

TMC 204

Statistical Data Analysis with R

Unit 5

Functions in R contd...

Presented By : Aditya Joshi

Asst. Professor

Department of Computer Application

Graphic Era Deemed to be University

18-04-2020

R Vector Functions:

They are the functions that can be applied over R vectors.

Functions that we use in R vectors are known as the vector functions.

For example: `rep()`, `seq()`, `using all()` and `any()`, more on `c()` etc.

1. R rep() Function:

`rep()` is used for replicating the values in `x`. The two common cases that exhibit faster-simplified versions are `rep.int` and `rep_len`. Furthermore, these functions are not generic.

Let's now discuss how we can apply this `rep()` to any vector with the help of examples.

1.1. How to repeat vectors in R

You can use the `rep()` function in several ways if you want to repeat the complete vector.

For example:

a) To repeat the vector `c(0, 0, 7)` three times, use this code:

```
> rep(c(0, 0, 7), times = 4)
```

```
[1] 0 0 7 0 0 7 0 0 7 0 0 7
```

b) We can also repeat every value by specifying each argument, like this:

```
> rep(c(2, 4, 2), each = 2)
```

```
[1] 2 2 4 4 2 2
```

c) For each value, we can tell R how often it has to repeat:

```
> rep(c(0, 7), times = c(4,3))
```

```
[1] 0 0 0 0 7 7 7
```

d) In `seq`, we use the argument, `length.out` to define R. It will repeat the vector until it reaches the specified length, even if the last repetition is incomplete.

```
> rep(1:3,length.out=9)
```

```
[1] 1 2 3 1 2 3 1 2 3
```

2. R seq() Function:

It generates regular sequences.

2.1. How to create vectors in R

a) In order to use integers to create vectors:

For example:

To create a list of vectors over a specified range, we use the colon (:) symbol.

The code 1:5 gives you a vector with the numbers 1 to 5, and 2:-5 create a vector with the numbers 2 to -5.

b) Using the seq(), we make steps in a sequence. Seq() function is used to describe the intervals by which numbers should decrease or increase.

For example:

In R, a vector with numbers 4.5 to 3.0 in steps of 0.5.

```
> seq(from = 4.5, to = 3.0, by = -0.5)
```

```
[1] 4.5 4.0 3.5 3.0
```

c) You can specify the length of the sequence by using the argument, length.out. Afterwards, R can calculate the step size by itself.

For example:

You can make a vector of nine values going from -2.7 to 1.3 like this:

```
> seq(from = -2.7, to = 1.3, length.out = 9)
```

```
[1] -2.7 -2.2 -1.7 -1.2 -0.7 -0.2  0.3  0.8  1.3
```

3. R any() Function

It takes the set of vectors and returns a set of logical vectors, in which at least one of the value is true.

3.1. Usage of R any() Function

Check if any or all the elements of a vector are TRUE. Both functions also accept many objects.

`any(..., na.rm=FALSE)`

Here,

... means one or more R objects that need to be checked.

na.rm means state whether NA values should be ignored or not.

4. R all() Function

It takes the set of vectors and returns a set of logical vectors, in which all of the values are TRUE.

4.1. Usage of R all() Function

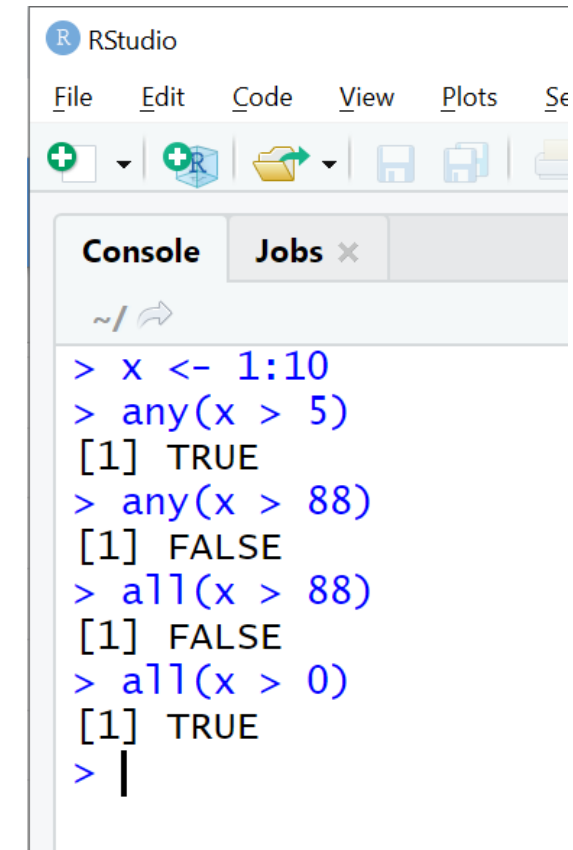
`all(..., na.rm=FALSE)`

Here,

... means one or more R objects that need to be checked.

na.rm means state whether NA values should be ignored or not.

The `any()` and `all()` functions are shortcuts because they report any or all of their arguments as TRUE.



```
RStudio
File Edit Code View Plots Se
+ R
Console Jobs x
~/
> x <- 1:10
> any(x > 5)
[1] TRUE
> any(x > 88)
[1] FALSE
> all(x > 88)
[1] FALSE
> all(x > 0)
[1] TRUE
> |
```

Suppose that R executes the following:

```
any(x > 5)
```

It first evaluates $x > 5$:

```
(FALSE, FALSE, FALSE, FALSE, FALSE)
```

`any()` function reports whether any of those values are TRUE, while `all()` function works and reports if all the values are TRUE.

Numeric and Character Functions in R:

we will learn about R built-in functions, in which we will focus on different types of numeric and character functions in R. Along with this, we will also understand different types of properties of character functions.

R Built-in Functions

We use R built-in functions to perform almost everything in R. Here we are only referring to numeric and character functions that are generally used in creating or recording variables.

R Numeric Functions

Let us see R Numeric functions:

Function

abs(x)

ceiling(x)

sqrt(x)

floor(x)

log(x)

trunc(x)

round(x, digits=n)

log10(x)

signif(x, digits=n)

exp(x)

cos(x), sin(x), tan(x)

Description

absolute value

ceiling(3.475) is 4

square root

floor(3.475) is 3

natural logarithm

trunc(5.99) is 5

round(3.475, digit=2) is 3.48

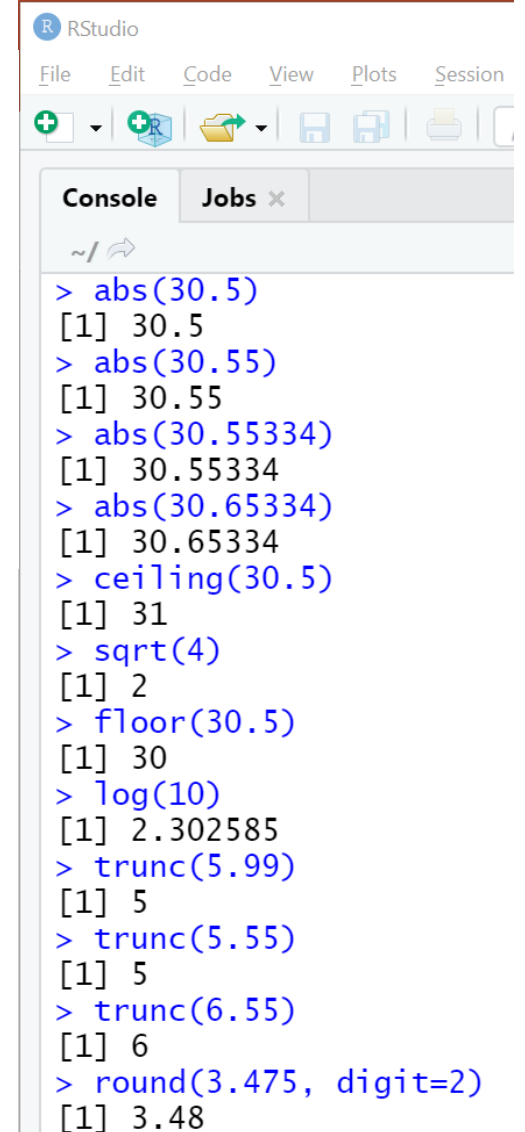
common logarithm

signif(3.475, digit=2) is 3.5

e^x

also acos(x), cosh(x), acosh(zx) etc

```
> log10(10)
[1] 1
> log10(2)
[1] 0.30103
> signif(3.475, digit=2)
[1] 3.5
> signif(3.475, digit=1)
[1] 3
> signif(3.475, digit=3)
[1] 3.48
> exp(3)
[1] 20.08554
```



The screenshot shows the RStudio interface. The top menu bar includes File, Edit, Code, View, Plots, and Session. Below the menu bar is a toolbar with icons for adding files, saving, and other functions. The main pane is the Console, which displays the following R code and its output:

```
> abs(30.5)
[1] 30.5
> abs(30.55)
[1] 30.55
> abs(30.55334)
[1] 30.55334
> abs(30.65334)
[1] 30.65334
> ceiling(30.5)
[1] 31
> sqrt(4)
[1] 2
> floor(30.5)
[1] 30
> log(10)
[1] 2.302585
> trunc(5.99)
[1] 5
> trunc(5.55)
[1] 5
> trunc(6.55)
[1] 6
> round(3.475, digit=2)
[1] 3.48
```


Character Functions in R

Let us see various R character functions:

1. `grep(pattern, x , ignore.case=FALSE, fixed=FALSE)`

Description:

Search for a pattern in x.

If `fixed =FALSE`, then the pattern is a regular expression.

If `fixed=TRUE`, then the pattern is a text string.

Returns matching indices.

```
> grep("A", c("b", "A", "c"), fixed = TRUE)
```

```
[1] 2
```

2. `substr(x, start=n1, stop=n2)`

Description:

Extract or replace substrings in a character vector.

```
> string = "abcdef"
```

```
> substr(string,2,4)
```

```
[1] "bcd"
```

```
> substr(string,2,4) <- "2222"
```

```
> string
```

```
[1] "a222ef"
```

3. strsplit

```
> strsplit("abc", "")
```

```
[[1]]
```

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

4. sub(pattern, replacement, x, ignore.case = FALSE, fixed=FALSE)

Description:

Find a pattern in x and replace it with the replacement text.

If a fixed = FALSE, then a pattern is a regular expression.

If fixed = TRUE, then a pattern is a text string.

```
> sub("\\s",".","GraphicEra")
```

```
[1] "GraphicEra"
```

```
> sub("\\s",".","Graphic Era")
```

```
[1] "Graphic.Era"
```

5. toupper(string)

Description:

Uppercase

```
> toupper(string)
```

```
[1] "A222EF"
```

6. tolower(string)

Description:

Lowercase

```
> tolower(string)
```

```
[1] "a222ef"
```

7. paste(..., sep="")

Description:

Concatenate strings after using sep string to separate them.

```
> paste("string", 1:3, sep = "")
```

```
[1] "string1" "string2" "string3"
```

```
> paste("string", 1:3, sep = "M")
```

```
[1] "stringM1" "stringM2" "stringM3"
```

```
> paste("Today is", date())
```

```
[1] "Today is Sat Apr 18 15:32:03 2020"
```

R Character Function – Create Strings

- In R, we store strings in a character vector. We can create strings with a single quote/double quote.

- **For example –**

```
> x="R Programming at Graphic Era"
```

1. Convert Object into Character type

We use an `as.character` function that converts arguments to a character type.

For example – We are storing 20 as a character.

```
> Y = as.character(20)
```

```
> class(Y)
```

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

2. Check the character type

```
> x="R Programming at Graphic Era"
```

```
> is.character(x)
```

```
[1] TRUE
```

3. Concatenate Strings in R

In order to join the two strings (also known as concatenation), we make use of the paste function. It is one of the most important strings manipulation task. Every analyst performs it almost daily to structure the data.

Paste Function Syntax:

paste (objects, sep = " ", collapse = NULL)

- sep – Keyword which denotes a separator or delimiter.
- default separator – A single space.
- collapse – Keyword which is used to separate the results.

```
> x="R Programming"
```

```
> y="at Graphic Era"
```

```
> paste(x,y)
```

```
[1] "R Programming at Graphic Era"
```

4. String Formatting in R

Suppose the value is stored in fraction and you need to convert it to percentage. In order to format our string in a C-style, sprintf function is used.

Sprintf Function Syntax:

```
sprintf(fmt, ...)
```

The keyword fmt denotes string format. The numbers and letters in the format start with the % symbol.

```
> x = 0.25
```

```
> sprintf("%.2f%%",x*100)
```

```
[1] "25.00%"
```

5. Extract or replace substrings in R

Extract

substr(x, starting position, end position)

```
> name = "abcdef"
```

```
> substr(name,1,4)
```

```
[1] "abcd"
```

Replace

substr(x, starting position, end position) = Value

```
> substr(name,1,3) = "111"
```

```
> name
```

```
[1] "111def"
```

6. String Length

The `nchar` function is used to compute the length of a character value.

```
> x = "c++ at Graphic Era"
```

```
> nchar(x)
```

```
[1] 18
```

7. Extract word from a string

Suppose you need to pull a first or last word from a character string.

word Function Syntax (Library : `stringr`)

`word(string, position of word to extract, separator)`

```
> name="c++ at Graphic Era"
```

```
> library(stringr)
```

```
> word(name,1)
```

```
[1] "c++"
```

In the example above, '1' denotes the first word to be extracted from a string.

Extract Last Word

```
> word(name,-1)
```

```
[1] "Era"
```

In the example above, '-1' denotes the first word to be extracted from the right of the string.

8. Convert Character to Uppercase / Lowercase /Propercase

```
> name="Graphic Era"
```

```
> tolower(name)
```

```
[1] "graphic era"
```

```
> toupper(name)
```

```
[1] "GRAPHIC ERA"
```

```
> name="GRAPHIC ERA"
```

```
> library(stringr)
```

```
> str_to_title(name)
```

```
[1] "Graphic Era"
```

9. Repeat the character N times

We can use strrep base R function to repeat the character N times.

```
> strrep(name, 5)
```

```
[1] "GRAPHIC ERAGRAPHIC ERAGRAPHIC ERAGRAPHIC ERAGRAPHIC ERA"
```


10. Find String in a Character Variable

The `str_detect()` function helps to check whether a substring exists in a string. It is equal to 'contain' function of SAS. It returns TRUE/FALSE against each value.

```
> names = c("Machine Learning", "Hadoop", "C++", "R Programming")
```

```
> library(stringr)
```

```
> str_detect(names, "Machine Learning")
```

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

11. Splitting a Character Vector

In the case of text mining, it is required to split a string to calculate the used keywords in the list. We use 'strsplit()' in base R to perform this operation.

```
> sentence <- "Big Data at Graphic Era"
```

```
> strsplit(sentence, "")
```

```
[[1]]
```

```
[1] "B" "i" "g" " " "D" "a" "t" "a" " " "a" "t" " " "G" "r" "a" "p" "h" "i" "c" " " "E"
```

```
[22] "r" "a"
```