Introduction to R

Dr. PSR Murty
Department of Information Technology,
Vishnu Institute of Technology,
Bhimavaram.

Why R?

R has a number of virtues:

- It is a public-domain implementation of the widely used S statistical language, and the R/S platform is a de facto standard among professional statisticians.
- It is comparable, and often superior, in power to commercial products in most of the significant senses - variety of operations available, programmability, graphics, and so on.
- In addition to providing statistical operations, R is a general-purpose programming language, so you can use it to automate analyses and create new functions that extend the existing language features.

Why R?

- It is available for the Windows, Mac, and Linux operating systems.
- It incorporates features found in object-oriented and functional programming languages.
- The system saves data sets between sessions, so you don’t need to reload them each time. It saves your command history too.
- Because R is open source software, it’s easy to get help from the user community. Also, a lot of new functions are contributed by users, many of whom are prominent statisticians.

How to run R?

- R operates in two modes, interactive and batch mode.
- **Interactive Mode**: Interactive session prompts the user for input as data or commands.
- **Batch Mode**: when we want to run our computer program or a series of programs with out manual intervention then we use batch mode.
  - R CMD BATCH options infile outfile

R Sessions

- To start an R session, type 'R' from the command line in windows or linux OS. For example, from shell prompt '$' in linux, type $ R.
- This opens the R session and you can see the '>' prompt of R.
- **Working with R Session**: Once we are inside R session, we can directly execute R commands by typing them line by line. Pressing the enter key terminates the R command and brings back '>' R prompt.

R Sessions

- **Exiting R Session**: To exit the R session type quit() in the R prompt.
- **Saving R Session**: when you type quit() in R session, it asks weather to save the current session. You can choose ‘y’ to save the current session, i.e., all the commands, variables and objects we created will be save and available for the next session.

Basic Math

- Basic Math:- R is a powerful tool for all mathematical calculations, data manipulation and scientific computations. R can certainly be used to do basic math.
- R follows the basic order of operations: Parenthesis, Exponents, Multiplication, Division, Addition and Subtraction (PEMDAS). This means the operations inside parenthesis take priority over other operations. Next on the priority list is exponentiation. After that multiplication and division are performed, followed by addition and subtraction.

Functions in R

- A function is a piece of code written to carry out a specified task; it can or can not accept arguments or parameters and it can or can not return one or more values.
- A function is formally a part of a computer program that performs some specific action, but is not itself a complete executable program.
- In **R**, a function is an object which has the mode function.
- Function_name<-function(arglist){body}

Variables

- **Variable:** A variable provides us with named storage that our programs can manipulate.
- A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number.
- Typically, compiled languages require that you declare variables; that is, warn the interpreter/compiler of the variables' existence before using them.

Mode

- As with most scripting languages (such as Python and Perl), you do not declare variables in R.
  - Z<-3#this is perfectly legal
- R data types (variable types) are called modes.
- To check the mode of a variable x query it by the call **typeof(x)**.

Data Structures in R

- R has a wide variety of objects for holding data, including scalars, vectors, matrices, data frames, and lists.
- They differ in the type of data they can hold, how they're created, their structural complexity, and the notation used to identify and access individual elements.

Vectors

- The fundamental data structure in R is the vector.
- Vectors are one dimensional arrays that can hold numeric data, character data, or logical data.
- All elements in a vector must have the **same mode**, which can be integer, numeric (floating-point number), character (string), logical (Boolean), and so on.

Vectors

- Properties of vectors are
  - **Recycling** The automatic lengthening of vectors in certain settings
  - **Filtering** The extraction of subsets of vectors
  - **Vectorization** Where functions are applied element-wise to vectors

Vectors

- In R, numbers are actually considered as one-element vectors, and there is really no such thing as a **scalar**.
- vector indices in R begin at 1.
- **Declaration:** To declare a vector with two components,
  - > y <- vector(length=2)
  - > y[1] <- 5
  - > y[2] <- 12
- or y<-c(5,12) #c is concatenate function

Vectors

- **Inserting or deleting elements**: Vectors are stored like arrays in C, contiguously, and thus you cannot insert or delete elements.
- The size of a vector is determined at its creation, so if you wish to add or delete elements,
  - > x <- c(88,5,12,13)
  - > x <- c(x[1:3],168,x[4]) # insert 168 before the 13
  - >x [1] 88 5 12 168 13 you'll need to reassign the vector.

Vectors

- **Obtaining the length of the vector:** You can obtain the length of a vector by using the length() function.
  - > x <- c(1,2,4)
  - > length(x)
  - [1] 3
- To iterate over the vector, we use
  - for (i in 1:length(x))

Vectors

- **Matrices and Arrays as Vectors:** Arrays and matrices (and even lists, in a sense) are actually vectors.
- They merely have extra class attributes.
  - For example, matrices have the number of rows and columns.
- arrays and matrices are vectors, and that means that everything we say about vectors applies to them, too.

Vectors

- **Recycling:** When applying an operation to two vectors that requires them to be the same length, R automatically recycles, or repeats, the shorter one, until it is long enough to match the longer one.
  - > c(1,2,4) + c(6,0,9,20,22)
  - [1] 7 2 13 21 24
  - Warning message: longer object length is not a multiple of shorter object length in: c(1, 2, 4) + c(6, 0, 9, 20, 22)

Vectors

- Common Vector Operations:
  - arithmetic and logical operations
  - vector indexing and
  - ways to create vectors
- **Arithmetic and Logical operators:** R is a functional language.
- Every operator is actually a function.
  - > 2+3
  - [1] 5
  - > "+"(2,3)
  - [1] 5

Vectors

- When we add vectors, + operation will be applied element-wise.
  - > x <- c(1,2,4)
  - > x + c(5,0,-1) #+ operation is done element wise
  - [1] 6 2 3

  - >x* c(5,0,-1) #Multiplication is done element wise
  - [1] 5 0 -4

Vectors

- > x <- c(1,2,4)
- > x / c(5,4,-1)
- [1] 0.2 0.5 -4.0
- > x %% c(5,4,-1)
- [1] 1 2 0

Vectors

- **Vector Indexing:** One of the most important and frequently used operations in R is that of indexing vectors, in which we form a subvector by picking elements of the given vector for specific indices.
- The format is vector1[vector2], with the result that we select those elements of vector1 whose indices are given in vector2.
  - > y <- c(1.2,3.9,0.4,0.12)
  - > y[c(1,3)] # extract elements 1 and 3 of y
  - [1] 1.2 0.4

Vectors

- > y[2:3]
- [1] 3.9 0.4
- > v <- 3:4
- > y[v]
- [1] 0.40 0.12

- Note that duplicates are allowed.
- Negative subscripts mean that we want to exclude the given elements in our output.

Vectors

- **Generating Useful Vectors:** There are a few R operators that are especially useful for creating vectors.
  - with : operator
  - with seq()
  - With rep()
- : operator: It produces a vector consisting of a range of numbers.
  - > 5:8
  - [1] 5 6 7 8
  - > 5:1
  - [1] 5 4 3 2 1

Vectors

- with seq(): A generalization of : is the seq() (or sequence) function, which generates a sequence in arithmetic progression.
  - > seq(from=12,to=30,by=3)
  - [1] 12 15 18 21 24 27 30
- The spacing can be a noninteger value, too.
  - for (i in seq(x))
  - > x <- c(5,12,13)
  - > seq(x)
  - [1] 1 2 3
  - > x <- NULL
  - > seq(x)
  - integer(0)

Vectors

- with rep(): The rep() (or repeat) function allows us to conveniently put the same constant into long vectors.
- The call form is rep(x, times), which creates a vector of times*length(x) elements—that is, times copies of x.
- There is also a named argument each, which interleaves the copies of x.

Vectors

- Vectorization: One of the most effective ways to achieve speed in R code is to use operations that are vectorized, meaning that a function applied to a vector is actually applied individually to each element.
  - > u <- c(5,2,8)
  - > v <- c(1,3,9)
  - >u>v
  - [1] TRUE FALSE FALSE
- if an R function uses vectorized operations, it, too, is vectorized, thus enabling a potential speedup:
  - > w <- function(x) return(x+1) # w() uses +, which is vectorized, so w() is vectorized
  - > w(u)
  - [1] 6 3 9

Vectors

- Filtering: This allows us to extract a vector's elements that satisfy certain conditions.
- Filtering is one of the most common operations in R, as statistical analyses often focus on data that satisfies conditions of interest.
  - > x<-1:10
  - > x[x>3]<-5
  - > x
  - [1] 1 2 3 5 5 5 5 5 5 5

Vectors

- Filtering can also be done with subset() function.
- The difference between using this function and ordinary filtering lies in the manner in which NA values are handled.
  - > x <- c(6,1:3,NA,12)
  - >x
  - [1] 6 1 2 3 NA 12
  - > x[x > 5]
  - [1] 6 NA 12
  - > subset(x, x > 5)
  - [1] 6 12

Vectors

- **Vectorized if-then-else:** R also includes a vectorized version, the ifelse() function. The form is as follows:
  - ifelse(b,u,v)
- where b is a Boolean vector, and u and v are vectors. The return value is itself a vector; element i is u[i] if b[i] is true, or v[i] if b[i] is false.
  - > x <- 1:10
  - > y <- ifelse(x %% 2 == 0,5,12) # %% is the mod operator
  - >y
  - [1] 12 5 12 5 12 5 12 5 12 5

Vectors

- how vectors relate to some other data types in R.
- Explain about recycling, filtering and vectorization.
- How to check weather two vectors are equal?
- Filtering vs subset function.
- What are the functions to create vectors.
- Vectorized ifelse().

Matrices

- A matrix is a vector with two additional attributes: the number of rows and the number of columns.
- Since matrices are vectors, they also have modes, such as numeric and character.
- Matrices are special cases of a more general R type of object: arrays.
- Much of R's power comes from the various operations you can perform on matrices. Analogous to vector subsetting and vectorization.

Matrices

- **Creating :** create a matrix is by using the matrix() function.
  - > y <- matrix(c(1,2,3,4),nrow=2,ncol=2)
  - >y
  - [,1] [,2]
  - [1,] 1 3
  - [2,] 2 4
- The internal storage of a matrix is in column-major order, meaning that first all of column 1 is stored, then all of column 2, and so on.
- Matrix row and column subscripts begin with 1. For example, the upper-left corner of the matrix a is denoted a[1,1].
- Another way to build y is to specify elements individually:

Matrices

- > y <- matrix(nrow=2,ncol=2)
- > y[1,1] <- 1
- > y[2,1] <- 2
- > y[1,2] <- 3
- > y[2,2] <- 4
- >y
- [,1] [,2]
- [1,] 1 3
- [2,] 2 4

Matrices

- Though internal storage of a matrix is in column-major order, you can set the byrow argument in matrix() to true to indicate that the data is coming in row-major order.
  - > m <- matrix(c(1,2,3,4,5,6),nrow=2,byrow=T)
  - >m
  - [,1] [,2] [,3]
  - [1,] 1 2 3
  - [2,] 4 5 6

Matrices

- General Matrix Operations: These include performing linear algebra operations, matrix indexing, and matrix filtering.
- matrix multiplication
  - >y% *% y # mathematical matrix multiplication
  - [,1] [,2]
  - [1,] 7 15
  - [2,]10 22
- matrix scalar multiplication
  - >3 *y # mathematical multiplication of matrix by scalar
  - [,1] [,2]
  - [1,] 3 9
  - [2,] 6 12

Matrices

- matrix addition
  - > y+y # mathematical matrix addition
  - [,1] [,2]
  - [1,] 2 6
  - [2,] 4 8
- **Matrix Indexing:** Similar to vector indexing
  - Mat[1, 4]#refers to the element at row1 and col4 in Mat
  - Mat[2,]#refers to row2 and all cols
  - Mat[,4]#refers to col4 all rows
  - Mat[1:2,]#refers to row1 and row2 and all cols
  - Mat[c(1:3),]#refers to row1,3 and all cols

Matrices

- Filtering Matrices: Filtering can be done with matrices, just as with vectors. You must be careful with the syntax, though.
  - >x
  - x
  - [1,] 1 2
  - [2,] 2 3
  - [3,] 3 4
  - > x[x[,2] >= 3,] #elements >=3 in col 2 will return true
  - x
  - [1,] 2 3
  - [2,] 3 4

Matrices

- >m
- [,1] [,2]
- [1,] 1 4
- [2,] 2 5
- [3,] 3 6
- > m[m[,1]>1&m[,2] > 5,]
- [1] 3 6

- First, the expression m[,1] > 1 compares each element of the first column of m to 1 and returns (FALSE,TRUE,TRUE). The second expression, m[,2] > 5, similarly returns (FALSE,FALSE,TRUE). We then take the logical AND of (FALSE,TRUE,TRUE) and (FALSE,FALSE,TRUE), yielding (FALSE,FALSE,TRUE).

Matrices

- Since matrices are vectors, you can also apply vector operations to them.
  - >m
  - [,1] [,2]
  - [1,] 5 -1
  - [2,] 2 10
  - [3,] 9 11
  - > which(m > 2)
  - [1] 1 3 5 6

Matrices

- **Applying Functions to Matrix Rows and Columns :** One of the most famous and most used features of R is the *apply() family of functions, such as apply(), tapply(), and lapply().
- Here, we'll look at apply(), which instructs R to call a user-specified function on each of the rows or each of the columns of a matrix.
- This is the general form of apply for matrices:
  - apply(m,dimcode,f,fargs)
- where the arguments are as follows:
  - m is the matrix.
  - dimcode is the dimension, equal to 1 if the function applies to rows or 2 for columns.

Matrices

  - f is the function to be applied.
  - fargs is an optional set of arguments to be supplied to f.
- For example, here we apply the R function mean() to each column of a matrix z:
  - >z
  - [,1] [,2]
  - [1,] 1 4
  - [2,] 2 5
  - [3,] 3 6
  - > apply(z,2,mean)
  - [1] 2 5

Data Frame

- A **data frame** is used for storing data tables.
- Technically, a data frame is a list, with the components of that list being equal length vectors.
- Data frame is like a matrix, with two-dimensional rows-and-columns structure.
- It differs from a matrix in that each column may have a different mode.

Data Frame

- **Creating a Data Frames:** We can create a data frame using **data.frame().**
  - Students<-c(“Sanjay”,”Durga Prasanth”,”Hari Chandana”)
  - RollNos<-c(“17pa1a1248”, “17pa1a1221”, “17pa1a1230”)
  - df = data.frame(Students, RollNos)
- The top line of the table, called the **header**, contains the column names.
- Each horizontal line afterward denotes a **data row**, which begins with the name of the row, and then followed by the actual data.
- Each data member of a row is called a **cell**.

Data Frame

- **Accessing Data Frames:** To retrieve data in a cell, we would enter its row and column coordinates in the *single square bracket* "[]" operator. The two coordinates are separated by a comma.
    - > mtcars[1, 2]
    - [1] 6
- We can also use the row and column names instead of the numeric coordinates.
    - > mtcars["Mazda RX4", "cyl"]
    - [1] 6
- We reference a data frame column with the *double square bracket* "[[]]" operator.
  - df[[1]] or df[“students”] or df$students

Data Frame

- The number of data rows in the data frame is given by the nrow function.
  - > nrow(mtcars) # number of data rows
    [1] 32
- And the number of columns of a data frame is given by the ncol function.
  - > ncol(mtcars) # number of columns
    [1] 11

Lists

- A list is a R object which can contain many different types of elements inside it like vectors, matrices, data frames, functions and even other lists.
- Creating a list:
  - L<-list(1:3, c(“chairs”,”tables”,”computers”), c(1000,2500,25000))
- Naming the objects in a list: names(L)<-c(“S.no”, “items”, “price”)

Lists

- Accessing the elements in a list: Double square brackets are used for accessing the elements of a list.
  - L[[2]][1] # will return the item ‘table’
  - L[[“prices”]] #displays prices vector
  - length(L) #displays length of the list

Factors

- A factor is a statistical datatype that stores categorical variables, which are used in statistical modelling.
- A categorial variable is one that has two or more categories, but their is no intrinsic ordering to the categories. Eg: male and female.
- An R factor might be viewed simply as a vector with a bit more information added which consists of a record of the distinct values in that vector, called levels.
  - > x <- c(1,2,4,56,1,4,6)
  - > factor(x) # Displays factors of the object
  - [1] 1 2 4 56 1 4 6
  - Levels: 1 2 4 6 56

Arrays

- An array is essentially a multidimensional vector.
- It must all be of the same type and individual elements are accessed in a similar fashion using brackets.
- The first element is the row index, the second is the column index and the remaining elements are for outer dimensions. The array function takes a dim attribute which creates the required number of dimension. In the below example we create an array with two elements which are 2x3 matrices each. Creating an array:
  - > m< - array(1:12, dim=c(2,3,2))
- In the above example, “my.array” is the name of the array we have given. There are 12 units in this array mentioned as “1:12” and are divided in three dimensions “(2, 3, 2)”.

Arrays

- Alternative: with existing vector and using dim() : my.vector<- 1:24
- To convert my.vector vector to an array exactly like my.array simply by assigning the dimensions, like this: > dim(my.vector) <- c(3,4,2)
- Accessing elements of the array :-
  - m[1, ,] # Display every first row of the matrices
  - m[1, ,1] # Display first row of matrix 1
  - m[, , 1] # Display first matrix