Basic-R Programming

Karthik V

2023-02-03

Identifiers

Rules for writing Identifiers in R

Identifiers can be a combination of letters, digits, period (.) and underscore (_). It must start with a letter or a period. If it starts with a period, it cannot be followed by a digit.Reserved words in R cannot be used as identifiers.

```
# Example for Identifier
# variable
.a_number<-344
print(.a_number)

## [1] 344
```

Valid identifiers in R

`total, Sum, .fine.with.dot, this_is_acceptable, Number5`

Invalid identifiers in R

`tot@l, 5um, _fine, TRUE, .0ne`

Best Practices

- Earlier versions of R used underscore (_) as an assignment operator. So, the period (.) was used extensively in variable names having multiple words.
- Current versions of R support underscore as a valid identifier but it is good practice to use period as word separators.
- For example, `a.variable.name is preferred over a_variable_name or alternatively we could use camel case as aVariableName`

Reserved Words in R

```
#print(?reserved)
```

if else repeat while function for in next break

TRUE FALSE NULL Inf NaN NA NA_integer_ NA_real_ NA_complex_ NA_character_

Constants in R

- Constants, as the name suggests, are entities whose value cannot be altered.
- Basic types of constant are numeric constants and character constants.

Numeric Constants

- All numbers fall under this category. They can be of type integer, double or complex.
- It can be checked with the typeof() function.
- Numeric constants followed by L are regarded as integer and those followed by i are regarded as complex.

```
typeof(5)

## [1] "double"

typeof(5L)

## [1] "integer"

typeof(5i)

## [1] "complex"
```

Character Constants

Character constants can be represented using either single quotes (') or double quotes (") as delimiters.

```
typeof("5")

## [1] "character"

name<-'Edubridge'
typeof(name)

## [1] "character"
```

Built-in Constants

Some of the built-in constants defined in R along with their values is shown below

```
print(LETTERS)

##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
"R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"

print(letters)

##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
"r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"

print(pi)

## [1] 3.141593

print(month.abb)
```

```
##  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov"
"Dec"

print(month.name)

##  [1] "January"   "February"  "March"     "April"     "May"       "June"
##  [7] "July"      "August"    "September" "October"   "November"
"December"
```

DATA TYPES

In contrast to other programming languages like C and java in R, the variables are not declared as some data type. The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable. There are many types of R-objects. The frequently used ones are –

```
Vectors
Lists
Matrices
Arrays
Factors
Data Frames
```

Basic Data Types:

1.Vector:

- Vector is a basic data structure in R that contains element of similar ie. Homogeneous type.
- Vectors can be of all basic data types i.e. logical, integer, double, character, complex and raw.
- Vector can be created by using function c().
- In R using the function, typeof() one can check the data type of vector.
- function length() gives us the number of elements in the vector. #### integer

```
#integer
v1<-35L
typeof(v1)

## [1] "integer"

class(v1)

## [1] "integer"

is.vector(v1)

## [1] TRUE
```

double/numeric

```
#double/numeric
v2<-569
typeof(v2)

## [1] "double"

class(v2)

## [1] "numeric"
```

complex

```
#complex
v3<-4+3i
cat(typeof(v3),class(v3))

## complex complex
```

Logical

```
# Boolean Value canbe TRUE/T  FALSE/F
TRUE->b1  #b1<-TRUE
b2<-F
cat(typeof(b1),class(b2))

## logical logical
```

Character

```
c1<-'Edubridge'
c2<-"Data Analytics"
cat(typeof(c1),class(c1))

## character character

cat(typeof(c2),class(c2))

## character character
```

Raw

```
v <- charToRaw("Hello")
print(v)

## [1] 48 65 6c 6c 6f

cat(typeof(v),class(v))

## raw raw
```

creating Vector using c()

```
a.vector1<-c(12,15,'a',3L) # coerced to a common type
print(a.vector1)

## [1] "12" "15" "a"  "3"
```

```
typeof(a.vector1)

## [1] "character"
```

Creating Vector Using vector()

```
vector(mode='character',length = 4)

## [1] "" "" "" ""
```

Creating Vector using seq()

```
#seq(from, to, by= )
even<-seq(10,20,2)
print(even)

## [1] 10 12 14 16 18 20

print(typeof(even))

## [1] "double"
```

checking Vector or not using is.vector()

```
even<-seq(10,20,2)
print(even)

## [1] 10 12 14 16 18 20

print(is.vector(even))

## [1] TRUE
```

Creating Vector Using colon operator with numeric data

```
#from:to
v.1<-2:10
print(v.1)

## [1]  2  3  4  5  6  7  8  9 10

cat(typeof(v.1),is.vector(v.1))

## integer TRUE
```

Indexing of Vectors in R

- Indexing is used to extract, updating the individual or multiple values from the vector.
- In R, the indexing starts with 1. If we want to select a range of elements from the vector we can use [starting point : Ending Point]. There is concept of negative indexing as well,
- in R we use negative indexing to exclude the element while performing any action.

```
int.vector<-c(3L,5L,7L,8L,10L)
typeof(int.vector)

## [1] "integer"
```

```
#access the element using index
print(int.vector[1:3]) #indexing starts with 1.

## [1] 3 5 7

print(int.vector[-3])

## [1]  3  5  8 10

print(int.vector[c(1,3,15)]) #select the values in given index vector

## [1]  3  7 NA

print(int.vector[c(-1,-3,-4)]) # drop the values in given index vector

## [1]  5 10

days <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
days[-2:-4]

## [1] "Sun"   "Thurs" "Fri"   "Sat"
```

Logical Index

```
days <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
days=='Sun'

## [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE

days[days=='Sun']

## [1] "Sun"

int.vector<-c(3L,5L,7L,8L,10L)
int.vector<8

## [1]  TRUE  TRUE  TRUE FALSE FALSE

int.vector[int.vector<8]

## [1] 3 5 7
```

Modifying a vector

- Modification of a Vector is the process of applying some operation on an individual element of a vector to change its value in the vector. There are different ways through which we can modify a vector:

```
X <- c(2, 7, 9, 7, 8, 2)
X[1:3]<-10
print(X)

## [1] 10 10 10  7  8  2
```

Vector Manipulation

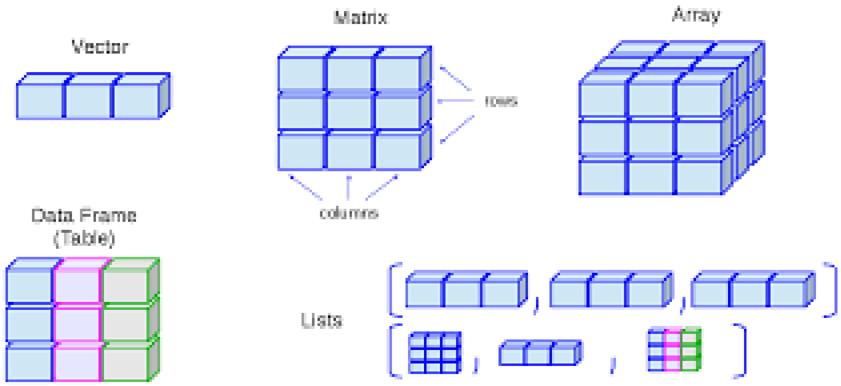
- Vector arithmetic

- Two vectors of same length can be added, subtracted, multiplied or divided giving the result as a vector output.

```
v1 <- c(3,8,4,5,3,2,1,7)
v2 <- c(4,11,0,8)
v1%%v2

## [1]   3   8 NaN   5   3   2 NaN   7

sort(v1,decreasing = TRUE)

## [1] 8 7 5 4 3 3 2 1

rep(v1,3)

##  [1] 3 8 4 5 3 2 1 7 3 8 4 5 3 2 1 7 3 8 4 5 3 2 1 7
```

Matrix

Matrix is a rectangular arrangement of numbers in rows and columns. In a matrix, as we know rows are the ones that run horizontally and columns are the ones that run vertically. In R programming, matrices are two-dimensional, homogeneous data structures



Creating a Matrix

- To create a matrix in R you need to use the function called matrix().
- The arguments to this matrix() are the set of elements in the vector. You have to pass how many numbers of rows and how many numbers of columns you want to have in your matrix.
- Note: By default, matrices are in column-wise order.

```
mdat <- matrix(c(1,2,3,11,12,13), nrow = 2, ncol = 3, byrow = TRUE)
mdat

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]   11   12   13
```

Logical Matrix

```
m1 <- matrix(c(T, T, F, F, T, F), nrow = 3)
m1

##       [,1]  [,2]
## [1,]  TRUE FALSE
## [2,]  TRUE  TRUE
## [3,] FALSE FALSE

#column-wise
m2 <- matrix(c("a", "b","c", "d"), nrow = 2,byrow = F)
m2

##      [,1] [,2]
## [1,] "a"  "c"
## [2,] "b"  "d"

#row-wise
m2 <- matrix(c("a", "b","c", "d"), nrow = 2,byrow = T)
m2

##      [,1] [,2]
## [1,] "a"  "b"
## [2,] "c"  "d"
```

checking matrix or not using is.matrix()

```
m2 <- matrix(c("a", "b","c", "d"), nrow = 2,byrow = T)
m2

##      [,1] [,2]
## [1,] "a"  "b"
## [2,] "c"  "d"

is.matrix(m2)

## [1] TRUE

is.vector(m2)

## [1] FALSE
```

typecast Matrix to vector

```
char.vect<-as.vector(m2)
char.vect

## [1] "a" "c" "b" "d"

is.vector(char.vect)

## [1] TRUE
```

Example:

```
# R program to create a matrix
A = matrix(1:9,nrow = 3,ncol = 3,byrow =
TRUE,dimnames=list(c('a','b','c'),c('d','e','f')))
print(A)
```

```
##   d e f
## a 1 2 3
## b 4 5 6
## c 7 8 9
```

```
list(c('a','b','c'),c('d','e','f'))
```

```
## [[1]]
## [1] "a" "b" "c"
##
## [[2]]
## [1] "d" "e" "f"
```

row and Column names

```
print(row.names(A))
```

```
## [1] "a" "b" "c"
```

```
print(colnames(A))
```

```
## [1] "d" "e" "f"
```

Modifiy the Row and Columns names

```
row.names(A)=c('AX','BX','CX')
colnames(A)=c('DY','EY','FY')
print(A)
```

```
##    DY EY FY
## AX  1  2  3
## BX  4  5  6
## CX  7  8  9
```

Acessing Matrix Elements using index

```
# matrix[row-index,column-index]
A[c(1,2,3),c(2,3)]
```

```
##    EY FY
## AX  2  3
## BX  5  6
## CX  8  9
```

Matrix metrics

- dimension of the Matrix
- rows
- columns

- elements

```
# Create a 3x3 matrix
A = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9),nrow = 3,ncol = 3,byrow = TRUE)
cat("The 3x3 matrix:\n")

## The 3x3 matrix:

print(A)

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

###Dimension of the matrix

```
dim(A)

## [1] 3 3
```

Number of rows

```
nrow(A)

## [1] 3
```

Number of columns

```
ncol(A)

## [1] 3
```

Number of elements

```
length(A)

## [1] 9

prod(dim(A))

## [1] 9
```

Creating constant filled matrices

- R allows creation of various different types of matrices with the use of arguments passed to the matrix() function.
- Matrix where all rows and columns are filled by a single constant ‘k’
  - Syntax: matrix(k, m, n) Parameters:
  - k: the constant
  - m: no of rows
  - n: no of columns

```
matrix(100,4,5)

##      [,1] [,2] [,3] [,4] [,5]
## [1,]  100  100  100  100  100
## [2,]  100  100  100  100  100
## [3,]  100  100  100  100  100
## [4,]  100  100  100  100  100
```

Diagonal matrix:

- A diagonal matrix is a matrix in which the entries outside the main diagonal are all zero. To create such a matrix the syntax is given below
- Syntax: diag(k, m, n)
  - Parameters:
  - k: the constants/array
  - m: no of rows
  - n: no of columns

```
diag(c(10,20,30),4,4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   10    0    0    0
## [2,]    0   20    0    0
## [3,]    0    0   30    0
## [4,]    0    0    0   10
```

Identity matrix:

- A square matrix in which all the elements of the principal diagonal are ones and all other elements are zeros. To create such a matrix the syntax is given below:
- Syntax: diag(k, m, n)
  - Parameters:
  - k: 1
  - m: no of rows
  - n: no of columns

```
diag(1,3,3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

Matrix Computations

- Various mathematical operations are performed on the matrices using the R operators.
- The result of the operation is also a matrix.
- The dimensions (number of rows and columns) should be same for the matrices involved in the operation.

Matrix Addition

```
# Create two 2x3 matrices.
matrix1 <- matrix(c(3,9,-1,4,2,6), nrow =2)
print(matrix1)
```

```
##      [,1] [,2] [,3]
## [1,]    3   -1    2
## [2,]    9    4    6
```

```
matrix2 <- matrix(c(5,2,0,9,3,4), nrow =2)
print(matrix2)
```

```
##      [,1] [,2] [,3]
## [1,]    5    0    3
## [2,]    2    9    4
```

Matrix Operation

```
matrix1+matrix2
```

```
##      [,1] [,2] [,3]
## [1,]    8   -1    5
## [2,]   11   13   10
```

Element wise Multilication

```
matrix1*matrix2
```

```
##      [,1] [,2] [,3]
## [1,]   15    0    6
## [2,]   18   36   24
```

Product Multiplication

```
matrix1%*%t(matrix2)
```

```
##      [,1] [,2]
## [1,]   21    5
## [2,]   63   78
```

Matrix Concatenation

- Matrix concatenation refers to the merging of rows or columns of an existing matrix.

Concatenation of a row:

- The concatenation of a row to a matrix is done using rbind().

```
A = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9),nrow = 3,ncol = 3,byrow = TRUE)
cat("The 3x3 matrix:\n")
```

```
## The 3x3 matrix:
```

```
print(A)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

```
B = matrix(c(10, 11, 12),nrow = 1,ncol = 3)
cat("The 1x3 matrix:\n")
```

```
## The 1x3 matrix:

print(B)

##      [,1] [,2] [,3]
## [1,]   10   11   12

# Add a new row using rbind()
C = rbind(A, B)
cat("After concatenation of a row:\n")

## After concatenation of a row:

print(C)

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   11   12
```

Concatenation of a column:

- The concatenation of a column to a matrix is done using cbind().

```
B = matrix(c(10, 11, 12),nrow = 3,ncol = 1,byrow = TRUE)
cat("The 3x1 matrix:\n")

## The 3x1 matrix:

print(B)

##      [,1]
## [1,]   10
## [2,]   11
## [3,]   12

# Add a new column using cbind()
C = cbind(A,B)
cat("After concatenation of a column:\n")

## After concatenation of a column:

print(C)

##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3   10
## [2,]    4    5    6   11
## [3,]    7    8    9   12
```

Lists

- A list is a data structure, much like a vector, in that it is used for storing an ordered set of elements.

- However, where a vector requires all its elements to be the same type, a list allows different types of elements to be collected.
- Due to this flexibility, lists are often used to store various types of input and output data and sets of configuration parameters for machine learning models.

Create a list

- The List is been created using list() Function in R.

```
list1<-list(c(3,5,6),c(T,F,F),4+5i,"Edubridge")
print(list1)
```

```
## [[1]]
## [1] 3 5 6
##
## [[2]]
## [1]  TRUE FALSE FALSE
##
## [[3]]
## [1] 4+5i
##
## [[4]]
## [1] "Edubridge"
```

Acess list Elements using index

```
list1[1]  # output as list
```

```
## [[1]]
## [1] 3 5 6
```

```
list1[[1]] #output as a element type
```

```
## [1] 3 5 6
```

```
typeof(list1[[4]])
```

```
## [1] "character"
```

structure of List object in R using str()

```
str(list1)
```

```
## List of 4
##  $ : num [1:3] 3 5 6
##  $ : logi [1:3] TRUE FALSE FALSE
##  $ : cplx 4+5i
##  $ : chr "Edubridge"
```

checking list or not using is.list()

```
is.list(list1)
```

```
## [1] TRUE
```

Example

```
# creating Employee List
empId <- c(1, 2, 3, 4)
empName <- c("Debi", "Sandeep", "Subham", "Shiba")
numberOfEmp = 4
emplist<-list(empId,empName,numberOfEmp)
print(emplist)
```

```
## [[1]]
## [1] 1 2 3 4
##
## [[2]]
## [1] "Debi"    "Sandeep" "Subham"  "Shiba"
##
## [[3]]
## [1] 4
```

```
print(str(emplist))
```

```
## List of 3
##  $ : num [1:4] 1 2 3 4
##  $ : chr [1:4] "Debi" "Sandeep" "Subham" "Shiba"
##  $ : num 4
## NULL
```

Naming List Elements

- Access components by names: All the components of a list can be named and we can use those names to access the components of the list using the dollar$ command.

```
names(emplist)<-c('empId','empName','numberOfEmp')
print(emplist)
```

```
## $empId
## [1] 1 2 3 4
##
## $empName
## [1] "Debi"    "Sandeep" "Subham"  "Shiba"
##
## $numberOfEmp
## [1] 4
```

Acess the list element using $names of the elements

```
emplist$empId
```

```
## [1] 1 2 3 4
```

```
emplist$empName
```

```
## [1] "Debi"    "Sandeep" "Subham"  "Shiba"
```

```
emplist$numberOfEmp
```

```
## [1] 4
```

To add an item to the end of the list, use the append() function:

```
salary<-c(12,15.5,10.2,8.4)
list2<-list(income=salary)
list2

## $income
## [1] 12.0 15.5 10.2  8.4

append(emplist,list2,after=1) #default  after = length(list Obj)

## $empId
## [1] 1 2 3 4
## 
## $income
## [1] 12.0 15.5 10.2  8.4
## 
## $empName
## [1] "Debi"    "Sandeep" "Subham"  "Shiba"
## 
## $numberOfEmp
## [1] 4
```

Concatenation/Merge of lists

Two lists can be concatenated using the concatenation function. So, when we want to concatenate two lists we have to use the concatenation operator.

```
c(emplist,list2)

## $empId
## [1] 1 2 3 4
## 
## $empName
## [1] "Debi"    "Sandeep" "Subham"  "Shiba"
## 
## $numberOfEmp
## [1] 4
## 
## $income
## [1] 12.0 15.5 10.2  8.4
```

Converting List to Vector

Here we are going to convert the list to vector, for this we will create a list first and then unlist the list into the vector.

```
unlist.vector<-unlist(emplist)
unlist.vector
```

```
##      empId1      empId2      empId3      empId4    empName1    empName2
##         "1"         "2"         "3"         "4"      "Debi"   "Sandeep"
##    empName3    empName4 numberOfEmp
##    "Subham"     "Shiba"         "4"
```

```
typeof(unlist.vector)
```

```
## [1] "character"
```

```
names(unlist.vector)
```

```
## [1] "empId1"      "empId2"      "empId3"      "empId4"      "empName1"
## [6] "empName2"    "empName3"    "empName4"    "numberOfEmp"
```

```
str(unlist.vector)
```

```
##  Named chr [1:9] "1" "2" "3" "4" "Debi" "Sandeep" "Subham" "Shiba" "4"
##  - attr(*, "names")= chr [1:9] "empId1" "empId2" "empId3" "empId4" ...
```

Array

In R, Arrays can contain multi-dimensional rectangular shaped data storage structure. “Rectangular” in the sense, each row is having the same length and similarly for each column and other dimensions. Matrices are a special type of two — dimensional arrays.

Arrays can contain only homogeneous data, i.e. elements having similar data type. In R, an array can be created in using the array() function. Arrays take vectors in the form of input and use the values in the dim parameter for creating an array. #array(data = NA, dim = length(data), dimnames = NULL)

```
array(1:3, c(2,4)) # recycle 1:3 "2 2/3 times"
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    2    1
## [2,]    2    1    3    2
```

```
print(array(LETTERS,dim=c(3,3,2)))
```

```
## , , 1
## 
##      [,1] [,2] [,3]
## [1,] "A"  "D"  "G" 
## [2,] "B"  "E"  "H" 
## [3,] "C"  "F"  "I" 
## 
## , , 2
## 
##      [,1] [,2] [,3]
## [1,] "J"  "M"  "P" 
## [2,] "K"  "N"  "Q" 
## [3,] "L"  "O"  "R"
```

```
vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
vec2 <- c(10, 11, 12)

arr1<-array(c(vec1,vec2),dim=c(2,3,2 ))
print(arr1)
```

```
## , , 1
## 
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
## 
## , , 2
## 
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
```

Checking R object is array or not?

```
is.array(arr1)
```

```
## [1] TRUE
```

```
str(arr1)
```

```
##  num [1:2, 1:3, 1:2] 1 2 3 4 5 6 7 8 9 10 ...
```

```
names(arr1)
```

```
## NULL
```

Naming of Arrays

The row names, column names and matrices names are specified as a vector of the number of rows, number of columns and number of matrices respectively. By default, the rows, columns and matrices are named by their index values.

```
row_names <- c("row1", "row2")
col_names <- c("col1", "col2", "col3")
mat_names <- c("Mat1", "Mat2")
arr = array(2:14, dim = c(2, 3, 2),dimnames = list(row_names,col_names,
mat_names))
print (arr)
```

```
## , , Mat1
## 
##      col1 col2 col3
## row1    2    4    6
## row2    3    5    7
## 
## , , Mat2
## 
```

```
##      col1 col2 col3
## row1    8   10   12
## row2    9   11   13

rownames(arr)

## [1] "row1" "row2"

colnames(arr)

## [1] "col1" "col2" "col3"

print(str(arr))

##  int [1:2, 1:3, 1:2] 2 3 4 5 6 7 8 9 10 11 ...
##  - attr(*, "dimnames")=List of 3
##   ..$ : chr [1:2] "row1" "row2"
##   ..$ : chr [1:3] "col1" "col2" "col3"
##   ..$ : chr [1:2] "Mat1" "Mat2"
## NULL
```

Accessing arrays

The arrays can be accessed by using indices for different dimensions separated by commas. Different components can be specified by any combination of elements' names or positions.

```
#arr[row,colum,matrix]  for 3D
arr[,c(1),1:2]

##      Mat1 Mat2
## row1    2    8
## row2    3    9
```

apply

Calculations Across Array Elements We can do calculations across the elements in an array using the apply() function.

```
arr

## , , Mat1
## 
##      col1 col2 col3
## row1    2    4    6
## row2    3    5    7
## 
## , , Mat2
## 
##      col1 col2 col3
## row1    8   10   12
## row2    9   11   13
```

```
apply(arr,c(1,2),sum)

##      col1 col2 col3
## row1   10   14   18
## row2   12   16   20
```

Factors

- Categorical (nominal) and ordered categorical (ordinal) variables in R are called factors.
- Factors are the r-objects which are created using a vector.
- It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector. They are useful in statistical modeling.

Creation of factors

- Factors are created using the factor() function.
- factor(x = character(), levels, labels = levels, exclude = NA, ordered = is.ordered(x), nmax = NA)
- The nlevels functions gives the count of levels.

```
# Create a vector.
apple_colors <- c('green','green','yellow','red','red','red','green')
apple_colors

## [1] "green"  "green"  "yellow" "red"    "red"    "red"    "green"

# character vector
```

Unique elements in vector

```
unique(apple_colors)

## [1] "green"  "yellow" "red"
```

find the duplicated values

```
apple_colors[duplicated(apple_colors)]

## [1] "green" "red"   "red"   "green"

fact.apple.color<-factor(apple_colors)
fact.apple.color

## [1] green  green  yellow red    red    red    green 
## Levels: green red yellow

str(fact.apple.color)

##  Factor w/ 3 levels "green","red",..: 1 1 3 2 2 2 1
```

Check isfactor and isordered

```
is.factor(fact.apple.color)
```

```
## [1] TRUE

is.ordered(fact.apple.color)

## [1] FALSE

nlevels(fact.apple.color)

## [1] 3
```

#Acess and Modifiying Factors elements

```
fact.apple.color[1]

## [1] green
## Levels: green red yellow

fact.apple.color[1]<-'yellow'
fact.apple.color

## [1] yellow green  yellow red    red    red    green
## Levels: green red yellow

fact.apple.color[1]<-'pink'

## Warning in `[<-.factor`(`*tmp*`, 1, value = "pink"): invalid factor level,
NA
## generated
```

Update the Levels in Factor

```
levels(fact.apple.color)

## [1] "green"  "red"    "yellow"

levels(fact.apple.color)<-c(levels(fact.apple.color),'pink')
fact.apple.color[1]<-'pink'
print(fact.apple.color)

## [1] pink   green  yellow red    red    red    green
## Levels: green red yellow pink
```

ordered

```
ratings <- c("Poor", "Good", "Excellent", "Good", "Poor", "Excellent")
ratings_factor <- factor(ratings, levels = c("Poor", "Good", "Excellent"),
ordered = TRUE)
print(ratings_factor)

## [1] Poor      Good      Excellent Good      Poor      Excellent
## Levels: Poor < Good < Excellent

is.ordered(ratings_factor)

## [1] TRUE
```

DataFrame

A DataFrame in R is a table-like structure for storing and manipulating data. It is similar to a spreadsheet or a database table and is an essential data structure for working with data in R.

To create a DataFrame in R, you can use the data.frame() function and provide the data as argument.

```
gender = c("Male", "Male","Female")  # character vector
height = c(152, 171.5, 165) # double vector
weight = c(81,93, 78)## double vector
age = c(42L,38L,26L)#integer vector
BMI.data<-data.frame(gender,height,weight,age,stringsAsFactors =F)
BMI.data
```

```
##   gender height weight age
## 1   Male  152.0     81  42
## 2   Male  171.5     93  38
## 3 Female  165.0     78  26
```

```
is.data.frame(BMI.data)
```

```
## [1] TRUE
```

#Access the Column in Data Frame

```
colnames(BMI.data)
```

```
## [1] "gender" "height" "weight" "age"
```

```
BMI.data$gender
```

```
## [1] "Male"   "Male"   "Female"
```

```
BMI.data$height
```

```
## [1] 152.0 171.5 165.0
```

```
BMI.data$weight
```

```
## [1] 81 93 78
```

```
BMI.data$age
```

```
## [1] 42 38 26
```

access the column using [ ]

```
BMI.data['gender']
```

```
##   gender
## 1   Male
## 2   Male
## 3 Female
```

Character Index

```
BMI.data[c('gender','age')]

##   gender age
## 1   Male  42
## 2   Male  38
## 3 Female  26
```

numeric Index

```
BMI.data[2,c(2,3)]  #[row,column]

##   height weight
## 2  171.5     93
```

Built-in datasets

Built-in datasets in R can be found in the datasets package, which is included with the base installation of R. To access these datasets, you can load the datasets package into the environment using the library() function:

```
#library(help=datasets)
library(datasets)  #importing Datasets in to R environment
```

Some Sample Datasets

```
mtcars  # dataFrame

##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
tail(mtcars)
```

```
##                 mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

```
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```