When pivot-wider goes wrong

Packages

The inevitable:

```
library(tidyverse)
```

Some long data that should be wide

```
# A tibble: 6 x 3
    obs time      y
  <dbl> <chr> <dbl>
1     1 pre      19
2     2 post     18
3     3 pre      17
4     4 post     16
5     5 pre      15
6     6 post     14
```

- Six observations of variable `y`, but three measured before some treatment and three measured after.
- Really matched pairs, so want column of `y`-values for `pre` and for `post`.
- `pivot_wider`.

What happens here?

```
d %>% pivot_wider(names_from = time, values_from = y)
```

```
# A tibble: 6 x 3
    obs   pre  post
  <dbl> <dbl> <dbl>
1     1    19    NA
2     2    NA    18
3     3    17    NA
4     4    NA    16
5     5    15    NA
6     6    NA    14
```

- Should be *three* `pre` values and *three* `post`. Why did this happen?
- `pivot_wider` needs to know which *row* to put each observation in.
- Uses combo of columns *not* named in `pivot_wider`, here `obs` (only).

The problem

```
d %>% pivot_wider(names_from = time, values_from = y)
```

```
# A tibble: 6 x 3
    obs   pre  post
  <dbl> <dbl> <dbl>
1     1    19    NA
2     2    NA    18
3     3    17    NA
4     4    NA    16
5     5    15    NA
6     6    NA    14
```

- There are 6 different `obs` values, so 6 different rows.
- No data for `obs` 2 and `pre`, so that cell missing (`NA`).
- Not enough data (6 obs) to fill 12 ($= 2 \times 6$) cells.
- `obs` needs to say which subject provided which *2* observations.

Fixing it up

```
# A tibble: 6 x 3
  subject time      y
    <dbl> <chr> <dbl>
1       1 pre      19
2       1 post     18
3       2 pre      17
4       2 post     16
5       3 pre      15
6       3 post     14
```

- column `subject` shows which subject provided each `pre` and `post`.
- when we do `pivot_wider`, now only *3* rows, one per subject.

Coming out right

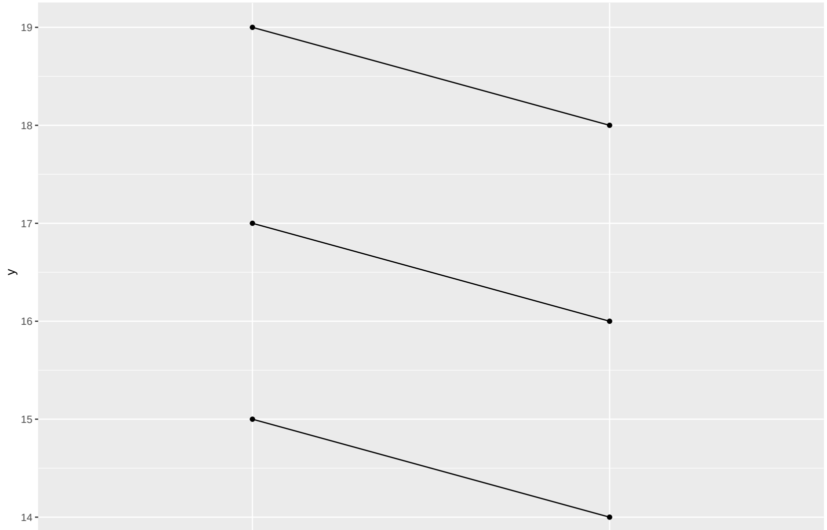
```
d2 %>% pivot_wider(names_from = time, values_from = y)
```

```
# A tibble: 3 x 3
  subject   pre  post
    <dbl> <dbl> <dbl>
1       1    19    18
2       2    17    16
3       3    15    14
```

- row each observation goes to determined by other column `subject`, and now a `pre` and `post` for each `subject`.
- right layout for matched pairs t or to make differences for sign test or normal quantile plot.
- “spaghetti plot” needs data longer, as `d2`.

Spaghetti plot

```
d2 %>% mutate(time = fct_inorder(time)) %>%
  ggplot(aes(x = time, y = y, group = subject)) +
    geom_point() + geom_line()
```



Another example

- Two independent samples this time

```
# A tibble: 8 x 2
  group         y
  <chr>     <dbl>
1 control       8
2 control      11
3 control      13
4 control      14
5 treatment    12
6 treatment    15
7 treatment    16
8 treatment    17
```

- These should be arranged like this
- but what if we make them wider?

Wider

```r
d3 %>% pivot_wider(names_from = group, values_from = y)
```

```
# A tibble: 1 x 2
  control   treatment
  <list>    <list>
1 <dbl [4]> <dbl [4]>
```

- row determined by what not used for `pivot_wider`: nothing!
- everything smooshed into *one* row!
- this time, too *much* data for the layout.
- Four data values squeezed into each of the two cells: “list-columns”.

Get the data out

- To expand list-columns out into the data values they contain, can use `unnest`:

```
d3 %>% pivot_wider(names_from = group, values_from = y) %>%
  unnest(c(control, treatment))
```

```
# A tibble: 4 x 2
  control treatment
    <dbl>     <dbl>
1       8        12
2      11        15
3      13        16
4      14        17
```

- in this case, wrong layout, because data values not paired.

A proper use of list-columns

```
d3 %>% nest_by(group) %>%
  summarize(n = nrow(data),
            mean_y = mean(data$y),
            sd_y = sd(data$y))
```

```
# A tibble: 2 x 4
# Groups:   group [2]
  group         n mean_y  sd_y
  <chr>     <int>  <dbl> <dbl>
1 control       4   11.5  2.65
2 treatment     4   15    2.16
```

- another way to do `group_by` and `summarize` to find stats by group.
- run this one piece at a time to see what it does.