Understanding Factors

swcarpentry.github.io/r-novice-inflammation/12-supp-factors/

> **Questions**
>
> - How is categorical data represented in R?
> - How do I work with factors?
>
> **Objectives**
>
> - Understand how to represent categorical data in R.
> - Know the difference between ordered and unordered factors.
> - Be aware of some of the problems encountered when using factors.

Factors are used to represent categorical data. Factors can be ordered or unordered and are an important class for statistical analysis and for plotting.

Factors are stored as integers, and have labels associated with these unique integers. While factors look (and often behave) like character vectors, they are actually integers under the hood, and you need to be careful when treating them like strings.

Once created, factors can only contain a pre-defined set values, known as*levels*. By default, R always sorts *levels* in alphabetical order. For instance, if you have a factor with 2 levels:

> ## The `factor()` Command
>
> The `factor()` command is used to create and modify factors in R:
>
> ```
> sex <- factor(c("male", "female", "female", "male"))
> ```

R will assign `1` to the level `"female"` and `2` to the level `"male"` (because `f` comes before `m`, even though the first element in this vector is `"male"`). You can check this by using the function `levels()`, and check the number of levels using `nlevels()`:

```
levels(sex)
```

```
[1] "female" "male"
```

```
nlevels(sex)
```

```
[1] 2
```

Sometimes, the order of the factors does not matter, other times you might want to specify the order because it is meaningful (e.g., “low”, “medium”, “high”) or it is required by particular type of analysis. Additionally, specifying the order of the levels allows us to compare levels:

```
food <- factor(c("low", "high", "medium", "high", "low", "medium", "high"))
levels(food)

[1] "high"   "low"    "medium"

food <- factor(food, levels = c("low", "medium", "high"))
levels(food)

[1] "low"    "medium" "high"

min(food) ## doesn't work

Error in Summary.factor(structure(c(1L, 3L, 2L, 3L, 1L, 2L, 3L), .Label = c("low", :
'min' not meaningful for factors

food <- factor(food, levels = c("low", "medium", "high"), ordered=TRUE)
levels(food)

[1] "low"    "medium" "high"

min(food) ## works!

[1] low
Levels: low < medium < high
```

In R's memory, these factors are represented by numbers (1, 2, 3). They are better than using integer labels because factors are self describing: `"low"`, `"medium"`, and `"high"`" is more descriptive than `1`, `2`, `3`. Which is low? You wouldn't be able to tell with just integer data. Factors have this information built in. It is particularly helpful when there are many levels (like the subjects in our example data set).

> ## Representing Data in R
>
> You have a vector representing levels of exercise undertaken by 5 subjects
>
> **"l","n","n","i","l"** ; n=none, l=light, i=intense
>
> What is the best way to represent this in R?
>
> a) exercise <- c("l", "n", "n", "i", "l")
>
> b) exercise <- factor(c("l", "n", "n", "i", "l"), ordered = TRUE)
>
> c) exercise < -factor(c("l", "n", "n", "i", "l"), levels = c("n", "l", "i"), ordered = FALSE)
>
> d) exercise <- factor(c("l", "n", "n", "i", "l"), levels = c("n", "l", "i"), ordered = TRUE)

> > ## Solution

Converting Factors

Converting from a factor to a number can cause problems:

```
f <- factor(c(3.4, 1.2, 5))
as.numeric(f)
```

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

This does not behave as expected (and there is no warning).

The recommended way is to use the integer vector to index the factor levels:

```
levels(f)[f]
```

```
[1] "3.4" "1.2" "5"
```

This returns a character vector, the `as.numeric()` function is still required to convert the values to the proper type (numeric).

```
f <- levels(f)[f]
f <- as.numeric(f)
```

Using Factors

Lets load our example data to see the use of factors:

```
dat <- read.csv(file = 'data/sample.csv', stringsAsFactors = TRUE)
```

> ### Default Behavior
>
> `stringsAsFactors=TRUE` is the default behavior for R. We could leave this argument out. It is included here for clarity.

```
str(dat)
```

```
'data.frame': 100 obs. of  9 variables:
 $ ID           : Factor w/ 100 levels "Sub001","Sub002",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ Gender       : Factor w/ 4 levels "f","F","m","M": 3 3 3 1 3 4 1 3 3 1 ...
 $ Group        : Factor w/ 3 levels "Control","Treatment1",..: 1 3 3 2 2 3 1 3 3 1
...
 $ BloodPressure: int  132 139 130 105 125 112 173 108 131 129 ...
 $ Age          : num  16 17.2 19.5 15.7 19.9 14.3 17.7 19.8 19.4 18.8 ...
 $ Aneurisms_q1 : int  114 148 196 199 188 260 135 216 117 188 ...
 $ Aneurisms_q2 : int  140 209 251 140 120 266 98 238 215 144 ...
 $ Aneurisms_q3 : int  202 248 122 233 222 320 154 279 181 192 ...
 $ Aneurisms_q4 : int  237 248 177 220 228 294 245 251 272 185 ...
```

Notice the first 3 columns have been converted to factors. These values were text in the data file so R automatically interpreted them as categorical variables.

```
summary(dat)
```

```
      ID     Gender        Group    BloodPressure        Age
Sub001 : 1   f:35   Control   :30   Min.   : 62.0   Min.   :12.10
Sub002 : 1   F: 4   Treatment1:35   1st Qu.:107.5   1st Qu.:14.78
Sub003 : 1   m:46   Treatment2:35   Median :117.5   Median :16.65
Sub004 : 1   M:15                   Mean   :118.6   Mean   :16.42
Sub005 : 1                          3rd Qu.:133.0   3rd Qu.:18.30
Sub006 : 1                          Max.   :173.0   Max.   :20.00
(Other):94
 Aneurisms_q1    Aneurisms_q2    Aneurisms_q3    Aneurisms_q4
Min.   : 65.0   Min.   : 80.0   Min.   :105.0   Min.   :116.0
1st Qu.:118.0   1st Qu.:131.5   1st Qu.:182.5   1st Qu.:186.8
Median :158.0   Median :162.5   Median :217.0   Median :219.0
Mean   :158.8   Mean   :168.0   Mean   :219.8   Mean   :217.9
3rd Qu.:188.0   3rd Qu.:196.8   3rd Qu.:248.2   3rd Qu.:244.2
Max.   :260.0   Max.   :283.0   Max.   :323.0   Max.   :315.0
```

Notice the `summary()` function handles factors differently to numbers (and strings), the occurrence counts for each value is often more useful information.

> ## The `summary()` Function
>
> The `summary()` function is a great way of spotting errors in your data (look at the *dat$Gender* column). It's also a great way for spotting missing data.

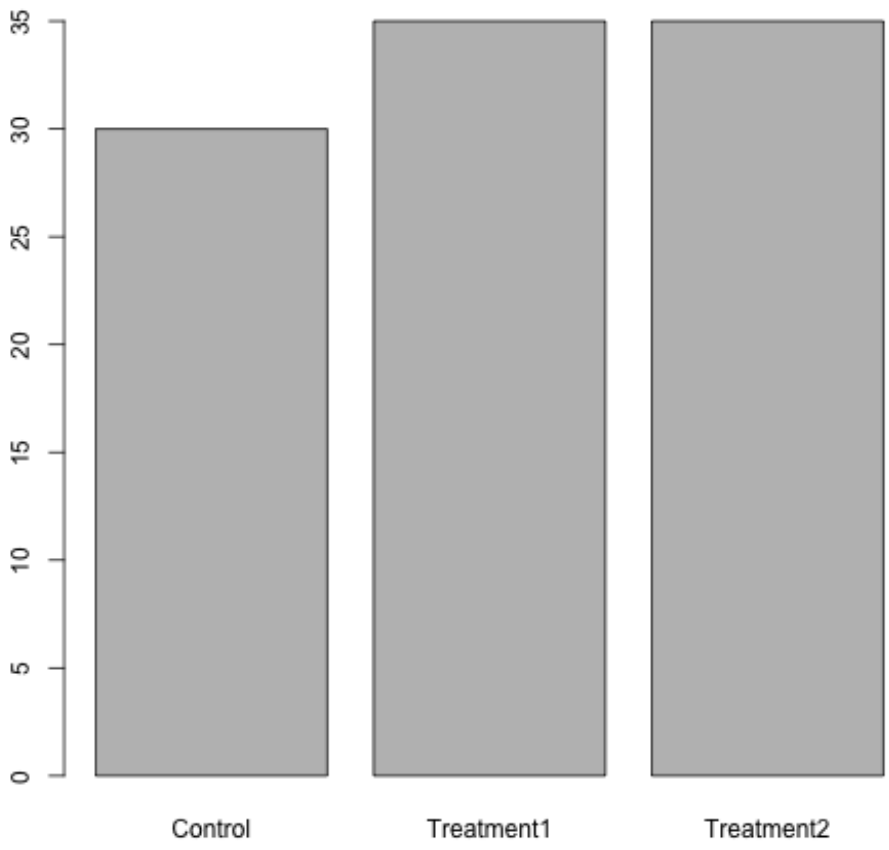
Reordering Factors

The function `table()` tabulates observations and can be used to create bar plots quickly. For instance:

```
table(dat$Group)
```

```
   Control Treatment1 Treatment2
        30         35         35
```

```
barplot(table(dat$Group))
```

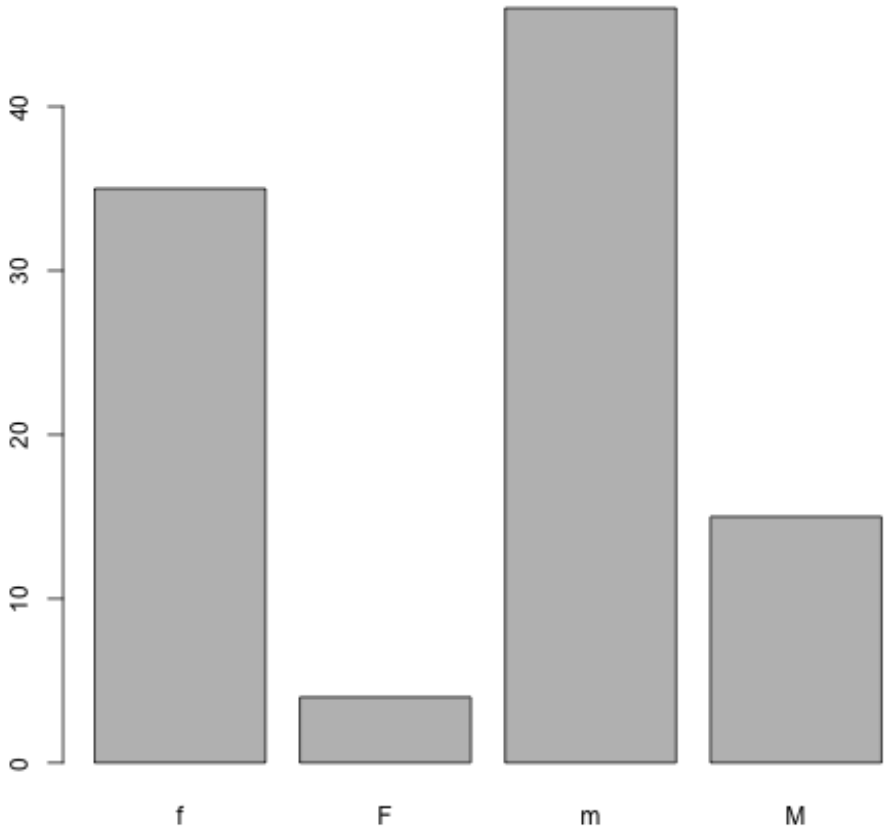


Use the `factor()` command to modify the column dat$Group so that the *control* group is plotted last

Removing Levels from a Factor

Some of the Gender values in our dataset have been coded incorrectly. Let's remove factors.

```
barplot(table(dat$Gender))
```

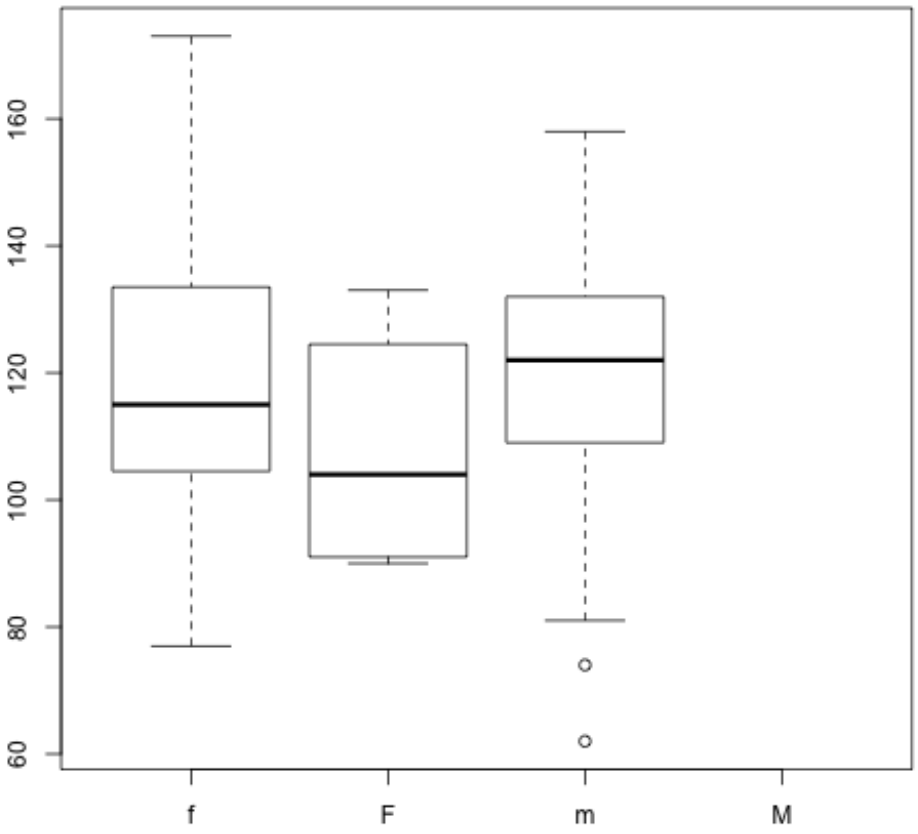


Values should have been recorded as lowercase ‘m’ & ‘f’. We should correct this.

```
dat$Gender[dat$Gender == 'M'] <- 'm'
```

Updating Factors

```
plot(x = dat$Gender, y = dat$BloodPressure)
```

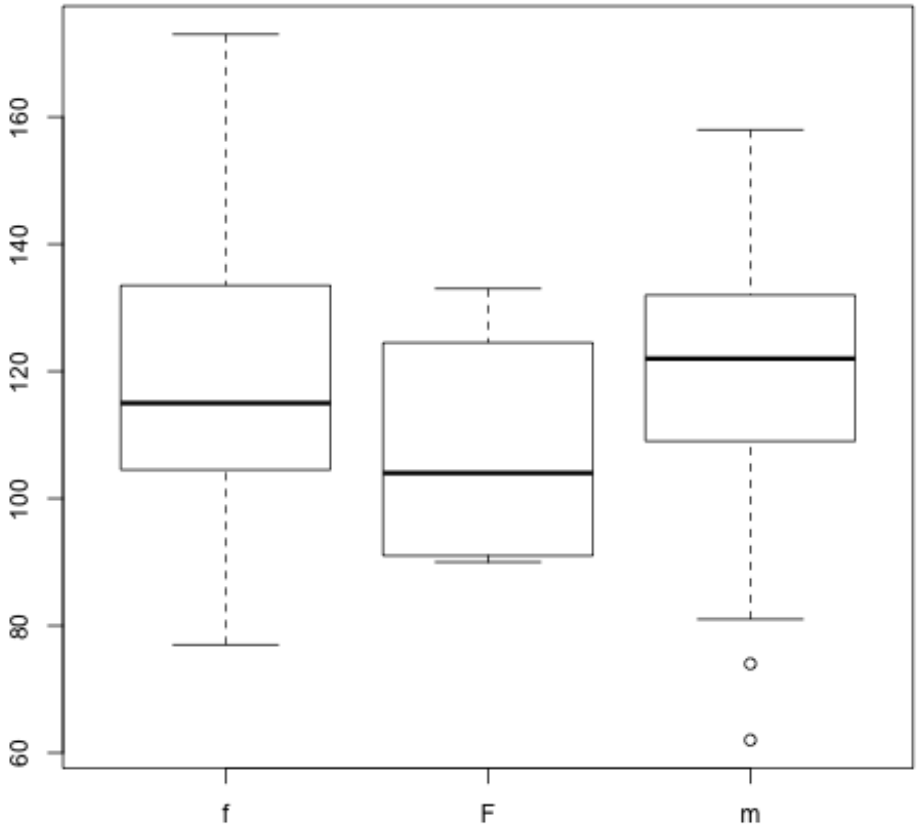


Why does this plot show 4 levels?

Hint how many levels does dat$Gender have?

We need to tell R that “M” is no longer a valid value for this column. We use the `droplevels()` function to remove extra levels.

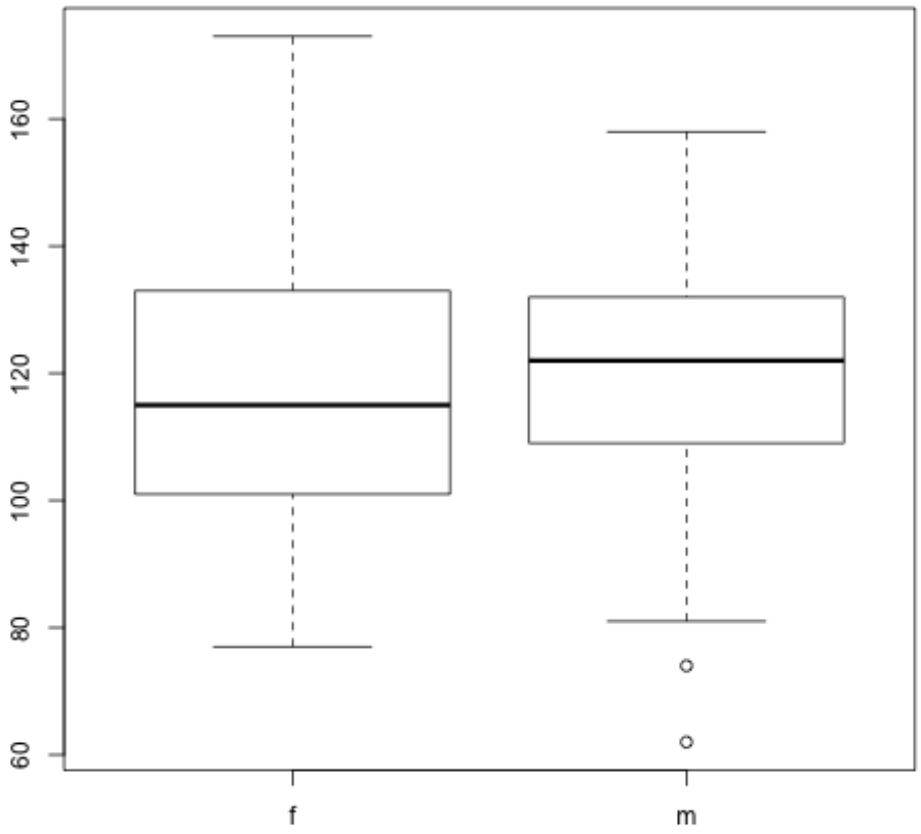
```
dat$Gender <- droplevels(dat$Gender)
plot(x = dat$Gender, y = dat$BloodPressure)
```


160
140
120
100
80
60
f
F
m

Adjusting Factor Levels

Adjusting the `levels()` of a factor provides a useful shortcut for reassigning values in this case.

```
levels(dat$Gender)[2] <- 'f'
plot(x = dat$Gender, y = dat$BloodPressure)
```



Key Points

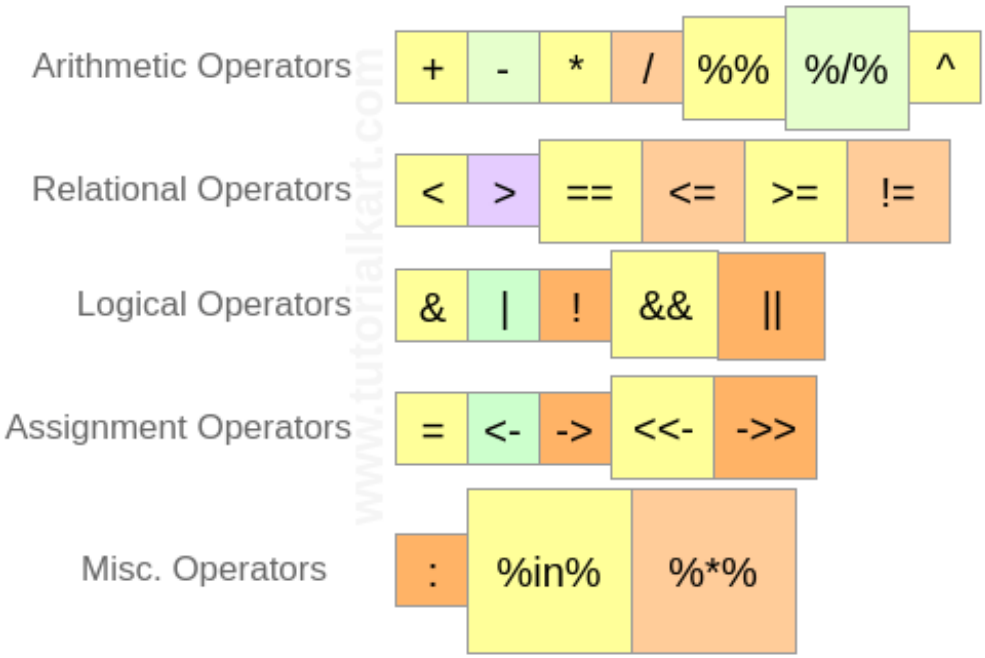
- Factors are used to represent categorical data.
- Factors can be *ordered* or *unordered*.
- Some R functions have special methods for handling factors.

R Operators – Arithmetic, Relational, Logical, Assignment

R Tutorial – We shall learn about R Operators – Arithmetic, Relational, Logical, Assignment and some of the Miscellaneous Operators that R programming language provides.

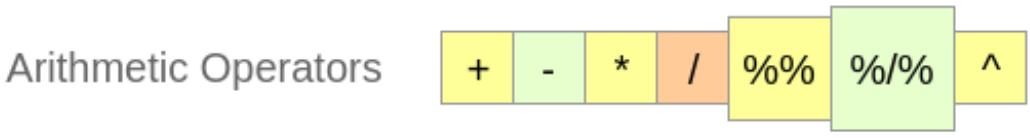
R Operators

There are four main categories of Operators in R programming language. They are shown in the following picture :



We shall learn about these operators in detail with Example R programs.

R Arithmetic Operators



Arithmetic Operators are used to accomplish arithmetic operations. They can be operated on the basic data types Numericals, Integers, Complex Numbers. Vectors with these basic data types can also participate in arithmetic operations, during which the operation is performed on one to one element basis.

Operator	Description	Usage
+	Addition of two operands	a + b
–	Subtraction of second operand from first	a – b
*	Multiplication of two operands	a * b
/	Division of first operand with second	a / b
%%	Remainder from division of first operand with second	a %% b
%/%	Quotient from division of first operand with second	a %/% b
^	First operand raised to the power of second operand	a^b

An example for each of the arithmetic operator on Numerical values is provided below :

r_op_arithmetic.R R Script File

```
1    # R Arithmetic Operators Example for integers
2    a<-7.5
3    b<-2
4    print(a+b)#addition
5    print(a-b)#subtraction
6    print(a*b)#multiplication
7    print(a/b)#Division
8    print(a%%b)#Reminder
9    print(a%/%b)#Quotient
10   print(a^b)#Power of
11
12
```

Output

```
1   $Rscript r_op_arithmetic.R
2   [1]9.5
3   [1]5.5
4   [1]15
5   [1]3.75
6   [1]1.5
7   [1]3
8   [1]56.25
```

An example for each of the arithmetic operator on Vectors is provided below :

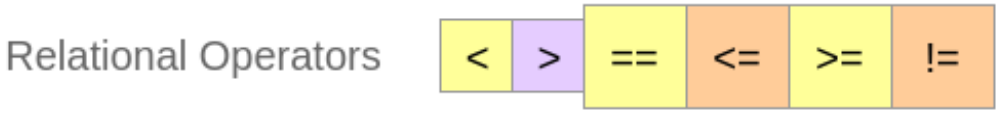
r_op_arithmetic_vector.R R Script File

```
1    # R Operators - R Arithmetic Operators Example for vectors
2    a<-c(8,9,6)
3    b<-c(2,4,5)
4    print(a+b)#addition
5    print(a-b)#subtraction
6    print(a*b)#multiplication
7    print(a/b)#Division
8    print(a%%b)#Reminder
9    print(a%/%b)#Quotient
10   print(a^b)#Power of
11
12
```

Output

```
$Rscript r_op_arithmetic_vector.R
[1]101311
[1]651
[1]163630
[1]4.002.251.20
[1]011
[1]421
[1]6465617776
```

R Relational Operators



Relational Operators are those that find out relation between the two operands provided to them. Following are the six relational operations R programming language supports.The output is boolean (TRUE or FALSE) for all of the Relational Operators in R programming language.

Operator	Description	Usage
<	Is first operand **less than** second operand	a < b
>	Is first operand **greater than** second operand	a > b
==	Is first operand **equal to** second operand	a == b
<=	Is first operand **less than** or equal to second operand	a <= b
>=	Is first operand **greater than** or equal to second operand	a > = b
!=	Is first operand **not equal** to second operand	a!=b

An example for each of the relational operator on Numberical values is provided below :

r_op_relational.R R Script File

```
# R Operators - R Relational Operators Example for Numbers
a<-7.5
b<-2
print(a<b)# less than
print(a>b)# greater than
print(a==b)# equal to
print(a<=b)# less than or equal to
print(a>=b)# greater than or equal to
print(a!=b)# not equal to


```

Output

```
$Rscript r_op_relational.R
[1]FALSE
[1]TRUE
[1]FALSE
[1]FALSE
[1]TRUE
[1]TRUE
```

An example for each of the relational operator on Vectors is provided below :

r_op_relational_vector.R R Script File

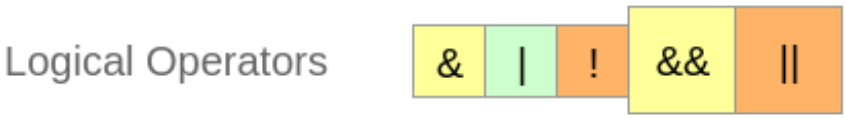
```
# R Operators - R Relational Operators Example for Numbers
a<-c(7.5,3,5)
b<-c(2,7,0)
print(a<b)# less than
print(a>b)# greater than
print(a==b)# equal to
print(a<=b)# less than or equal to
print(a>=b)# greater than or equal to
print(a!=b)# not equal to


```

Output

```
$Rscript r_op_relational_vector.R
[1]FALSETRUEFALSE
[1]TRUEFALSETRUE
[1]FALSEFALSEFALSE
[1]FALSETRUEFALSE
[1]TRUEFALSETRUE
[1]TRUETRUETRUE
```

R Logical Operators



Logical Operators in R programming language work only for the basic data types logical, numeric and complex and vectors of these basic data types.

Operator	Description	Usage
&	Element wise logical AND operation.	a & b
|	Element wise logical OR operation.	a | b
!	Element wise logical NOT operation.	!a
&&	Operand wise logical AND operation.	a && b
||	Operand wise logical OR operation.	a || b

An example for each of the logical operators on Numerical values is provided below :

r_op_logical.R R Script File

```
# R Operators - R Logical Operators Example for basic logical elements
a<-0# logical FALSE
b<-2# logical TRUE
print(a&b)# logical AND element wise
print(a|b)# logical OR element wise
print(!a)# logical NOT element wise
print(a&&b)# logical AND consolidated for all elements
print(a||b)# logical OR consolidated for all elements


```

Output

```
$Rscript r_op_logical.R
[1]FALSE
[1]TRUE
[1]TRUE
[1]FALSE
[1]TRUE
```

An example for each of the logical operators on Vectors is provided below :

R Script File

```
# R Operators - R Logical Operators Example for boolean vectors
a<-c(TRUE,TRUE,FALSE,FALSE)
b<-c(TRUE,FALSE,TRUE,FALSE)
print(a&b)# logical AND element wise
print(a|b)# logical OR element wise
print(!a)# logical NOT element wise
print(a&&b)# logical AND consolidated for all elements
print(a||b)# logical OR consolidated for all elements


```

Output

```
$Rscript r_op_logical_vector.R
[1]TRUEFALSEFALSEFALSE
[1]TRUETRUETRUEFALSE
[1]FALSEFALSETRUETRUE
[1]TRUE
[1]TRUE
```

R Assignment Operators

Assignment Operators are those that help in assigning a value to the variable.

Operator	Description	Usage
=	Assigns right side value to left side operand	a = 3

Operator	Description	Usage
<-	Assigns right side value to left side operand	a <- 5
->	Assigns left side value to right side operand	4 -> a
<<-	Assigns right side value to left side operand	a <<- 3.4
->>	Assigns left side value to right side operand	c(1,2) ->> a

An example for each of the assignment operators is provided below :

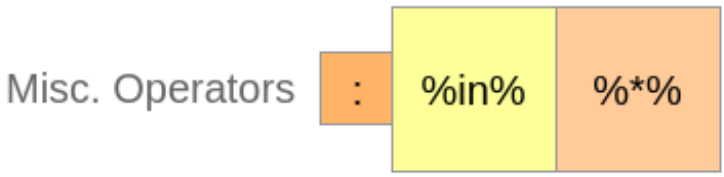
r_op_assignment.R R Script File

```
1   # R Operators - R Assignment Operators
2   a=2
3   print(a)
4   a<-TRUE
5   print(a)
6   454->a
7   print(a)
8   a<<-2.9
9   print(a)
10  c(6,8,9)->a
11  print(a)
12
13
14
15
16
```

Output

```
1  $Rscript r_op_assignment.R
2  [1]2
3  [1]TRUE
4  [1]454
5  [1]2.9
6  [1]689
```

R Miscellaneous Operators



These operators does not fall into any of the categories mentioned above, but are significantly important during R programming for manipulating data.

Operator	Description	Usage
:	Creates series of numbers from left operand to right operand	a:b
%in%	Identifies if an element(a) belongs to a vector(b)	a %in% b
%*%	Performs multiplication of a vector with its transpose	A %*% t(A)

An example for each of the Miscellaneous operators is provided below :

r_op_misc.R R Script File

```
1    # R Operators - R Misc Operators
2    a=23:31
3    print(a)
4    a=c(25,27,76)
5    b=27
6    print(b%in%a)
7    M=matrix(c(1,2,3,4),2,2,TRUE)
8    print(M%*%t(M))
9
10
11
```

Output

```
1   $Rscript r_op_misc.R
2   [1]232425262728293031
3   [1]TRUE
4   [,1][,2]
5   [1,]511
6   [2,]1125
```

Conclusion :

In this R Tutorial, we have learnt about R Operators – R Arithmetic Operators, R Relational Operators, R Logical Operators, R Assignment Operators, R Miscellaneous Operators with example R commands and R script files.

R Arithmetic Operators

The R Arithmetic operators includes operators like Arithmetic Addition, Subtraction, Division, Multiplication, Exponent, Integer Division and Modulus. All these operators are binary operators, which means they operate on two operands. Below table shows all the Arithmetic Operators in R Programming language with examples.

R Arithmetic Operators	Operation	Example
+	Addition	15 + 5 = 20
–	Subtraction	15 – 5 = 10
*****	Multiplication	15 * 5 = 75
/	Division	15 / 5 = 3
%/%	Integer Division – Same as Division but it will return the integer value by flooring the extra decimals	16 %/% 3 = 5. If you divide 16 with 3 you get 5.333 but the Integer division operator will trim the decimal values and outputs the integer
^	Exponent – It returns the Power of One variable against the other	15 ^ 3 = 3375 (It means 15 Power 3 or 10^3).
%%	Modulus – It returns the remainder after the division	15 %% 5 = 0 (Here remainder is zero). If it is 17 %% 4 then it will be 1.

R Arithmetic Operators Example

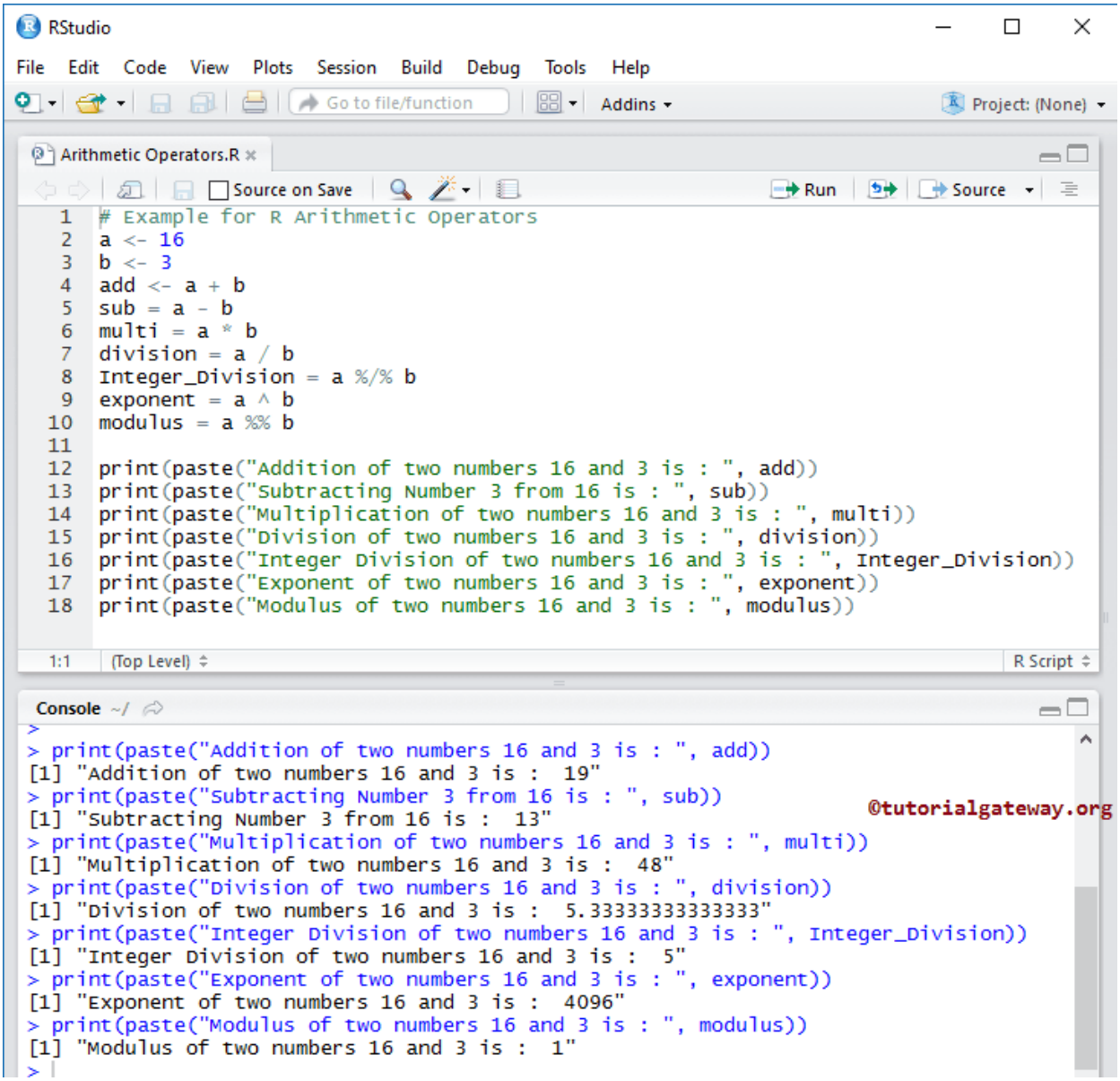
In this example, We are using two variables a and b and their values are 16 and 3. We are going to use these two variables to perform various arithmetic operations present in R programming language

R CODE

```
# Example for R Arithmetic Operators
a<-16
b<-3
add<-a+b
sub=a-b
multi=a*b
division=a/b
Integer_Division=a%/%b
exponent=a^b
modulus=a%%b
print(paste("Addition of two numbers 16 and 3 is : " ,add))
print(paste("Subtracting Number 3 from 16 is : ",sub))
print(paste("Multiplication of two numbers 16 and 3 is : " ,multi))
print(paste("Division of two numbers 16 and 3 is : " ,division))
print(paste("Integer Division of two numbers 16 and 3 is : " ,Integer_Division))
print(paste("Exponent of two numbers 16 and 3 is : " ,exponent))
print(paste("Modulus of two numbers 16 and 3 is : " ,modulus))

```

OUTPUT



Following statement will find the exponent. It means 16 power 3 = 16 * 16 * 16 = 4096

```
1    exponent=a^b
```

NOTE: When we are using division (/) operator the result will be float or decimal value. If you want to display the output as integer value by rounding the value then use Integer Division (%/%) operator

Thank You for Visiting Our Blog.

Comparison Operators in R

The Comparison operators in R Programming are mostly used either in If Conditions or Loops. R Relational operators are commonly used to check the relationship between two variables.

- If the relation is true then it will return Boolean True
- If the relation is false then it will return Boolean False.

Below table shows all the Relational Operators in R Programming with examples.

Comparison Operators in R	Usage	Description	Example
>	i > j	i is greater than j	25 > 14 returns True
<	i < j	i is less than j	25 < 14 returns False
>=	i >= j	i is greater than or equal to j	25 >= 14 returns True
<=	i <= j	i is less than or equal to j	25 <= 14 return False
==	i == j	i is equal to j	25 == 14 returns False
!=	i != j	i is not equal to j	25 != 14 returns True

Comparison Operators in R Programming Example

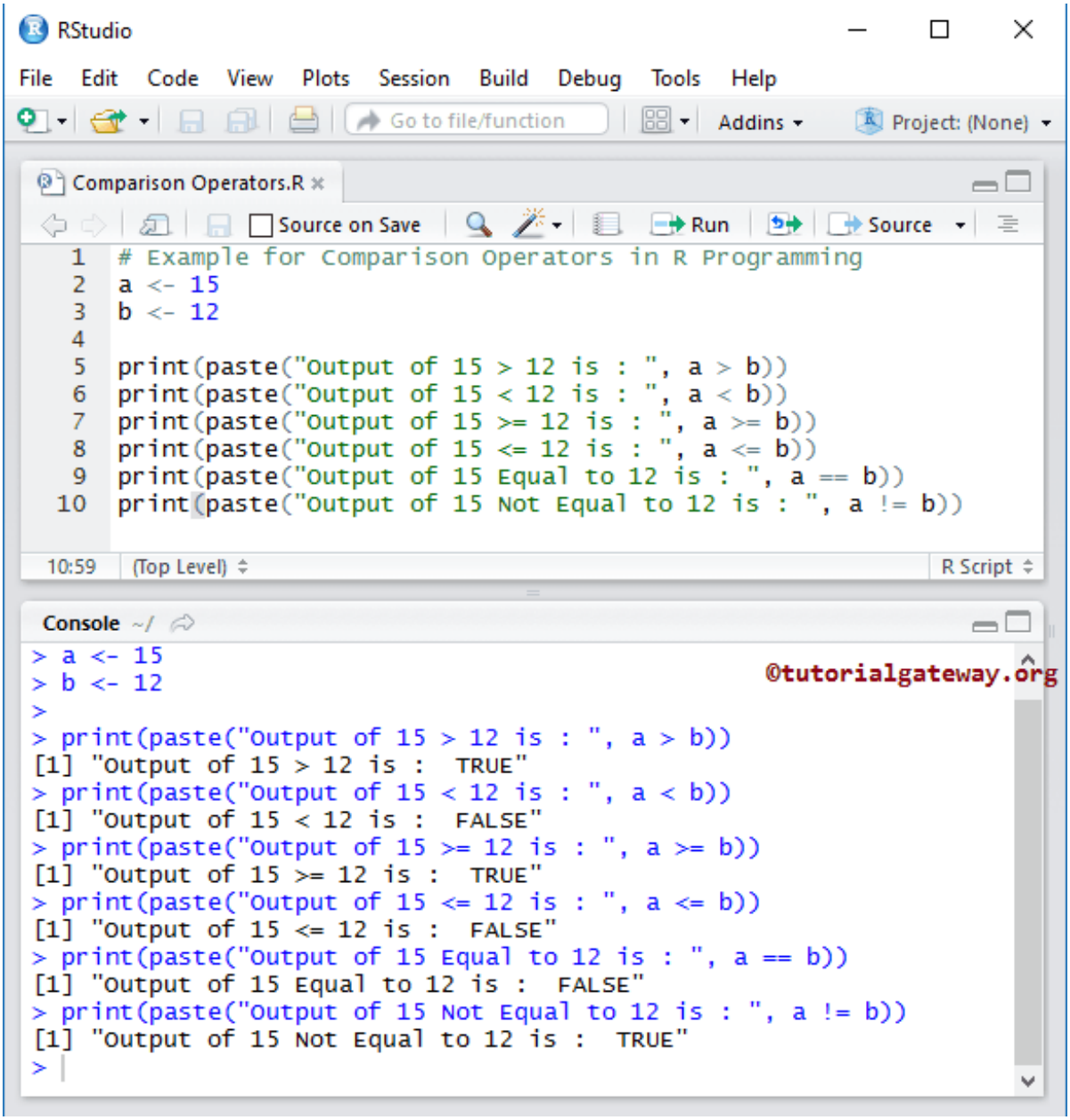
This example helps you to understand the Comparison Operators in R Programming language practically. For this example, We are using two variables a and b and their respective values are 15 and 12. We are going to use these two variables to perform various relational operations present in R Programming

R CODE

```
# Example for Comparison Operators in R Programming
a<-15
b<-12
print(paste("Output of 15 > 12 is : " ,a>b))
print(paste("Output of 15 < 12 is : " ,a<b))
print(paste("Output of 15 >= 12 is : " ,a>=b))
print(paste("Output of 15 <= 12 is : " ,a<=b))
print(paste("Output of 15 Equal to 12 is : " ,a==b))
print(paste("Output of 15 Not Equal to 12 is : " ,a!=b))

```

OUTPUT



ANALYSIS

We assigned two integer values a, b and we assigned the values 15 and 12 using the below statement.

```
1  a<-15
2  b<-12
```

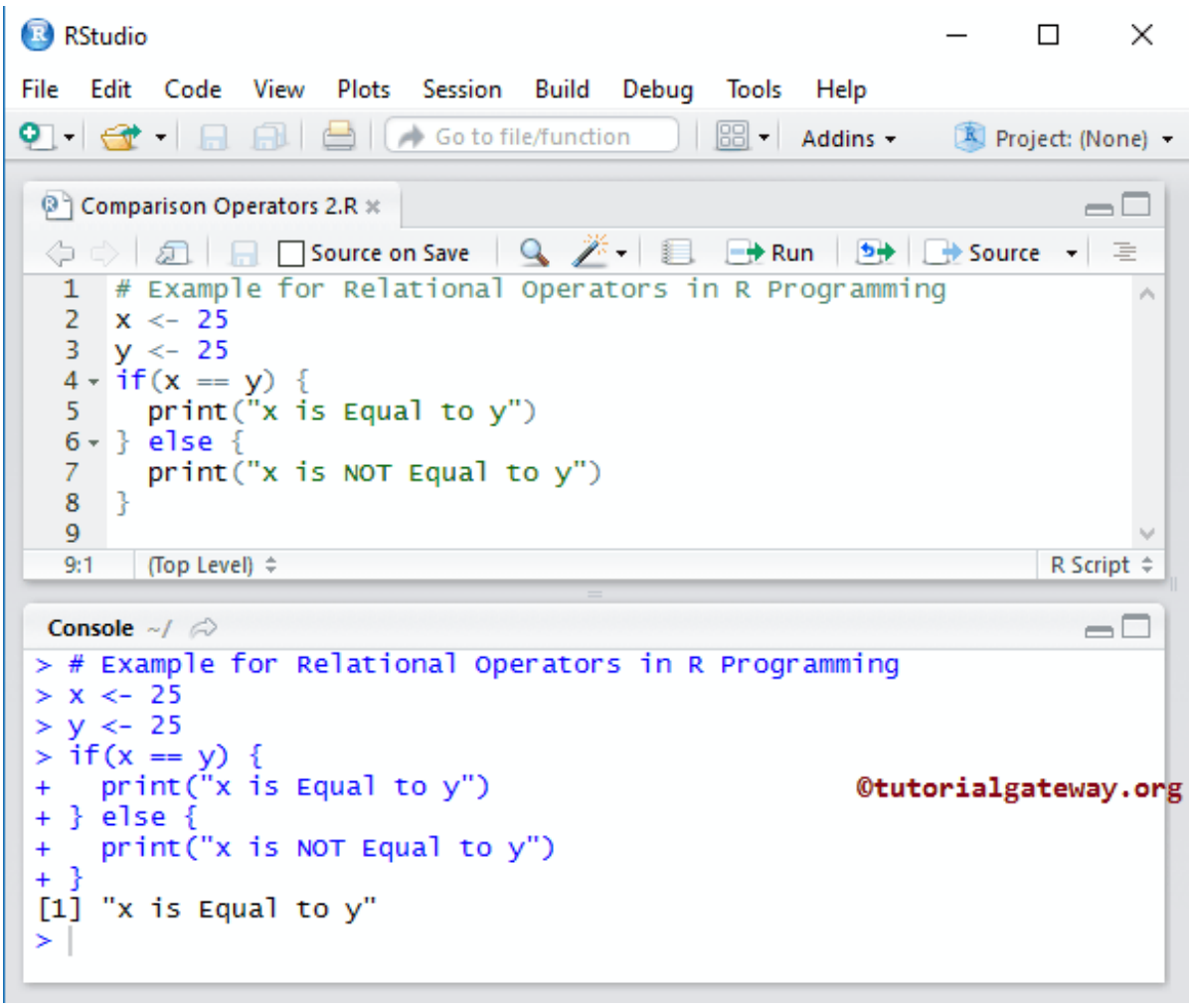
In the next lines, We checked these values against each and every comparison operator (relational operator) present in R Programming language.

R Comparison Operators in IF Statement

This example helps you to understand, how to use Comparison operators inside the If statements. For this example, We are using two variables x and y and their values are 25 and 25. We are going to use these two variables inside the If condition along with one of the comparison operator present in R programming to check the condition.

```
1  # Example for Relational Operators in R Programming
2  x<-25
3  y<-25
4  if(x==y){
5  print("x is Equal to y")
6  }else{
7  print("x is NOT Equal to y")
8  }
```

OUTPUT



ANALYSIS

We assigned two integer values x, y and assigned the values 25 and 25 using the below statement.

```
1  x<-25
2  y<-25
```

If Condition

If x is equal to y then first print statement will be executed

```
1  print("x is Equal to y")
```

If the first condition fails then the second print statement will be executed.

```
1  print("x is NOT Equal to y")
```

Here 25 is equal to 25 so, First print statement is printed

Thank You for Visiting Our Blog.

Logical Operators in R

The Comparison Operators in R are used to compare two variables, and what if we want to compare more than one condition? Very simple, R logical operators will do the trick for you.

The Logical operators in R programming are used to combine two or more conditions, and perform the logical operations using & (Logical AND), | (Logical OR) and ! (Logical NOT). Below table describes them

OPERATORS	NAME	DESCRIPTION	EXAMPLE
&	logical AND	It will return true when both conditions are true	c(20, 30) & c(30, 10)
&&	logical AND	Same as above but, It will work on single element	If (age > 18 && age <= 25)
|	logical OR	It will returns true when at-least one of the condition is true	c(20, 30) | c(30, 10)
||	logical OR	Same as above but, It will work on single element	If (age == 35 || age < 60)
!	logical NOT	If the condition is true, logical NOT operator returns as false	If age = 18 then !(age = 18) returns false.

Let us see the truth tables behind the logical operators in R programming for better understanding

LOGICAL AND Truth table

Truth table behind the logical AND operator is as shown below:

Condition 1	Condition 2	Condition 1 && Condition 2
True	True	True
True	False	False
False	True	False
False	False	False

LOGICAL OR Truth table

The Truth table behind the logical OR operator is as shown below:

Condition 1	Condition 2	Condition 1 || Condition 2
True	True	True
True	False	True
False	True	True

False	False	False

Basic Logical Operators in R example

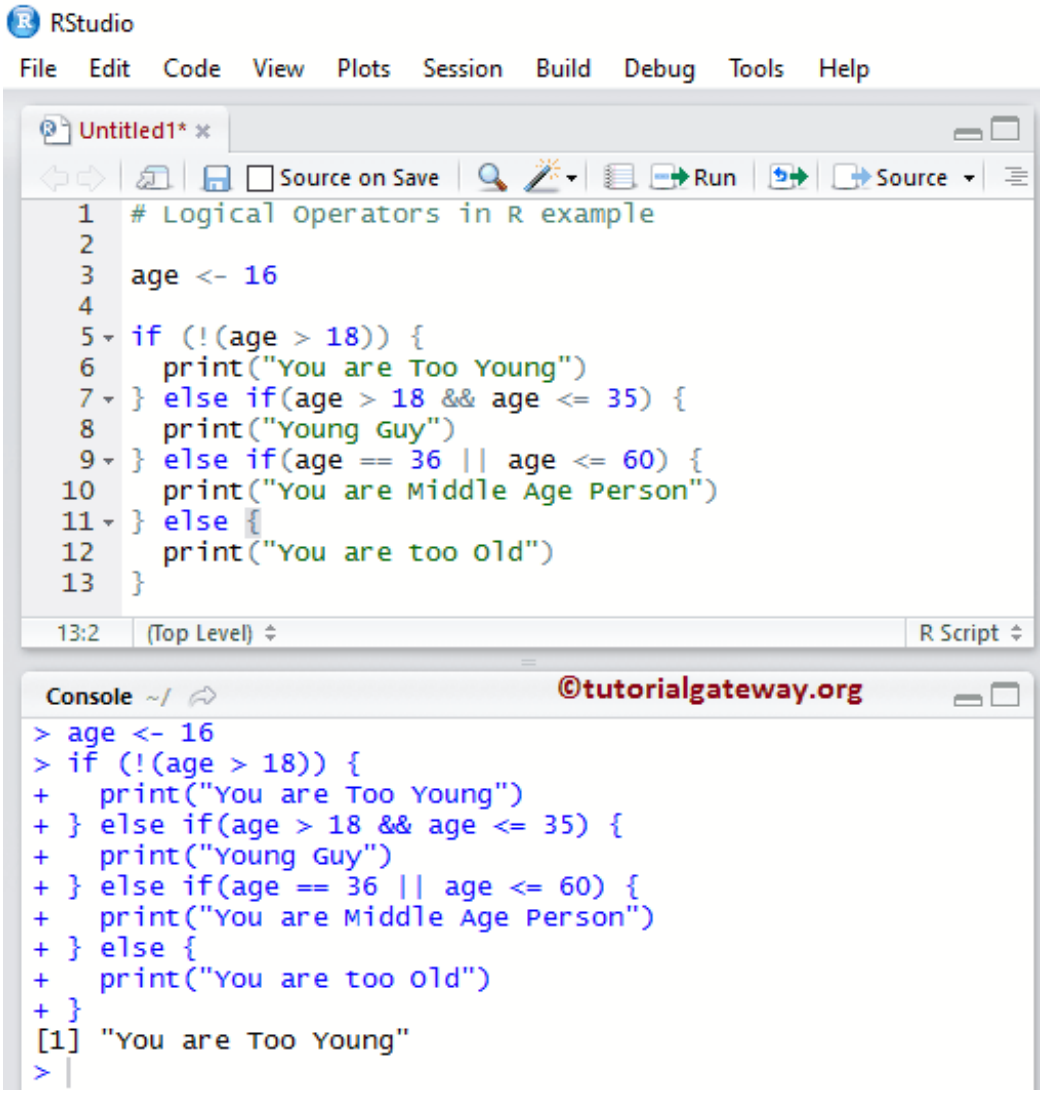
This example will help you understand, how the logical operators in R Programming are used in If statements. For this example, We are assigned one integer variables and then, inside the If Statement we are using basic logical operators such as : &&, ||, and !.

R CODE

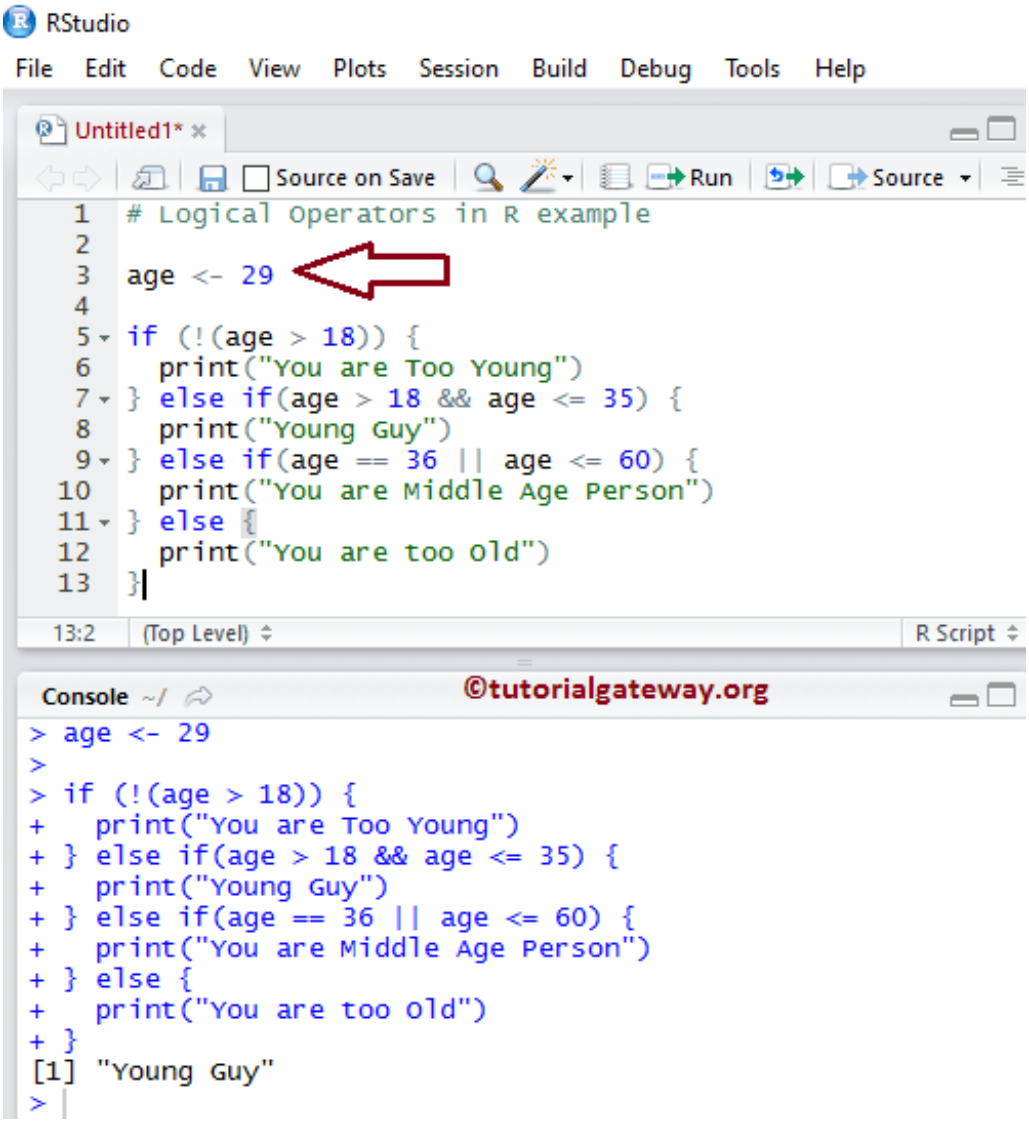
```
1   # Logical Operators in R example
2   age<-16
3   if(!(age>18)){
4   print("You are Too Young" )
5   }elseif(age>18&&age<=35){
6   print("Young Guy")
7   }elseif(age==36||age<=60){
8   print("You are Middle Age Person" )
9   }else{
10  print("You are too Old" )
11  }
12
```

OUTPUT

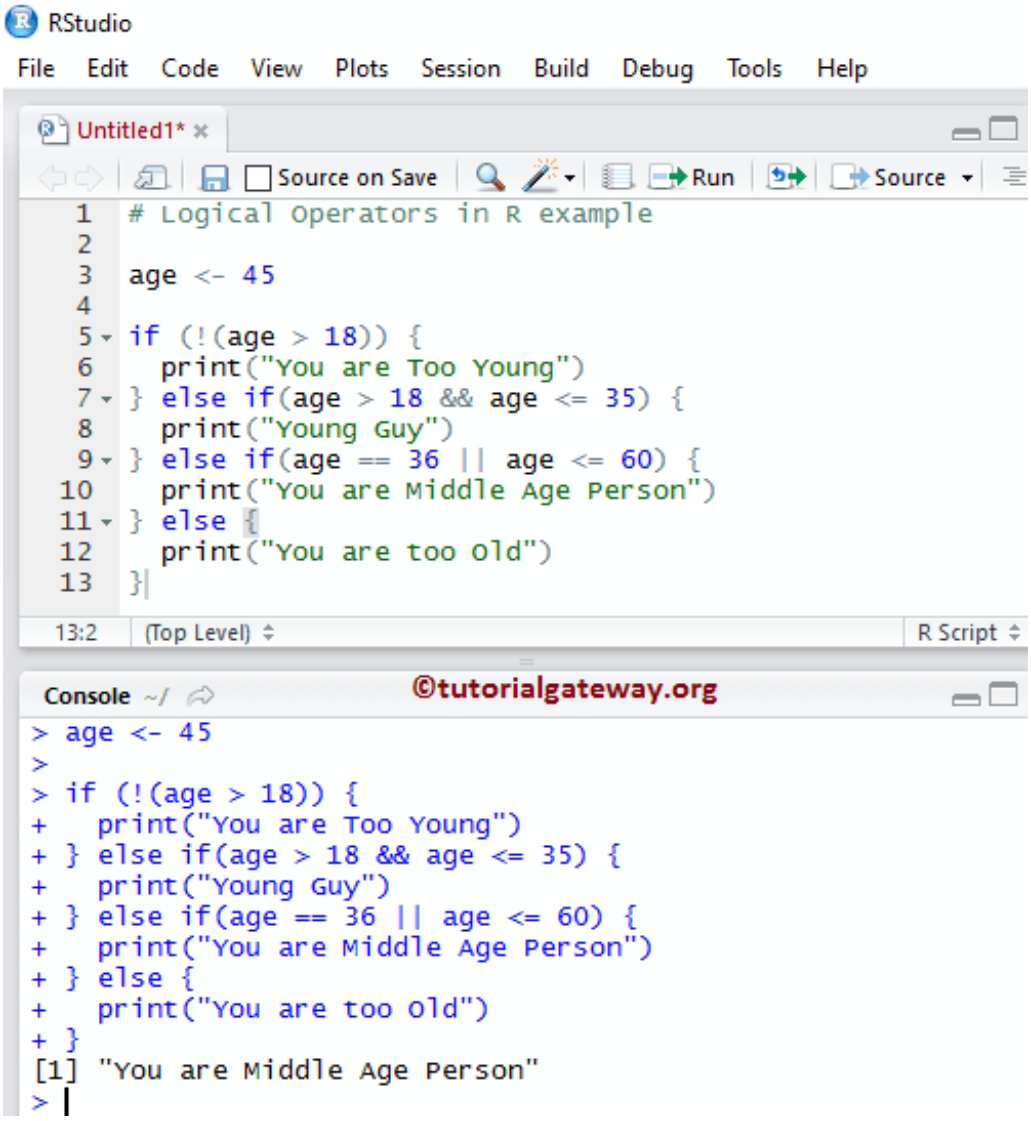
From the below screenshot you can observe that, we entered age = 16. It means, age is not greater than 18 so, First statement is printed



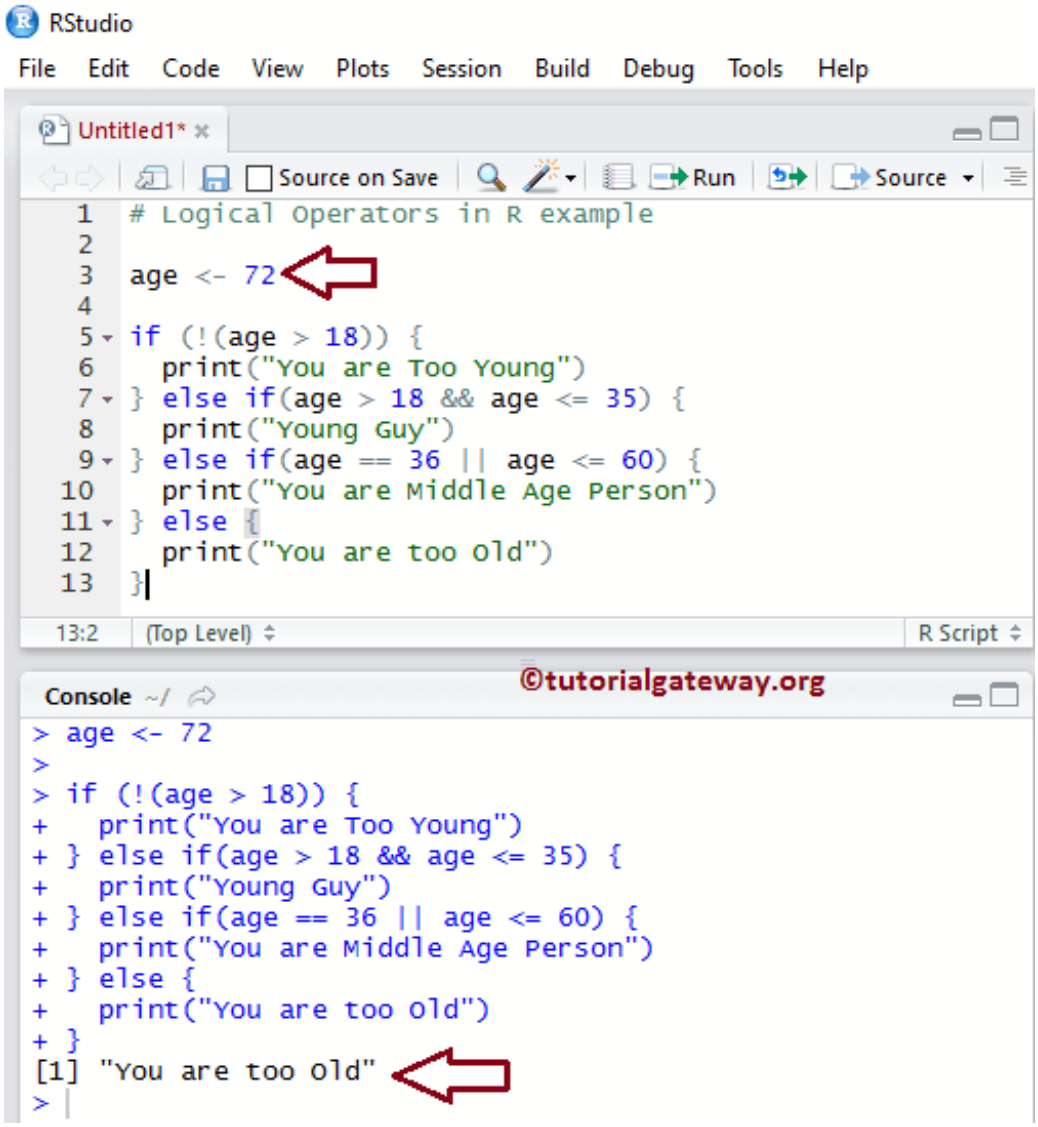
Let us see what will happen when we change the values. From the below screenshot you can observe that, we have entered age = 29. It means age is between 18 and 35 so, Second statement is printed



From the below screenshot you can observe that, we have entered age = 45. It means age is between 36 and 60 so, third statement is printed



From the below screenshot you can observe that, we have entered age = 72.



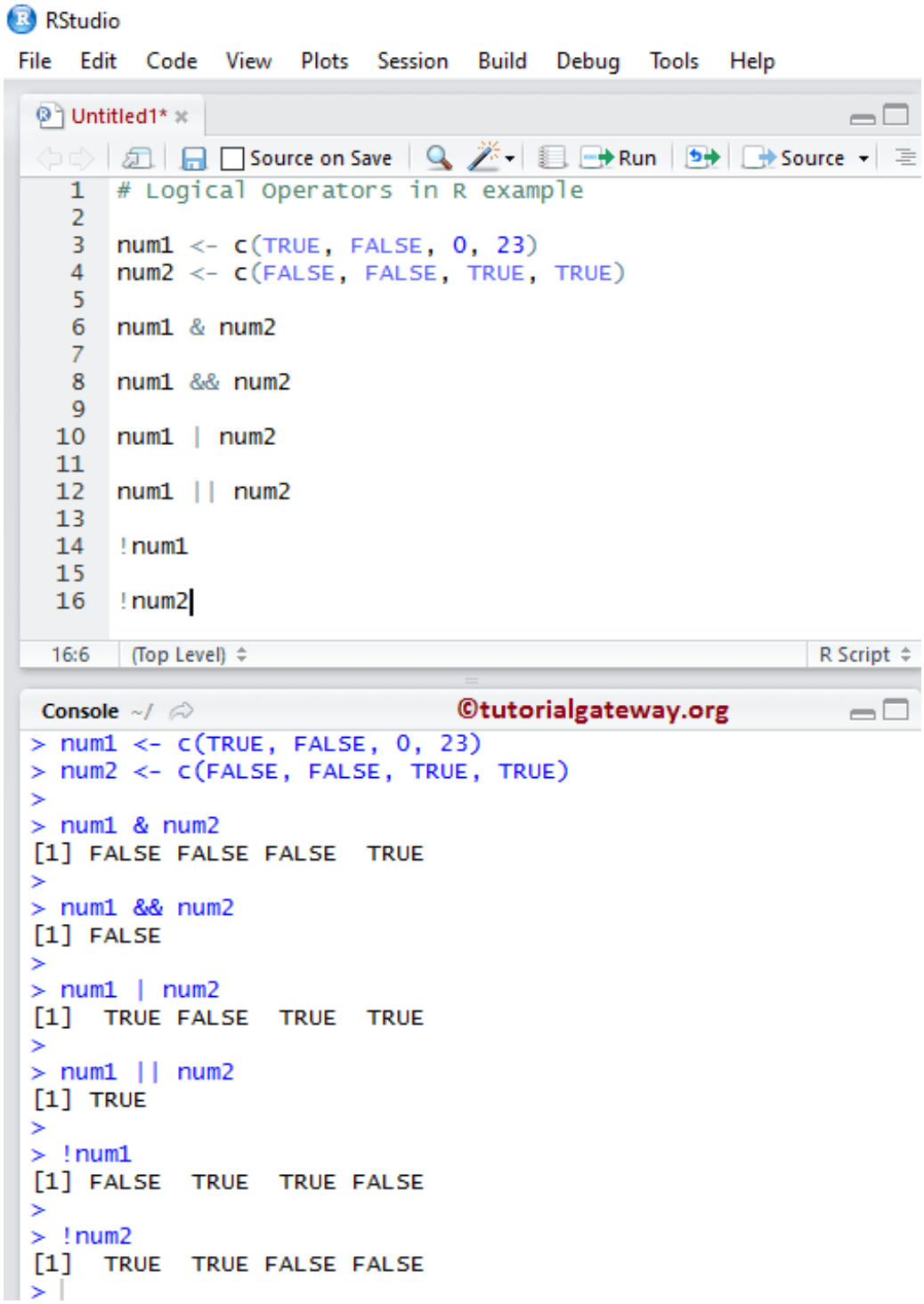
R Logical Operators example

This example will help you understand, how each and every R logical operator will work. Remember, any positive integer value greater than zero is considered as Boolean TRUE, and 0 is considered as Boolean False.

R CODE

```
1  # Logical Operators in R example
2  num1<-c(TRUE,FALSE,0,23)
3  num2<-c(FALSE,FALSE,TRUE,TRUE)
4  # Performs logical AND operation on each and every element in both num1, num2
5  num1&num2
6  # Performs logical AND operation on first element in both num1, num2
7  num1&&num2
8  # Performs logical OR operation on each and every element in both num1, num2
9  num1|num2
10 # Performs logical OR operation on first element in both num1, num2
11 num1||num2
12 Thiswill convert all the num1 TRUEvalues toFALSE,andFALSEvalues toTRUE
13 !num1
14 # From num2 Vector - This will convert all the TRUE values to FALSE, and FALSE to TRUE
15 !num2
16
17
18
19
20
21
22
```

OUTPUT



ANALYSIS

First we declared two vectors

```
1  num1<-c(TRUE,FALSE,0,23)
2  num2<-c(FALSE,FALSE,TRUE,TRUE)
```

Below statement will compare each and every vector element, and find the logical relation.

```
1  num1&num2
```

Following statement will compare the first element of num1 vector and the first element of num2 vector. It means, TRUE && FALSE = FALSE.

```
1    num1&&num2
```

Thank You for Visiting Our Blog.