# Fix in R: Error: attempt to apply non-function

Authored by

**Mohammed looti**

October 30, 2025

One of the most frequently encountered syntax errors when working with the R programming language is the cryptic message:

**Error: attempt to apply non-function**

This error typically indicates that the R interpreter has encountered an object--such as a variable, a vector, or a column within a data frame--followed immediately by an opening parenthesis. In R's syntax rules, this structure signals the invocation of a function, where the content inside the parentheses is expected to be the arguments passed to that function. When the object preceding the parenthesis is not a callable function, the execution fails, triggering this specific error.

In the vast majority of cases involving arithmetic operations, this error arises simply because a crucial operator, usually the multiplication sign (**`*`**), has been omitted. Unlike standard mathematical notation where multiplication can be implied (e.g., 5x), R requires explicit mathematical operators for all calculations. This comprehensive guide details the exact steps required to diagnose and resolve this issue across two common data structures: the data frame and the atomic vector.

## Decoding the "Attempt to Apply Non-Function" Error

To properly debug this issue, it is vital to understand the underlying logic of the R language parser. When R executes a script, it processes expressions sequentially. If it encounters a sequence like `object(argument)`, it inherently assumes that `object` is a reference to a defined function. The interpreter then attempts to look up the object, confirm its type is indeed a function, and execute it using the supplied arguments.

If the object is instead a numeric scalar, a character string, or a vector of values (i.e., data), R cannot proceed with the function call. Because data structures are not executable, R reports that you are attempting to "apply" something that is not a function. This scenario is most frequent when performing scalar multiplication against a data structure, or when attempting element-wise multiplication between two data structures without specifying the required operator.

Understanding this distinction--between an operation that requires an explicit arithmetic operator and a function call that uses parentheses for arguments--is the first step toward writing cleaner and more reliable R code. We will now explore practical examples demonstrating how to correctly insert the multiplication operator to ensure R processes the arithmetic operation as intended.

## Scenario 1: Resolving Error in Data Frame Column Multiplication

Working with data frames is central to data manipulation in R. A common task involves creating a new column derived from calculations performed on existing columns. Suppose we start by defining a simple data frame named `df`, containing columns `x` and `y`.

**#create data frame**
**df <- data.frame(x=c(1, 2, 6, 7),**
**y=c(3, 5, 5, 8))**

#view data frame
df

x y
1 1 3
2 2 5
3 6 5
4 7 8

The objective is to generate a new column, `x_times_10`, where each element is the corresponding value in column `x` multiplied by 10. If we mistakenly rely on implicit multiplication, or use parentheses immediately following the column reference, R interprets this as a function call targeting the contents of `df$x`.

**#attempt to create new column without the multiplication operator**
**df$x_times_10 <- df$x(10)**

Error: attempt to apply non-function

The error occurs because `df$x` returns a numeric vector, which is data, not a callable function. The parser sees `df$x` followed by `(10)` and attempts to execute the vector as if it were a function named after the column. To resolve this, we must explicitly insert the multiplication operator (**`*`**) between the column reference and the scalar value.

By including the required operator, we instruct R to perform the element-wise multiplication correctly. The corrected code snippet below successfully generates the desired new column, illustrating the strict requirement for explicit arithmetic syntax within the R environment.

**#correct syntax using the explicit multiplication operator**
**df$x_times_10 <- df$x*(10)**

#view updated data frame
df

x y x_times_10
1 1 3 10
2 2 5 20

3 6 5 60
4 7 8 70

## Scenario 2: Resolving Errors in Vector Multiplication

The "attempt to apply non-function" error is equally common when dealing directly with atomic vectors, particularly when attempting to multiply two vectors together. Suppose we define two numeric vectors, `x` and `y`, and our goal is to calculate the product of their corresponding elements (element-wise multiplication).

A common mistake, especially for programmers transitioning from other languages or mathematical backgrounds, is to enclose both vectors in parentheses and place them adjacent to each other, such as `(x)(y)`. This syntax is disastrous in R because the first parenthetical expression, `(x)`, evaluates to the vector `x`. R then immediately interprets this result as a function that should be applied to the next expression, `(y)`, which is treated as the argument.

**#create two vectors**
**x <- c(1, 2, 2, 2, 4, 5, 6)**
**y <- c(5, 6, 8, 7, 8, 8, 9)**

#erroneous attempt to multiply corresponding elements in vectors
(x)(y)

Error: attempt to apply non-function

Since vector `x` is composed of numeric values and is not a callable function, the interpreter halts and produces the error. The solution, just as in the data frame scenario, lies in inserting the explicit multiplication operator (*****) between the two vectors. This signals to R that the desired operation is element-wise arithmetic, which is the default behavior for vector operations in R when using binary operators.

The corrected command yields the expected result: a new vector containing the products of the corresponding elements of `x` and `y`, confirming that the proper use of the ***** operator is the key to executing arithmetic calculations successfully within the R environment.

**#correctly multiply corresponding elements in vectors**
**(x)*(y)**

5 12 16 14 32 40 54

## Best Practices for Avoiding Syntax Pitfalls in R

While the "attempt to apply non-function" error is often a simple fix, adopting strong coding habits can prevent its occurrence entirely, saving significant debugging time. The fundamental principle to remember when writing mathematical expressions in R is the absolute necessity of explicit operators. Never assume implied multiplication will work, regardless of context--whether you are multiplying two variables, a variable by a constant, or two complex data structures.

To maintain robust and readable code, always double-check the placement of common arithmetic operators. These operators must be clearly placed between the operands to define the intended mathematical relationship. Key operators that demand explicit placement include:

***** (Multiplication)
**+** (Addition)
**-** (Subtraction)
**/** (Division)
**^** (Exponentiation)

Furthermore, when writing complex expressions, use parentheses judiciously to control the order of operations, but be extremely cautious of placing parentheses immediately following a variable name when arithmetic is the goal. If you encounter this specific error, your debugging process should prioritize checking any variable or data structure reference that is immediately followed by a set of parentheses. Replacing the implicit function call syntax with the correct arithmetic operator (e.g., changing `A(B)` to `A * B`) will almost certainly resolve the issue.

## Conclusion and Additional Resources

The "Error: attempt to apply non-function" is a classic example of an R syntax trap that misinterprets intended arithmetic as a function call due to the omission of an explicit operator. By understanding how R parses expressions and consistently using the multiplication sign (*****) when performing arithmetic, developers can easily avoid this common frustration when working with data frames and vectors. Mastering this fundamental syntax rule is crucial for writing efficient and error-free scripts in R.

The following tutorials explain how to fix other common errors in R: