# Tidyr

#### Recap

- tidyverse is an *opinionated* collection of pacakges
- All packages within it's ecosystem use the same syntax:
  - %>% pipe operators at the end of the line read as "and then"

"I took my original data.frame %>%

I kept only 5 out of the original 20 columns %>%

I added a new column that was based on the 2nd column %>%

I grouped the data based on a categorical column %>%

I got descriptive statistics per level of the categorical var"

#### Recap

- tidyverse is an *opinionated* collection of pacakges
- All packages within it's ecosystem use the same syntax:
  - %>% pipe operators at the end of the line read as "and then"

```
originalData %>%
select(1:5) %>%
mutate(newVar = sqrt(var1)) %>%
group_by(factorVar) %>%
summarize(meanVar = mean(var))
```

#### This time

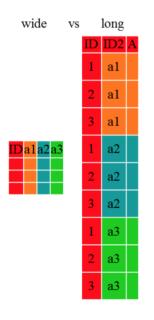
Functions from the tidyr package (but DON'T memorize which functions come from which packages!)

- Go from long to wide format
- Split columns and combine them
- Missing data

## Long vs. Wide data

**Long data** - Each column is a variable and each row is an observation. Each row does NOT need to be a unique participant.

**Wide data** - Each row is a particular participant, and columns can contain multiple observations for the same data.



##		Subject	Time1	Time	2 Time3
##	1	1	0.2	0.	4 0.3
##	2	2	0.8	0.	9 0.7
##	3	3	1.3	1.	0 1.1
##		Subject	TimePo	oint	Score
##	1	1		1	0.2
##	2	2		1	0.8
##	3	3		1	1.3
##	4	1		2	0.4
##	5	2		2	0.9
##	6	3		2	1.0
##	7	1		3	0.3
##	8	2		3	0.7
##	9	3		3	1.1

### Long vs. Wide data

For the most part, you want your data to be in the **long** format

- Especially for plotting in ggplot2!
- (some analyses, like reliability, require the wide format, but most stick with long)

However, we often receive data in the wide format. It is useful to be able to go between the two. tidyr makes this easy with:

- pivot\_wider() to go from long to wide
- pivot\_longer() to go from wide to long

This function takes in long data and makes it wide. Important arguments:

- names\_from = which columns to get the *name* of the output column.
- values\_from = which columns to get the *value* of the output column.

Let's take the example data.frame I showed earlier. Since it's completely arbitrary, I'm going to call it generic

#### generic

```
##
    Subject TimePoint Score
## 1
                        0.2
           1
## 2
                        0.8
## 3
                      1.3
## 4
                      0.4
## 5
                       0.9
                    2 1.0
## 6
                    3 0.3
## 7
                    3 0.7
## 8
## 9
                        1.1
```

This generic data.frame is in the **long** format. To make it into the **wide** format, let's use pivot\_wider()

Sometimes, it's a bit more complicated. Let's add some more variables to generic to test this out.

- hairColor factor with 2 levels (brown & blonde)
- happiness scale of 1 to 10 measured at each time point

```
Subject TimePoint Score hairColor happiness
##
## 1
          1
                   1
                      0.2
                              brown
                                          10
                             blonde
## 2
                     0.8
## 3
                   1 1.3 blonde
                                           6
## 4
                   2 0.4 brown
## 5
                   2 0.9 blonde
## 6
                   2 1.0 blonde
                                           5
                   3 0.3 brown
## 7
                                          10
                       0.7
                             blonde
## 8
                                           3
```

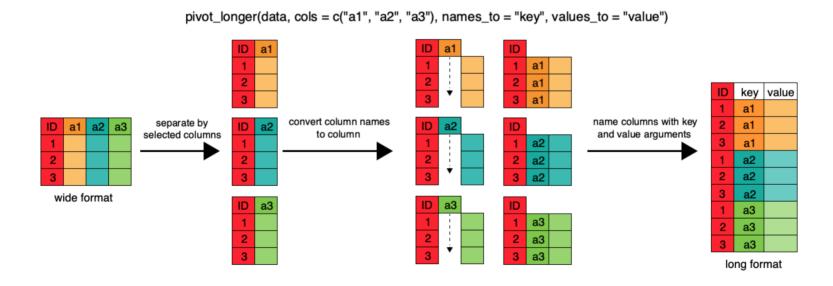
Now, let's say we want each time point's Score and happiness variables in the wide format...

```
## # A tibble: 3 x 8
    Subject hairColor Score_1 Score_2 Score_3 happiness_1 happiness_2 happine
##
                                           <dbl>
     <dbl> <chr>
                    <dbl>
                           <dbl>
                                 <dbl>
                                                     <dbl>
##
                    0.2 0.4
## 1
        1 brown
                                   0.3
                                             10
## 2 2 blonde 0.8 0.9 0.7
## 3 3 blonde
                     1.3 1
                                1.1
```

### pivot\_longer() function

The exact opposite of pivot\_wider() is pivot\_longer. This takes a wide data.frame and makes it into a **long** data.frame. Arguments are now names\_to = and values\_to =. You also need to include a cols = argument to say which columns you want into the longer format.

Before doing this with code, here's a schematic that might be helpful:



#### pivot\_longer() function

Let's keep going with our current example, starting from wideGenericMore

```
## # A tibble: 18 x 4
     Subject hairColor valueType allScores
##
##
       <dbl> <chr>
                      <chr>
                                     <fdb>>
## 1
           1 brown
                      Score 1
                                       0.2
                      Score_2
## 2
                                       0.4
           1 brown
                                       0.3
##
           1 brown
                      Score 3
## 4
                      happiness_1
           1 brown
                                      10
                      happiness_2
##
           1 brown
                                       9
                      happiness_3
##
           1 brown
                                      10
           2 blonde
                      Score 1
##
                                       0.8
##
           2 blonde
                      Score 2
                                       0.9
           2 blonde
                      Score 3
##
                                       0.7
```

#### pivot\_longer() function

For both of these pivot functions, you can use the – (minus) sign to say "everything except this column". For example:

```
## # A tibble: 18 x 4
     Subject hairColor valueType allScores
##
       <dbl> <chr>
                       <chr>
                                       <dbl>
##
## 1
           1 brown
                       Score 1
                                         0.2
           1 brown
                       Score_2
                                         0.4
## 2
                       Score 3
                                         0.3
##
           1 brown
           1 brown
                       happiness_1
##
                                        10
                       happiness_2
##
           1 brown
                                         9
##
           1 brown
                       happiness_3
                                        10
           2 blonde
                       Score_1
##
                                         0.8
           2 blonde
                       Score 2
##
                                         0.9
           2 blonde
##
   9
                       Score 3
                                         0.7
```

#### The pivot functions

#### Some things to notice:

- In pivot\_longer, the arguments take in strings (aka, need quotations!). That's because you need to tell R what to name something.
- In pivot\_wider, the arguments take in variable names that already exist. So you do not need to wrap those in quotation marks.
- These are the types of functions that I mess up ALL. THE. TIME. Use your History tab!

#### separate() function

In our latest iteration, longGeneric, we have a column called valueType where it is a name, then an underscore (\_), and a number, ex: Score\_1.

We can use separate() to make split valueType into 2 separate columns...1 for the Score and another for the 1.

```
longGeneric %>%
  separate(col = valueType,
        into = c("variableName", "timePoint"))
```

```
## # A tibble: 18 x 5
      Subject hairColor variableName timePoint allScores
##
        <dbl> <chr>
                                      <chr>
                         <chr>
                                                     <dbl>
##
##
            1 brown
                         Score
                                                       0.2
##
            1 brown
                         Score
                                                       0.4
                         Score
##
            1 brown
                                                       0.3
                         happiness
##
            1 brown
                                                      10
                         happiness
##
            1 brown
                                                       9
##
            1 brown
                         happiness
                                       3
                                                      10
            2 blonde
                         Score
                                                       0.8
##
                                       1
            2 blonde
##
                         Score
                                                       0.9
```

#### separate() function

Note that I did not specify that I wanted to separate based on the underscore.

- When it is simple like this, R can automatically detect it.
- But if it's a bit trickier, you can specify how to separate in the sep = argument.
  - For example, sep = ": " if you want to separate on a colon + space.

#### unite() function

The opposite of separate is unite(). For instance, let's say we want to create a variable called bogus that looks something like brown: Score or blonde: happiness. The separator is a colon + space.

```
## # A tibble: 18 x 3
##
    Subject bogus
                             allScores
      <dbl> <chr>
                                <dbl>
##
## 1
          1 brown: Score_1
                                  0.2
## 2
          1 brown: Score 2
                                0.4
## 3
          1 brown: Score_3
                                 0.3
## 4
          1 brown: happiness_1
                                 10
## 5
          1 brown: happiness_2
                                  9
          1 brown: happiness_3
## 6
                                 10
## 7
         2 blonde: Score_1
                                 0.8
          2 blonde: Score 2
##
                               0.9
##
          2 blonde: Score_3
                                  0.7
```

## Missing values in tidyverse

- Like base R and others, many tidyverse functions have an argument for na.rm =.
- You can add a drop\_na() function to your tidyverse chunk. This function is part of tidyr and it will get rid of any rows that contain missing values. It's the equivalent of na.omit()
- Do everything in your power to make sure missing values are treated as NA and *not* something else. Ex:
  - o 999 -- Many measurements can have a value of 999...
  - " " -- Spaces are treated as a character string, not truly missing!
     Remember, the class of your object is based on the least specific object. So if you have a vector of integers, but one missing value that is " ", the class of your vector will be a character! Same thing goes for . (periods).
- If you have something like 999 and you want to replace that with an NA, either of the following will work:
  - o data[data == 999] <- NA (for the entire dataset)</pre>
  - o data\$column[data\$column == 999] <- NA (for a single column)</pre>
  - o data <- gsub(pattern = 999, replacement = NA, x = data)
    (but this will find anything with 999, so be careful!)</pre>