Continuing our introduction to tidy data

Daniel Anderson
Week 7, Class 1

Agenda

- Review tidy data
- Cover a few more examples together
- Lab
    - Work through Alison tidy data lab (based on Jenny Bryan's LOTR tidy data example)

Agenda

- Review tidy data
- Cover a few more examples together
- Lab
    - Work through Alison tidy data lab (based on Jenny Bryan's LOTR tidy data example)

Learning objectives for today

- Get more comfortable with tidy data
- Get more fluent with `gather`

What questions do you have?

Revisiting *git*

Talk with neighbor. What do these terms mean?

- clone
- pull
- stage
- commit
- push
- repo
- remote

Load the data

Let's look at the **exam1.csv** data

```
library(tidyverse)
library(rio)
library(here)
d <- import(here("data", "exam1.csv"),
            setclass = "tbl_df")
```

Quick trial

- Try to tidy these data so they look like the below

```
## # A tibble: 630 x 4
##    stu_name gender  item correct
##    <chr>    <chr>  <dbl>   <int>
##  1 Adam     M          1       1
##  2 Adam     M          2       1
##  3 Adam     M          3       1
##  4 Adam     M          4       1
##  5 Adam     M          5       1
##  6 Adam     M          6       1
##  7 Adam     M          7       1
##  8 Adam     M          8       0
##  9 Adam     M          9       0
## 10 Adam     M         10       0
## # ... with 620 more rows
```

Step 1: `gather` the item variables

- Change all item variables into two variables: `item` and `score`

gather {tidyr} R Documenta

Gather columns into key-value pairs.

Description

Gather takes multiple columns and collapses into key-value pairs, duplicating all other columns as needed. You use `gather()` when you notice that you have columns that are not variables.

Usage

```
gather(data, key, value, ..., na.rm = FALSE, convert = FALSE,
  factor_key = FALSE)
```

Arguments

`data`	A data frame.
`key, value`	Names of key and value columns to create in output.
`...`	Specification of columns to gather. Use bare variable names. Select all variables between x and z with `x:z`, exclude y with `-y`. For more options, see the select documentation.

How does `gather` *work?*

arg 3
Columns to Gather

```
# A tibble: 3 x 5
        Country `1979` `1989` `1999` `2009`
          <chr>  <dbl>  <lgl>  <dbl>  <dbl>
1        France     NA     NA    0.3    0.4
2  South Africa     NA     NA   14.8   17.2
3 United States 0.0318     NA    0.5    0.6
```

gather(**year**, **percentage**, **-1**)

arg 1 arg 2

```
# A tibble: 12 x 3
        Country  year percentage
          <chr> <chr>      <dbl>
1        France  1979         NA
2  South Africa  1979         NA
3 United States  1979     0.0318
4        France  1989         NA
5  South Africa  1989         NA
6 United States  1989         NA
7        France  1999     0.3000
8  South Africa  1999    14.8000
9 United States  1999     0.5000
10       France  2009     0.4000
11 South Africa  2009    17.2000
12 United States  2009     0.6000
```

Try running the following code

```
d %>%
  gather(key = item, value = score, -1:-2)
```

- Third argument to `...` says we want to omit the first and second columns in when gathering.

What do you get? Are these data tidy now?

- The code on the previous slide basically puts our data in a tidy format.
- To "clean up" some, could transform the `item` variable to numeric

```
## # A tibble: 630 x 4
##    stu_name gender item   score
##    <chr>    <chr>  <chr>  <int>
##  1 Adam     M      item_1     1
##  2 Anne     F      item_1     1
##  3 Audrey   F      item_1     1
##  4 Barbara  F      item_1     1
##  5 Bert     M      item_1     1
##  6 Betty    F      item_1     1
##  7 Blaise   M      item_1     1
##  8 Brenda   F      item_1     1
##  9 Britton  F      item_1     1
## 10 Carol    F      item_1     1
## # ... with 620 more rows
```

Finish tidying the data

```
td <- d %>%
  gather(item, score, -1:-2) %>%
  mutate(item = parse_number(item))
```

- `parse_number()` comes from the *readr* package.

```
head(td)
```

```
## # A tibble: 6 x 4
##   stu_name gender  item score
##   <chr>    <chr>  <dbl> <int>
## 1 Adam     M          1     1
## 2 Anne     F          1     1
## 3 Audrey   F          1     1
## 4 Barbara  F          1     1
## 5 Bert     M          1     1
## 6 Betty    F          1     1
```

An alternative

(please run this code, following the explanation)

```
td <- d %>%
  gather(item, score, -1:-2) %>%
  separate(item, c("discard", "item"), sep = "_") %>%
  select(-discard)
```

Why are tidy data useful?

- When used in conjunction with `dplyr`, tidy data can result in large gains in efficiency.

For example, suppose we want to calculate the proportion of students responding correctly to each item.

```
td %>%
  group_by(item) %>%
  summarize(prop = mean(score))
```

```
## # A tibble: 18 x 2
##    item        prop
##    <chr>      <dbl>
##  1 1      1
##  2 10     0.6857143
##  3 11     0.3428571
##  4 12     0.1714286
##  5 13     0.2
##  6 14     0.08571429
##  7 15     0.02857143
##  8 16     0.02857143
##  9 17     0.02857143
## 10 18     0
## 11 2      1
## 12 3      1
## 13 4      0.9142857
## 14 5      0.8857143
## 15 6      0.8571429
## 16 7      0.8857143
## 17 8      0.7714286
## 18 9      0.8571429
```

What if we also wanted to know the standard deviation?

```
td %>%
  group_by(item) %>%
  summarize(prop = mean(score),
            sd = sd(score)) %>%
  head()
```

```
## # A tibble: 6 x 3
##   item        prop        sd
##   <chr>      <dbl>     <dbl>
## 1 1     1          0
## 2 10    0.6857143  0.4710082
## 3 11    0.3428571  0.4815940
## 4 12    0.1714286  0.3823853
## 5 13    0.2        0.4058397
## 6 14    0.08571429 0.2840286
```

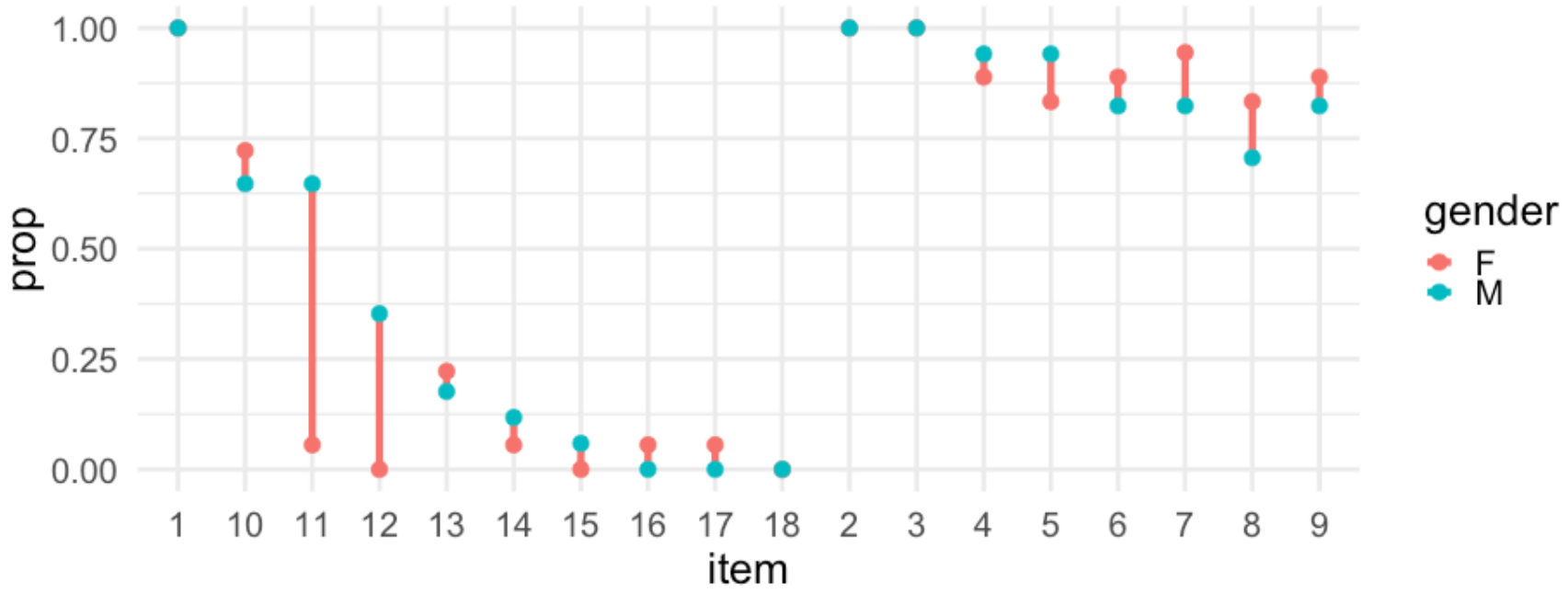
What if we wanted to know the proportion correct for each item by gender?

```
td %>%
  group_by(item, gender) %>%
  summarize(prop = mean(score))
```

```
## # A tibble: 36 x 3
## # Groups:   item [?]
##    item  gender       prop
##    <chr> <chr>       <dbl>
##  1 1     F       1
##  2 1     M       1
##  3 10    F       0.7222222
##  4 10    M       0.6470588
##  5 11    F       0.05555556
##  6 11    M       0.6470588
##  7 12    F       0
##  8 12    M       0.3529412
##  9 13    F       0.2222222
## 10 13    M       0.1764706
## # ... with 26 more rows
```

We can take the previous example further, by piping the output into a plot

```
td %>%
  group_by(item, gender) %>%
  summarize(prop = mean(score)) %>%
  mutate(gender = as.factor(gender)) %>%
  ggplot(aes(x = item, y = prop, color = gender)) +
  geom_line(aes(group = item)) +
  geom_point()
```



But, probably better (clearer) to do it in two steps.

First produce the data

```
pd <- td %>%
  group_by(item, gender) %>%
  summarize(prop = mean(score)) %>%
  mutate(gender = as.factor(gender))
```

Then produce the plot

```
ggplot(pd, aes(x = item, y = prop, color = gender)) +
  geom_point() +
  geom_line(aes(group = item))
```

An alternative

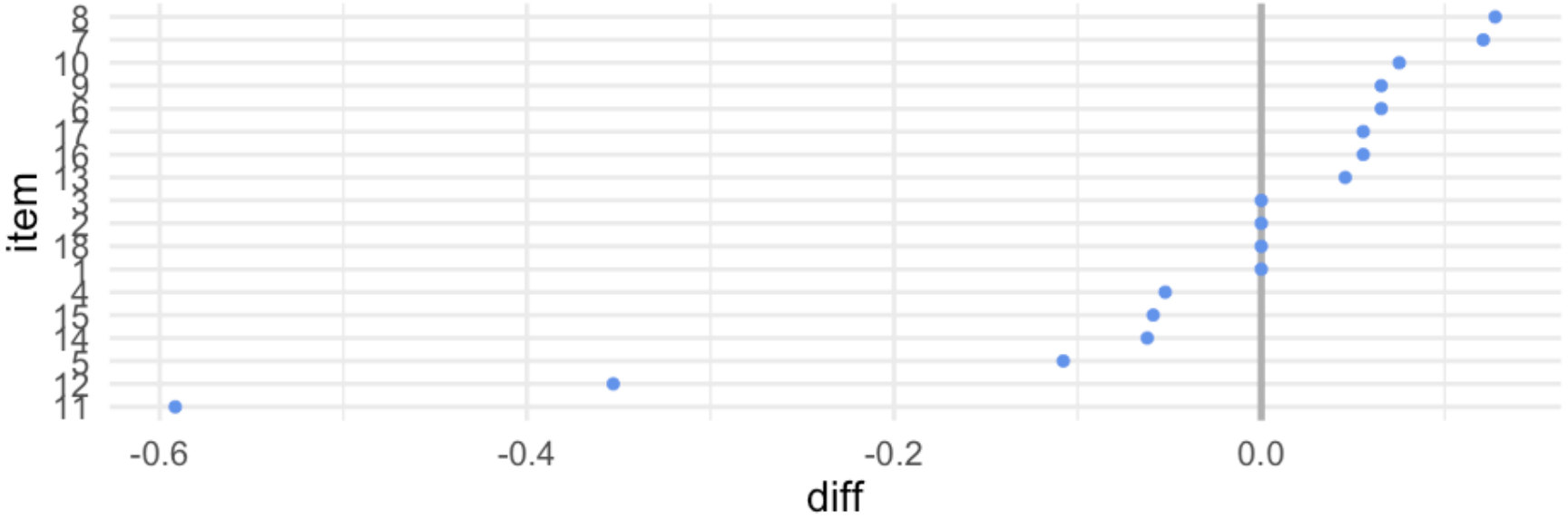
```
prop_diff <- td %>%
  group_by(item, gender) %>%
  summarize(prop = mean(score)) %>%
  mutate(gender = as.factor(gender)) %>%
  spread(gender, prop) %>%
  ungroup() %>%
  mutate(diff = F - M,
         item = fct_reorder(item, diff))

head(prop_diff)
```

```
## # A tibble: 6 x 4
##   item            F          M         diff
##   <fct>       <dbl>      <dbl>        <dbl>
## 1 1     1           1           0
## 2 10    0.7222222   0.6470588   0.07516340
## 3 11    0.05555556  0.6470588  -0.5915033
## 4 12    0           0.3529412  -0.3529412
## 5 13    0.2222222   0.1764706   0.04575163
## 6 14    0.05555556  0.1176471  -0.06209150
```

Plot it

```
ggplot(prop_diff, aes(x = item, y = diff)) +
  geom_hline(yintercept = 0,
             color = "gray70",
             size = 2) +
  geom_point(size = 3,
             color = "cornflowerblue")  +
  coord_flip()
```



Challenge

Remember, the following code calculates the mean score for each item.

```
td %>%
  group_by(item) %>%
  summarize(prop = mean(score))
```

- Try to modify the above code to produce raw scores for every student.
- If you're successful, try to also calculate the percent correct.

Calculate Raw Scores

Modify the prior code to:

- `group_by` *stu_name* (rather than *item*)
- `sum` score (rather than average it with `mean`)

```
td %>%
  group_by(stu_name) %>%
  summarize(raw_score = sum(score)) %>%
  head()
```

```
## # A tibble: 6 x 2
##   stu_name raw_score
##   <chr>        <int>
## 1 Adam             7
## 2 Anne            10
## 3 Audrey          11
## 4 Barbara          6
## 5 Bert             8
## 6 Betty            9
```

Calculate percent correct

```
td %>%
  group_by(stu_name) %>%
  summarize(total_poss = max(n()),
            raw_score = sum(score),
            pct_correct = raw_score / total_poss)
```

```
## # A tibble: 35 x 4
##    stu_name total_poss raw_score pct_correct
##    <chr>          <int>     <int>       <dbl>
##  1 Adam              18         7   0.3888889
##  2 Anne              18        10   0.5555556
##  3 Audrey            18        11   0.6111111
##  4 Barbara           18         6   0.3333333
##  5 Bert              18         8   0.4444444
##  6 Betty             18         9   0.5
##  7 Blaise            18        13   0.7222222
##  8 Brenda            18        10   0.5555556
##  9 Britton           18         8   0.4444444
## 10 Carol             18         6   0.3333333
## # ... with 25 more rows
```

Another common format with longitudinal data

Are these tidy? If not, what's wrong?

```
## # A tibble: 5 x 9
##     sid wave_1_math wave_2_math wave_3_math wave_4_math wave_1_rdg
##   <int>       <dbl>       <dbl>       <dbl>       <dbl>      <dbl>
## 1     1          95          98         102         105         96
## 2     2         101         103         107         109        108
## 3     3          99         103         106         110        103
## 4     4         109         111         115         116        108
## 5     5         101         104         107         113         92
## # ... with 3 more variables: wave_2_rdg <dbl>, wave_3_rdg <dbl>,
## #   wave_4_rdg <dbl>
```

Variable names include data

```
ld %>%
  gather(var, score, -1)
```

```
## # A tibble: 40 x 3
##       sid var          score
##     <int> <chr>        <dbl>
##  1      1 wave_1_math     95
##  2      2 wave_1_math    101
##  3      3 wave_1_math     99
##  4      4 wave_1_math    109
##  5      5 wave_1_math    101
##  6      1 wave_2_math     98
##  7      2 wave_2_math    103
##  8      3 wave_2_math    103
##  9      4 wave_2_math    111
## 10      5 wave_2_math    104
## # ... with 30 more rows
```

```
ld %>%
  gather(var, score, -1) %>%
  separate(var,
           c("dis", "wave", "subject"),
           sep = "_",
           convert = TRUE)
```

```
## # A tibble: 40 x 5
##       sid dis    wave subject score
##     <int> <chr> <int> <chr>   <dbl>
##  1      1 wave      1 math       95
##  2      2 wave      1 math      101
##  3      3 wave      1 math       99
##  4      4 wave      1 math      109
##  5      5 wave      1 math      101
##  6      1 wave      2 math       98
##  7      2 wave      2 math      103
##  8      3 wave      2 math      103
##  9      4 wave      2 math      111
## 10      5 wave      2 math      104
## # ... with 30 more rows
```

```
tidy_ld <- ld %>%
  gather(var, score, -1) %>%
  separate(var,
           c("dis", "wave", "subject"),
           sep = "_",
           convert = TRUE) %>%
  select(-dis)

tidy_ld
```

```
## # A tibble: 40 x 4
##      sid  wave subject score
##    <int> <int> <chr>   <dbl>
##  1     1     1 math       95
##  2     2     1 math      101
##  3     3     1 math       99
##  4     4     1 math      109
##  5     5     1 math      101
##  6     1     2 math       98
##  7     2     2 math      103
##  8     3     2 math      103
##  9     4     2 math      111
## 10     5     2 math      104
## # ... with 30 more rows
```

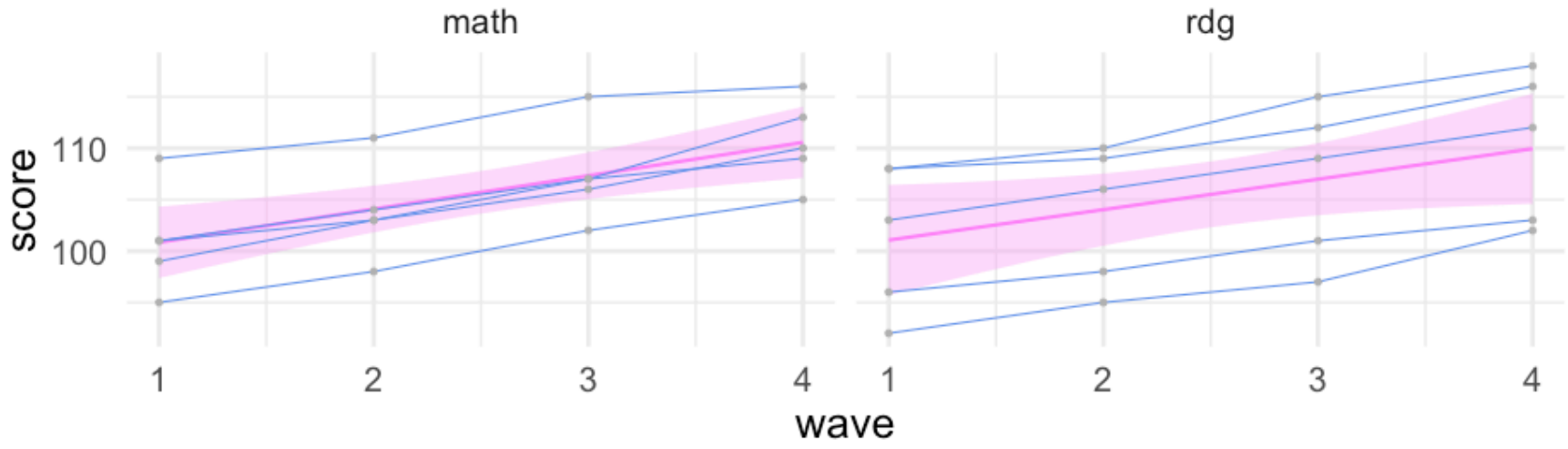
Again - why so useful? Quick summaries!

```
tidy_ld %>%
  group_by(wave, subject) %>%
  summarize(n = n(),
            mean = mean(score),
            sd = sd(score)) %>%
  arrange(subject, wave) %>%
  head()
```

```
## # A tibble: 6 x 5
## # Groups:   wave [4]
##    wave subject     n  mean       sd
##   <int> <chr>   <int> <dbl>    <dbl>
## 1     1 math        5 101   5.099020
## 2     2 math        5 103.8 4.658326
## 3     3 math        5 107.4 4.722288
## 4     4 math        5 110.6 4.159327
## 5     1 rdg         5 101.4 7.197222
## 6     2 rdg         5 103.6 6.730527
```

Plotting

```
ggplot(tidy_ld, aes(wave, score)) +
  geom_smooth(method = "lm",
              color = "orchid1",
              fill = "orchid1",
              alpha = 0.3) +
  geom_line(color = "cornflowerblue", aes(group = sid)) +
  geom_point(color = "gray70") +
  facet_wrap(~subject)
```



Spreading the data back out

Tidy data are great when conducting preliminary descriptives and for plotting the data. But if you're using other packages for analysis, it may need to be in a different format.

spread {tidyr} R Documentation

Spread a key-value pair across multiple columns.

Description

Spread a key-value pair across multiple columns.

Usage

```
spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE,
  sep = NULL)
```

Arguments

`data`
: A data frame.

`key`
: The bare (unquoted) name of the column whose values will be used as column headings.

`value`
: The bare (unquoted) name of the column whose values will populate the cells.

`fill`
: If set, missing values will be replaced with this value. Note that there are two types of missingness in the input: explicit missing values (i.e. `NA`), and implicit missings, rows that simply aren't present. Both types of missing value will be replaced by `fill`.

`convert`
: If `TRUE`, `type.convert` with `asis = TRUE` will be run on each of the new columns. This is useful if the value column was a mix of variables that was coerced to a string. If the class of the value column was factor or date, note that will not be true of the new columns that are produced, which are coerced to character before type conversion.

Spread *td*

Reminder what the tidy data look like

```
## # A tibble: 630 x 4
##    stu_name gender item  score
##    <chr>    <chr>  <chr> <int>
##  1 Adam     M      1         1
##  2 Anne     F      1         1
##  3 Audrey   F      1         1
##  4 Barbara  F      1         1
##  5 Bert     M      1         1
##  6 Betty    F      1         1
##  7 Blaise   M      1         1
##  8 Brenda   F      1         1
##  9 Britton  F      1         1
## 10 Carol    F      1         1
## # ... with 620 more rows
```

```
s_d <- td %>%
  spread(item, score)

s_d
```

```
## # A tibble: 35 x 20
##    stu_name gender   `1`  `10`  `11`  `12`  `13`  `14`  `15`  `16`  `17`
##    <chr>    <chr>  <int> <int> <int> <int> <int> <int> <int> <int> <int>
##  1 Adam     M          1     0     0     0     0     0     0     0     0
##  2 Anne     F          1     1     0     0     0     0     0     0     0
##  3 Audrey   F          1     1     0     0     1     0     0     0     0
##  4 Barbara  F          1     1     0     0     0     0     0     0     0
##  5 Bert     M          1     1     0     0     0     0     0     0     0
##  6 Betty    F          1     0     0     0     0     0     0     0     0
##  7 Blaise   M          1     1     1     1     1     0     0     0     0
##  8 Brenda   F          1     1     0     0     0     0     0     0     0
##  9 Britton  F          1     0     0     0     0     0     0     0     0
## 10 Carol    F          1     0     0     0     0     0     0     0     0
## # ... with 25 more rows, and 9 more variables: `18` <int>, `2` <int>,
## #   `3` <int>, `4` <int>, `5` <int>, `6` <int>, `7` <int>, `8` <int>,
## #   `9` <int>
```

More on spreading

- It's also common to have to `gather` beyond where you really need to, manipulate the variable, then spread it back out.

```
head(sim2)
```

```
## # A tibble: 6 x 19
##     SID male_g6 male_g7 male_g8 ell_g6 ell_g7 ell_g8 sped_g6 sped_g7
##   <int>   <int>   <int>   <int>  <int>  <int>  <int>   <int>   <int>
## 1     1       0       0       0      0      0      0       0       0
## 2     2       1       1       1      0      0      0       0       0
## 3     3       1       1       1      1      1      1       0       0
## 4     4       1       1       1      0      0      0       0       0
## 5     5       0       0       0      0      0      0       0       0
## 6     6       1       1       1      0      0      0       0       0
## # ... with 10 more variables: sped_g8 <int>, pullouts_g6 <int>,
## #   pullouts_g7 <int>, pullouts_g8 <int>, disability_g6 <chr>,
## #   disability_g7 <chr>, disability_g8 <chr>, score_g6 <dbl>,
## #   score_g7 <dbl>, score_g8 <dbl>
```

First `gather` all vars

```
sim2 %>%
    gather(var, val, -1)
```

```
## # A tibble: 1,800 x 3
##       SID var     val
##     <int> <chr>   <chr>
##  1      1 male_g6 0
##  2      2 male_g6 1
##  3      3 male_g6 1
##  4      4 male_g6 1
##  5      5 male_g6 0
##  6      6 male_g6 1
##  7      7 male_g6 1
##  8      8 male_g6 0
##  9      9 male_g6 0
## 10     10 male_g6 1
## # ... with 1,790 more rows
```

Next, separate

```
sim2 %>%
    gather(var, val, -1) %>%
    separate(var, c("var", "grade"), sep = "_")
```

```
## # A tibble: 1,800 x 4
##       SID var   grade val
##     <int> <chr> <chr> <chr>
##  1      1 male  g6    0
##  2      2 male  g6    1
##  3      3 male  g6    1
##  4      4 male  g6    1
##  5      5 male  g6    0
##  6      6 male  g6    1
##  7      7 male  g6    1
##  8      8 male  g6    0
##  9      9 male  g6    0
## 10     10 male  g6    1
## # ... with 1,790 more rows
```

Parse Numeric

```
sim2 %>%
    gather(var, val, -1) %>%
    separate(var, c("var", "grade"), sep = "_") %>%
    mutate(grade = parse_number(grade))
```

```
## # A tibble: 1,800 x 4
##       SID var   grade val
##     <int> <chr> <dbl> <chr>
##  1      1 male      6 0
##  2      2 male      6 1
##  3      3 male      6 1
##  4      4 male      6 1
##  5      5 male      6 0
##  6      6 male      6 1
##  7      7 male      6 1
##  8      8 male      6 0
##  9      9 male      6 0
## 10     10 male      6 1
## # ... with 1,790 more rows
```

spread for final produce

```
sim2 %>%
    gather(var, val, -1) %>%
    separate(var, c("var", "grade"), sep = "_") %>%
    mutate(grade = parse_number(grade)) %>%
    spread(var, val)
```

```
## # A tibble: 300 x 8
##      SID grade disability ell   male  pullouts score            sped
##    <int> <dbl> <chr>      <chr> <chr> <chr>    <chr>            <chr>
##  1     1     6 none       0     0     0        208.434415681363 0
##  2     1     7 none       0     0     0        212.52698033647  0
##  3     1     8 none       0     0     0        219.999423463527 0
##  4     2     6 none       0     1     0        193.235211343351 0
##  5     2     7 none       0     1     0        200.918088606708 0
##  6     2     8 none       0     1     0        205.561434813331 0
##  7     3     6 asd        1     1     0        196.085670969229 0
##  8     3     7 asd        1     1     0        203.05758537086  0
##  9     3     8 asd        1     1     0        210.530524529609 0
## 10     4     6 none       0     1     0        204.049406440706 0
## # ... with 290 more rows
```

Same thing with our longitudinal data from before

Say we wanted a wave column, but wanted separate columns by subject

```
tidy_ld %>%
  spread(subject, score) %>%
  head()
```

```
## # A tibble: 6 x 4
##     sid  wave  math   rdg
##   <int> <int> <dbl> <dbl>
## 1     1     1    95    96
## 2     1     2    98    98
## 3     1     3   102   101
## 4     1     4   105   103
## 5     2     1   101   108
## 6     2     2   103   110
```

Lab