Today, we’re going to introduce and animate the *separate()* function, which converts data with a single column containing multiple variables into a tidy, one-column-per-variable format.

First, let’s create and view some sample data, where one column is actually the concatenation of three variables: two treatments, and one value.

library(tidyverse)

library(gganimate)

sample\_data <- tibble(

ID = 1:10,

`TRT1\_TRT2\_Value` = paste(sample(LETTERS[1:3], 10, replace = TRUE),

sample(LETTERS[1:3], 10, replace = TRUE),

round(rnorm(10)), sep = "\_")

)

sample\_data

# A tibble: 10 x 2

ID TRT1\_TRT2\_Value

1 1 B\_A\_1

2 2 B\_A\_-4

3 3 B\_B\_2

4 4 C\_B\_0

5 5 A\_C\_2

6 6 A\_B\_0

7 7 A\_C\_1

8 8 B\_A\_-1

9 9 B\_B\_0

10 10 A\_A\_0

Next, we will use the *separate()* function and specify the *into* parameter, as well as the *sep* character (an underscore, in this case):

sample\_data\_separated <- sample\_data %>%

separate("TRT1\_TRT2\_Value", into = c("Treatment 1", "Treatment 2", "Value"), sep = "\_")

sample\_data\_separated

# A tibble: 10 x 4

ID `Treatment 1` `Treatment 2` Value

1 1 B A 1

2 2 B A -4

3 3 B B 2

4 4 C B 0

5 5 A C 2

6 6 A B 0

7 7 A C 1

8 8 B A -1

9 9 B B 0

10 10 A A 0

Next, we perform a similar routine to the previous blogs and combine the two datasets into one dataset which will be used to build the animation:

longDat <- function(x) {

names(x) %>%

rbind(x) %>%

setNames(seq\_len(ncol(x))) %>%

mutate(row = row\_number()) %>%

tidyr::gather(column, value, -row) %>%

mutate(column = as.integer(column)) %>%

ungroup() %>%

arrange(column, row)

}

long\_tables <- map(list(sample\_data, sample\_data\_separated), longDat)

combined\_table <- long\_tables[[1]] %>%

mutate(tstep = "a")

separated\_table <- long\_tables[[2]] %>%

mutate(tstep = "b")

both\_tables <- bind\_rows(combined\_table, separated\_table)

both\_tables$celltype[both\_tables$column == 1] <- c("header", rep("id", 10), "header2", rep("id", 10))

both\_tables$celltype[both\_tables$column == 2] <- c("header", rep("value\_treatment", 10), "header2", rep("treatment", 10))

both\_tables$celltype[both\_tables$column == 3] <- c("header2", rep("treatment", 10))

both\_tables$celltype[both\_tables$column == 4] <- c("header2", rep("value", 10))

both\_tables

# A tibble: 66 x 5

row column value tstep celltype

1 1 1 ID a header

2 2 1 1 a id

3 3 1 2 a id

4 4 1 3 a id

5 5 1 4 a id

6 6 1 5 a id

7 7 1 6 a id

8 8 1 7 a id

9 9 1 8 a id

10 10 1 9 a id

# … with 56 more rows

From this, we can produce static versions of the two images which will form the basis for the animation:

base\_plot <- ggplot(both\_tables, aes(column, -row, fill = celltype)) +

geom\_tile(color = "black") +

geom\_text(aes(label = value), size = 6, fontface = "bold") +

theme\_void() +

scale\_fill\_manual(values = c("grey85", "grey85", "#ffebcc", "#d6e5ff", "#ffd6d7", "#f2d6ff"),

name = "",

labels = c("Header", "", "ID", "Treatment", "Value", "Value\_Treatment"),

breaks = c("header", "", "id", "treatment", "value", "value\_treatment")) +

theme(

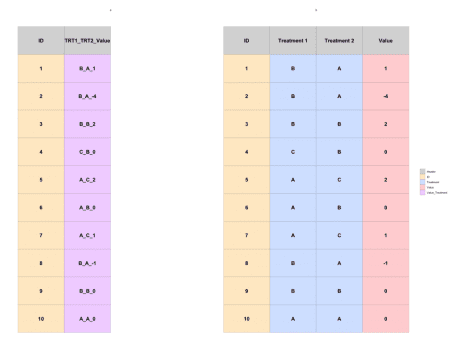
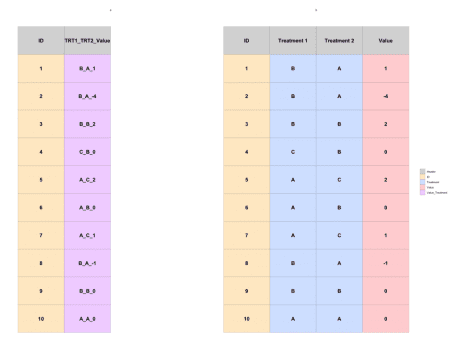
plot.margin = unit(c(1, 1, 1, 1), "cm")

)

p0 <- base\_plot +

facet\_wrap(~tstep)

p0



Finally, we use gganimate to build the final animation!

p1 <- base\_plot +

transition\_states(

states = tstep,

transition\_length = 1,

state\_length = 1

) +

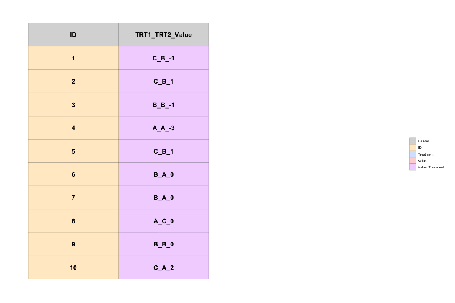
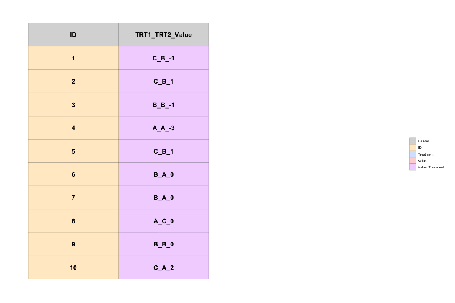
enter\_fade() +

exit\_fade() +

ease\_aes('sine-in-out')

p1\_animate <- animate(p1, height = 800, width = 1200, fps = 20, duration = 10)

anim\_save("separate\_animate.gif")



We hope you’ve enjoyed this third installment in our animating data transformations series! Stay tuned for more!