Basic Data Objects in R

Norman Lo

1/10/2022

General Data Objects in R

In the previous section, we have mentioned about the most common data structure in R, “vector”. However, in many cases, data needs to be stored in a higher dimension data structure. In R, several data structures are offered to satisfy our need to manage different data sources. In this section, we are going to cover the most popular data structures in R, which includes data.frame, matrix, list, and array.

2.1 - data.frame

data.frame is a two dimensional data structure in R. It is a special case of a vector which has each component of equal length. Each component form the column and contents of hte component form the rows. Generally speaking, data.frame is similar to Excel spreadsheet.

There are many data.frame methods in R, but the simplest way to create a data.frame is to use data.frame() function. Let’s create a simple data.frame with with columns x, y, and q, which came from three individual vectors.

```
# Create 3 vectors x, y, and q
x <- 1:10
y <- -4:5
q <- c("Hockey", "Football", "Baseball", "Curling", "Rugby",
       "Lacrosse", "Basketball", "Tennis", "Cricket", "Soccer")

# Create a data frame using data.frame()
theDF <- data.frame(x, y, q)
theDF
```

```
##     x  y          q
## 1   1 -4     Hockey
## 2   2 -3   Football
## 3   3 -2   Baseball
## 4   4 -1    Curling
## 5   5  0      Rugby
## 6   6  1   Lacrosse
## 7   7  2 Basketball
## 8   8  3     Tennis
## 9   9  4    Cricket
## 10 10  5     Soccer
```

```
# We can check the class of a data frame object using class()
class(theDF)
```

```
## [1] "data.frame"
```

In this example, we created a data.frame with three individual vectors and the dimension of the data.frame is 10x3. As noticed, "theDF" in the example is a variable assigned with the data.frame. data.frame also allows us to define the column names, so it can be easily read. In the following example, we are assigning the name of the columns to each vector pass into the data.frame() function.

```
# Assign column names to the data frame
theDF <- data.frame(First=x, Second=y, Sport=q)
theDF
```

```
##    First Second      Sport
## 1      1     -4     Hockey
## 2      2     -3   Football
## 3      3     -2   Baseball
## 4      4     -1    Curling
## 5      5      0      Rugby
## 6      6      1   Lacrosse
## 7      7      2 Basketball
## 8      8      3     Tennis
## 9      9      4    Cricket
## 10    10      5     Soccer
```

data.frame is a complex object with many different **attributes**. The most popular attributes are number of rows, number of columns, and dimenion of the data.frame. R offers the functions to check these attributes at ease.

nrow() - check the number of rows of the data.frame ncol() - check the number of columns of the data.frame dim() - check the dimension (row x column) of hte data.frame.

```
# Number of row
nrow(theDF)
```

```
## [1] 10
```

```
# Number of column
ncol(theDF)
```

```
## [1] 3
```

```
# Dimension of the data frame
dim(theDF)
```

```
## [1] 10  3
```

We can also check the name of each column using the names() function, which will return a vector of character. We can also check the name of a specific column by index and assign new names to the columns after it's created. Here are some examples:

```
# Check the column names of the data frame
names(theDF)
```

```
## [1] "First"  "Second" "Sport"
```

```
# Check only the third column name by indexing
names(theDF)[3]
```

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

```
# Check the row names (index)
rownames(theDF)
```

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

```
# Changing the row names
rownames(theDF) <- c("one", "two", "three", "four", "five",
                     "six", "seven", "eight", "nine", "ten")
rownames(theDF)
```

```
##  [1] "one"   "two"   "three" "four"  "five"  "six"   "seven" "eight" "nine"
## [10] "ten"
```

```
# Reset row names back to index
rownames(theDF) <- NULL
rownames(theDF)
```

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

When a data.frame is so big or contains many rows, we may only want to print the first few rows to check the data within. In this case, we can use head() function to check the first 6 rows of the data. tail() function, on the other hand, returns the last 6 rows of the data in the data.frame.

```
# View only the first six rows of the data frame (default)
head(theDF)
```

```
##   First Second    Sport
## 1     1     -4   Hockey
## 2     2     -3 Football
## 3     3     -2 Baseball
## 4     4     -1  Curling
## 5     5      0    Rugby
## 6     6      1 Lacrosse
```

```
# View the first seven rows of the data frame (n=7)
head(theDF, n=7)
```

```
##   First Second      Sport
## 1     1     -4     Hockey
## 2     2     -3   Football
## 3     3     -2   Baseball
## 4     4     -1    Curling
## 5     5      0      Rugby
## 6     6      1   Lacrosse
## 7     7      2 Basketball
```

```
# View the last six rows of the data frame (default)
tail(theDF)
```

```
##    First Second      Sport
## 5      5      0      Rugby
## 6      6      1   Lacrosse
## 7      7      2 Basketball
## 8      8      3     Tennis
## 9      9      4    Cricket
## 10    10      5     Soccer
```

If you want to check the type of data structure, you can use the class() function.

```
# Check the type of the data structure
class(theDF)
```

```
## [1] "data.frame"
```

```
# Extracting a column of elements from the data frame using $
theDF$Sport
```

```
##  [1] "Hockey"     "Football"   "Baseball"   "Curling"    "Rugby"
##  [6] "Lacrosse"   "Basketball" "Tennis"     "Cricket"    "Soccer"
```

```
# Extracting an element from the third row and second column using [ ]
theDF[3, 2]
```

```
## [1] -2
```

```
# Extracting the elements from the third row and second to third columns
theDF[3, 2:3]  # Note that the return data frame has column names
```

```
##   Second    Sport
## 3     -2 Baseball
```

```
# Extracting the elements from the third and fifth rows and second column
theDF[c(3,5), 2]  # Note that the return vector does not have name title
```

```
## [1] -2  0
```

```r
# Extracting the elments from the third and fifth rows and second to third columns
theDF[c(3,5), 2:3]  # Note that the return data frame has column names
```

```
##   Second    Sport
## 3     -2 Baseball
## 5      0    Rugby
```

```r
# Extracting all the elements from the third column
theDF[ , 3]  # Note the return vector does not have name title
```

```
##  [1] "Hockey"     "Football"   "Baseball"   "Curling"    "Rugby"
##  [6] "Lacrosse"   "Basketball" "Tennis"     "Cricket"    "Soccer"
```

```r
# Extracting all the elements from second to third columns
theDF[ , 2:3]  # Note the return data frame has column names
```

```
##    Second      Sport
## 1      -4     Hockey
## 2      -3   Football
## 3      -2   Baseball
## 4      -1    Curling
## 5       0      Rugby
## 6       1   Lacrosse
## 7       2 Basketball
## 8       3     Tennis
## 9       4    Cricket
## 10      5     Soccer
```

```r
# Extracting all the elements in the second row
theDF[2, ]  # Note the return data frame has column names
```

```
##   First Second    Sport
## 2     2     -3 Football
```

```r
# Extracting all the elements in the second to fourth rows
theDF[2:4, ]
```

```
##   First Second    Sport
## 2     2     -3 Football
## 3     3     -2 Baseball
## 4     4     -1  Curling
```

```r
# Extracting all the elements in columns "First" and "Sport"
theDF[ , c("First", "Sport")]
```

```
##    First      Sport
## 1      1     Hockey
## 2      2   Football
## 3      3   Baseball
## 4      4    Curling
## 5      5      Rugby
## 6      6   Lacrosse
## 7      7 Basketball
## 8      8     Tennis
## 9      9    Cricket
## 10    10     Soccer
```

It's often confusing for people to use different methods to extract data from the data.frame. For instance, we can use df[, "column name"], df["column name"], and df[["column name"]] to extract a column of elements, but most people don't realize the difference between these methods. Here are some examples to demontrate:

```
# Extract only "Sport" column with df[ , "column name"] method
theDF[ , "Sport"]
```

```
##  [1] "Hockey"     "Football"   "Baseball"   "Curling"    "Rugby"
##  [6] "Lacrosse"   "Basketball" "Tennis"     "Cricket"    "Soccer"
```

```
# Check the class of the method
class(theDF[ , "Sport"])  # return factor object, categorical data
```

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

```
# Extract only "Sport" column with df["column name"] method
theDF["Sport"]
```

```
##         Sport
## 1      Hockey
## 2    Football
## 3    Baseball
## 4     Curling
## 5       Rugby
## 6    Lacrosse
## 7  Basketball
## 8      Tennis
## 9     Cricket
## 10     Soccer
```

```
# Check the class of the method
class(theDF["Sport"])  # return data.frame object
```

```
## [1] "data.frame"
```

```
# Extract only "Sport" column with df[["column name"]] method
theDF[["Sport"]]
```

```
##  [1] "Hockey"     "Football"   "Baseball"   "Curling"    "Rugby"
##  [6] "Lacrosse"   "Basketball" "Tennis"     "Cricket"    "Soccer"
```

```
# Check the class of the method
class(theDF[["Sport"]])  # return factor object, categorical data
```

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

As observed in the previous examples, different methods return different R objects. In fact, when we are using the df[, “column name”] method, we can pass in a third argument: drop=FALSE, so the return object is a data.frame.

```
# Extract only "Sport" column applying drop=FALSE argument
theDF[ , "Sport", drop=FALSE]
```

```
##          Sport
## 1       Hockey
## 2     Football
## 3     Baseball
## 4      Curling
## 5        Rugby
## 6     Lacrosse
## 7   Basketball
## 8       Tennis
## 9      Cricket
## 10      Soccer
```

```
# Check the class of the method
class(theDF[ , "Sport", drop=FALSE])  # return data.frame object
```

```
## [1] "data.frame"
```

```
# Extract only the thrid column applying drop=FALSE argument
theDF[ , 3, drop=FALSE]
```

```
##          Sport
## 1       Hockey
## 2     Football
## 3     Baseball
## 4      Curling
## 5        Rugby
## 6     Lacrosse
## 7   Basketball
## 8       Tennis
## 9      Cricket
## 10      Soccer
```

```
# Check the class of the method
class(theDF[ , 3, drop=FALSE])  # return data.frame object
```

```
## [1] "data.frame"
```

Note: Many people first learning a statistical software like R do not pay attention to the type of data structure in different analysis packages. Indeed, it's especially important and essential to understand what is required to pass into a function in R.

As described in the previous section, factor is a special data structure that each unique category (string) in the data is defined into levels. R has an easy way to transform a factor into a data.frame format using model.matrix() function. The function returns columns of dummy variables with value 0 and 1 that indicates the level in the factor.

```
# Create a new factor
newFactor <- factor(c("Pennsylvania", "New York", "New Jersey", "New York",
                      "Tennessee", "Massachusetts", "Pennsylvania", "New York"))

# Create a data.frame based on the factor newFactor
model.matrix(~newFactor - 1)
```

```
##   newFactorMassachusetts newFactorNew Jersey newFactorNew York
## 1                      0                   0                 0
## 2                      0                   0                 1
## 3                      0                   1                 0
## 4                      0                   0                 1
## 5                      0                   0                 0
## 6                      1                   0                 0
## 7                      0                   0                 0
## 8                      0                   0                 1
##   newFactorPennsylvania newFactorTennessee
## 1                     1                  0
## 2                     0                  0
## 3                     0                  0
## 4                     0                  0
## 5                     0                  1
## 6                     0                  0
## 7                     1                  0
## 8                     0                  0
## attr(,"assign")
## [1] 1 1 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$newFactor
## [1] "contr.treatment"
```

2.2 - List

When we are trying to have a collection of different data objects, we can use the **list** data structure to collect them. List can store any type or length of data object, such as numeric, character, mixed of numeric and character data, data.frame, or a list.

```
# Create a list with three elements
list("a", "b", "c")
```

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

```
## [[2]]
## [1] "b"
##
## [[3]]
## [1] "c"
```

```
# Create a list with a single element, which is a vector
list(c("a", "b", "c"))
```

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

```
# Create a list with two vector elements
(list3 <- list(c("a", "b", "c"), 3:7))
```

```
## [[1]]
## [1] "a" "b" "c"
##
## [[2]]
## [1] 3 4 5 6 7
```

```
# Create a list with two elements, a data.frame and vector
list(theDF, 1:10)
```

```
## [[1]]
##    First Second      Sport
## 1      1     -4     Hockey
## 2      2     -3   Football
## 3      3     -2   Baseball
## 4      4     -1    Curling
## 5      5      0      Rugby
## 6      6      1   Lacrosse
## 7      7      2 Basketball
## 8      8      3     Tennis
## 9      9      4    Cricket
## 10    10      5     Soccer
##
## [[2]]
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
# Create a list with three elements, a data.frame, vector, and list of two vectors
list5 <- list(theDF, 1:10, list3)
list5
```

```
## [[1]]
##    First Second      Sport
## 1      1     -4     Hockey
## 2      2     -3   Football
## 3      3     -2   Baseball
## 4      4     -1    Curling
## 5      5      0      Rugby
## 6      6      1   Lacrosse
## 7      7      2 Basketball
## 8      8      3     Tennis
## 9      9      4    Cricket
## 10    10      5     Soccer
##
## [[2]]
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## [[3]]
## [[3]][[1]]
## [1] "a" "b" "c"
##
## [[3]][[2]]
## [1] 3 4 5 6 7
```

Note: As you may noticed in the above examples, when we are assigning a list of elements to list3 inside the parentheses, not only it create the list3 variable, but it also display the contents assigned to the variable.

Similar to data.frame, we can also assign name to each of the element in a list and the names can be check by names() function.

```
# Check the name of each element in list5
names(list5)
```

```
## NULL
```

```
# Assign name to each element in list5
names(list5) <- c("data.frame", "vector", "list")

# Check the name of each element in list5 again
names(list5)
```

```
## [1] "data.frame" "vector"     "list"
```

```
# Print the list with the newly assigned names
list5
```

```
## $data.frame
##    First Second      Sport
## 1      1     -4     Hockey
## 2      2     -3   Football
## 3      3     -2   Baseball
## 4      4     -1    Curling
## 5      5      0      Rugby
## 6      6      1   Lacrosse
## 7      7      2 Basketball
## 8      8      3     Tennis
## 9      9      4    Cricket
## 10    10      5     Soccer
##
## $vector
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
##
## $list
## $list[[1]]
## [1] "a" "b" "c"
##
## $list[[2]]
## [1] 3 4 5 6 7
```

```
# We can also assign name to each element when creating the list
list6 <- list(TheDataFrame = theDF, TheVector = 1:10, TheList = list3)

# Check the name of each element in list6
names(list6)
```

```
## [1] "TheDataFrame" "TheVector"    "TheList"
```

```
# Print the list with assigned names
list6
```

```
## $TheDataFrame
##    First Second      Sport
## 1      1     -4     Hockey
## 2      2     -3   Football
## 3      3     -2   Baseball
## 4      4     -1    Curling
## 5      5      0      Rugby
## 6      6      1   Lacrosse
## 7      7      2 Basketball
## 8      8      3     Tennis
## 9      9      4    Cricket
## 10    10      5     Soccer
##
## $TheVector
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $TheList
## $TheList[[1]]
## [1] "a" "b" "c"
##
## $TheList[[2]]
## [1] 3 4 5 6 7
```

```
# Create an empty list with 4 empty spaces
(emptyList <- vector(mode="list", length=4))
```

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

To extract a single element from a list, we can use double square brackets with the element's index or name. For more complex extraction, such as list within a list or an element within a data.frame or vector, we can extract the data by **nested indexing**. Furthermore, we can also add new element to a list by assigning it to a new index or name.

```
# Extracting an element from a list using index
list5[[1]]
```

```
##    First Second      Sport
## 1      1     -4     Hockey
## 2      2     -3   Football
## 3      3     -2   Baseball
## 4      4     -1    Curling
## 5      5      0      Rugby
## 6      6      1   Lacrosse
## 7      7      2 Basketball
## 8      8      3     Tennis
## 9      9      4    Cricket
## 10    10      5     Soccer
```

```
# Extracting an element form a list using the element's name
list5[["data.frame"]]
```

```
##    First Second      Sport
## 1      1     -4     Hockey
## 2      2     -3   Football
## 3      3     -2   Baseball
## 4      4     -1    Curling
## 5      5      0      Rugby
## 6      6      1   Lacrosse
## 7      7      2 Basketball
## 8      8      3     Tennis
## 9      9      4    Cricket
## 10    10      5     Soccer
```

```
# Extracting an element from an element in a list using nested indexing
list5[[1]]$Sport
```

```
##  [1] "Hockey"     "Football"   "Baseball"   "Curling"    "Rugby"
##  [6] "Lacrosse"   "Basketball" "Tennis"     "Cricket"    "Soccer"
```

```
# Extracting an element from an element in a list using nested indexing
list5[[1]][ , "Second"]
```

```
##  [1] -4 -3 -2 -1  0  1  2  3  4  5
```

```
# Extracting an element from an element in a list using nested indexing and return a data.frame object
list5[[1]][ , "Second", drop=FALSE]
```

```
##    Second
## 1      -4
## 2      -3
## 3      -2
## 4      -1
## 5       0
## 6       1
## 7       2
## 8       3
## 9       4
## 10      5
```

```
# Check the length of a list
length(list5)
```

```
## [1] 3
```

```
# Add the 4th element to a list without a name
list5[[4]] <- 2
length(list5)
```

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

```
# Add the 5th element to a list with an assigned name
list5[[5]] <- 3:6
length(list5)
```

```
## [1] 5
```

```
# Check the name of each element in a list
names(list5)
```

```
## [1] "data.frame" "vector"     "list"       ""           ""
```

```
# Print all elements in a list
list5
```

```
## $data.frame
##    First Second      Sport
## 1      1     -4     Hockey
## 2      2     -3   Football
## 3      3     -2   Baseball
## 4      4     -1    Curling
## 5      5      0      Rugby
## 6      6      1   Lacrosse
## 7      7      2 Basketball
## 8      8      3     Tennis
## 9      9      4    Cricket
## 10    10      5     Soccer
##
## $vector
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $list
## $list[[1]]
## [1] "a" "b" "c"
##
## $list[[2]]
## [1] 3 4 5 6 7
##
##
## [[4]]
## [1] 2
##
## [[5]]
## [1] 3 4 5 6
```

Note: In most cases, adding new element to a list is not going to create efficient problem. However, as a list expands, especially with data.frame or list, it would increase the computational complexity. Therefore, it's suggested to set the length of the list at the beginning and fill it with the proper index.

2.3 - Matrix

Matrix is an important mathematical structure in statistical computation. It's similar to data.frame, which represents a collection of numbers arranged in an order of rows and columns. The only different is that matrix requires all elements to be the same data type, which is usually numeric. Same as vector, matrix calculation is based on vectorized computation. The data.frame attributes, such as nrow(), ncol(), and dim(), also apply to matrix. Here are some examples to demonstrate:

```
# Create a 5x2 matrix with elements 1 to 10
A <- matrix(1:10, nrow=5)
A
```

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

```
# Create a 5x2 matrix with elements 21 to 30
B <- matrix(21:30, nrow=5)
B
```

```
##      [,1] [,2]
## [1,]   21   26
## [2,]   22   27
## [3,]   23   28
## [4,]   24   29
## [5,]   25   30
```

```
# Create a 2x10 matrix with elements 21 to 40
C <- matrix(21:40, nrow=2)
C
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]   21   23   25   27   29   31   33   35   37    39
## [2,]   22   24   26   28   30   32   34   36   38    40
```

```
# Check the row attribute of Matrix A
nrow(A)
```

```
## [1] 5
```

```
# Check the column attribute of Matrix A
ncol(A)
```

```
## [1] 2
```

```
# Check the dimension of Matrix A
dim(A)
```

```
## [1] 5 2
```

```
# Adding Matrix A and B (must be the same dimension)
A + B
```

```
##      [,1] [,2]
## [1,]   22   32
## [2,]   24   34
## [3,]   26   36
## [4,]   28   38
## [5,]   30   40
```

```
# Multiplying Matrix A and B (not matrix multiplication)
A * B
```

```
##      [,1] [,2]
## [1,]   21  156
## [2,]   44  189
## [3,]   69  224
## [4,]   96  261
## [5,]  125  300
```

```
# Check if the elements are the same in two matrices
A == B
```

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

Matrix Multiplication is often used in mathematical operations. The requirment for matrix multiplication is that the number of rows of the first matrix needs to match with the number of columns of the second matrix. To demonstrate, we transpose matrix B in the following example and multiple the two matrices.

```
# Transpose Matrix B for matrix multiplication
A %*% t(B)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]  177  184  191  198  205
## [2,]  224  233  242  251  260
## [3,]  271  282  293  304  315
## [4,]  318  331  344  357  370
## [5,]  365  380  395  410  425
```

Similar to data.frame, matrix can also assign the names to the rows and columns.

```
# Check column names of Matrix A
colnames(A)
```

```
## NULL
```

```
# Check row names of Matrix A
rownames(A)
```

```
## NULL
```

```
# Assign column and row names to Matrix A
colnames(A) <- c("Left", "Right")
rownames(A) <- c("1st", "2nd", "3rd", "4th", "5th")

# Check the assigned column and row names of Matrix A
colnames(A)
```

```
## [1] "Left"  "Right"
```

```
rownames(A)
```

```
## [1] "1st" "2nd" "3rd" "4th" "5th"
```

```
# Assign column and row names to Matrix B
colnames(B) <- c("First", "Second")
rownames(B) <- c("One", "Two", "Three", "Four", "Five")

# Check the assigned column and row names of Matrix B
colnames(B)
```

```
## [1] "First"  "Second"
```

```
rownames(B)
```

```
## [1] "One"   "Two"   "Three" "Four"  "Five"
```

```
# Assign column and row names to Matrix C
colnames(C) <- LETTERS[1:10]
rownames(C) <- c("Top", "Bottom")

# Check the assigned column and row names of Matrix C
colnames(C)
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
```

```
rownames(C)
```

```
## [1] "Top"    "Bottom"
```

R has two special vectors called "letters" and "LETTERS". They are used to create vector of lowercase letters and uppercase letters. In the previous example, LETTER[1:10] represents a vector of uppercase letters A~J.

When we are tranposing a matrix with assigned row and column names, the names will be transposed to match with the data.

```
# Transpose Matrix A
t(A)
```

```
##       1st 2nd 3rd 4th 5th
## Left    1   2   3   4   5
## Right   6   7   8   9  10
```

```
# Matrix multiplication for Matrix A and C
A %*% C
```

```
##       A   B   C   D   E   F   G   H   I   J
## 1st 153 167 181 195 209 223 237 251 265 279
## 2nd 196 214 232 250 268 286 304 322 340 358
## 3rd 239 261 283 305 327 349 371 393 415 437
## 4th 282 308 334 360 386 412 438 464 490 516
## 5th 325 355 385 415 445 475 505 535 565 595
```

2.4 - Array

Array is the R data object that can store data in more than two dimensions. Array can store only the same data type. An array is created using the array() function in R. Similar to a vector, we can use the square brackets to extract elements from an array. The first argument refers to the row of each element, the second argument refers to the column of each element, and finally the third argument refers to the index of the element. Here is an example to demonstrate:

```r
# Create a array with 2 two dimensional vectors
theArray <- array(1:12, dim=c(2, 3, 2))
theArray
```

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

```r
# Extract the first row from each vector element by index
theArray[1, , ]
```

```
##      [,1] [,2]
## [1,]    1    7
## [2,]    3    9
## [3,]    5   11
```

```r
# Extract the first row from the first vector element by index
theArray[1, , 1]
```

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

```r
# Extract the first vector by index
theArray[ , , 1]
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Summary:

Most people do not realize the importance of the data type they are dealing with in their project. In fact, it could be frustrating when you are trying to debug some the errors with different statistical package in R. Most of the time, the errors are related to the type of data required in those packages. Therefore, it's very important to learn all the different data structures in R before using different statistical packages.