…Data can be and usually is messy in all kinds of ways. One of the most common, particularly in the case of summary tables obtained from some source or other, is that the values aren't directly usable. The following summary table was copied and pasted into Excel from an external source, saved as a CSV file, and arrived looking like this:

```
library(tidyverse)

rfm_tbl <- read_csv("data/rfm_table.csv")
```

```
## Parsed with column specification:
## cols(
##   SEGMENT = col_character(),
##   DESCRIPTION = col_character(),
##   R = col_character(),
##   F = col_character(),
##   M = col_character()
## )
```

```
rfm_tbl
```

```
## # A tibble: 23 x 5
##    SEGMENT        DESCRIPTION                              R     F     M
##
##  1
##  2 Champions      Bought recently, buy often and spend t… 4- 5  4- 5  4- 5
##  3
##  4 Loyal Custome… Spend good money. Responsive to promot… 2- 5  3- 5  3- 5
##  5
##  6 Potential Loy… Recent customers, spent good amount, b… 3- 5  1- 3  1- 3
##  7
##  8 New Customers  Bought more recently, but not often     4- 5  <= 1  <= 1
##  9
## 10 Promising      Recent shoppers, but haven't spent much 3- 4  <= 1  <= 1
## # … with 13 more rows
```

This is messy and we can't do anything with the values in `R`, `F`, and `M`. Ultimately we want a table with separate columns containing the low and high values for these variables. If no lower bound is shown, the lower bound is zero. We're going to use a few tools, notably `separate()` to get where we want to be. I'll step through this pipeline one piece at a time, so you can see how the table is being changed from start to finish.

First let's clean clean the variable names and remove the entirely blank lines.

```
rfm_tbl %>%
  janitor::clean_names() %>%
  filter_all(any_vars(!is.na(.)))
```

```
## # A tibble: 11 x 5
##    segment        description                              r     f     m
##
```

```
##  1 Champions      Bought recently, buy often and spend t… 4- 5  4- 5  4- 5
##  2 Loyal Custome… Spend good money. Responsive to promot… 2- 5  3- 5  3- 5
##  3 Potential Loy… Recent customers, spent good amount, b… 3- 5  1- 3  1- 3
##  4 New Customers  Bought more recently, but not often     4- 5  <= 1  <= 1
##  5 Promising      Recent shoppers, but haven't spent much 3- 4  <= 1  <= 1
##  6 Need Attention Above average recency, frequency & mon… 2- 3  2- 3  2- 3
##  7 About To Sleep Below average recency, frequency & mon… 2- 3  <= 2  <= 2
##  8 At Risk        Spent big money, purchased often but l… <= 2  2- 5  2- 5
##  9 Can't Lose Th… Made big purchases and often, but long… <= 1  4- 5  4- 5
## 10 Hibernating    Low spenders, low frequency, purchased… 1- 2  1- 2  1- 2
## 11 Lost           Lowest recency, frequency & monetary s… <= 2  <= 2  <= 2
```

Next we start work on the values. I thought about different ways of doing this, notably working out a way to apply or map `separate()` to each of the columns I want to change. I got slightly bogged down doing this, and instead decided to lengthen the `r`, `f`, and `m` variables into a single key-value pair, do the recoding there, and then widen the result again. First, lengthen the data:

```
rfm_tbl %>%
  janitor::clean_names() %>%
  filter_all(any_vars(!is.na(.))) %>%
  pivot_longer(cols = r:m)
```

```
## # A tibble: 33 x 4
##    segment         description                                    name  value
##
##  1 Champions       Bought recently, buy often and spend the mo… r     4- 5
##  2 Champions       Bought recently, buy often and spend the mo… f     4- 5
##  3 Champions       Bought recently, buy often and spend the mo… m     4- 5
##  4 Loyal Customers Spend good money. Responsive to promotions   r     2- 5
##  5 Loyal Customers Spend good money. Responsive to promotions   f     3- 5
##  6 Loyal Customers Spend good money. Responsive to promotions   m     3- 5
##  7 Potential Loya… Recent customers, spent good amount, bought… r     3- 5
##  8 Potential Loya… Recent customers, spent good amount, bought… f     1- 3
##  9 Potential Loya… Recent customers, spent good amount, bought… m     1- 3
## 10 New Customers   Bought more recently, but not often          r     4- 5
## # … with 23 more rows
```

I'm quite sure that there's an elegant way to use one of the `map()` functions to process the `r`, `f`, and `m` columns in sequence. But seeing as I couldn't quickly figure it out, this alternative strategy works just fine. In fact, as a general approach I think it's always worth remembering that the tidyverse really "wants" your data to be in long form, and lots of things that are awkward or conceptually tricky can suddenly become *much* easier if you get the data into the shape that the function toolbox wants it to be in. Lengthening the data you're working with is very often the right approach, and you know you can widen it later on once you're done cleaning or otherwise manipulating it.

With our table in long format we can now use `separate()` on the value column. The `separate()` function is very handy for pulling apart variables that should be in different columns. Its defaults are good, too. In this case I didn't have to write a regular expression to specify the characters that are dividing up the values. In the function call we use `convert = TRUE` to turn the results into integers, and `fill = "left"` because there's an implicit zero on the left of each entry that looks like e.g. `<= 2`.

```
rfm_tbl %>%
  janitor::clean_names() %>%
  filter_all(any_vars(!is.na(.))) %>%
  pivot_longer(cols = r:m) %>%
```

```
  separate(col = value, into = c("lo", "hi"),
           remove = FALSE, convert = TRUE,
           fill = "left")
```

```
## # A tibble: 33 x 6
##    segment       description                          name  value    lo    hi
##
##  1 Champions     Bought recently, buy often and sp… r     4- 5       4     5
##  2 Champions     Bought recently, buy often and sp… f     4- 5       4     5
##  3 Champions     Bought recently, buy often and sp… m     4- 5       4     5
##  4 Loyal Custom… Spend good money. Responsive to p… r     2- 5       2     5
##  5 Loyal Custom… Spend good money. Responsive to p… f     3- 5       3     5
##  6 Loyal Custom… Spend good money. Responsive to p… m     3- 5       3     5
##  7 Potential Lo… Recent customers, spent good amou… r     3- 5       3     5
##  8 Potential Lo… Recent customers, spent good amou… f     1- 3       1     3
##  9 Potential Lo… Recent customers, spent good amou… m     1- 3       1     3
## 10 New Customers Bought more recently, but not oft… r     4- 5       4     5
## # … with 23 more rows
```

Before widening the data again we drop the `value` column. We don't need it anymore. (It will mess up the widening if we keep it, too: try it and see what happens.)

```
rfm_tbl %>%
  janitor::clean_names() %>%
  filter_all(any_vars(!is.na(.))) %>%
  pivot_longer(cols = r:m) %>%
  separate(col = value, into = c("lo", "hi"),
           remove = FALSE, convert = TRUE,
           fill = "left") %>%
  select(-value)
```

```
## # A tibble: 33 x 5
##    segment        description                                  name     lo    hi
##
##  1 Champions      Bought recently, buy often and spend t… r         4     5
##  2 Champions      Bought recently, buy often and spend t… f         4     5
##  3 Champions      Bought recently, buy often and spend t… m         4     5
##  4 Loyal Custome… Spend good money. Responsive to promot… r         2     5
##  5 Loyal Custome… Spend good money. Responsive to promot… f         3     5
##  6 Loyal Custome… Spend good money. Responsive to promot… m         3     5
##  7 Potential Loy… Recent customers, spent good amount, b… r         3     5
##  8 Potential Loy… Recent customers, spent good amount, b… f         1     3
##  9 Potential Loy… Recent customers, spent good amount, b… m         1     3
## 10 New Customers  Bought more recently, but not often     r         4     5
## # … with 23 more rows
```

Now we can widen the data, with `pivot_wider()`.

```
rfm_tbl %>%
  janitor::clean_names() %>%
  filter_all(any_vars(!is.na(.))) %>%
  pivot_longer(cols = r:m) %>%
  separate(col = value, into = c("lo", "hi"),
           remove = FALSE, convert = TRUE,
```

```
          fill = "left") %>%
  select(-value) %>%
  pivot_wider(names_from = name,
              values_from = lo:hi)
```

```
## # A tibble: 11 x 8
##    segment     description               lo_r  lo_f  lo_m  hi_r  hi_f  hi_m
##
##  1 Champions   Bought recently, buy of…     4     4     4     5     5     5
##  2 Loyal Cust… Spend good money. Respo…     2     3     3     5     5     5
##  3 Potential … Recent customers, spent…     3     1     1     5     3     3
##  4 New Custom… Bought more recently, b…     4    NA    NA     5     1     1
##  5 Promising   Recent shoppers, but ha…     3    NA    NA     4     1     1
##  6 Need Atten… Above average recency, …     2     2     2     3     3     3
##  7 About To S… Below average recency, …     2    NA    NA     3     2     2
##  8 At Risk     Spent big money, purcha…    NA     2     2     2     5     5
##  9 Can't Lose… Made big purchases and …    NA     4     4     1     5     5
## 10 Hibernating Low spenders, low frequ…     1     1     1     2     2     2
## 11 Lost        Lowest recency, frequen…    NA    NA    NA     2     2     2
```

Finally we put back those implicit zeros using `replace_na()` and reorder the columns to our liking. Using `replace_na()` is fine here because we know that every missing value should in fact be a zero.

```
rfm_tbl %>%
  janitor::clean_names() %>%
  filter_all(any_vars(!is.na(.))) %>%
  pivot_longer(cols = r:m) %>%
  separate(col = value, into = c("lo", "hi"),
           remove = FALSE, convert = TRUE,
           fill = "left") %>%
  select(-value) %>%
  pivot_wider(names_from = name,
              values_from = lo:hi) %>%
  mutate_if(is.integer, replace_na, 0) %>%
  select(segment,
         lo_r, hi_r,
         lo_f, hi_f,
         lo_m, hi_m,
         description)
```

```
## # A tibble: 11 x 8
##    segment      lo_r  hi_r  lo_f  hi_f  lo_m  hi_m description
##
##  1 Champions       4     5     4     5     4     5 Bought recently, buy of…
##  2 Loyal Cust…     2     5     3     5     3     5 Spend good money. Respo…
##  3 Potential …     3     5     1     3     1     3 Recent customers, spent…
##  4 New Custom…     4     5     0     1     0     1 Bought more recently, b…
##  5 Promising       3     4     0     1     0     1 Recent shoppers, but ha…
##  6 Need Atten…     2     3     2     3     2     3 Above average recency, …
##  7 About To S…     2     3     0     2     0     2 Below average recency, …
##  8 At Risk         0     2     2     5     2     5 Spent big money, purcha…
##  9 Can't Lose…     0     1     4     5     4     5 Made big purchases and …
## 10 Hibernating     1     2     1     2     1     2 Low spenders, low frequ…
## 11 Lost            0     2     0     2     0     2 Lowest recency, frequen…
```

Much nicer.