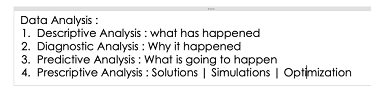
Instructor: Nimisha Pandey

Google Drive Link : <https://drive.google.com/drive/folders/1bY7fssaEixjKYoMysNcF-xxysQHiagQH>

Community Link: <https://community.simplilearn.com/threads/ds-with-r-nimisha-pandey-may-25.51200/>





# Comments in R

Comments in R is done using #

# This is a comment

# Tips

1. To clear the console – Press Ctrl + L or click the broom symbol 
2. NA is a way to represent a missing value. It’s class is “logical”
3. Workspace Image – Rstudio Desktop. If you close out Rstudio Desktop edition, it will ask you if you want to save your current workspace image. Saving it will preserve the variables you have created. Else with every new session, the variables are lost.

# Identifier

1. It should start with either a character or a period (.)
2. Then numbers, underscore, period are allowed
3. If a variable name is started with a period (.) it should not be immediately followed by a number.
4. Reserve words cannot be used (if, break, next, continue etc.)
5. R is case sensitive

# Assignment of value

a <- 25 (Official Way)

Assigning the value of 25 to variable “a”

Other forms of assignment:

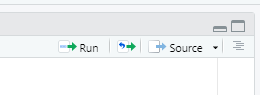
25 -> a

a = 25

# Running the code

Select the code and press Ctrl + Enter or Command + Enter

Or use the Run button



# Basic datatypes in R

1. Numeric (With or without decimal)
2. Integer (Explicit integers)
3. Character
4. Logical (TRUE | FALSE | T | F) It has to be in capital

a <- 25 (a is Numeric)

a <- 25L (a is now Integer)

class() is used to know the datatype of the variable

> a <- 10

> class(a)

[1] "numeric"

> b <- 25L

> class(b)

[1] "integer"

# Operators in R

1. Arithmetic operator (+, - , \* , /)
   1. +, - , \* , /
   2. ^ for power (2^3) = 8
   3. %% for Modulus – Remainder of a division
   4. %/% for Integer divide (Quotient of a division)
2. Relational operators
   1. == for equals to
   2. !=
   3. >, >=, <, <=
3. Logical Operators
   1. && (Logical and), & (bitwise and) - for “and”
   2. || , | for “OR”

# Basic data structures

## Vector

1. Vector – one dimensional and it is homogeneous

prime\_numbers <- c(2,3,5,7,11,13,17,19,23,29,31)

> class(prime\_numbers)

[1] "numeric"

But if I do this, see the change:

> prime\_numbers <- c(2,3,5,7,11,13,17,19,23,29,"A")

> class(prime\_numbers)

[1] "character"

> prime\_numbers

[1] "2" "3" "5" "7" "11" "13" "17" "19" "23" "29" "A"

Everything is now changed to character

Creating some mixed vectors – see the trend

> mix\_1 <- c(1,2L,"A",T)

> mix\_1

[1] "1" "2" "A" "TRUE"

> class(mix\_1)

[1] "character"

> mix\_2 <- c(1,TRUE)

> mix\_2

[1] 1 1

> class(mix\_2)

[1] "numeric"

> mix\_3 <- c("A",TRUE)

> mix\_3

[1] "A" "TRUE"

> class(mix\_3)

[1] "character"

> mix\_4 <- c(1.25,3,4L,TRUE)

> mix\_4

[1] 1.25 3.00 4.00 1.00

> class(mix\_4)

[1] "numeric"

Create a vector from 1 to 20 in sequence:

> c(1:20)

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

## Sequence

seq() creates a sequence a number. It returns a vector.

seq(start,end,[step]) also creates a sequence of numbers

> seq(1,20)

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

> c(seq(1,20))

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

seq() is also used to generate a series

Create a series from 2 to 50 for even numbers

> seq(2,50,2)

[1] 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50

Create a sequence from 2 to 5 by skipping 3 numbers

> seq(2,50,3)

[1] 2 5 8 11 14 17 20 23 26 29 32 35 38 41 44 47 50

?seq() will show the help page

### Examples

seq(0, 1, length.out = 11)

seq(stats::rnorm(20)) # effectively 'along'

seq(1, 9, by = 2) # matches 'end'

seq(1, 9, by = pi) # stays below 'end'

seq(1, 6, by = 3)

seq(1.575, 5.125, by = 0.05)

seq(17) # same as 1:17, or even better seq\_len(17)

> seq(1, 9, by = 2)

[1] 1 3 5 7 9

> seq(1, 9, by = pi)

[1] 1.000000 4.141593 7.283185

> seq(1, 9, by = 3)

[1] 1 4 7

Reverse sequence

seq(10,1)

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

Incrementing by decimal values

> seq(1,2,0.1)

[1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0

> seq(1,100)

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

[20] 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38

[39] 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57

[58] 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76

[77] 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95

[96] 96 97 98 99 100

The yellow highlighted number is the nth element in that row. The number of elements in one row depends on the screen size.

Character has the largest space in memory.

# Class Conversion

> mix\_4

[1] 1.25 3.00 4.00 1.00

> as.integer(mix\_4)

[1] 1 3 4 1

Class conversion is done using **as.classfunction()**

Consider the vector “a” shown below and notice what happens when we try to convert this to integer

> a <- c(1,2.567,TRUE,FALSE,"ABC","23")

> as.integer(a)

[1] 1 2 NA NA NA 23

Warning message:

NAs introduced by coercion

Note that even TRUE and FALSE are converted to NA. This has happened because a character was present in the vector. This is because of the presence of character “ABC” or “23” present in the vector. As a result everything was converted to character as “1”,”2.567”, “TRUE”, “FALSE”, “ABC”,”23”. That is why “TRUE” (treated as character) could not be converted to number.

as.integer(TRUE) will give you 1 as output.

Consider the below example and you will see that now TRUE and FALSE are converted to 1 and 0 respectively.

> as.integer(c(1,1.2,TRUE,FALSE))

[1] 1 1 1 0

> class(NA)

[1] "logical"

# Basic working with Vectors

length() gives you the number of elements in a vector

> temp <- c(34,32,30,29,28,31,36,39,41,42,40,36)

> length(temp)

[1] 12

To access elements of the vector

temp[1] – the first element of the vector “temp”

Note: index in R starts from 1

SLICING in Vector

To get the values from 1st to 6th element including both of them

> temp[1:6]

[1] 34 32 30 29 28 31

> temp[5:15]

[1] 28 31 36 39 41 42 40 36 NA NA NA

Since “temp” originally had only 12 elements, the 13th, 14th and 15th are printed as NA

Accessing random elements

Let’s say we want 1st element, then 9th element, then 4th element, then 7th element.

For that first create a vector of indices. And pass this vector as an index to the “temp” vector

temp[c(1,9,4,7)]

> temp[c(1,9,4,7)]

[1] 34 41 29 36

This will not work

> temp[1,9,4,7]

Error in temp[1, 9, 4, 7] : incorrect number of dimensions

Remember last index of the vector = length(vector)

First index of the vector = 1

Extract every alternate values from the vector “temp”

> temp[seq(1,length(temp),2)]

[1] 34 30 28 36 41 40

## Negative index in vector

Using a negative index, it will skip that index position value and rest the rest of the vector

> temp

[1] 34 32 30 29 28 31 36 39 41 42 40 36

> temp[-2]

[1] 34 30 29 28 31 36 39 41 42 40 36

## Vector operators

Convert each element of “temp” vector to degree Fahrenheit

9C/5 + 32 = F

> ((9\*temp)/5) + 32

[1] 93.2 89.6 86.0 84.2 82.4 87.8 96.8 102.2 105.8 107.6 104.0 96.8

Vector allows element wise calculation.

Each element is multiplied with 9, divided by 5 and then added to 32

You can pass a logical vector as index to a vector. In that case all the positions where the value is TRUE, the value will be returned

Consider the vector prime\_numbers

prime\_numbers <- c(2,3,5,7,9,11)

This is a 6 member vector. Now, we will pass a 6 element logical vector as index to prime\_numbers

prime\_numbers[c(T,T,F,F,F,T)]

This operation will return the 1st, 2nd and 6th element of prime\_numbers vector

> prime\_numbers[c(T,T,F,F,F,T)]

[1] 2 3 11

Now, let’s say you only pass a 2 element logical vector to prime\_numbers, then the pattern of this 2 element logical vector will be repeated till all 6 elements of prime\_numbers are traversed.

> prime\_numbers[c(T,F)]

[1] 2 5 9

What will happen if we pass a 4 element logical vector to prime\_numbers?

> prime\_numbers[c(T,T,F,T)]

[1] 2 3 7 9 11

The pattern is repeated till all members are traversed.

We can also pass a logic condition as an index to temp vector.

Note: temp is a vector. If we do temp > 35, this will perform a logical check on each member of temp vector and return a logical vector

> temp

[1] 34 32 30 29 28 31 36 39 41 42 40 36

> temp > 35

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

Now, if we pass temp > 35 as an index to temp vector, all the index position value which is true, the element will be returned

> temp[temp > 35]

[1] 36 39 41 42 40 36

### which() function

If you want to know which index position is satisfying the condition (i.e. returned as TRUE) – you can use the **which()** function

> which(temp>35)

[1] 7 8 9 10 11 12

Notice the 4 statements below

> temp>30

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

> temp< 35

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

> temp>30 & temp < 35

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

> temp>30 && temp < 35

[1] TRUE

When we did temp > 30 & temp < 35 – using a single “&” operator, it did a element wise “and” operation

When we did temp > 30 && temp < 35 – using a (2 “and”) logical “and” - “&&” – it took the whole as a logical condition and checked if there is an element which is > 30 and < 35. If it satisfies, the equation returned TRUE.

So, if you want to extract the values which is > 30 and < 35, use the below format

> temp[temp>30 & temp < 35]

[1] 34 32 31

## Named Vector

Suppose you have a vector storing marks of 5 students.

marks <- c(67,78,81,73,89)

Now, the only way to access the elements of this vector is using the index.

In case of named vector, we provide a name for each index position.

See example below:

marks\_named <- c(jake=67,jill=78,james=81,jake=73,jimmy=89)

> marks\_named

jake jill james jake jimmy

67 78 81 73 89

Note here that we have two “jake”. We will see what happens in such a case.

If you want to access Jill’s marks, you can use the command shown below:

> marks\_named['jill']

jill

78

If you issue a command for “jake” (remember there are two of them)

> marks\_named['jake']

jake

67

It returns only for the first one. Currently I do not know how to access the next one.

How to get all the index names of the named vector

> names(marks\_named)

[1] "jake" "jill" "james" "jake" "jimmy"

For this you should use the **names()** function. This actually returns a vector with all the index names.

If you want to just access the first name index, you can modify the command as

> names(marks\_named)[1]

[1] "jake"

Using the same syntax, you can reassign a new name to this index position. Suppose you want to change the 4th index name to “joy”. In that case, the command would be:

> names(marks\_named)[4] <- "joy"

> names(marks\_named)

[1] "jake" "jill" "james" "joy" "jimmy"

> marks\_named

jake jill james joy jimmy

67 78 81 73 89

So, there is another way to create a named vector. Suppose you already have a vector called marks\_2 which is not a named vector. You can now create a separate vector storing the names. Then using the names() function, we can assign names to marks vector or may be store it in a new vector.

> marks\_2 <- c(67,78,81,73,89,53,67)

> name <- c('jake','jill','james','joy','jimmy','arvind','neha')

> names(marks\_2) <- name

> marks\_2

jake jill james joy jimmy arvind neha

67 78 81 73 89 53 67

This makes marks\_2 as a named vector

What will you do if you need marks of “jake” and “joy” from marks\_2 vector?

> marks\_2[c("jake","joy")]

jake joy

67 73

Put “jake” and “joy” in a vector and pass this as an index to marks\_2 vector.

What will you do if you do not want the marks of “jake” and “joy”.

We know that negative index excludes them from result set. So somehow we need to find the index position of “jake” and “joy” and negate them. We also know what which() function gives us the index position if we pass the value.

So, we can do the following steps:

1. First use the names() function to get the names (index names) as a form of vector
2. In this vector – perform a which() operation for “jake” and “joy” – which will return a vector with these two index position. Store it in a vector.
3. Negate this vector and pass this as index to marks\_2 vector

> nam <- names(marks\_2)

> nam

[1] "jake" "jill" "james" "joy" "jimmy" "arvind" "neha"

> pos <- which(nam == 'jake' | name == 'joy')

> pos

[1] 1 4

> marks\_2[-pos]

jill james jimmy arvind neha

78 81 89 53 67

OR

You can put this entire thing in one line:

> marks\_2[-which(names(marks\_2)=='jake' | names(marks\_2)=='joy')]

jill james jimmy arvind neha

78 81 89 53 67

OR

There is another way of doing it.

Remember if the index position is FALSE at a certain place, the value is skipped. Using that principal the below solution works

> names(marks\_2) == "jake"

[1] TRUE FALSE FALSE FALSE FALSE FALSE FALSE

> names(marks\_2) == "joy"

[1] FALSE FALSE FALSE TRUE FALSE FALSE FALSE

Using an OR condition

> names(marks\_2) == "jake" | names(marks\_2) == "joy"

[1] TRUE FALSE FALSE TRUE FALSE FALSE FALSE

Negate the above result set using a NOT operator (!)

> !(names(marks\_2) == "jake" | names(marks\_2) == "joy")

[1] FALSE TRUE TRUE FALSE TRUE TRUE TRUE

> marks\_2[!(names(marks\_2) == "jake" | names(marks\_2) == "joy")]

jill james jimmy arvind neha

78 81 89 53 67

## Creating vector using rep()

rep() repeats the elements of the vector

> rep(1:10)

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

> rep(1:10,each=2)

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

In the above example, each element of the vector is repeated twice

> rep(1:10,times=2)

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

In the above example, the vector itself is repeated twice.

> rep(1:10,each=2,length.out=125)

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

[27] 4 4 5 5 6 6 7 7 8 8 9 9 10 10 1 1 2 2 3 3 4 4 5 5 6 6

[53] 7 7 8 8 9 9 10 10 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9

[79] 10 10 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10 1 1 2 2

[105] 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10 1 1 2 2 3

In the above example, each element is repeated twice, until the length of 125 is reached.

> name

[1] "jake" "jill" "james" "joy" "jimmy" "arvind" "neha"

> rep(name,each=3)

[1] "jake" "jake" "jake" "jill" "jill" "jill" "james" "james"

[9] "james" "joy" "joy" "joy" "jimmy" "jimmy" "jimmy" "arvind"

[17] "arvind" "arvind" "neha" "neha" "neha"

In the above example, name was a vector. rep(name, each=3) repeated each element 3 times.

### length.out

Note: length.out is the minimum length of the vector

> rep(1:5,each=2,times=2)

[1] 1 1 2 2 3 3 4 4 5 5 1 1 2 2 3 3 4 4 5 5

## Modifying values of a vector

Revision:

What does names(marks\_2) returns?

It returns a vector of names associated with marks\_2 vector

> names(marks\_2)

[1] "jake" "jill" "james" "joy" "jimmy" "arvind" "neha"

What does names(marks\_2) == "jimmy" returns?

It returns a vector where only the position where “jimmy” matches is returned as true

> names(marks\_2) == "jimmy"

[1] FALSE FALSE FALSE FALSE TRUE FALSE FALSE

What will happen if we pass this logical vector to another vector?

Ans: Only that position where the value is TRUE, the value of the vector is returned

> names(marks\_2)[names(marks\_2)=="jimmy"]

[1] "jimmy"

Remember : names(marks\_2) is a vector

Now, what will happen if we assign a new value here

> names(marks\_2)[names(marks\_2)=="jimmy"] <- "jim"

> marks\_2

jake jill james joy jim arvind neha

67 78 81 73 89 53 67

Note that “jimmy” is now replaced with “jim”.

This is how you change the name of a named vector.

So you get hold of the names as a vector which is names(marks\_2) and then try to reach the exact index position where you want to change the value:

names(marks\_2][x]. x can be a number or a vector. We either need a number or a logic vector with only jimmy’s position as TRUE with rest of the values as FALSE

To figure of x – we wrote names(marks\_2)==”jimmy”. This will create a logical vector with jimmy’s position as TRUE and rest will be FALSE.

Once you get hold of it, set a new value “jim” to that same position.

Now, let’s say you want to modify marks of “jim” to 88

> marks\_2["jim"] <- 88

> marks\_2

jake jill james joy jim arvind neha

67 78 81 73 88 53 67

## Vector concatenation

> marks1 = c(23,31,30,c(28,17),c(33,29,40))

A vector can take other vectors and concatenate them as a single vector

> marks1

[1] 23 31 30 28 17 33 29 40

> marks2 = c(26,31,39)

> c(marks1,marks2)

[1] 23 31 30 28 17 33 29 40 26 31 39

The above example took vector marks1 and marks2 and created a new vector

## Vector Arithmetic

Let’s assume two vectors score1 and score2 of 5 elements each

> score1 <- c(25,20,18,34,31)

> score2 <- c(30,24,29,33,38)

>

> score1 + score2

[1] 55 44 47 67 69

Adding them will perform an element wise add

Similarly you can do

> score1 - score2

[1] -5 -4 -11 1 -7

> score1 \* score2

[1] 750 480 522 1122 1178

> score1 / score2

[1] 0.8333333 0.8333333 0.6206897 1.0303030 0.8157895

Now let’s see what happens when the lengths are unequal:

> vec1 <- c(1:6)

> vec1

[1] 1 2 3 4 5 6

> vec2 <- c(10,100)

> vec1 \* vec2

[1] 10 200 30 400 50 600

vec1 is of 6 element length and vec2 is of 2 element length. If the shorter vector is a multiple of longer vector then there is no problem and shorter vector will repeat itself to match the longer vector. You can see this in the example above.

What happens if the shorter vector is not a multiple of longer vector?

Let’s see

> vec3 <- c(1:5)

> vec3\*vec2

[1] 10 200 30 400 50

Warning message:

In vec3 \* vec2 :

longer object length is not a multiple of shorter object length

vec3 was created with 5 elements. Now vec2 is no longer a multiple of vec3. However, the repetition does happen but also a warning is thrown.

# Matrix

Let’s say I want to store temperature of 4 different cities for 6 different time interval.

So, I might have 4 vectors representing 4 cities storing 6 element each for 6 time interval.

Something like shown below:

c(33, 35, 37, 42, 44, 29)

c(18, 17, 19, 16, 18, 15)

c(31, 30, 30, 30, 32, 31)

c(21, 22, 27, 23, 25, 28)

However, a better way to represent them is using a matrix

To create a matrix we will have to use the **maxtrix()** function. matrix() takes a single vector and then based on row and column configuration, creates a matrix.

So, let’s combine these 4 vectors into a single vector.

c(c(33, 35, 37, 42, 44, 29),

c(18, 17, 19, 16, 18, 15),

c(31, 30, 30, 30, 32, 31),

c(21, 22, 27, 23, 25, 28))

and then pass it to the matrix() function

matrix(c(c(33, 35, 37, 42, 44, 29),

c(18, 17, 19, 16, 18, 15),

c(31, 30, 30, 30, 32, 31),

c(21, 22, 27, 23, 25, 28))

)

We also need to tell, how to arrange these elements –

matrix(c(c(33, 35, 37, 42, 44, 29),

c(18, 17, 19, 16, 18, 15),

c(31, 30, 30, 30, 32, 31),

c(21, 22, 27, 23, 25, 28)),

byrow = TRUE)

byrow = TRUE means start arranging the elements row wise. Complete the first row and then move on to the next row.

But how many rows?

We need to tell that

matrix(c(c(33, 35, 37, 42, 44, 29),

c(18, 17, 19, 16, 18, 15),

c(31, 30, 30, 30, 32, 31),

c(21, 22, 27, 23, 25, 28)),

byrow = TRUE,

nrow = 4)

This means make 4 rows.

What about columns?

You can also tell that like

matrix(c(c(33, 35, 37, 42, 44, 29),

c(18, 17, 19, 16, 18, 15),

c(31, 30, 30, 30, 32, 31),

c(21, 22, 27, 23, 25, 28)),

byrow = TRUE,

nrow = 4, ncol=6)

Otherwise if we have mentioned nrows = 4, it will take the total elements (in our example = 24), divide that by 4 (4 rows) = 6 columns

> matrix(c(c(33, 35, 37, 42, 44, 29),

+ c(18, 17, 19, 16, 18, 15),

+ c(31, 30, 30, 30, 32, 31),

+ c(21, 22, 27, 23, 25, 28)),

+ byrow = TRUE,

+ nrow = 4, ncol=6)

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

[1,] 33 35 37 42 44 29

[2,] 18 17 19 16 18 15

[3,] 31 30 30 30 32 31

[4,] 21 22 27 23 25 28

If we skip ncol = 6

> matrix(c(c(33, 35, 37, 42, 44, 29),

+ c(18, 17, 19, 16, 18, 15),

+ c(31, 30, 30, 30, 32, 31),

+ c(21, 22, 27, 23, 25, 28)),

+ byrow = TRUE,

+ nrow = 4)

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

[1,] 33 35 37 42 44 29

[2,] 18 17 19 16 18 15

[3,] 31 30 30 30 32 31

[4,] 21 22 27 23 25 28

What if we suddenly decide to go for 5 rows instead of 4 with the same dataset?

> matrix(c(c(33, 35, 37, 42, 44, 29),

+ c(18, 17, 19, 16, 18, 15),

+ c(31, 30, 30, 30, 32, 31),

+ c(21, 22, 27, 23, 25, 28)),

+ byrow = TRUE,

+ nrow = 5)

[,1] [,2] [,3] [,4] [,5]

[1,] 33 35 37 42 44

[2,] 29 18 17 19 16

[3,] 18 15 31 30 30

[4,] 30 32 31 21 22

[5,] 27 23 25 28 33

Warning message:

In matrix(c(c(33, 35, 37, 42, 44, 29), c(18, 17, 19, 16, 18, 15), :

data length [24] is not a sub-multiple or multiple of the number of rows [5]

What just happened?

The number of elements were 25. Row requested was 5. If we see the multiples of 5, the nearest multiple of 5 from 24 is 25 (it will never go backwards).

So 25/5 = 5 columns. This operation will now make 5 columns and one element will be repeated to make it 25 elements. Of course there is a warning because the repetition was partial.

matrix() will always try to do the minimum possible repetition.