How to use the R case_when function

March 17, 2021 by Joshua Ebner

This tutorial will show you how to use the case_when function in R to implement conditional logic like if/else and if/elif/else.

It explains the syntax, and also shows clear examples in the examples section.

You can click on any of the links below, and it will take you to the appropriate section in the tutorial.

Table of Contents:

Having said that, the tutorial might make more sense if you read it start to finish.

With that in mind, let's jump in.

A QUICK INTRODUCTION TO CASE_WHEN

Frequently, when we're doing data manipulation in R, we need to modify data based on various possible conditions.

This is particularly true when we're creating new variables with the mutate function from dplyr.

To show you this, let's look at an example.

Let's say that there's a class of students in a statistics class. These students take a test, and they get a score of 0 to 100 on the test.

Based on their test score, each student will get a test grade:

- If the score is greater than or equal to 90, assign an 'A'
- Else if the score is greater than or equal to 80, assign a 'B'
- Else if the score is greater than or equal to 70, assign a 'C'
- Else if the score is greater than or equal to 60, assign a 'D'
- Else, assign an 'F'

You have some information

student	test_score	grade
natascha	94	A
arun	90	A
mike	88	B
steve	75	C
james	65	D
ashley	65	D
oscar	45	F

You want to generate new information, based on different cases

So you have one piece of information, and based on that information, you're trying to generate new values based on conditions. You need to generate new information with some if-elif-else style logic.

How do you do this in R?

You can do it in R with the `case_when()` function.

To understand how, let’s look at the syntax.

THE SYNTAX OF CASE_WHEN

Here, we’ll look at the syntax of `case_when`.

The case_when syntax can be little bit complex, especially if you use it with multiple possible cases and conditions.

That being the case, I’ll try to explain this in stages, to help you understand.

We’ll first look at the syntax for a very simple use of case_when, and then we’ll move on to a use that has multiple conditions.

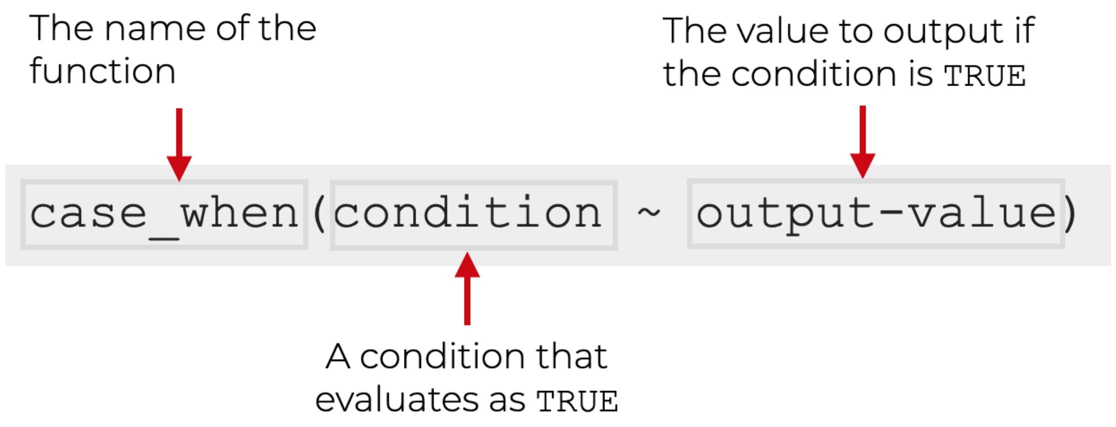
CASE_WHEN WITH A SINGLE CASE

Let’s first look at a simple example of the syntax.

We can use case_when to implement a simple sort of logic, where the function just tests for s single condition, and outputs a value if that condition is `TRUE`.

To do this syntactically, we simply type the name of the function: `case_when()`.

Then, inside the parenthesis, there is an expression with a “left hand side” and a “right hand side,” which are separated by a tilde (`~`).


The name of the function
The value to output if the condition is TRUE
case_when(condition ~ output-value)
A condition that evaluates as TRUE

THE LEFT HAND SIDE IS A CONDITION

Inside the parenthesis of case_when, the left hand side is a conditional statement that should evaluate as TRUE or FALSE.

This condition is the condition that we're looking for that indicates membership in a particular case.

This will almost always be a:

- Comparison operation (i.e., >=)
- Compound logical expression that combines multiple comparison operations with the and/or/not operators (&, |, !)

Essentially, the left hand side of the expression needs to be a logical expression that evaluates as TRUE or FALSE.

This is the "match condition" that we're looking for to match a particular "case."

THE RIGHT HAND SIDE PROVIDES A REPLACEMENT VALUE

The right hand side of the expression provides the replacement value.

So if the left hand side is looking for the values that match a particular case, the right hand side of the expression provides the output of case_when() for that case.

This explanation above explains how case_when() works if we have a single condition and case that we're looking for.

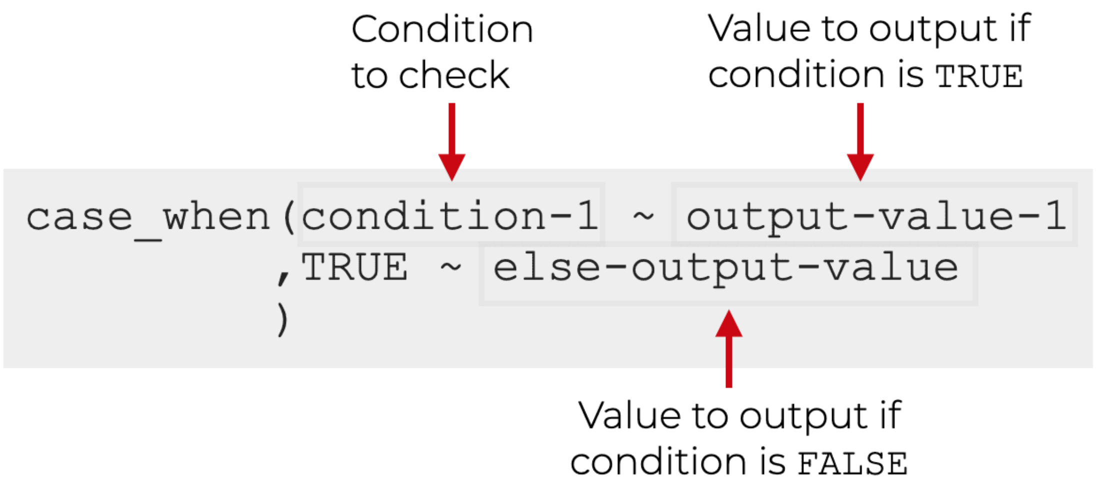
But the real power of case_when() comes in when you're using it to implement if/else logic, or if/elif/else logic with multiple cases.

Let's take a look at the syntax for those

USING CASE_WHEN TO IMPLEMENT IF/ELSE LOGIC

In the syntax explanation immediately above, I showed you how to use case_when with a simple condition, but nothing else.

Here, we'll look at the syntax that searches for a condition and assigns an output if that condition is TRUE. But if the condition is FALSE, output a different value.


Condition to check
Value to output if condition is TRUE
Value to output if condition is FALSE

```
case_when(condition-1 ~ output-value-1
         ,TRUE ~ else-output-value
         )
```

In this syntax for if-else using case_when, you might have noticed the TRUE syntax in the second line. Why do we need this?

```
case_when(condition-1 ~ output-value-1
         ,TRUE ~ else-output-value
         )
```

This forces `case_when` to output the "`else-output-value`", if none of the previous conditions were `TRUE`

Remember from the earlier section that when we use case_when, we use two-sided expressions to evaluate a condition, and then output a value if that condition is TRUE. If the left hand side is TRUE, then case_when() outputs the value on the right hand side.

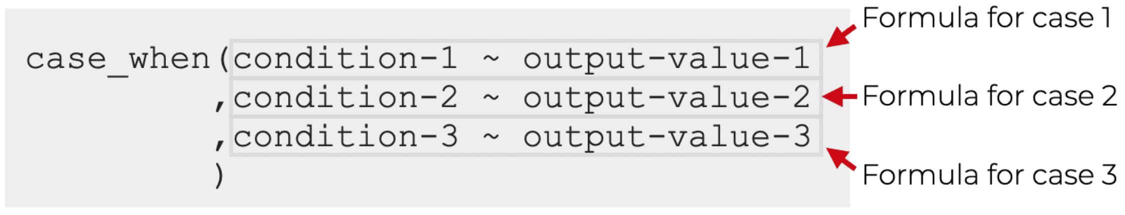
In this syntax example here, the second line hard-codes the value TRUE in that final two-sided expression. This forces case_when to output the "else-output-value" if none of the previous conditions were TRUE.

CASE_WHEN WITH MULTIPLE CASES

Now that we've looked at two examples with one condition, let's look at how case_when() works when we have multiple cases.

The case_when syntax that tests for different cases is similar to the syntax for one case.

When we have multiple cases, we have "a sequence of two-sided formulas." Said differently, the syntax will have a sequence of multiple formulas for a "test condition" and "output".



```
case_when(condition-1 ~ output-value-1
         ,condition-2 ~ output-value-2
         ,condition-3 ~ output-value-3
         )
```

So in the above image, condition-1 is a logical condition that tests for the first case, and output-value-1 is the output. Then condition-2 is a logical condition that tests for the second case. And so on.

Although the above image shows equations for three cases, we can technically have many more (although, the code would get messy).

SYNTAX FOR AN IF/ELIF/ELSE" STATEMENT

Before we look at some examples, there's one last bit of syntax that I'll explain.

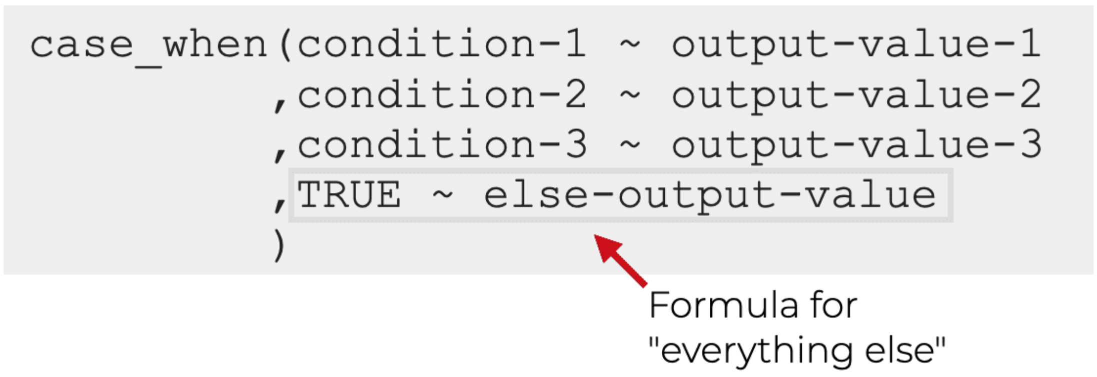
When you're using case_when with multiple cases, it's like using multiple if-else statements, where you test the first condition, and then output a value if condition 1 is true. Then you test the second condition, and output a different value if condition 2 is true. And so on.

But typically, when you do multiple if-else statements, there's a final "else" that provides an output if none of the previous conditions were true.

How do we do that with case_when?

I actually showed you this earlier in the syntax explanation for if/else logic, but let's look at it here in the context of if/elif/else.

If we're implementing if/elif/else logic, we need to have a final two-sided formula (after the other two-sided formulas), that specifies a value to output if none of the other conditions were true.



```
case_when(condition-1 ~ output-value-1
         ,condition-2 ~ output-value-2
         ,condition-3 ~ output-value-3
         ,TRUE ~ else-output-value
         )
```

Notice exactly how we do this.

On the right hand side of the final two-sided equation, we have the “else” output value.

But the *left* hand side of the final two-sided equation is the boolean value TRUE.

Why?

Remember, for every two-sided formula, if the left hand side is TRUE, then it outputs the right hand side.

So for this final formula, we *force* this to evaluate as TRUE by literally using the value TRUE. This forces case_when to output the “ else-output-value” for any remaining values that weren’t previously categorized.

It’s a bit of a syntactical hack to force case_when to categorize “everything else”.

I realize that all of this might seem a little abstract, and possibly a little difficult to understand.

Because of that, I think it's very useful to look at examples of how to use case_when with real data.

So let's do that.

EXAMPLES OF HOW TO USE CASE_WHEN IN R

Here we'll take a look at several examples of how to use the R case_when function.

For simplicity and clarity, we're going to start with a simple example of how to use case_when on an R vector.

But since we commonly use case_when with *dataframes*, the remaining examples will show you how to use case_when on an R dataframe.

You can click on any of the following links, and it will take you to the appropriate example.

Examples:

- Use case_when to perform a simple if_else
- Use case_when to perform if-elif-else
- Use case_when to do if-else, and create a new variable in a dataframe
- Create new variable by multiple conditions via mutate (if-elif-else)
- Create a new variable in a dataframe with case_when, using compound logical conditions

RUN THIS CODE FIRST

Before you run the examples, you’ll need to run some code to import the case_when function, and also to create some data that we’ll work with.

IMPORT DPLYR

The case_when function is part of the dplyr library in R.

Having said that, you’ll need to import dplyr explicitly or import the tidyverse package (which includes dplyr).

You can do that by running the following:

```
library(dplyr)
```

Or alternatively, you can import the Tidyverse like this:

```
library(tidyverse)
```

CREATE DATA

In the following examples, we’re going to work with a vector of data, and also a dataframe.

You can run this code to create the vector:

```
test_score_vector <- c(94,90,88,75,66,65,45)
```

This vector contains several numbers that represent student test scores.

We'll also create a dataframe called test_score_df that contains related data.

```
test_score_df <- tribble(~student, ~major, ~test_score

                  ,'natascha', 'business', 94
```

```
                    ,'arun', 'statistics', 90
                    ,'mike', 'statistics', 88
                    ,'steve', 'statistics', 75
                    ,'james', 'business', 66
                    ,'ashley', 'statistics', 65
                    ,'oscar', 'statistics', 45
                    )
```

The numbers in the `test_score` variable are the same numbers from `test_score_vector`.

But the `test_score_df` dataframe also contains student names and each student's major (in the `student` variable and `major` variable, respectively).

Once you run the code to create these datasets, you'll be ready to go.

EXAMPLE 1: USE CASE_WHEN TO PERFORM A SIMPLE IF_ELSE

First, we'll do a very simple example.

Here, we're going to operate on the vector `test_score_vector`, which contains test scores for seven students.

We're going to use case_when to assign a Pass/Fail grade for each score.

If the test score is greater than or equal to 60, case_when will return '`Pass`'.

Otherwise, case_when will return 'Fail'.

Let's take a look:

```
case_when(test_score_vector >= 60 ~ 'Pass'

          ,TRUE ~ 'Fail'

          )
```

OUT:

```
[1] "Pass" "Pass" "Pass" "Pass" "Pass" "Pass" "Fail"
```

EXPLANATION

This is fairly simple, but let me explain.

Inside the parenthesis of case_when, we have the expression test_score_vector >= 60 ~ 'Pass'. This checks each value of test_score_vector to see if the value is greater than or equal to 60. If the value meets this condition, case_when returns 'Pass'.

However, if a value does *not* match that condition, then case_when moves to the next condition.

You'll see on the second line, we have the expression TRUE ~ 'Fail'. This effectively assigns the value 'Fail' to all of the values that didn't match the first condition.

This is like a catch-all "else" statement in a typical if/else statement.

EXAMPLE 2: USE CASE_WHEN TO PERFORM IF-ELIF-ELSE

Next, we're going to use case_when() on a vector of data, test_score_vector, but we're going to use it to test multiple cases and assign the following values:

- If test_score_vector is greater than or equal to 90, assign 'A'
- Else if test_score_vector is greater than or equal to 80, assign 'B'
- Else if test_score_vector is greater than or equal to 70, assign 'C'
- Else if test_score_vector is greater than or equal to 60, assign 'D'
- Else, assign 'F'

So we're going to use case_when() as an if-elif-else statement, applied to a vector of data.

Let's take a look.

```
case_when(test_score_vector >= 90 ~ 'A'

          ,test_score_vector >= 80 ~ 'B'

          ,test_score_vector >= 70 ~ 'C'

          ,test_score_vector >= 60 ~ 'D'

          ,TRUE ~ 'F'

          )
```

OUT:

```
[1] "A" "A" "B" "C" "D" "D" "F"
```

EXPLANATION

So what happened here?

The input was the vector test_score_vector, which contained the values c(94,90,88,75,66,65,45).

The output was the values "A" "A" "B" "C" "D" "D" "F".

Essentially, case_when evaluated each number in the input vector, and assigned an output value depending on that input:

- If the value was greater than or equal to 90, it assigned the value 'A'.
- Then, if the value was greater than or equal to 80, but less than 90, it assigned the value 'B'.
- etc

So depending on the input number, it assigned a letter score of A, B, C, D, or F ... just like most grading schemes in the USA.

Notice as well the final line of the case_when statement. The final line TRUE ~ 'F' effectively assigns the value 'F' as an "else" value, if none of the previous conditions were TRUE.

EXAMPLE 3: USE CASE_WHEN TO DO IF-ELSE, AND CREATE A NEW VARIABLE IN A DATAFRAME

Next, we're going to use case_when in the context of manipulating a *dataframe*.

This example will actually be almost exactly the same as example 1, but instead of operating on a vector, we'll operate on a dataframe.

So here, we're going to add a new variable to our dataframe, test_score_df. Specifically, we're going to add a variable called pass_fail_grade which will assign 'Pass' if the test score is greater than or equal to 60, and will assign 'Fail' otherwise.

To do this, we're going to use case_when, but we're going to use it *inside* of the dplyr mutate function.

Remember: the dplyr mutate function adds new variables to an R dataframe.

Let's take a look.

```
test_score_df %>%

  mutate(pass_fail_grade = case_when(test_score_vector >=
60 ~ 'Pass'

                                     ,TRUE ~ 'Fail'

                                     )

         )
```

OUT:

```
# A tibble: 7 x 4

  student   major      test_score pass_fail_grade

1 natascha business           94 Pass

2 arun     statistics         90 Pass

3 mike     statistics         88 Pass

4 steve    statistics         75 Pass

5 james    business           66 Pass
```

```
6 ashley   statistics         65 Pass

7 oscar    statistics         45 Fail
```

EXPLANATION

What happened here?

Notice that the output dataframe has a new variable called pass_fail_grade.

This variable contains the values Pass or Fail, which have been assigned depending on the value of test_score. If test_score is greater than or equal to 60, then the assigned value is Pass, else the assigned value is Fail.

Also take note that in order to do this, we needed to use case_when *inside* of mutate.

So the code starts at the top of the code with the name of the dataframe, test_score_df.

We used the pipe operator to pipe the dataframe into mutate, to create a new variable.

Inside of mutate, we call case_when.

case_when looks at the test_score variable, and tests different conditions for different cases, assigning a 'Pass' if test_score is greater than or equal to 60, else the assigning a value of Fail.

But importantly, the Pass/Fail output of case_when is being assigned to the new variable pass_fail_grade. This all happens inside of the mutate function.

I realize that this is a slightly more complicated application, but in reality, this is a very common way to use case_when in R. We commonly use case_when to create new variables in a dataframe, in conjunction with the mutate function.

EXAMPLE 4: CREATE NEW VARIABLE BY MULTIPLE CONDITIONS VIA MUTATE (IF-ELIF-ELSE)

Now, let's increase the complexity.

This example will be somewhat similar to example 3, in that we're going to operate on a dataframe.

But it's also similar to example example 2, in the sense that we'll use case_when to look for multiple different cases.

Here, we're going to start with the test_score_df dataframe. We'll pipe that into the mutate function, to create a new variable called test_grade. Inside of mutate, to generate the specific values of test_grade, we'll use case_when.

Let's take a look.

```
test_score_df %>%

  mutate(test_grade = case_when(test_score_vector >= 90 ~
'A'

                                ,test_score_vector >= 80 ~
'B'

                                ,test_score_vector >= 70 ~
'C'

                                ,test_score_vector >= 60 ~
'D'
```

```
                                ,TRUE ~ 'F'
                                )
  )
```
OUT:

```
# A tibble: 7 x 4
  student  major      test_score test_grade
1 natascha business           94 A
2 arun     statistics         90 A
3 mike     statistics         88 B
4 steve    statistics         75 C
5 james    business           66 D
6 ashley   statistics         65 D
7 oscar    statistics         45 F
```
EXPLANATION

If you understood example 2 and example 3, then this should make some sense.

Here, we're using case_when inside of mutate to create a new categorical variable.

The case_when function is operating on test_score, and outputs:

- 'A' if test_score is greater than or equal to 90
- 'B' if test_score is greater than or equal to 80
- 'C' if test_score is greater than or equal to 70

- 'D' if test_score is greater than or equal to 60
- 'F' if none of the previous conditions where true

It evaluates these conditions one at a time, from top to bottom, and if a condition is false, it just moves on to the next.

The output of case_when is being saved with the name test_grade, which mutate adds to the output dataframe.

EXAMPLE 5: CREATE A NEW VARIABLE IN A DATAFRAME WITH CASE_WHEN, USING COMPOUND LOGICAL CONDITIONS

Let's do one final example.

Here, we're going to add a variable with a Pass/Fail grade to our dataframe, test_score_df.

This is somewhat similar to example 3. Like example 3, we'll be adding a pass/fail variable to the dataframe.

But, there will be an important difference here.

In this example, we're going to use slightly more complex conditions to assign Pass or Fail.

We're going to assign the Pass/Fail grade based on two variables: test score and major.

Here, case_when will use the following logic:

- everyone who gets a score over 70 will pass
- If a person gets above a 60, and is **not** a statistics major, they will also pass

- everyone else will fail

So effectively, if a person gets between a 60 and 70 on the test, the Pass/Fail grade will depend on their major. In that range, people with a statistics major will fail, but everyone else will pass.

Let's take a look.

```
test_score_df %>%

  mutate(pass_fail_grade = case_when(test_score_vector >= 70 ~ 'Pass'

                                     ,(test_score_vector >= 60) & (major != 'statistics') ~ 'Pass'

                                     ,TRUE ~ 'Fail'

                                     )

         )
```

OUT:

```
# A tibble: 7 x 4

  student  major      test_score pass_fail_grade

1 natascha business           94 Pass

2 arun     statistics         90 Pass

3 mike     statistics         88 Pass

4 steve    statistics         75 Pass

5 james    business           66 Pass

6 ashley   statistics         65 Fail
```

```
7 oscar    statistics         45 Fail
```

EXPLANATION

So what happened here?

Notice that everyone with a test score above 70 received a Pass grade.

Notice that everyone with a test score below 60 received a Fail grade.

But in the range between 60 and 70, there are two special cases (the records for james and ashley).

James had a test score of 66, but he's a business major, so he passed.

Ashley received a score of 65, but she's a statistics major, so she failed.

The logic for this was in the second line of case_when, with the code

```
(test_score_vector >= 60) & (major != 'statistics') ~ 'Pass' .
```

This code assigned a Pass grade if test score was greater than or equal to 60 AND major was **not** equal to 'statistics'. Effectively, any row of data that had grade between 60 and 70 and was anything other than a statistics major would evaluate as True on the left hand side of the expression, and would receive a Pass.

Rows of data with a test grade between 60 and 70 and a statistics major would evaluate as False, which would then cause case_when to evaluate the row of data with the expression TRUE ~ 'Fail', which would automatically assign a grade of 'Fail'.

Effectively, with this grading scheme, statistics majors are evaluated more strictly and must earn a test score above 70 in order to pass, but other majors only need to score above 60.

FREQUENTLY ASKED QUESTIONS ABOUT CASE_WHEN

Now that you've seen some examples of case_when, let's review some frequently asked questions about this function.

Frequently asked questions:

- How do you use case_when to perform if-else?
- How do you use case_when to perform if-elif-else?
- How do you use case_when to add a new variable to a dataframe?

QUESTION 1: HOW DO YOU USE CASE_WHEN TO PERFORM IF-ELSE?

To use case_when as an if-else generator, you simply have one test expression, and then a second catch-all expression at the end with the form TRUE ~ 'else-value'.

I covered this in example 1 and example 3.

Example 1 shows you how to do this with a vector of data.

Example 3 shows you how to do this with an R dataframe to create a new variable.

QUESTION 2: HOW DO YOU USE CASE_WHEN TO PERFORM IF-ELIF-ELSE?

To use case_when as an if-elif-else function, you will have several test conditions in sequence, and then a final catch-all expression at the end with the form TRUE ~ 'else-value'.

I covered this in example 2 and example 4.

Example 2 shows you how to do if/elif/else with a vector of data.

Example 4 shows you how to do if/elif/else with an R dataframe to create a new variable.

QUESTION 3: HOW DO YOU USE CASE_WHEN TO ADD A NEW VARIABLE TO A DATAFRAME?

To create a new variable in a dataframe using case_when, you need to use case_when inside of the dplyr mutate function.

I show examples of this in example 3, example 4, and example 5.

LEAVE YOUR OTHER QUESTIONS IN THE COMMENTS BELOW

Do you have other questions about case_when?

If so, leave your question in the comments section below.

JOIN OUR PREMIUM R DATA SCIENCE COURSE

The case_when function is extremely useful for doing data manipulation in R.

But, it's really one tool among several dozen tools in dplyr and the Tidyverse.

If you want to master data manipulation in R, you really need to master all of the other functions like mutate, filter, group_by, and many more.

And beyond that, there's more to learn about data visualization and data analysis in R too.

Having said that, if you're serious about learning dplyr, and data science in R, you should consider joining our premium course called *Starting Data Science with R*.

Starting Data Science will teach you all of the essentials you need to do data science in R, including:

- How to manipulate your data with dplyr
- How to visualize your data with ggplot2
- Tidyverse helper tools, like tidyr and forcats
- How to analyze your data with ggplot2 + dplyr
- and more ...

Moreover, it will help you completely *master* the syntax within a few weeks. We'll show you a practice system that will enable you to *memorize* all of the R syntax you learn. If you have trouble remembering R syntax, this is the course you've been looking for.

Find out more here: Learn More About Starting Data Science with R

Joshua Ebner

Joshua Ebner is the founder, CEO, and Chief Data Scientist of Sharp Sight. Prior to founding the company, Josh worked as a Data Scientist at Apple. He has a degree in Physics from Cornell University.