Data Structure, Data Types and Data Manipulation in R

Baburao Kamble
August,2014

1 Introduction to Data Structure and Data Types

A data type is a (potentially infinite) class of concrete objects that all share some property. For example, "integer" is a data type containing all of the infinitely many integers, "string" is a data type containing all of the infinitely many strings, and "32-bit integer" is a data type containing all integers expressible in thirty-two bits.
R has a wide variety of data types including double, integer, complex, logical, character, factor, dates and times, Missing data (NaN) and Infinity

A data structure is an abstract description of a way of organizing data to allow certain operations on it to be performed efficiently. For example, a binary tree is a data structure, as is a Fibonacci heap, AVL tree, or skiplist. Theoreticians describe data structures and prove their properties in order to show that certain algorithms or problems can be solved efficiently under certain assumptions.

R has a wide variety of data structure including vectors (numerical, character, and logical), matrices, arrays, data frames, Time-Series objects, string and lists

Data type can be regarded as data structure because we can define our own data type by using typed function. Data types are the category in which the variable are listed to hold the specific value and data structures are the way in which the values and variables are stored in the memory. The use of a data type called structures allows a collection of values possibly of different types to be treated as a single item. Each data structure is built up from the basic data types of the underlying programming language using the available data structuring facilities, such as arrays & pointers

2 Data Types in R

2.1 Integer

Integers are natural numbers. Integers are whole number with no decimal point. It can have a leading sign (+/-). Integers can be used to represent counting variables, for example the population of country, or number of countries in South America.

```
>SouthAmericaCountries<- as.integer(12)
>is.integer(SouthAmericaCountries)
[1] TRUE
```

2.2 Double

The Double data type provides the largest and smallest possible magnitudes for a number.
Doubles are numbers like 145.335, 6.67 and 9.81. Doubles are used to represent continuous variables like the weight (160.45 lb) or length (145.55 miles).

```
> x<-145.335
> typeof(x)
[1] "double"
> y<-6.67
> typeof(y)
[1] "double"
```

2.3 Complex

A complex variable or value is usually represented as a pair of floating point numbers. R support a complex data type and provide special syntax for building such values, and extend the basic arithmetic operations ('+', '−', '×', '÷') to act on them.
In statistical data analysis you will not need them often for scientific calculations.
A complex number is a number that can be expressed in the form a + bi, where a and b are real numbers and i is the imaginary unit, which satisfies the equation i2 = −1.[1]

```
> x <- 1
```

```
> y <- 1
> z <- complex(real = x, imaginary = y)
> z
[1] 1+1i
> as.complex(-1)
```

2.4 Logical

A logical data item is a primitive data structure that can assume the value of either "TRUE" or "False"

Most commonly used logical operators are *and, or and not* represented by &, — and !, respectively.

```
> x<-100
> y <- x > 10
> y
[1] TRUE
> y<- (30>x) & (x <= 100)
> y
[1] FALSE
```

2.5 Character

A character object is used to represent string values in R. We convert objects into character values with the as.character() function:

A character object is represented by a collection of characters between double quotes (" ").

For example: "u" and "UNL".

```
> x<-c("UNL")
> x
[1] "UNL"
> y <- c("U","N","L")
> y
[1] "U" "N" "L"
```

The double quotes indicate that we are dealing with an object of type 'character'.

2.6 Factor

Conceptually, factors are variables in R which take on a limited number of different values; such variables are often referred to as categorical variables. One of the most important uses of factors is in categorical data analysis. Factors in R are stored as a vector of integer values with a corresponding set of character values to use when the factor is displayed.

Factor objects can be created from character objects or from numeric objects, using the function factor.

```
>Wildlife<-
c("lion","leopard","tiger","leopard","l
eopard","tiger","lion","leopard")
>Wildlife
[1]   "lion"          "leopard"   "tiger"
"leopard"  "leopard"  "tiger"     "lion"
"leopard"
>FactorWildlife<-factor(Wildlife)
>FactorWildlife
[1] lion      leopard tiger     leopard
leopard tiger   lion    leopard
Levels: leopard lion tiger
```

You can change these levels at the time you create a factor by passing a vector with the new values through the labels= argument.

```
>MyWLdata<-     factor(Wildlife,labels<-
c("TypeI","TypeII","TypeIII"))
> MyWLdata
[1] TypeII  TypeI   TypeIII TypeI   TypeI
TypeIII TypeII  TypeI
Levels: TypeI TypeII TypeIII
```

```
>MyWLdataNumbers<-
factor(Wildlife,labels=c(1,2,3))
> MyWLdataNumbers
[1] 2 1 3 1 1 3 2 1
Levels: 1 2 3
```

You can transform factor variables to double or integer variables using the as.double or as.integer function.

```
>FactorWildlife.numeric<as.double(Facto
rWildlife)
> FactorWildlife.numeric
[1] 2 1 3 1 1 3 2 1
```

The 1 is assigned to the leopard level, only because alphabetically leopard comes first.

2.7 Dates and Times

The builtin function in R *as.Date* handles calendar dates only (without times).

```
>Adate <- c("01-01-1990", "28-08-1999")
>Adate
[1] "01-01-1990" "28-08-1999"
> Bdate <- as.Date(Adate, "%d-%m-%Y")
> Bdate
[1] "1990-01-01" "1999-08-28"
> data.class(Bdate)
[1] "Date"
```

You can add a number to a date object, the number is interpreted as the number of day to add to the date.

```
> Bdate+123
[1] "1990-05-04" "1999-12-29"
```

You can subtract one date from another, the result is an object of class 'difftime'

```
diffBdate = Bdate[2] - Bdate[1]
>diffBdate
Time difference of 3526 days
```

The POSIXct and POSIXlt classes allow for dates and times with control for time zones.

```
>TimeData1<-as.POSIXct("2010-01-20
15:34:55")
> TimeData1
[1] "2010-01-20 15:34:55 CST"
```

Change Timezone

```
>TimeData1<-as.POSIXct("2010-01-20
15:34:55", tz="Europe/London")
> TimeData1
[1] "2010-01-20 15:34:55 GMT"
>weekdays(TimeData1)
[1] "Wednesday"
```

2.8 Missing data and Infinite values

Missing data in R appears as NA(*Not Available*). NA is not a string or a numeric value, but an indicator of missing value. We can create vectors with missing values

```
>x<-c(1,2,3,4,5,NA)
>is.na(x)
[1] FALSE FALSE FALSE FALSE FALSE  TRUE
>mean(x)
[1] NA
>mean(x, na.rm=TRUE)
[1] 3
```

Recoding missing values

Weather data or some data formats often use codes such as -9999 for *not applicable (NA)* values. There is need to recode -9999 to NA.

```
>y<-c(1,2,3,4,5,8,9,10,11,-9999)
>y[y==-9999] <- NA
>y
 [1]  1  2  3  4  5  8  9 10 11 NA
```

Infinite values are represented by Inf or –Inf.

In R there are two function to check infinite values, is.finite and is.infinite. These two functions return a vector of the same length as original, indicating which elements are finite (not infinite and not missing) or infinite.

The first function returns TRUE if the number is finite; the second one returns TRUE if the number is infinite.

```
>x<-(10^(290:320))
>is.finite(x)
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE
[17]  TRUE  TRUE  TRUE FALSE FALSE FALSE
FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
FALSE FALSE FALSE
>is.infinite(x)
 [1] FALSE FALSE FALSE FALSE FALSE FALSE
FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
FALSE FALSE FALSE FALSE
[17] FALSE FALSE FALSE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE
```

3 Data Structure in R

Data structure are the ways of organizing data in R. Data structures play a central role in the data analysis using R. You interact with data structures even more often than with function written in R. In addition, data structures are essential building blocks for efficient data analysis in R.

Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. R has the following structures:

- Vector
- Matrices and arrays
- Data frame
- List

3.1 Vectors

Vectors can be thought of as contiguous cells containing data. Vectors are one-dimensional arrays that can hold numeric data, character data, or logical data.

R has six basic ('atomic') vector types: logical, integer, real, complex, string (or character) and raw. The modes and storage modes for the different vector types are listed in the following table.

typeof	mode	storage.mode
logical	logical	logical
integer	numeric	integer
double	numeric	double
complex	complex	complex
character	character	character
raw	raw	raw

4444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444

The Vector ADT extends the notion of array by storing a sequence of arbitrary objects
An element can be accessed, inserted or removed by specifying its rank (number of elements preceding it)
An exception is thrown if an incorrect rank is specified (e.g., a negative rank)

```
a <- c(1,2,-2, 4,4,5,6,7)
b <- c("Lion","Goat","Cow")
c <- c(TRUE, FALSE, TRUE, FALSE)
```

a is numeric vector, b is a character vector, and c is a logical vector.

3.2 Matrices and arrays

Matrices are nothing more than 2-dimensional vectors.

```
>MyMatrix=matrix(data=c(9,2,3,4,5,6,2,3,4),ncol=3)
>MyMatrix
```

Arrays are similar to matrices but can have more than two dimensions. An array is a number of elements in a specific order. Arrays are one of the most efficient data structures in inserting, retrieving, indexing data as it does those function in O(1) time (1 CPU call).

```
>dim1<- c("A1","A2")
>dim2<- c("B1","B2","B3")
>dim3<- c("C1","C2","C3","C4")
>z<-array(1:24,c(2,3,4),
dimnames=list(dim1, dim2, dim3))
```

Arrays are a natural extension of matrices.

3.3 Data Frame

A data frame is more general than a matrix in that different columns can contain different modes of data (numeric, character, etc.).
Data frames, which we often think of and treat like arrays. But, a data.frame, df, is a list of vectors, v_i, and so it inherits some of the flexibility of lists.
In particular, a data frame looks very much like an array, but each column is a list element in the data frame. Since the columns are separate objects in data frames, it's possible to have the i column contain a character vector and the j column a logical vector, and so on. Thus, a data frame is a convenient way to store mixed data in R.

```
> ObservationID<-c(1, 2, 3, 4)
> Tmin<-c(25, 34, 28, 52)
> WindType<-c("Type1", "Type2", "Type1", "Type1")
>CropHealthStatus<-c("Poor", "Improved", "Excellent", "Poor")
>CropWeatherdata<-
data.frame(ObservationID,Tmin, WindType, CropHealthStatus)
```

Specifying elements of a data frame

```
> CropWeatherdata[1:2]
```

3.4 List

Lists are the most complex of the R data types. A list is a dynamically allocated structure where each list element may be any object. A data.frame is in fact a list with the restriction that each list element must be a vector, and some methods defined that let you access it like a matrix. Any data stored in a data.frame can be (and is) stored

in a list, but it is not the case that every list can be represented as a data.frame.

```
>g<-"My First List"
>h<-c(25,26,18,39)
>j<-matrix(1:10, nrow=5)
>k<-c("one", "two", "three")
>mylist<-list(title=g, ages=h, j, k)
```