Hacky Hour 20231128 - Pivots with tidyr

Goal of Today: Converting data between LONG and WIDE format

Often, we as humans like to enter our data in wide format. For many statistical tests and visualizations in R, however, our data needs to be in long format. As such we need to perform an operation called a “pivot” in order to reshape our data. There are many ways to do this in R:

1. The `reshape` package (e.g. `melt()`) was popular, but is declining in its use
2. The old `tidyverse` functions of `gather` and `spread`
3. The new `tidyverse` operations `pivot_longer` and `pivot_wider`.

Let’s read in our data and look at it’s format.

```
library(tidyverse)

growth_data <- read_csv("data/growth_data.csv")

growth_data
```

```
# A tibble: 11 x 10
   Time     A1     B1     C1    A2    B2    C2    A3    B3    C3
  <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1     0 0.05   0.045  0.05   0.05  0.06  0.055 0.02  0.01  0.022
2     1 0.0944 0.0857 0.0941 0.178 0.213 0.191 0.022 0.012 0.023
```

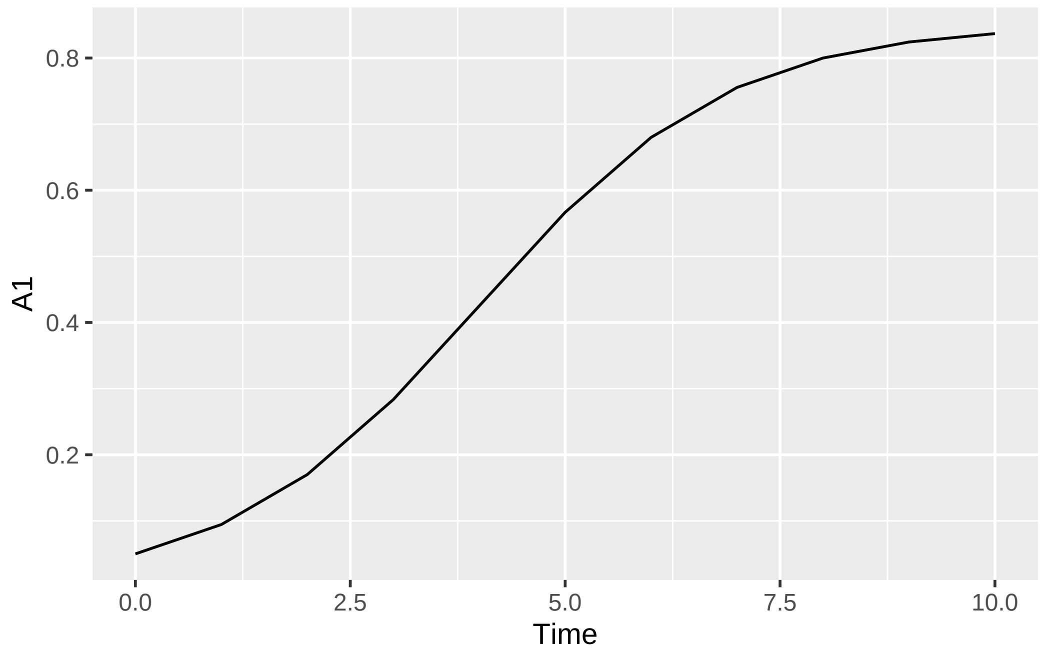
```
 3      2 0.17   0.157  0.168  0.492 0.584 0.503 0.024 0.012 0.025
 4      3 0.283  0.267  0.278  0.883 1.04  0.848 0.022 0.013 0.025
 5      4 0.425  0.411  0.413  1.10  1.29  1.02  0.031 0.014 0.023
 6      5 0.567  0.565  0.545  1.17  1.37  1.08  0.03  0.015 0.022
 7      6 0.68   0.694  0.648  1.19  1.39  1.09  0.033 0.02  0.03
 8      7 0.756  0.784  0.716  1.20  1.40  1.10  0.033 0.022 0.03
 9      8 0.8    0.838  0.756  1.20  1.40  1.10  0.031 0.025 0.029
10      9 0.824  0.868  0.777  1.20  1.40  1.10  0.032 0.025 0.03
11     10 0.837  0.884  0.788  1.20  1.40  1.10  0.029 0.03  0.031
```

This is data from a growth curve I ran in a well plate. I ran it for 10 hours, and each well has it's own column where I recorded the OD. This growth curve included three strains, which I did in triplicate. Below is where each strain was in the well plate:

	1	2	3
A	WT	gene1	gene2
B	WT	gene1	gene2
C	WT	gene1	gene2

What happens if we try to plot this?

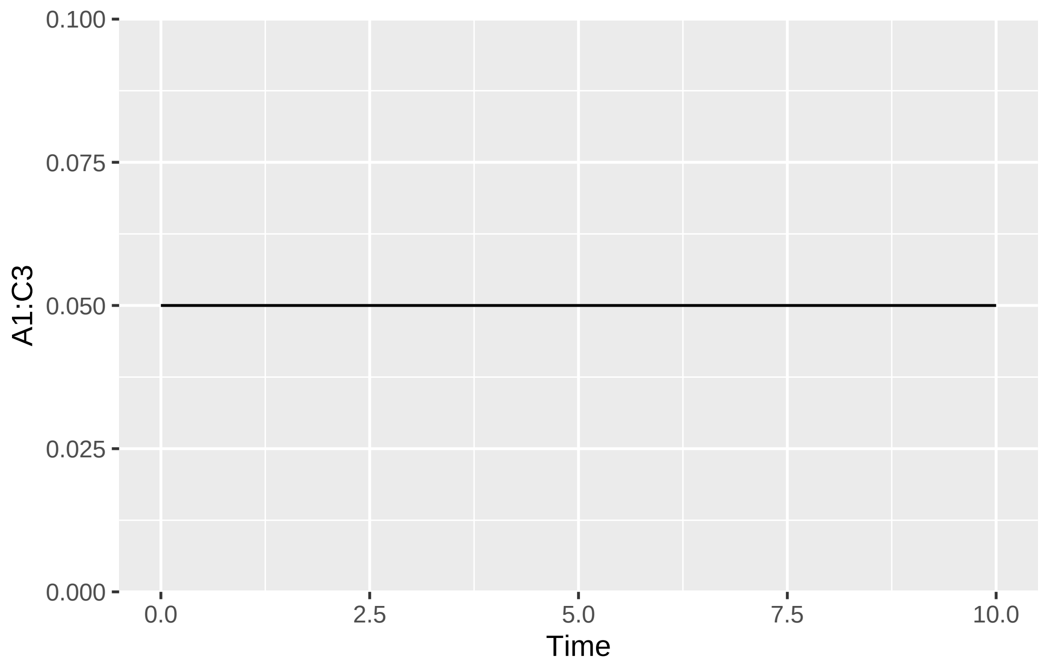
```r
growth_data %>%
  ggplot(aes(x = Time, y = A1)) +
  geom_line()
```



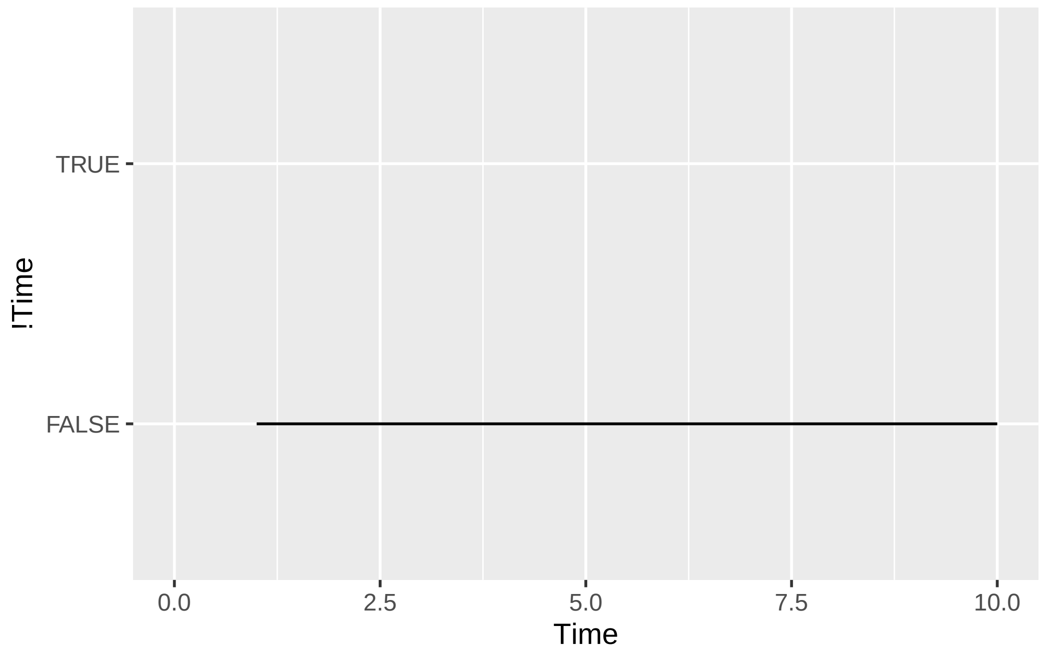
Okay, it's easy to plot one of the strains...

But how do I plot all of them?

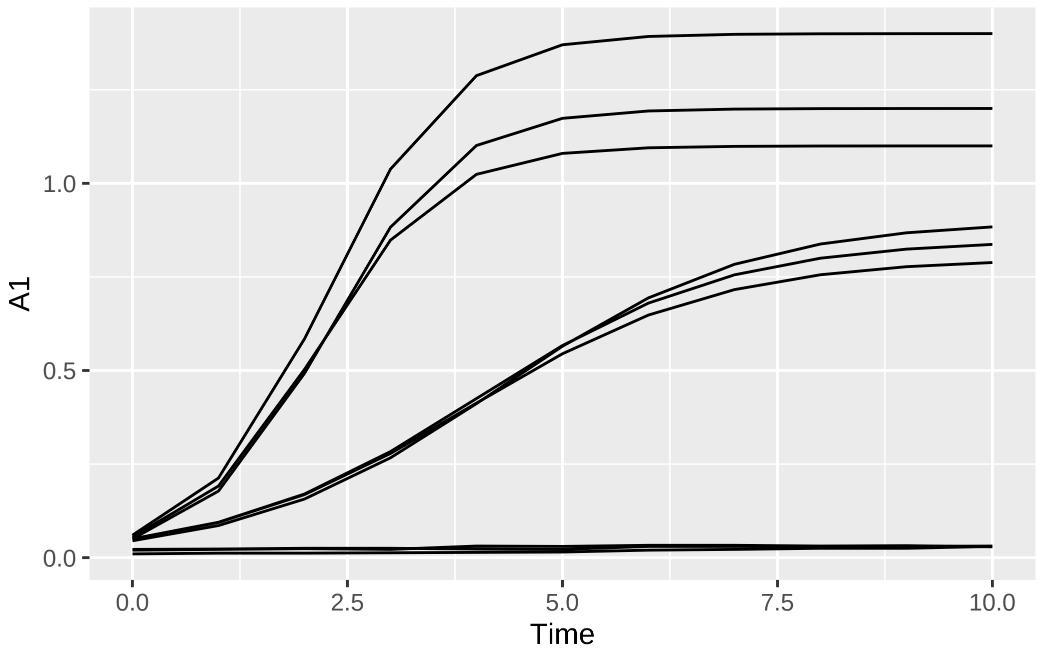
```
# This didn't work
growth_data %>%
  ggplot(aes(x = Time, y = A1:C3)) +
  geom_line()
```



```
# Nope
growth_data %>%
  ggplot(aes(x = Time, y = !Time)) +
  geom_line()
```



```r
# Yes, but wow it's annoying

growth_data %>%
  ggplot(aes(x = Time)) +
  geom_line(aes(y = A1)) +
  geom_line(aes(y = A2)) +
  geom_line(aes(y = A3)) +
  geom_line(aes(y = B1)) +
  geom_line(aes(y = B2)) +
  geom_line(aes(y = B3)) +
  geom_line(aes(y = C1)) +
  geom_line(aes(y = C2)) +
  geom_line(aes(y = C3))
```



Clearly, there needs to be a better way. Ggplot want's it data to be in long format, where we'll have one column with all of the "y's" (which is OD), and another column which separates them into groups (like Well). Let's perform a pivot and then discuss what we did:

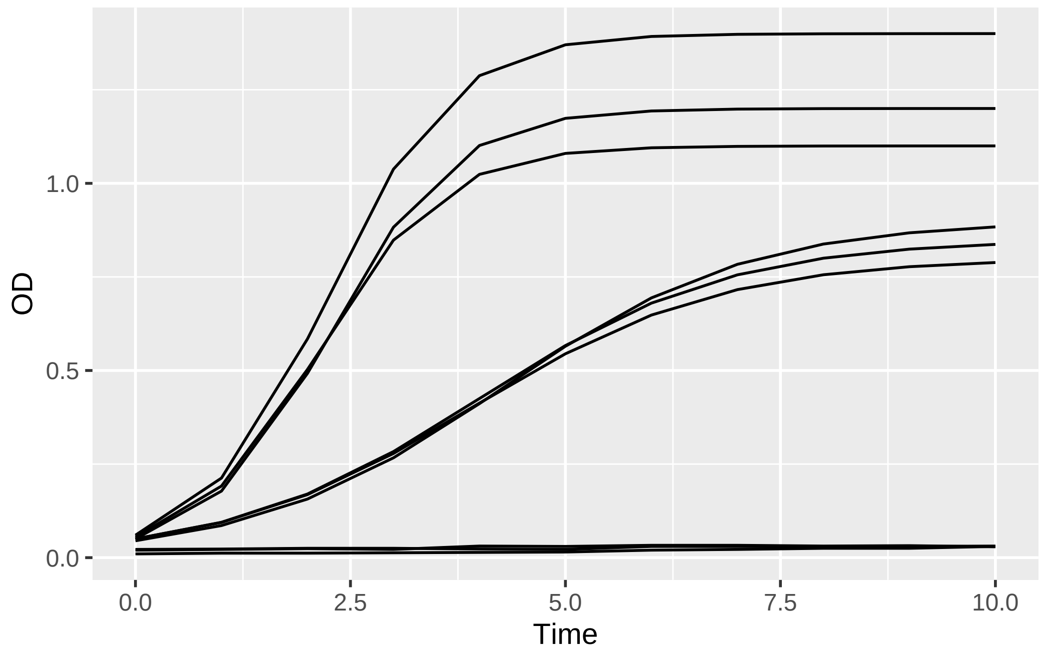
```
long_data <- growth_data %>%
  pivot_longer(cols = c(A1, A2, A3, B1, B2, B3, C1, C2, C3),
               names_to = "Well", values_to = "OD")

# We can do this even easier, by using the `!` symbol to select everything BUT Time

long_data <- growth_data %>%
  pivot_longer(cols = !Time, names_to = "Well", values_to = "OD")

long_data
```

```
# A tibble: 99 x 3
```

```
    Time Well      OD
   <dbl> <chr>  <dbl>
 1     0 A1    0.05
 2     0 B1    0.045
 3     0 C1    0.05
 4     0 A2    0.05
 5     0 B2    0.06
 6     0 C2    0.055
 7     0 A3    0.02
 8     0 B3    0.01
 9     0 C3    0.022
10     1 A1    0.0944
# i 89 more rows
```

Wow - our data is much longer! Now let's see how we plot that in ggplot:

```
long_data %>%
  ggplot(aes(x = Time, y = OD, group = Well)) +
  geom_line()
```



Much better, and much easier!

Hmmm, we're interesting in calculating the average of strain at each timepoint, so we need to figure out how to tell R what strain matches which well. There are actually lots of ways to do this; I'll demonstrate three below:

```
# Using case_whens and writing out all the wells

long_data %>%
  mutate(Strain = case_when(Well %in% c("A1", "B1", "C1") ~ "WT",
                            Well %in% c("A2", "B2", "C2") ~ "saeS",
                            Well %in% c("A3", "B3", "C3") ~ "lpdA"))
```

```
# A tibble: 99 x 4
    Time Well      OD Strain
   <dbl> <chr>  <dbl> <chr>
 1     0 A1    0.05   WT
 2     0 B1    0.045  WT
 3     0 C1    0.05   WT
 4     0 A2    0.05   saeS
 5     0 B2    0.06   saeS
 6     0 C2    0.055  saeS
 7     0 A3    0.02   lpdA
 8     0 B3    0.01   lpdA
 9     0 C3    0.022  lpdA
10     1 A1    0.0944 WT
# i 89 more rows
```

```
# Making a "dictionary" that you then use to join:

well_dict <- data.frame(Well = c("A1","B1","C1","A2","B2","C2","A3","B3","C3"),
                        Strain = rep(c("WT","saeS","lpdA"),3))
well_dict
```

```
  Well Strain
1   A1     WT
2   B1   saeS
3   C1   lpdA
4   A2     WT
5   B2   saeS
6   C2   lpdA
7   A3     WT
8   B3   saeS
9   C3   lpdA
```

```r
long_data %>%
  inner_join(well_dict)
```

```
# A tibble: 99 x 4
    Time Well      OD Strain
   <dbl> <chr> <dbl> <chr>
 1     0 A1    0.05   WT
 2     0 B1    0.045  saeS
 3     0 C1    0.05   lpdA
 4     0 A2    0.05   WT
 5     0 B2    0.06   saeS
 6     0 C2    0.055  lpdA
 7     0 A3    0.02   WT
 8     0 B3    0.01   saeS
 9     0 C3    0.022  lpdA
10     1 A1    0.0944 WT
# i 89 more rows
```

```r
# Looking for patterns (WT starts with A) with str_detect

long_data %>%
  mutate(Strain = case_when(str_detect(Well, "1") ~ "WT",
                            str_detect(Well, "2") ~ "saeS",
                            str_detect(Well, "3") ~ "lpdA"))
```

```
# A tibble: 99 x 4
    Time Well      OD Strain
   <dbl> <chr> <dbl> <chr>
 1     0 A1    0.05   WT
 2      0 B1    0.045  WT
 3      0 C1    0.05   WT
 4      0 A2    0.05   saeS
 5      0 B2    0.06   saeS
 6      0 C2    0.055  saeS
 7      0 A3    0.02   lpdA
 8      0 B3    0.01   lpdA
 9      0 C3    0.022  lpdA
10      1 A1    0.0944 WT
# i 89 more rows
```

This is entirely up to you. If I'm doing many samples/combinations, I will usually use the joining method, and write out my well dictionary in Excel beforehand.

Let's go ahead and use the first optionto assign strains, and then calculate averages for each strain and each timepoint

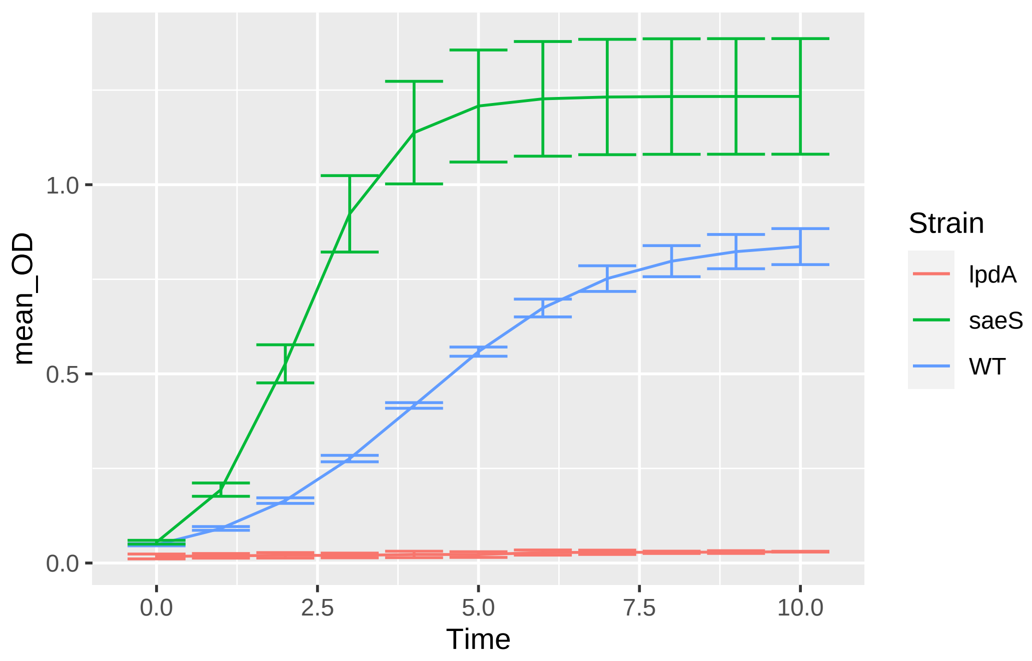
```
strain_data <- long_data %>%
  mutate(Strain = case_when(Well %in% c("A1", "B1", "C1") ~ "WT",
                            Well %in% c("A2", "B2", "C2") ~ "saeS",
                            Well %in% c("A3", "B3", "C3") ~ "lpdA"))

summarized_data <- strain_data %>%
 group_by(Time, Strain) %>%
  summarize(mean_OD = mean(OD),
            sd_OD = sd(OD))

summarized_data
```

```
# A tibble: 33 x 4
# Groups:   Time [11]
```

```
    Time Strain mean_OD   sd_OD
   <dbl> <chr>    <dbl>   <dbl>
 1     0 WT      0.0483 0.00289
 2     0 lpdA    0.0173 0.00643
 3     0 saeS    0.055  0.0050
 4     1 WT      0.0914 0.00495
 5     1 lpdA    0.019  0.00608
 6     1 saeS    0.194  0.0176
 7     2 WT      0.165  0.00737
 8     2 lpdA    0.0203 0.00723
 9     2 saeS    0.527  0.0504
10     3 WT      0.276  0.00854
# i 23 more rows
```

Great! Now we can plot:

```
summarized_data %>%
  ggplot(aes(x = Time, y = mean_OD, group = Strain, color = Strain)) +
  geom_line() +
  geom_errorbar(aes(ymin = mean_OD - sd_OD, ymax = mean_OD + sd_OD))
```



Woohoo! Finally, maybe your collaborator wants you to send them the data, but in wide format so they can plot it in excel. We can convert our data back by using the `pivot_wider` function.

```
strain_data %>%
  pivot_wider(names_from = Strain, values_from = OD)
```

```
# A tibble: 99 x 5
    Time Well      WT   saeS   lpdA
   <dbl> <chr>  <dbl>  <dbl>  <dbl>
 1     0 A1     0.05   NA     NA
 2     0 B1     0.045  NA     NA
 3     0 C1     0.05   NA     NA
 4     0 A2    NA       0.05  NA
 5     0 B2    NA       0.06  NA
 6     0 C2    NA       0.055 NA
 7      0 A3     NA       NA       0.02
 8      0 B3     NA       NA       0.01
 9      0 C3     NA       NA       0.022
10      1 A1      0.0944 NA      NA
# i 89 more rows
```

Hm, that doesn't look right. What's going on?

It looks like R still has a row for each well at each timepoint, but made new columns for each strain, and so we have NAs for strains that don't match a given well. That's not really pretty. Let's try again, removing the well column first.

```
strain_data %>%
  select(-Well) %>%
  pivot_wider(names_from = Strain, values_from = OD)
```

```
# A tibble: 11 x 4
    Time WT        saeS      lpdA
   <dbl> <list>    <list>    <list>
 1     0 <dbl [3]> <dbl [3]> <dbl [3]>
 2     1 <dbl [3]> <dbl [3]> <dbl [3]>
 3     2 <dbl [3]> <dbl [3]> <dbl [3]>
 4     3 <dbl [3]> <dbl [3]> <dbl [3]>
 5     4 <dbl [3]> <dbl [3]> <dbl [3]>
 6     5 <dbl [3]> <dbl [3]> <dbl [3]>
 7     6 <dbl [3]> <dbl [3]> <dbl [3]>
 8     7 <dbl [3]> <dbl [3]> <dbl [3]>
 9     8 <dbl [3]> <dbl [3]> <dbl [3]>
10     9 <dbl [3]> <dbl [3]> <dbl [3]>
11    10 <dbl [3]> <dbl [3]> <dbl [3]>
```

Oooooooh, ok, we got a lot going on. First, we get a warning, saying that we don't have unique identifiers. This makes sense - each strain has three OD measurements at each time-point. Because of this, we get a fancy thing call a list column, where cell of the dataframe is holding all three measurements at once. While we can discuss list columns another day (and they can be very useful!), we don't want to today. Let's use both the Strain name and the well name to identify them:

```
strain_data %>%
  pivot_wider(names_from = c(Strain, Well), values_from = OD)
```

```
# A tibble: 11 x 10
    Time  WT_A1  WT_B1  WT_C1 saeS_A2 saeS_B2 saeS_C2 lpdA_A3 lpdA_B3 lpdA_C3
   <dbl>  <dbl>  <dbl>  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
 1     0 0.05   0.045  0.05     0.05    0.06    0.055   0.02    0.01    0.022
 2     1 0.0944 0.0857 0.0941   0.178   0.213   0.191   0.022   0.012   0.023
 3     2 0.17   0.157  0.168    0.492   0.584   0.503   0.024   0.012   0.025
 4     3 0.283  0.267  0.278    0.883   1.04    0.848   0.022   0.013   0.025
 5     4 0.425  0.411  0.413    1.10    1.29    1.02    0.031   0.014   0.023
 6     5 0.567  0.565  0.545    1.17    1.37    1.08    0.03    0.015   0.022
 7     6 0.68   0.694  0.648    1.19    1.39    1.09    0.033   0.02    0.03
 8     7 0.756  0.784  0.716    1.20    1.40    1.10    0.033   0.022   0.03
 9     8 0.8    0.838  0.756    1.20    1.40    1.10    0.031   0.025   0.029
10     9 0.824  0.868  0.777    1.20    1.40    1.10    0.032   0.025   0.03
11    10 0.837  0.884  0.788    1.20    1.40    1.10    0.029   0.03    0.031
```

R is smart and automatically glued together the strain and well names. We could also do this with our summarized data, where we are getting our values from two separate columns

```
summarized_data %>%
  pivot_wider(names_from = Strain, values_from = c(mean_OD, sd_OD))
```

```
# A tibble: 11 x 7
# Groups:   Time [11]
    Time mean_OD_WT mean_OD_lpdA mean_OD_saeS sd_OD_WT sd_OD_lpdA sd_OD_saeS
   <dbl>      <dbl>        <dbl>        <dbl>    <dbl>      <dbl>      <dbl>
 1     0     0.0483       0.0173        0.055  0.00289    0.00643     0.0050
 2     1     0.0914       0.019         0.194  0.00495    0.00608     0.0176
 3     2     0.165        0.0203        0.527  0.00737    0.00723     0.0504
 4     3     0.276        0.02          0.923  0.00854    0.00624     0.101
 5     4     0.416        0.0227        1.14   0.00745    0.00850     0.136
 6     5     0.559        0.0223        1.21   0.0122     0.00751     0.148
 7     6     0.674        0.0277        1.23   0.0235     0.00681     0.152
 8     7     0.752        0.0283        1.23   0.0340     0.00569     0.152
 9     8     0.798        0.0283        1.23   0.0411     0.00306     0.153
10     9     0.823        0.029         1.23   0.0453     0.00361     0.153
11    10     0.836        0.03          1.23   0.0476     0.00100     0.153
```

Great! Enjoy the wonderful world of pivoting :)