Making New Columns

Author: Cole Brookson **Date:** 20 July 2022

While summarizing columns is useful if we want to pare down our dataframe and get a summary, it's often the case we actually just want to add a new column to our dataframe, perhaps based on some set of other columns already in our dataframe.

In Tidy R, we will almost always use `dplyr::mutate()` for this task.

Mutating Columns with Single Values

The simplest use of `mutate()` is to make a new column that is made up of either a pre-existing object or a single value. For examples here, let's turn to a long-term dataset - sugar maple seedlings at Hubbard Brook Experimental Forest.

```
library(tidyverse)
library(lterdatasampler)
df = lterdatasampler::hbr_maples
head(df)
```

```
## # A tibble: 6 x 11
##    year watershed eleva~1 trans~2 sample stem_~3 leaf1~4 leaf2~5 leaf_~6 stem_~7
##   <dbl> <fct>     <fct>   <fct>   <fct>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1  2003 Reference Low     R1      1         86.9   13.8    12.1   0.0453  0.03
## 2  2003 Reference Low     R1      2        114     14.6    15.3   0.0476  0.0338
## 3  2003 Reference Low     R1      3         83.5   12.5     9.73  0.0423  0.0248
## 4  2003 Reference Low     R1      4         68.1    9.97   10.1   0.0397  0.0194
## 5  2003 Reference Low     R1      5         72.1    6.84    5.48  0.0204  0.018
## 6  2003 Reference Low     R1      6         77.7    9.66    7.64  0.0317  0.0246
## # ... with 1 more variable: corrected_leaf_area <dbl>, and abbreviated variable
## #   names 1: elevation, 2: transect, 3: stem_length, 4: leaf1area,
## #   5: leaf2area, 6: leaf_dry_mass, 7: stem_dry_mass
## # i Use ‘colnames()‘ to see all variable names
```

For this example, we only will work with some of the variables, so let's first select the columns we want (LINK TO SECTION ON SELECTION).

```
df = df %>%
  dplyr::select(year, watershed, elevation, leaf1area,
                leaf2area, corrected_leaf_area)
df
```

```
## # A tibble: 359 x 6
##     year watershed elevation leaf1area leaf2area corrected_leaf_area
##    <dbl> <fct>     <fct>         <dbl>     <dbl>               <dbl>
##  1  2003 Reference Low           13.8      12.1                29.1
##  2  2003 Reference Low           14.6      15.3                33.0
##  3  2003 Reference Low           12.5       9.73               25.3
##  4  2003 Reference Low            9.97     10.1                23.2
##  5  2003 Reference Low            6.84      5.48               15.5
##  6  2003 Reference Low            9.66      7.64               20.4
##  7  2003 Reference Low            8.82      9.23               21.2
## 8   2003 Reference Low           5.83      6.18               15.2
## 9   2003 Reference Low           8.11      7.13               18.4
## 10  2003 Reference Low           3.02      3.44                9.60
## # ... with 349 more rows
## # i Use ‘print(n = ...)‘ to see more rows
```

We could for example, want a column in this dataset that tells us what forest these data are from. That would be an easy use of the `mutate()` function as there's only one value. We could do this like so:

```
df = df %>%
  dplyr::mutate(forest = "HBEF")
df
```

```
## # A tibble: 359 x 7
##      year watershed elevation leaf1area leaf2area corrected_leaf_area forest
##     <dbl> <fct>     <fct>         <dbl>     <dbl>               <dbl> <chr>
##  1  2003 Reference Low          13.8      12.1                29.1  HBEF
##  2  2003 Reference Low          14.6      15.3                33.0  HBEF
##  3  2003 Reference Low          12.5       9.73               25.3  HBEF
##  4  2003 Reference Low           9.97     10.1                23.2  HBEF
##  5  2003 Reference Low           6.84      5.48               15.5  HBEF
##  6  2003 Reference Low           9.66      7.64               20.4  HBEF
##  7  2003 Reference Low           8.82      9.23               21.2  HBEF
##  8  2003 Reference Low           5.83      6.18               15.2  HBEF
##  9  2003 Reference Low           8.11      7.13               18.4  HBEF
## 10  2003 Reference Low           3.02      3.44                9.60 HBEF
## # ... with 349 more rows
## # i Use ‘print(n = ...)‘ to see more rows
```

And we see that our new column has been added onto the far right of the dataframe.

Mutating Columns as a Function of Existing Columns

A more common use case is if we want to make a new column from existing columns. Two typical uses here would be:

1. Making a new column that is made via a function operating on another column(s)
2. Making a factor column that binds together values from other columns

1 - New Column Via Numeric Function or Logical Test

Rowwise Function In these data, there are two measures of leaf area: `leaf1area` and `leaf2area`. While they're both very similar, it might still be useful to have a mean estimate for each observation in the data between the two leaf area measurements.

Say we wanted to calculate for row, the mean value of the two measures of leaf area. **To make this operation on a row-wise basis, we can use `dplyr::rowwise()`** which will ensure each calculation is done on a single row. Combining this with `mutate()` will produce our desired result.

```
df = df %>%
  dplyr::rowwise() %>%  # ensures calculation done for each row
  dplyr::mutate(
    mean_area = mean(c(leaf1area, leaf2area), na.rm = TRUE)
  )
df
```

```
## # A tibble: 359 x 8
## # Rowwise:
##     year watershed elevation leaf1area leaf2area corrected_leaf~1 forest mean_~2
##    <dbl> <fct>     <fct>         <dbl>     <dbl>            <dbl> <chr>    <dbl>
##  1  2003 Reference Low           13.8      12.1             29.1  HBEF     13.0
##  2  2003 Reference Low           14.6      15.3             33.0  HBEF     14.9
##  3  2003 Reference Low           12.5       9.73            25.3  HBEF     11.1
##  4  2003 Reference Low            9.97     10.1             23.2  HBEF     10.0
##  5  2003 Reference Low            6.84      5.48            15.5  HBEF      6.16
##  6  2003 Reference Low            9.66      7.64            20.4  HBEF      8.65
##  7  2003 Reference Low            8.82      9.23            21.2  HBEF      9.03
##  8  2003 Reference Low            5.83      6.18            15.2  HBEF      6.01
##  9  2003 Reference Low            8.11      7.13            18.4  HBEF      7.62
## 10  2003 Reference Low            3.02      3.44             9.60 HBEF      3.23
## # ... with 349 more rows, and abbreviated variable names
## #   1: corrected_leaf_area, 2: mean_area
## # i Use `print(n = ...)` to see more rows
```

Now, again, we have our new column tacked on at the end of our dataframe just as we wanted.

Since we have now a mean estimate of leaf area, we might want to compare that to our measure of `corrected_leaf_area`. We might want, for instance, to see if the `corrected_leaf_area` is a consistent proportion larger than the mean of the two leaf area measurements.

To go about this, we might want to get a ratio of the `mean_area` and the `corected_leaf_area` for each row. To do so, we can create an operation very similarly to above:

```
df = df %>%
  dplyr::rowwise() %>%  # we still need this to get a value for each row
  dplyr::mutate(
    corrected_mean_ratio = mean_area / corrected_leaf_area
  )
df
```

```
## # A tibble: 359 x 9
## # Rowwise:
##     year watershed elevation leaf1area leaf2area correc~1 forest mean_~2 corre~3
##    <dbl> <fct>     <fct>         <dbl>     <dbl>    <dbl> <chr>    <dbl>   <dbl>
##  1  2003 Reference Low           13.8      12.1      29.1 HBEF     13.0    0.446
##  2  2003 Reference Low           14.6      15.3      33.0 HBEF     14.9    0.452
##  3  2003 Reference Low           12.5       9.73     25.3 HBEF     11.1    0.438
##  4  2003 Reference Low            9.97     10.1      23.2 HBEF     10.0    0.432
##  5  2003 Reference Low            6.84      5.48     15.5 HBEF      6.16   0.399
##  6  2003 Reference Low            9.66      7.64     20.4 HBEF      8.65   0.423
##  7  2003 Reference Low            8.82      9.23     21.2 HBEF      9.03   0.426
##  8  2003 Reference Low            5.83      6.18     15.2 HBEF      6.01   0.396
##  9  2003 Reference Low            8.11      7.13     18.4 HBEF      7.62   0.415
## 10  2003 Reference Low            3.02      3.44     9.60 HBEF      3.23   0.337
## # ... with 349 more rows, and abbreviated variable names
## #   1: corrected_leaf_area, 2: mean_area, 3: corrected_mean_ratio
## # i Use `print(n = ...)` to see more rows
```

ifelse() A handy component of `mutate()` is that we can create a new column based on a conditional statement like an `ifelse()` statement (LINK TO PROGRAMMING CONCEPT OF IFELSE). In the `corrected_leaf_area` column, we may want to know if a particular value falls above or below the mean value for all of our observations. We could create a simple categorical variable based on this for each row with an `ifelse()` statement.

```
# overall mean
mean_val = mean(df$corrected_leaf_area, na.rm = TRUE)

# make new column
df = df %>%
  # still need to rowwise the calculation
  dplyr::rowwise() %>%
  dplyr::mutate(
    # we specify we want this as a factor
    corrected_above_below = as.factor(ifelse(
      # this is the logical test
      corrected_leaf_area > mean_val,
      # value if test is TRUE
      "Above",
      # value if test is FALSE
      "Below"
    ))
  )
df
```

```
## # A tibble: 359 x 10
## # Rowwise: 
##     year waters~1 eleva~2 leaf1~3 leaf2~4 corre~5 forest mean_~6 corre~7 corre~8
##    <dbl> <fct>    <fct>     <dbl>   <dbl>   <dbl> <chr>    <dbl>   <dbl> <fct>  
##  1  2003 Referen~ Low       13.8    12.1    29.1  HBEF     13.0    0.446 Above  
##  2  2003 Referen~ Low       14.6    15.3    33.0  HBEF     14.9    0.452 Above  
##  3  2003 Referen~ Low       12.5     9.73   25.3  HBEF     11.1    0.438 Below  
##  4  2003 Referen~ Low        9.97   10.1    23.2  HBEF     10.0    0.432 Below  
##  5  2003 Referen~ Low        6.84    5.48   15.5  HBEF      6.16   0.399 Below  
##  6  2003 Referen~ Low        9.66    7.64   20.4  HBEF      8.65   0.423 Below  
##  7  2003 Referen~ Low        8.82    9.23   21.2  HBEF      9.03   0.426 Below  
##  8  2003 Referen~ Low        5.83    6.18   15.2  HBEF      6.01   0.396 Below  
##  9  2003 Referen~ Low        8.11    7.13   18.4  HBEF      7.62   0.415 Below  
## 10  2003 Referen~ Low        3.02    3.44    9.60 HBEF      3.23   0.337 Below  
## # ... with 349 more rows, and abbreviated variable names 1: watershed,
## #   2: elevation, 3: leaf1area, 4: leaf2area, 5: corrected_leaf_area,
## #   6: mean_area, 7: corrected_mean_ratio, 8: corrected_above_below
## # i Use `print(n = ...)` to see more rows
```

2 - Binding Columns

While this is a less common use case, it is often handy to make a new column grouping variable that is informative and can be made for each row. For example, say we wanted to have some way to group our observations by three components: the year the data were collected (2003 vs. 2004), the watershed (reference vs. w1), and elevation (low or mid). In this toy example, we'll do this by pasting those elements together in a new row-wise column.

First, let's take a look at the summaries of each of these columns:

```
summary(df$year)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2003    2003    2003    2003    2004    2004
```

```
summary(df$watershed)
```

```
## Reference        W1
##       179       180
```

```
summary(df$elevation)
```

```
##  Low  Mid NA's
##  120  120  119
```

Importantly, it looks like there are some NA's in the elevation column. For our purposes, we'll want to keep those around as there are many observations that have NA's and we don't want to throw those all away.

Now, let's perform our calculation:

```
df = df %>%
  dplyr::rowwise() %>%
  dplyr::mutate(
    # we can use paste() from base R to paste the characters together
    group = as.factor(paste(year,
                   watershed,
                   elevation,
                  sep = "_")) # we want each component separated by a "_"
  )
df
```

```
## # A tibble: 359 x 11
## # Rowwise:
##     year waters~1 eleva~2 leaf1~3 leaf2~4 corre~5 forest mean_~6 corre~7 corre~8
##    <dbl> <fct>    <fct>     <dbl>   <dbl>   <dbl> <chr>    <dbl>   <dbl> <fct>
##  1  2003 Referen~ Low       13.8    12.1     29.1 HBEF     13.0    0.446 Above
##  2  2003 Referen~ Low       14.6    15.3     33.0 HBEF     14.9    0.452 Above
##  3  2003 Referen~ Low       12.5     9.73    25.3 HBEF     11.1    0.438 Below
##  4  2003 Referen~ Low        9.97   10.1     23.2 HBEF     10.0    0.432 Below
##  5  2003 Referen~ Low        6.84    5.48    15.5 HBEF      6.16   0.399 Below
##  6  2003 Referen~ Low        9.66    7.64    20.4 HBEF      8.65   0.423 Below
##  7  2003 Referen~ Low        8.82    9.23    21.2 HBEF      9.03   0.426 Below
##  8  2003 Referen~ Low        5.83    6.18   15.2  HBEF      6.01   0.396 Below
##  9  2003 Referen~ Low        8.11    7.13   18.4  HBEF      7.62   0.415 Below
## 10  2003 Referen~ Low        3.02    3.44    9.60 HBEF      3.23   0.337 Below
## # ... with 349 more rows, 1 more variable: group <fct>, and abbreviated
## #   variable names 1: watershed, 2: elevation, 3: leaf1area, 4: leaf2area,
## #   5: corrected_leaf_area, 6: mean_area, 7: corrected_mean_ratio,
## #   8: corrected_above_below
## # i Use ‘print(n = ...)‘ to see more rows, and ‘colnames()‘ to see all variable names
```

Now we can see how many unique combinations of these three variables there are:

```
table(df$group)
```

```
##
## 2003_Reference_Low 2003_Reference_Mid        2003_W1_Low        2003_W1_Mid
##                 60                 60                 60                 60
##  2004_Reference_NA         2004_W1_NA
##                 59                 60
```

We see that the observations are spread almost perfectly across the six different groups.