Writing functions that take data frame columns as arguments is a problem that most R users have been  
confronted with at some point. There are different ways to tackle this issue, and this blog post will  
focus on the solution provided by the latest release of the {rlang} package.

First, let’s load the {tidyverse}:

library(tidyverse)

library(rlang)

Let’s suppose that I need to write a function that takes a data frame, as well as a column from  
this data frame as arguments:

how\_many\_na <- function(dataframe, column\_name){

dataframe %>%

filter(is.na(column\_name)) %>%

count()

}

Let’s try this function out on the starwars data:

data(starwars)

head(starwars)

## # A tibble: 6 x 13

## name height mass hair\_color skin\_color eye\_color birth\_year gender

##

## 1 Luke… 172 77 blond fair blue 19 male

## 2 C-3PO 167 75 gold yellow 112

## 3 R2-D2 96 32 white, bl… red 33

## 4 Dart… 202 136 none white yellow 41.9 male

## 5 Leia… 150 49 brown light brown 19 female

## 6 Owen… 178 120 brown, gr… light blue 52 male

## # … with 5 more variables: homeworld , species , films ,

## # vehicles , starships

As you can see, there are missing values in the hair\_color column. Let’s try to count how many  
missing values are in this column:

how\_many\_na(starwars, hair\_color)

Error: object 'hair\_color' not found

R cannot find the hair\_color column, and yet it is in the data! Well, this is actually exactly  
the issue. The issue is that the column is inside the dataframe, but when calling the function  
with hair\_color as the second argument, R is looking for a variable called hair\_color that  
does not exist. What about trying with "hair\_color"?

how\_many\_na(starwars, "hair\_color")

## # A tibble: 1 x 1

## n

##

## 1 0

Now we get something, but something wrong!

One way to solve this issue, is to not use the filter() function, and instead rely on base R:

how\_many\_na\_base <- function(dataframe, column\_name){

na\_index <- is.na(dataframe[, column\_name])

nrow(dataframe[na\_index, column\_name])

}

how\_many\_na\_base(starwars, "hair\_color")

## [1] 5

This works, but not using the {tidyverse} at all is not an option, at least for me. For instance,  
the next function, which uses a grouping variable, would be difficult to implement without the  
{tidyverse}:

summarise\_groups <- function(dataframe, grouping\_var, column\_name){

dataframe %>%

group\_by(grouping\_var) %>%

summarise(mean(column\_name, na.rm = TRUE))

}

Calling this function results in the following error message:

Error: Column `grouping\_var` is unknown

Before the release of {rlang} 0.4.0 this is was the solution:

summarise\_groups <- function(dataframe, grouping\_var, column\_name){

grouping\_var <- enquo(grouping\_var)

column\_name <- enquo(column\_name)

mean\_name <- paste0("mean\_", quo\_name(column\_name))

dataframe %>%

group\_by(!!grouping\_var) %>%

summarise(!!(mean\_name) := mean(!!column\_name, na.rm = TRUE))

}

The core of the function remained very similar to the version from before, but now one has to  
use the enquo()–!! syntax. While not overly difficult to use, it is cumbersome.

Now this can be simplified using the new {{}} syntax:

summarise\_groups <- function(dataframe, grouping\_var, column\_name){

dataframe %>%

group\_by({{grouping\_var}}) %>%

summarise({{column\_name}} := mean({{column\_name}}, na.rm = TRUE))

}

Much easier and cleaner! You still have to use the := operator instead of = for the column name  
however. Also, from my understanding, if you want to modify the column names, for instance in this  
case return "mean\_height" instead of height you have to keep using the enquo()–!! syntax.