How to Create New Variables in R Using the Mutate Function

Authored by

stats writer

March 4, 2026

RECOMMENDED CITATION

stats writer (2026). *How to Create New Variables in R Using the Mutate Function*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=133843

The Mutate function in R allows users to create new variables by manipulating existing ones in a data frame. This function can be used to perform various operations such as mathematical calculations, logical comparisons, and text transformations. By specifying the desired formula or condition, the Mutate function can generate a new column with the resulting values. This feature is useful for data manipulation and analysis, as it allows for the creation of customized variables tailored to the specific needs of the user. With its flexible and efficient capabilities, the Mutate function is an essential tool for data scientists and analysts in R programming.

Use Mutate to Create New Variables in R

This tutorial explains how to use the mutate() function in R to add new variables to a data frame.

Adding New Variables in R

The following functions from the dplyr library can be used to add new variables to a data frame:

mutate() - adds new variables to a data frame while preserving existing variables

transmute() - adds new variables to a data frame and drops existing variables

mutate_all() - modifies all of the variables in a data frame at once

mutate_at() - modifies specific variables by name

mutate_if() - modifies all variables that meet a certain condition

mutate()

The mutate() function adds new variables to a data frame while preserving any existing variables. The basic synax for mutate() is as follows:

```
data <- mutate(new_variable = existing_variable/3)
```

data: the new data frame to assign the new variables tonew_variable: the name of the new variableexisting_variable: the existing variable in the data frame that you wish to perform some operation on to create the new variable

For example, the following code illustrates how to add a new variable *root_sepal_width* to the built-in *iris* dataset:

```
#define data frame as the first six lines of the iris dataset
data <- head(iris)

#view data
```

data

 # Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#1 5.1 3.5 1.4 0.2 setosa
#2 4.9 3.0 1.4 0.2 setosa
#3 4.7 3.2 1.3 0.2 setosa
#4 4.6 3.1 1.5 0.2 setosa
#5 5.0 3.6 1.4 0.2 setosa
#6 5.4 3.9 1.7 0.4 setosa

#load *dplyr* librarylibrary(dplyr)#define new column *root_sepal_width* as the square root of the *Sepal.Width* variable
data %>% mutate(root_sepal_width = sqrt(Sepal.Width))

 # Sepal.Length Sepal.Width Petal.Length Petal.Width Species root_sepal_width
#1 5.1 3.5 1.4 0.2 setosa 1.870829
#2 4.9 3.0 1.4 0.2 setosa 1.732051
#3 4.7 3.2 1.3 0.2 setosa 1.788854
#4 4.6 3.1 1.5 0.2 setosa 1.760682
#5 5.0 3.6 1.4 0.2 setosa 1.897367
#6 5.4 3.9 1.7 0.4 setosa 1.974842

transmute()

The transmute() function adds new variables to a data frame and drops existing variables. The following code illustrates how to add two new variables to a dataset and remove all existing variables:

```
#define data frame as the first six lines of the iris dataset
data <- head(iris)

#view data
data

# Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#1 5.1 3.5 1.4 0.2 setosa
#2 4.9 3.0 1.4 0.2 setosa
#3 4.7 3.2 1.3 0.2 setosa
#4 4.6 3.1 1.5 0.2 setosa
#5 5.0 3.6 1.4 0.2 setosa
#6 5.4 3.9 1.7 0.4 setosa

#define two new variables and remove all existing variables
```

```
data %>% transmute(root_sepal_width = sqrt(Sepal.Width),
root_petal_width = sqrt(Petal.Width))

# root_sepal_width root_petal_width
#1 1.870829 0.4472136
#2 1.732051 0.4472136
#3 1.788854 0.4472136
#4 1.760682 0.4472136
#5 1.897367 0.4472136
#6 1.974842 0.6324555
```

mutate_all()

The mutate_all() function modifies all of the variables in a data frame at once, allowing you to perform a specific function on all of the variables by using the funs()function. The following code illustrates how to divide all of the columns in a data frame by 10 using mutate_all():

```
#define new data frame as the first six rows of iris without the Species variable
data2 <- head(iris) %>% select(-Species)
```

```
#view the new data frame
data2

# Sepal.Length Sepal.Width Petal.Length Petal.Width
#1 5.1 3.5 1.4 0.2
#2 4.9 3.0 1.4 0.2
#3 4.7 3.2 1.3 0.2
#4 4.6 3.1 1.5 0.2
#5 5.0 3.6 1.4 0.2
#6 5.4 3.9 1.7 0.4

#divide all variables in the data frame by 10
data2 %>% mutate_all(funs(./10))

# Sepal.Length Sepal.Width Petal.Length Petal.Width
#1 0.51 0.35 0.14 0.02
#2 0.49 0.30 0.14 0.02
#3 0.47 0.32 0.13 0.02
#4 0.46 0.31 0.15 0.02
#5 0.50 0.36 0.14 0.02
#6 0.54 0.39 0.17 0.04
```

Note that additional variables can be added to the data frame by specifying a new name to be appended to the old variable name:

```
data2 %>% mutate_all(funs(mod = ./10))

# Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length_mod
#1 5.1 3.5 1.4 0.2 0.51
#2 4.9 3.0 1.4 0.2 0.49
#3 4.7 3.2 1.3 0.2 0.47
#4 4.6 3.1 1.5 0.2 0.46
#5 5.0 3.6 1.4 0.2 0.50
#6 5.4 3.9 1.7 0.4 0.54
# Sepal.Width_mod Petal.Length_mod Petal.Width_mod
#1 0.35 0.14 0.02
#2 0.30 0.14 0.02
#3 0.32 0.13 0.02
#4 0.31 0.15 0.02
#5 0.36 0.14 0.02
#6 0.39 0.17 0.04
```

mutate_at()

The mutate_at() function modifies specific variables by name. The following code illustrates how to divide two specific variables by 10 using mutate_at():

```
data2 %>% mutate_at(c("Sepal.Length", "Sepal.Width"),
```

```
funs(mod = ./10))

# Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length_mod
#1 5.1 3.5 1.4 0.2 0.51
#2 4.9 3.0 1.4 0.2 0.49
#3 4.7 3.2 1.3 0.2 0.47
#4 4.6 3.1 1.5 0.2 0.46
#5 5.0 3.6 1.4 0.2 0.50
#6 5.4 3.9 1.7 0.4 0.54
# Sepal.Width_mod
#1 0.35
#2 0.30
#3 0.32
#4 0.31
#5 0.36
#6 0.39
```


mutate_if()

The mutate_if() function modifies all variables that meet a certain condition. The following code illustrates how to use the mutate_if() function to convert any variables of type *factor* to type *character*:

```
#find variable type of each variable in a data frame
data <- head(iris)
sapply(data, class)

#Sepal.Length  Sepal.Width  Petal.Length  Petal.Width  Species
# "numeric" "numeric" "numeric" "numeric" "factor"

#convert any variable of type factor to type character
new_data <- data %>% mutate_if(is.factor, as.character)
sapply(new_data, class)

#Sepal.Length  Sepal.Width  Petal.Length  Petal.Width  Species
# "numeric" "numeric" "numeric" "numeric" "character"
```

The following code illustrates how to use the mutate_if() function to round any variables of type *numeric* to one decimal place:

```
#define data as first six rows of iris dataset
data <- head(iris)

#view data
data
```

```
# Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#1 5.1 3.5 1.4 0.2 setosa
#2 4.9 3.0 1.4 0.2 setosa
#3 4.7 3.2 1.3 0.2 setosa
#4 4.6 3.1 1.5 0.2 setosa
#5 5.0 3.6 1.4 0.2 setosa
#6 5.4 3.9 1.7 0.4 setosa

#round any variables of type numeric to one decimal place
data %>% mutate_if(is.numeric, round, digits = 0)

# Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#1 5 4 1 0 setosa
#2 5 3 1 0 setosa
#3 5 3 1 0 setosa
#4 5 3 2 0 setosa
#5 5 4 1 0 setosa
#6 5 4 2 0 setosa
```

Further reading:

A Guide to apply(), lapply(), sapply(), and tapply() in R

How to Arrange Rows in R

How to Filter Rows in R