val conf = new SparkConf()

   .setAppName("User mining")

   .setMaster("local[*]")


  val sc = new SparkContext(conf)

  var pathToFile = ""


  def main(args: Array[String]) {

    if (args.length != 1) {

      println()

      println("Dude, I need exactly one argument.")

      println("But you have given me " + args.length +".")

      println("The argument should be path to json file containing a bunch of tweets. esired.")

      System.exit(1)

    }


    pathToFile = args(0)
```

```
val tweets =
      sc.textFile(pathToFile).mapPartitions(TweetUtils.parseFromJson(_))
val tweetsByUser = tweets.map(x => (x.user, x)).groupByKey()
 val numTweetsByUser = tweetsByUser.map(x => (x._1, x._2.size))
 val sortedUsersByNumTweets = numTweetsByUser.sortBy(_._2, ascending=false)
sortedUsersByNumTweets.take(10).foreach(println)


  }
}
import com.google.gson._


object TweetUtils {
        case class Tweet (
                id : String,
                user : String,
                userName : String,
                text : String,
                place : String,
                country : String,
                lang : String
                )
        def parseFromJson(lines:Iterator[String]):Iterator[Tweet] = {
                val gson = new Gson
                lines.map(line => gson.fromJson(line, classOf[Tweet]))
        }
}
```