Base R Equivalents of dplyr Functions Part 1 – coalesce

Posted on May 15, 2024 by **Dave Rosenman** in **R bloggers** | 0 Comments

The dplyr `coalesce` function is incredibly useful and similar to the SQL `COALESCE` function. Given a set of vectors, it finds and keeps the first non-`NA` value at each position. For example, the following code returns the vector c(1, 2, 3, 4).

```
library(dplyr)
coalesce(c(1, NA, 3, NA), c(2, 2, 4, 4))
```

```
[1] 1 2 3 4
```

- At position 1, the first non-`NA` value is 1 (from the first vector).
- At position 2, the first non-`NA` value is 2 (from the second vector, because the value at position 2 in the first vector is NA).
- At position 3, the first non-`NA` value is 3 (from the first vector).
- Finally, at position 4, the first non-`NA` value is 4 (from the second vector, because the value at position 4 in the first vector is NA).

The `coalesce` function is not limited to two vectors. You can use as many vectors as you'd like.

```
coalesce(c(1, 2, NA, NA), c(3, 3, 3, NA), c(4, 4, 4, 4))
```

```
[1] 1 2 3 4
```

The vectors must be of equal length or length 1. Vectors of length 1 will be recycled. The following:

```
coalesce(c(1, NA, NA, 5), 3)
```

```
[1] 1 3 3 5
```

Is equivalent to:

```
dplyr::coalesce(c(1, NA, NA, 5), c(3, 3, 3, 3))
```

```
[1] 1 3 3 5
```

Most viewed posts (weekly)

PCA vs Autoencoders for Dimensionality Reduction
How to install (and update!) R and RStudio
{charcuterie} - What if Strings Were Iterable in R?
Auto XGBoost, Auto LighGBM, Auto CatBoost, Auto GradientBoosting
Object Oriented Programming in R (Part 4): Reference Classes & R6 Classes
Date Formats in R
Remove rows from dataframe based on condition in R

Sponsors

I most often use `coalesce` to replace all `NA` values in a vector with a single value. For example, the following code replaces all `NA` values with 0:

```
coalesce(c(1, NA, NA, 5, 6), 0)
```

```
[1] 1 0 0 5 6
```

Here are two alternative ways to do the same thing.

```
library(tidyr)
replace_na(c(1, NA, NA, 5, 6), 0) # this function is in the tidyr package
```

```
[1] 1 0 0 5 6
```

```
x <- c(1, NA, NA, 5, 6)
ifelse(is.na(x), 0, x)
```

```
[1] 1 0 0 5 6
```

Of the three options I have shown so far, I prefer `coalesce` and `replace_na`. (`coalesce` is a more general version of `replace_na`; `replace_na` takes a vector and a single value to replace the `NA` values in that vector with). If you want to go outside of base R and the tidyverse, `data.table::fcoalesce` is a much faster version of `dplyr::coalesce`.

Let's compare the speeds!

```
library(microbenchmark)
library(data.table)
set.seed(11)
x <- 1:10^7
x[sample(1:10^7, size = 10^6, replace = FALSE)] <- NA
microbenchmark(
  fcoalesce(x, 0L),
  coalesce(x, 0L),
  replace_na(x, 0L),
  ifelse(is.na(x), 0L, x),
  times = 5
  )
```

```
Unit: milliseconds
                     expr      min       lq      mean   median       uq      ma
         fcoalesce(x, 0L)  14.4582  16.1549  18.47224  18.5650  19.1356  24.047
          coalesce(x, 0L) 285.4586 289.1238 318.36890 292.7322 297.1620 427.367
        replace_na(x, 0L)  37.2726  41.0626  51.48104  41.4943  52.0923  85.483
 ifelse(is.na(x), 0L, x) 168.9397 178.8357 191.00360 180.9929 186.2458 240.003
 neval
     5
     5
     5
     5
```

`data.table:fcoalesce` is the winner in terms of speed, followed by `replace_na`. Of the four methods above, `coalesce` and `fcoalesce` are the most general, since they are not limited to replacing all `NA` values with a single value.

Is there a base R equivalent to `dplyr::coalesce`? No. But we can easily create one using just base R code.

To think about how we would do that, let's start with two vectors:

```
x <- c(1, 2, NA, NA)
y <- c(2, 2, 3, NA)
coalesce(x, y)
```

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

Our ads respect your privacy. Read our Privacy Policy page to learn more.

Contact us if you wish to place **your banner here**.

How could we get the same results using the `ifelse` function? It's simple. We return the value in `y` when the value in `x` is `NA`.

```
ifelse(is.na(x), y, x)
```

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

That's simple enough. But what if we want to use three vectors?

```
z <- c(4, 4, 4, 4)
coalesce(x, y, z)
```

```
[1] 1 2 3 4
```

We can start with our code from the case where we used two vectors.

```
output_step_1 <- ifelse(is.na(x), y, x)
```

When both `x` and `y` are `NA` (when `ifelse(is.na(x), y, x)` gives us `NA`), we want to use what is in z. Otherwise, we want to keep the results from step 1 above.

```
ifelse(is.na(output_step_1), z, output_step_1)
```

```
[1] 1 2 3 4
```

That worked! But what if we want to generalize this to any number of input vectors? We can use the base R function `Reduce`. For our case where we used `x`, `y`, and `z`, we could do:

```
 Reduce(function(x, y) ifelse(is.na(x), y, x), list(x, y, z))
```

```
[1] 1 2 3 4
```

`Reduce(f, list(x, y, z))`, where `f` is a function of two variables, is the equivalent of `f(f(x, y), z)`. And `Reduce(f, list(x, y, z, a))` is equivalent to `f(f(f(x, y), z), a)`. The Reduce function is used to iteratively apply a function to elements of a list, reducing it to a single value. It takes a function with two parameters and applies it to the first two elements of the list, then applies the same function to the result and the next element, and so on, until all elements are combined into a single value.

To use `Reduce` to mimic `coalesce(x, y, z, ...)`, we need to apply the logic `f <- function(x, y) { ifelse(is.na(x), y, x)}` over and over starting from left to right. In other words, for three vectors x, y, and z, we need to do:

```
f <- function(x, y) {
  ifelse(is.na(x), y, x)
}
f(f(x, y), z)
```

```
[1] 1 2 3 4
```

Which is equivalent to

```
Reduce(f, list(x, y, z))
```

```
[1] 1 2 3 4
```

So a very simple base R function equivalent to the `coalesce` function is:

```
coalesce_base_r <- function(...) {
  args <- list(...)
  Reduce(function(x, y) ifelse(is.na(x), y, x), args)
}
```

Recent Posts

Shiny in Production 2024: Full speaker lineup
How to Check if a String Contains Specific Characters in R: A Comprehensive Guide with Base R, string & stringi
PowerQuery Puzzle solved with R
Black Monday Crash
Checking If a Workbook is Open Using VBA and Executing from R
R Solution for Excel Puzzles
Object Oriented Programming in R (Part 4): Reference Classes & R6 Classes
Converting Text to Uppercase with toupper() in R
Git Tricks for Working with Large Repositories
How to Write Tests with shiny::testServer Similar to shinytest2
Systematic Sampleing in R with Base R
Auto XGBoost, Auto LighGBM, Auto CatBoost, Auto GradientBoosting
CopenhagenR, the 2024 spring season
{charcuterie} – What if Strings Were Iterable in R?
R Consortium Grants Committee Announces New Chair

Jobs for R-users

R Adoption Lead
R System Developer for The Institute of Marine Research (IMR) @ Bergen, Vestland, Norway
Principal Machine Learning Engineer @ New York, United States
Statistical Programmer for i360 @ Arlington, Virginia, United States
Biostatistician II

Let's see if it produces identical results to `dplyr::coalesce`:

```
set.seed(11)
x <- 1:10^7
y <- 1:10^7
x[sample(1:10^7, size = 10^6, replace = FALSE)] <- NA
y[sample(1:10^7, size = 10^6, replace = FALSE)] <- NA
z <- 1L

dplyr_result <- coalesce(x, y, z)
base_r_result <- coalesce_base_r(x, y, z)
identical(dplyr_result, base_r_result)
```

```
[1] TRUE
```

We get identical results!

Let's compare the speed:

```
microbenchmark(coalesce(x, y, z), coalesce_base_r(x, y, z), times = 5)
```

```
Unit: milliseconds
                     expr      min       lq     mean   median       uq      max
        coalesce(x, y, z) 299.6434 321.6311 355.9083 357.0767 385.4815 415.709
 coalesce_base_r(x, y, z) 349.0094 406.5152 400.0490 412.2628 413.9706 418.487
 neval
     5
     5
```

Our base R version of `coalesce` is almost identical in speed dplyr's!

But our function contains some flaws. dplyr's `coalesce` function forces the vectors passed to it to either be of the same length or be of length 1. If we try:

```
coalesce(c(1, 2, 3, NA, 6), c(4, 5))
```

```
Error in `coalesce()`:
! Can't recycle `..1` (size 5) to match `..2` (size 2).
```

We get an error, since the first vector has length 5 and the second has length 2.

Here's a better base R version of `coalesce`:

```
coalesce_base <- function(...) {
  args <- list(...)

  # Check for NULL, zero-length vectors, and collect lengths
  lengths <- sapply(args, function(x) {
    if (is.null(x) || length(x) == 0) {
      stop("Arguments must not be NULL or zero-length vectors")
    }
    length(x)
  })

  # Determine the maximum length
  max_length <- max(lengths)

  # Check if lengths are consistent.
  # Only allow vectors of length equal to max length or length of 1
  if (any(lengths != max_length & lengths != 1)) {
    stop("All arguments must have the same length,
         except for vectors of length 1 which can be recycled")
  }

  # Use Reduce with ifelse to coalesce
  Reduce(function(x, y) ifelse(is.na(x), y, x), args)
}

# Example usage:
v1 <- c(NA, 2, NA, 4, NA)
v2 <- c(1, NA, 3, NA, NA)
```

python-bloggers.com (python/data-science news)

Object Oriented Programming in R (Part 4): Reference Classes & R6 Classes
How to Write Tests with shiny::testServer Similar to shinytest2
MySQL EXPLAIN and EXPLAIN ANALYZE
T-test & ANOVA with Python VIDEO
Auto XGBoost, Auto LighGBM, Auto CatBoost, Auto GradientBoosting
SHAP from Scratch
All You Need to Know About Rhino 1.8 and 1.9 Updates: A Fireside Chat

Full list of contributing R-bloggers

R Posts by Year

Select Year

```
v3 <- 0

coalesce_base(v1, v2, v3)
```

```
[1] 1 2 3 4 0
```

Again, let's compare the speed of our function to `dplyr::coalesce` and `data.table::fcoalesce`.

```
microbenchmark(fcoalesce(x, y, z),
               coalesce(x, y, z),
               coalesce_base_r(x, y, z),
               times = 5)
```

```
Unit: milliseconds
                     expr      min       lq      mean   median       uq
       fcoalesce(x, y, z)  16.9233  17.5952  19.12082  17.6570  19.5384
        coalesce(x, y, z) 304.2171 321.0251 343.79622 355.3758 358.6747
 coalesce_base_r(x, y, z) 344.0853 357.0263 383.64736 386.5437 388.1294
      max neval
  23.8902     5
 379.6884     5
 442.4521     5
```

`data.table::fcoalesce` is the clear winner when it comes to speed! Our function is almost identical in speed to `dplyr::coalesce`!

Share Tweet

← **Previous post** **Next post** →