…Why you should know about it –

What does coalesce mean? In the English language, it is generally used to convey a coming together, or creating one whole body, mass or system. How does that help us when working with data? We spend a lot of time cleaning our data, surely the last thing we want to do is lump it all together?

In dplyr, coalesce takes after SQL's coalesce.

Any idea what this does?

```
SELECT COALESCE(colA,colB,colC) AS my_col
FROM my_table
```

It will find the first non NULL value in the 3 columns, and return it. If colA is NULL, but colB is populated, then colB is returned. if both colA and colB are NULL, and colC isn't, then colC is returned.

It's the same for R, but instead of NULL values, it is skipping over `NA`

The setup is simple:

```
library(tibble)
library(dplyr)
```

Here is some nice easy dummy data (don't ask about the names):

```
alpha <- c('a',NA,NA)
bravo <- c(NA,'b', NA)
cedric <- c(NA, NA,'c')
dave <-  c(NA,'dave','charlie')

data <- tibble(alpha,bravo,cedric,dave)
```

Which returns this tibble:

```
  alpha bravo cedric dave

1 a     NA    NA     NA
2 NA    b     NA     dave
3 NA    NA    c      charlie
```

OK , let's see what `coalesce` does.

```
coalesce(alpha)
1] "a" NA  NA
```

Now let's add in another column, where it might start to be more meaningful.

```
coalesce(alpha, bravo)
[1] "a" "b" NA
```

So we passed in :

```
  alpha bravo

1 a     NA
2 NA    b
3 NA    NA
```

In the first row, alpha has a value, so that is returned, in the second row, only bravo is populated, so that is returned.
The third row has 2 NA's, so NA is returned.

We can replace the NA, by supplying an alternative value:

```
 coalesce(alpha, bravo)
[1] "a" "b" NA

coalesce(alpha, bravo, 'replace')
[1] "a"       "b"       "replace"
```

You'll see the NA has gone, and our chosen value is in place.

```
coalesce(alpha, bravo, cedric)

# We pass in this:

#alpha bravo cedric
#
#1 a     NA    NA
#2 NA    b     NA
#3 NA    NA    c

# and we get back this:

[1] "a" "b" "c"
```

As expected, we get 'a' from alpha, 'b' from bravo, and 'c from cedric.

What about now? We pass in all 4 columns. Dave has values in both the second and third rows, so will any of them be returned?

```
coalesce(alpha, bravo, cedric,dave)
```

Can you guess the outputs?

```
[1] "a" "b" "c"
```

Remember, `coalesce` returns the *first* non NA value in each row, so in row 2, bravo supercedes dave, as does cedric in row 3.

If we change the order of the input vectors, and put dave first, then we get different results

```
coalesce(dave, bravo, cedric, alpha)
[1] "a"       "dave"    "charlie"
```

Why would you use this? Well, you might currently by doing a lot of `case_when()` statements, which you can avoid with this simple one liner. It's another little tool in the arsenal, and is truly one of the tidyverse's less glamourous, but no less useful functions.

One more example :

Here, we mutate a new column by coalescing three of our existing ones.

```
data %>%
  mutate(newcol = coalesce(alpha, bravo, dave))

  alpha bravo cedric dave    newcol

1 a     NA    NA     NA      a
2 NA    b     NA     dave    b
3 NA    NA    c      charlie charlie
```

You might think this would work :

```
data %>%
  mutate(newcol = coalesce(alpha:dave))
```

It doesn't , you'll get errors.

However , this does the job:

```
data %>%
  mutate(newcol = coalesce(!!!.))

 alpha bravo cedric dave    newcol

1 a     NA    NA     NA      a
2 NA    b     NA     dave    b
3 NA    NA    c      charlie c
```

I'm not sure how safe and reliable that is, so you are better to explicitly pass the column names:

```
data %>%
  select(alpha:cedric) %>%
  mutate(newcol = coalesce(.))

  alpha bravo cedric newcol$alpha $bravo $cedric

1 a     NA    NA     a            NA     NA
2 NA    b     NA     NA           b      NA
3 NA    NA    c      NA           NA     c
```

This is not what we intended.

How about this?

```
data %>%
  select(alpha:cedric) %>%
  mutate(newcol = coalesce(alpha, bravo, cedric))
```

```
alpha bravo cedric newcol

1 a     NA    NA     a
2 NA    b     NA     b
3 NA    NA    c      c
```

Yes, that works, and more importantly, future you will know what to expect.

To summarise, coalesce lets you

- select the first non NA value in one or more vectors / columns
- provide a value to replace the NA's
- skip having to write complicated case_when() statements to try and find the first non null value.