Adam theme

Adam Shen

Packages and setup

```
library(tidyverse)

set.seed(20)
```

The `rowwise` function

The `rowwise` function allows for the grouping of data by rows in order to perform some operation across the values found in its columns.

Some basic data:

```
nums <- tibble(
  x1 = sample(1:5, size=6, replace=TRUE),
  x2 = sample(1:5, size=6, replace=TRUE),
  x3 = sample(1:5, size=6, replace=TRUE),
  x4 = sample(1:5, size=6, replace=TRUE)
)

nums
```

```
## # A tibble: 6 x 4
##      x1    x2    x3    x4
##   <int> <int> <int> <int>
## 1     3     1     4     5
## 2     2     3     1     1
## 3     1     5     5     5
## 4     2     1     1     2
## 5     5     5     1     1
## 6     5     2     4     5
```

I won't actually be using `x4` in the demo below, but I'm including it to make the data a bit more realistic. Oftentimes, you will have more variables in your data set than the ones you are trying to operate on, i.e. you can't always just `tidyselect::everything()`!

Using `c_across` with `rowwise`

The `c_across` function is often used with `rowwise` in order to **combine values across columns**. What happens if we use `c_across` without `rowwise`?

```
nums %>%
  mutate(test = mean(c_across(x1:x3)))
```

```
## # A tibble: 6 x 5
```

```
##      x1    x2    x3    x4  test
##   <int> <int> <int> <int> <dbl>
## 1     3     1     4     5  2.83
## 2     2     3     1     1  2.83
## 3     1     5     5     5  2.83
## 4     2     1     1     2  2.83
## 5     5     5     1     1  2.83
## 6     5     2     4     5  2.83
```

It can be seen that the `test` column has a single value that is repeated throughout. Without grouping the data rowwise, the mean is computed using **all** the values across `x1`, `x2`, and `x3`:

```
mean(c(nums$x1, nums$x2, nums$x3))
```

```
## [1] 2.833333
```

Now, if we group the data rowwise before computing the mean across `x1`, `x2`, and `x3`:

```
nums %>%
  rowwise() %>%
  mutate(test = mean(c_across(x1:x3)))
```

```
## # A tibble: 6 x 5
## # Rowwise:
##      x1    x2    x3    x4  test
##   <int> <int> <int> <int> <dbl>
## 1     3     1     4     5  2.67
## 2     2     3     1     1  2
## 3     1     5     5     5  3.67
## 4     2     1     1     2  1.33
## 5     5     5     1     1  3.67
## 6     5     2     4     5  3.67
```

We can check that the results are what we wanted:

```
mean(c(3, 1, 4))
```

```
## [1] 2.666667
```

```
mean(c(5, 2, 4))
```

```
## [1] 3.666667
```

They match — great!

Note: `rowwise` is a type of grouping. From the first two lines of the tibble output above, we can see that after creating the `test` variable, the data is still grouped (rowwise)! Don't forget to `ungroup` when you're done mutating.

Rowwise matrix operations

Output is a vector of length 1

Suppose you wanted to perform the following computation using the values found in each row:

$$\mathbf{x}'\mathbf{A}\mathbf{x}$$

We will first create a function that:

1. Takes the combined values over a selection of columns (i.e. the result of `c_across`)
2. Converts the values to a matrix (a $n \times 1$ column vector)

3. Performs the matrix operation and reduces the resulting 1×1 matrix to a vector of length 1

```
quadratic <- function(x, A) {
  x <- as.matrix(x)

  drop(t(x) %*% A %*% x)
}
```

For the **A** matrix, let's use:

```
A <- matrix(
  sample(0:5, size=9, replace=TRUE),
  nrow=3, ncol=3
)

A
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    4
## [2,]    2    0    5
## [3,]    4    3    4
```

Putting it all together:

```
nums %>%
  rowwise() %>%
  mutate(test = quadratic(c_across(x1:x3), A)) %>%
  ungroup()
```

```
## # A tibble: 6 x 5
##      x1    x2    x3    x4  test
##   <int> <int> <int> <int> <dbl>
## 1     3     1     4     5   213
## 2     2     3     1     1    72
## 3     1     5     5     5   361
## 4     2     1     1     2    40
## 5     5     5     1     1   209
## 6     5     2     4     5   353
```

Checking our work:

```
quadratic(c(3, 1, 4), A)
```

```
## [1] 213
```

```
quadratic(c(5, 2, 4), A)
```

```
## [1] 353
```

Output is a vector of length greater than 1

Suppose you wanted to perform the following computation using the values found in each row:

$$\mathbf{Ax}$$

and wanted each value to go into its own column. This can be accomplished by first creating a function similar to the previous, but returning a list containing a named vector instead. Then we can use `unnest_wider` to

unnest the values within the list into their own columns.

Creating the right-multiplying function:

```
right_mult <- function(x, A) {
  x <- as.matrix(x)

  drop(A %*% x) %>%
    set_names(., paste0("new_x", 1:length(.))) %>%
    list()
}
```

Creating the new list-column:

```
nums %>%
  rowwise() %>%
  mutate(test = right_mult(c_across(x1:x3), A)) %>%
  ungroup()
```

```
## # A tibble: 6 x 5
##      x1    x2    x3    x4 test
##   <int> <int> <int> <int> <list>
## 1     3     1     4     5 <dbl [3]>
## 2     2     3     1     1 <dbl [3]>
## 3     1     5     5     5 <dbl [3]>
## 4     2     1     1     2 <dbl [3]>
## 5     5     5     1     1 <dbl [3]>
## 6     5     2     4     5 <dbl [3]>
```

Unnesting the list contents into their own columns:

```
nums %>%
  rowwise() %>%
  mutate(test = right_mult(c_across(x1:x3), A)) %>%
  ungroup() %>%
  unnest_wider(test)
```

```
## # A tibble: 6 x 7
##      x1    x2    x3    x4 new_x1 new_x2 new_x3
##   <int> <int> <int> <int>  <dbl>  <dbl>  <dbl>
## 1     3     1     4     5     21     26     31
## 2     2     3     1     1     12      9     21
## 3     1     5     5     5     31     27     39
## 4     2     1     1     2      8      9     15
## 5     5     5     1     1     19     15     39
## 6     5     2     4     5     25     30     42
```

Checking our work:

```
right_mult(c(3, 1, 4), A)
```

```
## [[1]]
## new_x1 new_x2 new_x3
##     21     26     31
```

```
right_mult(c(5, 2, 4), A)
```

```
## [[1]]
## new_x1 new_x2 new_x3
##     25     30     42
```