



DATA STRUCTURES IN R

Pavan Kumar A
Senior Project Engineer
Big Data Analytics Team
CDAC-KP

DATA STRUCTURES IN R

- Types of data structures in R
 - **Vector** : It is the structure that can contain one or more values of a single type or mixed (characters, integers)
 - It is represented as one dimensional data
 - **Matrices** : It is the 2-dimensional representation of data.
 - **Arrays** : It can be more than 2-dimensional representation of data.
 - **Data Frames**: It is the rectangular 2-dimensional representation of data



DATA STRUCTURES IN R- INTEGER VECTORS

- Following functions are used to create the character vectors
 - `c()` : Concatenate (joining items end to end)
 - `seq()` : Sequence (Generating equidistant series of numbers)
 - `rep()` : Replicate (used to generate repeated values)
- `c()` examples

```
> c(42,57,12,39,1,3,4)
[1] 42 57 12 39 1 3 4
```
- You can also concatenate vectors of more than one element

```
> x <- c(1, 2, 3)
> y <- c(10, 20)
```



DATA STRUCTURES IN R- INTEGER VECTORS

- **seq()**: It is used to generate the series of numbers which is of equidistant
- It accepts three arguments
 - Start element
 - Stop element
 - Jump element

```
> seq(4,9) #It generates the numbers from 4 to 9, only 2 arguments are given  
[1] 4 5 6 7 8 9
```

```
> seq(4,10,2) #Three arguments are given, jump by 2 elements  
[1] 4 6 8 10
```



DATA STRUCTURES IN R- INTEGER VECTORS

- `seq()` vector creation is used in plotting the x and y axis in the graphical analysis.

- For example:

- If x-axis co-ordinates are being created as

`c(1.65, 1.70, 1.75, 1.80, 1.85, 1.90)`

- Then simply using following command, can create the same

Syntax :

`seq(from, to, by)`

`Seq (1.65, 1.90, 0.05)`

`> 4:9` *#exactly the same as `seq(4, 9)`*

`[1] 4 5 6 7 8 9`

`> sum(1:10)`

`[1] 55`



DATA STRUCTURES IN R- INTEGER VECTORS

- Another Example of **seq()** command, Here we are adding **length.out** argument for the **seq()** command

```
> seq(1,4,length.out=4)
[1] 1 2 3 4
> seq(1,4,length.out=3)
[1] 1.0 2.5 4.0
> seq(1,4,length.out=2)
[1] 1 4
> seq(1,6,length.out=3)
[1] 1.0 3.5 6.0
> seq(1,6,length.out=4)
[1] 1.000000 2.666667 4.333333 6.000000
> seq(1,6,length.out=5)
[1] 1.00 2.25 3.50 4.75 6.00
> |
```

from = “Starting Element”
to = “Ending Element”
 $by = ((to - from)/(length.out - 1))$

```
> seq(from=1, to=4, by=4)
[1] 1
> seq(from=1, to=4, length.out=4)
[1] 1 2 3 4
> |
```



DATA STRUCTURES IN R- INTEGER VECTORS

- `rep()`, is used to generate repeated values.
- It is used in two variants, depending on whether the second argument is a **vector** or a **single number**

```
> oops <- c(7,9,13)
```

```
> rep(oops,3) # It repeats the entire vector oops 3 times
```

```
[1] 7 9 13 7 9 13 7 9 13
```

```
> rep(oops,1:3)
```

```
[1] 7 9 9 13 13 13
```

Here, oops should be repeated by vector of 1:3 values.

Indicating that **7 should be repeated once, 9 twice, and 13 three times**



DATA STRUCTURES IN R- INTEGER VECTORS

- Look at following examples

```
> rep(oops,1:4)
```

```
Error in rep(anow, 1:4) : invalid 'times' argument
```

```
> rep(1:2,c(10,15))
```

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

```
> rep(1:2,each=10)
```

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

```
> rep(1:2,c(10,10))
```

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



DATA STRUCTURES IN R- INTEGER VECTORS

- Integer vectors : Indexing

```
> length(a)
[1] 100
> a[1]
[1] 201
> a[50]
[1] 250
> a[100]
[1] 300
> |

> a[1:10]
[1] 201 202 203 204 205 206 207 208 209 210
> a[11:20]
[1] 211 212 213 214 215 216 217 218 219 220
> a[1:5,57:59]
Error in a[1:5, 57:59] : incorrect number of dimensions
> a[c(1:5,57:59)]
[1] 201 202 203 204 205 257 258 259
> |
```

```
> a[-1:-99]
[1] 300
> a[-1:-98]
[1] 299 300
> a[-1:-97]
[1] 298 299 300
```



DATA STRUCTURES IN R- CHARACTER VECTORS

- **Character Vector**: A character vector is a vector of **text strings**, whose elements are specified and printed in quotes

```
> c("Huey", "Dewey", "Louie")  
[1] "Huey" "Dewey" "Louie"
```
- Single quotes or Double quotes can be used for strings

```
> c('Huey', 'Dewey', 'Louie')  
[1] "Huey" "Dewey" "Louie"
```
- "Huey", it is a string of four characters, not six.
- The quotes are not actually part of the string, they are just there so that the system can tell the difference between a string and a variable name.

DATA STRUCTURES IN R- CHARACTER VECTORS

- If you print a character vector, it usually comes out with **quotes** added to each element. There is a way to avoid this, namely to use the `cat()` function.

- For instance,

```
> cat(c("Huey", "Dewey", "Louie"))
```

```
Huey Dewey Louie
```



DATA STRUCTURES IN R- CHARACTER VECTORS

- **Quoting and escape sequences**

- If the strings itself contains some quotations, new line characters.
- This is done using **escape sequences**

- Here, **\n** is an example of an escape sequence.

- The backslash (****) is known as the escape character

- If you want to insert quotes with in the string, the **\"** is used. For example

```
> cat("What is \"R\"?\n")
```

```
What is "R"?
```



DATA STRUCTURES IN R- CHARACTER VECTORS

- Logical vectors can take the value **TRUE** or **FALSE**
- In input, you may use the convenient abbreviations T and F

```
> c(T,T,F,T)
```

```
[1] TRUE TRUE FALSE TRUE
```

```
> c("apple",F,"Orange",T)
```

```
[1] "apple" "FALSE" "Orange" "TRUE"
```

```
> c("apple","F","Orange","T")
```

```
[1] "apple" "F"      "Orange" "T"
```

```
> |
```



DATA STRUCTURES IN R- CHARACTER VECTORS

- Example of Character Vector: **Indexing**

```
> a<-c("Huey","Dewey","Louie")
> a
[1] "Huey" "Dewey" "Louie"
> a[1]
[1] "Huey"
> a[2]
[1] "Dewey"
> a[3]
[1] "Louie"
> a[-1]
[1] "Dewey" "Louie"
> a[-2]
[1] "Huey" "Louie"
> a[-3]
[1] "Huey" "Dewey"
> |

> s = c("aa", "bb", "cc", "dd", "ee")
> s[1:3]
[1] "aa" "bb" "cc"
> s[3:5]
[1] "cc" "dd" "ee"
> s[1,2,3]
Error in s[1, 2, 3] : incorrect number of dimensions
> s[c(1,2,3)]
[1] "aa" "bb" "cc"
> s[c(1,3)]
[1] "aa" "cc"
> s[c(1:3,5)]
[1] "aa" "bb" "cc" "ee"
> |
```



DATA STRUCTURES IN R- CHARACTER VECTORS

○ Missing values

- In many data sets, you may find missing values.
- We need to have some method to deal with the missing values

○ R allows vectors to contain a special **NA** value.

○ Result of computations done on NA will be NA

```
> c("Name1", "Name2", "Name3", NA, "Name4")  
[1] "Name1" "Name2" "Name3" NA      "Name4"  
> c("Name1", "Name2", "Name3", "NA", "Name4")  
[1] "Name1" "Name2" "Name3" "NA"    "Name4"  
> c("Name1", "Name2", "Name3", "NA", "Name4")|
```



DATA STRUCTURES IN R- COMBINATION OF INT AND CHAR

- Example of `c()`

```
> anow<-c(1,2,3)
> bnow<-c(4,5,6,"name1","name2")
> cnow<-c(7,8,9,"name3",NA)
> anow
[1] 1 2 3
> bnow
[1] "4"      "5"      "6"      "name1" "name2"
> cnow
[1] "7"      "8"      "9"      "name3" NA
> full<-c(anow,bnow,cnow)
> full
[1] "1"      "2"      "3"      "4"      "5"      "6"      "name1" "name2" "7"      "8"      "9"      "name3" NA
> |
```

- It is also possible to assign names to the elements

```
> xnow <- c(red="Huey", blue="Dewey", green="Louie")
> xnow
      red      blue      green
"Huey" "Dewey" "Louie"
> |
```



DATA STRUCTURES IN R- MATRICES AND ARRAYS

- **Matrix**: It is 2 dimensional representation of numbers.
- Matrices and arrays are represented as vectors with dimensions

```
> x <- 1:12
```

```
> dim(x) <- c(3,4) #The dim assignment function sets or changes the  
dimension attribute of x, causing R to treat the vector of 12 numbers as a 3 × 4  
matrix
```

```
> x <- 1:12  
> dim(x) <- c(3,4)  
> x
```

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

```
> |
```

```
> dim(x) <- c(4,4)  
Error in dim(x) <- c(4, 4) :  
  dims [product 16] do not match the length of object [12]  
> |
```



DATA STRUCTURES IN R- MATRICES AND ARRAYS

- Another way to create Matrix is simply by using `matrix()` function
- **Syntax**

```
matrix(data = NA, nrow = 1, ncol = 1,  
byrow = FALSE)
```

```
> matrix(1:12,nrow=3,ncol=4)  
      [,1] [,2] [,3] [,4]  
[1,]    1    4    7   10  
[2,]    2    5    8   11  
[3,]    3    6    9   12  
> matrix(1:12,nrow=3,ncol=3)  
      [,1] [,2] [,3]  
[1,]    1    4    7  
[2,]    2    5    8  
[3,]    3    6    9  
> |
```

```
> ## Creating Matrix and filling  
> ## elements by row wise  
> matrix(1:12,nrow=3,byrow=T)  
      [,1] [,2] [,3] [,4]  
[1,]    1    2    3    4  
[2,]    5    6    7    8  
[3,]    9   10   11   12  
> ## Creating Matrix and filling  
> ## elements by column wise  
> matrix(1:12,nrow=3,byrow=F)  
      [,1] [,2] [,3] [,4]  
[1,]    1    4    7   10  
[2,]    2    5    8   11  
[3,]    3    6    9   12  
> |
```

DATA STRUCTURES IN R- MATRICES AND ARRAYS

- You can “glue” vectors together, **columnwise** or **rowwise**, using the **cbind** and **rbind** functions.

- The **cbind()** : Column bind

- The **rbind()** : Row bind

```
> cbind(A=1:4,B=5:8,C=9:12)
```

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

```
> rbind(A=1:4,B=5:8,C=9:12)
```

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

```
> |
```

- Arrays are similar to matrices but can have more than two dimensions.
See **help(array)** for details



DATA STRUCTURES IN R- MATRICES

○ Matrix Operations

- We can extract the desired rows from the matrix created as shown
- Functions like **rowSums()** and **rowMeans()** are used to calculate the sum of all row elements and mean of all row elements respectively
- Functions like **colSums()** and **colMeans()** are used to calculate the mean of all column elements and mean of all column elements respectively

```
> a
      a b  c
[1,] 1 5  9
[2,] 2 6 10
[3,] 3 7 11
[4,] 4 8 12
> a[,1]
[1] 1 2 3 4
> a[,2]
[1] 5 6 7 8
> a[,3]
[1] 9 10 11 12
> a[3,]
      a  b  c
      3  7 11
> |
```



DATA STRUCTURES IN R-DATA FRAMES

- Data Frame is also **2-dimensional object** just like Matrix, for storing data tables.
- Here, different columns can have different modes (numeric, character, factor, etc).
- All data frames are rectangular and R will remove out any 'short' using **NA**
- **Creating Data Frame**

```
> d <- c(1,2,3,4)
> e <- c("red", "white", "red", NA)
> f <- c(TRUE,TRUE,TRUE,FALSE)
> mydata <- data.frame(d,e,f)
> names(mydata) <- c("ID","Color","Passed") # variable names
> mydata
  ID Color Passed
1  1   red  TRUE
2  2 white  TRUE
3  3   red  TRUE
4  4  <NA> FALSE
> |
```



DATA STRUCTURES IN R-DATA FRAMES

- **Error:** Here, in the second vector 'e', is a 3 element vector and 'd' and 'f' are 4 element vectors.
- **It is a collection of vectors (Integer/Character) of equal lengths**

```
> d <- c(1,2,3,4)
> e <- c("red", "white", "red")
> f <- c(TRUE,TRUE,TRUE,FALSE)
> mydata <- data.frame(d,e,f)
Error in data.frame(d, e, f) :
  arguments imply differing number of rows: 4, 3
> |
```

- Each column in the Data Frame can be a separate type of data. In the previous example '**mydata**' data frame, it is the combination of numerical, character and factor data types.



ACCESSING DATA FRAMES

- There are a variety of ways to identify the elements of a data frame. Here are few screenshots.



```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA> FALSE
> mydata[1:2]
  ID Color
1  1   red
2  2 white
3  3   red
4  4  <NA>
> mydata[c("ID", "Color")]
  ID Color
1  1   red
2  2 white
3  3   red
4  4  <NA>
```

```
> mydata[c("ID", "Passed")]
  ID Passed
1  1   TRUE
2  2   TRUE
3  3   TRUE
4  4  FALSE
> mydata$ID
[1] 1 2 3 4
> mydata$Color
[1] red   white red   <NA>
Levels: red white
> mydata$Passed
NULL
> mydata$Passed
[1] TRUE TRUE TRUE FALSE
> |
```

```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA> FALSE
> mydata[1,2:3]
  Color Passed
1   red   TRUE
> mydata[2,2:3]
  Color Passed
2 white   TRUE
> mydata[2,]
  ID Color Passed
2  2 white   TRUE
> mydata[,3]
[1] TRUE TRUE TRUE FALSE
> mydata[1,]
  ID Color Passed
1  1   red   TRUE
> |
```

BUILD-IN DATA FRAMES IN R

- R has some build-in datasets. 'mtcars' is one datasets

```
> dim(mtcars)
[1] 32 11
> 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 ...
> |
```



DATA STRUCTURES IN R-LISTS

- **Lists:** It is the collection of objects that fall under similar category.
- A list is not fixed in length and can contain other lists.

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc", "dd", "ee")
> b = c(TRUE, FALSE, TRUE, FALSE, FALSE)
> x = list(n, s, b, 3)
> x
[[1]]
[1] 2 3 5

[[2]]
[1] "aa" "bb" "cc" "dd" "ee"

[[3]]
[1] TRUE FALSE TRUE FALSE FALSE

[[4]]
[1] 3

> |
```



CREATING DATA SUBSETS

- R deals with huge data, not all of which is useful.
- Therefore, first step is to **sort out the data containing the relevant information**.
- Extracted data sets are further divided into small subsets of data.
- Function used for extracting the data is **subset()**.
- The following operations are used for subset the data.
 - **\$ (Dollar)** : Used to select the single element of the data.
 - **[] (Single Square Brackets)** : Used to extract multiple elements of data.



CREATING DATA SUBSETS

- We can extract (subset) the part of the data table based on some condition using `subset()` function
- **Syntax** `subset(dataset, function)`
- **Example**

```
## Age.At.Death Age.As.Writer Name Surname Gender Death
## 1 22 16 Jane Doe FEMALE 2015-05-10
## 4 41 36 Jane Austen FEMALE 1817-07-18
```

```
writer_names_df <- subset(writers_df, Age.At.Death <= 40 & Age.As.Writer >= 18)
writer_names_df <- subset(writers_df, Name == "Jane")
male_writers <- writers_df[Gender == "MALE",]
writers_df[1,3] <- NULL #making null value
```

CREATING SUBSETS IN VECTORS

- To create subsets in vectors, `subset()` or `[]` can be used

A simple vector

```
v<-c(1,5,6,4,2,4,2)
```

#Using subset function

```
subset(v,v<4)
```

Creates the subset of numbers greater than 4 using
subset() function

#Using square brackets

```
v[v<4]
```

Creates the subset of numbers greater than 4 using []
brackets

#Another vector

```
t<-c("one", "one", "two", "three", "four", "two")
```

Remove "one" entries

```
subset(t, t!="one")
```

Creates the subset of texts after removing the word, "one"
using subset() function

```
t[t!="one"]
```

Creates the subset of texts after removing the word, "one"
using [] function

CREATING SUBSETS IN VECTORS

- Execution of code on R console

```
> v
[1] 1.0 3.0 0.2 1.5 1.7
> v[v>1]
[1] 3.0 1.5 1.7
> v[v>2]
[1] 3
> subset(v,v>2)
[1] 3
> subset(v,v>1)
[1] 3.0 1.5 1.7
> t<-c("one","two","three","three","one")
> t
[1] "one"    "two"    "three"  "three"  "one"
> t[t!="one"]
[1] "two"    "three"  "three"
> subset(t, t!="one")
[1] "two"    "three"  "three"
> |
```



CREATING SUBSETS IN DATA FRAMES

- Data Frames subsets can also be done using `subset()` and `[]` function

```
> sample1
```

	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

```
> sample1[sample1$mpg=="21",]
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4

```
> subset(sample1, mpg=="21")
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4



CREATING SUBSETS IN DATA FRAMES

- Data Frames subsets can also be done using `subset()` and `[]` function

```
> sample1[sample1$cyl<6,]  
      mpg  cyl disp  hp drat   wt  qsec vs am gear carb  
Datsun 710 22.8   4  108  93 3.85 2.32 18.61 1  1   4    1  
> subset(sample1, cyl<6)  
      mpg  cyl disp  hp drat   wt  qsec vs am gear carb  
Datsun 710 22.8   4  108  93 3.85 2.32 18.61 1  1   4    1  
> |
```



THANK YOU !!!

