Introduction to tidy data

(AKA what we should have covered before Alison's visit)

Daniel Anderson
Week 6, Class 2

UNIVERSITY OF OREGON

Agenda

- Introduce the concept of tidy data
- Tidy a simple dataset together with `tidyr`
- Summarize and transform tidy data with `dplyr`
- Lab

Agenda

- Introduce the concept of tidy data
- Tidy a simple dataset together with `tidyr`
- Summarize and transform tidy data with `dplyr`
- Lab

Learning objectives for today

- Understand the concept of tidy data is useful
- Understand and be able to apply the `gather` function

Revisiting *git*

Talk with neighbor. What do these terms mean?

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

> It is often said that 80% of data analysis is spent on the process of cleaning and preparing the data.
>
> -Hadley Wickham

- Persistent and varied challenge
- Little research on how to do it well
  - Enter Hadley

Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. http://www.jstatsoft.org/

Tidy Data

Hadley Wickham
RStudio

Abstract

A huge amount of effort is spent cleaning data to get it ready for analysis, but there has been little research on how to make data cleaning as easy and effective as possible. This paper tackles a small, but important, component of data cleaning: data tidying. Tidy datasets are easy to manipulate, model and visualise, and have a specific structure: each variable is a column, each observation is a row, and each type of observational unit is a table. This framework makes it easy to tidy messy datasets because only a small set of tools are needed to deal with a wide range of un-tidy datasets. This structure also makes it easier to develop tidy tools for data analysis, tools that both input and output tidy datasets. The advantages of a consistent data structure and matching tools are demonstrated with a case study free from mundane data manipulation chores.

Keywords: data cleaning, data tidying, relational databases, R.

1. Introduction

It is often said that 80% of data analysis is spent on the process of cleaning and preparing the data (Dasu and Johnson 2003). Data preparation is not just a first step, but must be repeated many over the course of analysis as new problems come to light or new data is collected. Despite the amount of time it takes, there has been surprisingly little research on how to clean data well. Part of the challenge is the breadth of activities it encompasses: from outlier checking, to date parsing, to missing value imputation. To get a handle on the problem, this paper focusses on a small, but important, aspect of data cleaning that I call data **tidying**: structuring datasets to facilitate analysis.

The principles of tidy data provide a standard way to organise data values within a dataset. A standard makes initial data cleaning easier because you don't need to start from scratch and reinvent the wheel every time. The tidy data standard has been designed to facilitate initial exploration and analysis of the data, and to simplify the development of data analysis tools that work well together. Current tools often require translation. You have to spend time

Tidy data

Definition

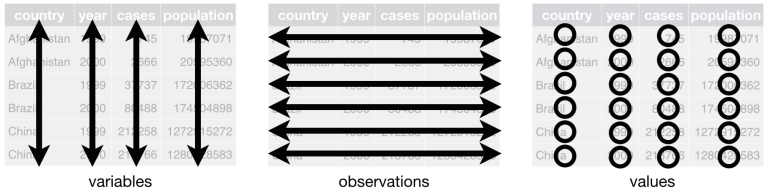
1. Each variable is a column
2. Each observation is a row
3. Each type of observational unit forms a table

Tidy data

Definition

1. Each variable is a column
2. Each observation is a row
3. Each type of observational unit forms a table

We won't talk about 3 so much...



Common ways data are "messy"

- Column headers are values, not variable names
- Multiple variables stored in one column
- Variables are stored in both rows and columns

Some examples

(from the JSS paper)

```
## # A tibble: 18 x 11
##    religion      `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k` `$50-75k`
##    <chr>           <int>     <int>     <int>     <int>     <int>     <int>
##  1 Agnostic           27        34        60        81        76       137
##  2 Atheist            12        27        37        52        35        70
##  3 Buddhist           27        21        30        34        33        58
##  4 Catholic          418       617       732       670       638      1116
##  5 Don't know/r…      15        14        15        11        10        35
##  6 Evangelical …     575       869      1064       982       881      1486
##  7 Hindu               1         9         7         9        11        34
##  8 Historically…     228       244       236       238       197       223
##  9 Jehovah's Wi…      20        27        24        24        21        30
## 10 Jewish             19        19        25        25        30        95
## # ... with 8 more rows, and 4 more variables: `$75-100k` <int>,
## #   `$100-150k` <int>, `>150k` <int>, `Don't know/refused` <int>
```


The tidied version

```
## # A tibble: 180 x 3
##    religion income             freq
##    <chr>    <fct>             <int>
##  1 Agnostic <$10k                27
##  2 Agnostic $10-20k              34
##  3 Agnostic $20-30k              60
##  4 Agnostic $30-40k              81
##  5 Agnostic $40-50k              76
##  6 Agnostic $50-75k             137
##  7 Agnostic $75-100k            122
##  8 Agnostic $100-150k           109
##  9 Agnostic >150k                84
## 10 Agnostic Don't know/refused   96
## 11 Atheist  <$10k                12
## 12 Atheist  $10-20k              27
## # ... with 168 more rows
```

Yet another example

```
## # A tibble: 10 x 13
##    country  year  m014 m1524 m2534 m3544 m4554    mu  f014 f1524 f2534
##    <chr>   <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
##  1 AD       2000     0     0     1     0     0    NA    NA    NA    NA
##  2 AE       2000     2     4     4     6     5    NA     3    16     1
##  3 AF       2000    52   228   183   149   129    NA    93   414   565
##  4 AG       2000     0     0     0     0     0    NA     1     1     1
##  5 AL       2000     2    19    21    14    24    NA     3    11    10
##  6 AM       2000     2   152   130   131    63    NA     1    24    27
##  7 AN       2000     0     0     1     2     0    NA     0     0     1
##  8 AO       2000   186   999  1003   912   482    NA   247  1142  1091
##  9 AR       2000    97   278   594   402   419    NA   121   544   479
## 10 AS       2000    NA    NA    NA    NA     1    NA    NA    NA    NA
## # ... with 2 more variables: f3544 <int>, f4554 <int>
```

In this example, *M* indicates if the data came from a male, while *F* indicates female. The subsequent numbers represent the age range. Tidying these data will be a two step process.

Step one

```
## # A tibble: 2,814 x 4
##    country  year variable cases
##    <chr>   <int> <chr>    <int>
##  1 AD       2000 m014         0
##  2 AE       2000 m014         2
##  3 AF       2000 m014        52
##  4 AG       2000 m014         0
##  5 AL       2000 m014         2
##  6 AM       2000 m014         2
##  7 AN       2000 m014         0
##  8 AO       2000 m014       186
##  9 AR       2000 m014        97
## 10 AS       2000 m014        NA
## # ... with 2,804 more rows
```

Notice this is much closer to what we want, but we have a problem now in that we have **two variables stored in one column**.

Step two: Tidied data

```
## # A tibble: 2,259 x 5
##    country  year sex   age_range cases
##    <chr>   <int> <chr> <chr>     <int>
##  1 AD       2000 m     0-14          0
##  2 AD       2000 m     15-24         0
##  3 AD       2000 m     25-34         1
##  4 AD       2000 m     35-44         0
##  5 AD       2000 m     45-54         0
##  6 AD       2000 m     55-64         0
##  7 AD       2000 m     65+           0
##  8 AE       2000 m     0-14          2
##  9 AE       2000 m     15-24         4
## 10 AE       2000 m     25-34         4
## # ... with 2,249 more rows
```


Variables as rows and columns

```
## # A tibble: 22 x 35
##    id       year month element    d1    d2    d3    d4    d5    d6    d7
##    <chr>   <int> <int> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 MX17004  2010     1 tmax       NA  NA    NA       NA  NA       NA    NA
##  2 MX17004  2010     1 tmin       NA  NA    NA       NA  NA       NA    NA
##  3 MX17004  2010     2 tmax       NA  27.3  24.1     NA  NA       NA    NA
##  4 MX17004  2010     2 tmin       NA  14.4  14.4     NA  NA       NA    NA
##  5 MX17004  2010     3 tmax       NA  NA    NA       NA  32.1     NA    NA
##  6 MX17004  2010     3 tmin       NA  NA    NA       NA  14.2     NA    NA
##  7 MX17004  2010     4 tmax       NA  NA    NA       NA  NA       NA    NA
##  8 MX17004  2010     4 tmin       NA  NA    NA       NA  NA       NA    NA
##  9 MX17004  2010     5 tmax       NA  NA    NA       NA  NA       NA    NA
## 10 MX17004  2010     5 tmin       NA  NA    NA       NA  NA       NA    NA
## # ... with 12 more rows, and 24 more variables: d8 <dbl>, d9 <dbl>,
## #   d10 <dbl>, d11 <dbl>, d12 <dbl>, d13 <dbl>, d14 <dbl>, d15 <dbl>,
## #   d16 <dbl>, d17 <dbl>, d18 <dbl>, d19 <dbl>, d20 <dbl>, d21 <dbl>,
## #   d22 <dbl>, d23 <dbl>, d24 <dbl>, d25 <dbl>, d26 <dbl>, d27 <dbl>,
## #   d28 <dbl>, d29 <dbl>, d30 <dbl>, d31 <dbl>
```

Two Steps

Step 1

```
## # A tibble: 66 x 6
##    id       year month element day_key value
##    <chr>   <int> <int> <chr>   <chr>   <dbl>
##  1 MX17004  2010    12 tmax    d1       29.9
##  2 MX17004  2010    12 tmin    d1       13.8
##  3 MX17004  2010     2 tmax    d2       27.3
##  4 MX17004  2010     2 tmin    d2       14.4
##  5 MX17004  2010    11 tmax    d2       31.3
##  6 MX17004  2010    11 tmin    d2       16.3
##  7 MX17004  2010     2 tmax    d3       24.1
##  8 MX17004  2010     2 tmin    d3       14.4
##  9 MX17004  2010     7 tmax    d3       28.6
## 10 MX17004  2010     7 tmin    d3       17.5
## # ... with 56 more rows
```


Step 2

```
## # A tibble: 33 x 4
##    id      date         tmax  tmin
##    <chr>   <chr>       <dbl> <dbl>
##  1 MX17004 2010-01-01 27.8    14.5
##  2 MX17004 2010-02-02 29.7    13.4
##  3 MX17004 2010-02-02 27.3    14.4
##  4 MX17004 2010-02-02 29.9    10.7
##  5 MX17004 2010-02-02 24.1    14.4
##  6 MX17004 2010-03-03 34.5    16.8
##  7 MX17004 2010-03-03 31.1    17.6
##  8 MX17004 2010-03-03 32.1    14.2
##  9 MX17004 2010-04-04 36.300  16.7
## 10 MX17004 2010-05-05 33.2    18.2
## # ... with 23 more rows
```


Defining tidy data

(slightly differently)

Two rules essentially define tidy data[1]

1. Each row is a case representing the same underlying attribute.
2. Each column is a variable containing the same type of value for each case.

1. From Modern Data Science with R

Defining tidy data

(slightly differently)

Two rules essentially define tidy data[1]

1. Each row is a case representing the same underlying attribute.
2. Each column is a variable containing the same type of value for each case.

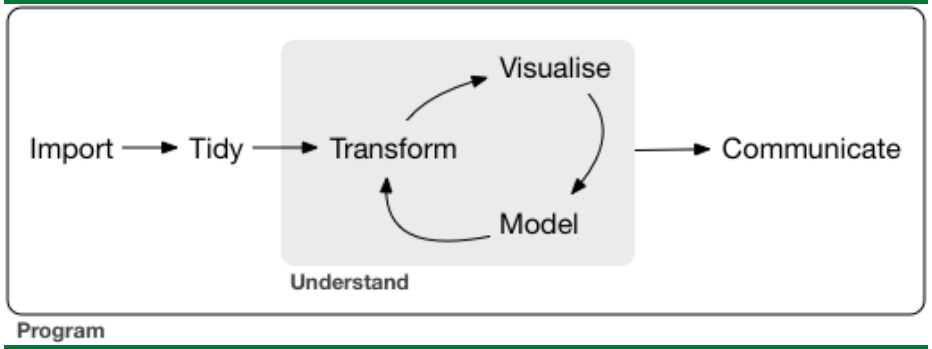
The combination of rows and columns make each case (row) unique, even though cells may be repeated many times (e.g., student identifier).

1. From Modern Data Science with R

Slides available at: http://www.datalorax.com/vita/ds/ds1-slides/w6p2/ 17 / 54

Other formats

- There are many reasons why you might want to have "messy" data. However, tidy data is an extremely useful format generally, and particularly useful when applying tools within the *tidyverse*.
- All packages within the tidyverse are designed to either help you get your data in a tidy format, or assume your data are already in a tidy format.
- Assuming a common data format leads to large jumps in efficiency, as the output from certain functions can be directly input into others.



tidyverse data analysis philosophy

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")
```


Pop Quiz Time

- Are these data tidy?
- If not, what needs to happen to make them tidy?
- What are the variables? What are the values?

```
## # A tibble: 35 x 20
##    stu_name gender item_1 item_2 item_3 item_4 item_5 item_6 item_7 item_8
##    <chr>    <chr>   <int>  <int>  <int>  <int>  <int>  <int>  <int>  <int>
##  1 Adam     M           1      1      1      1      1      1      1      0
##  2 Anne     F           1      1      1      1      1      1      1      1
##  3 Audrey   F           1      1      1      1      1      1      1      1
##  4 Barbara  F           1      1      1      1      0      0      1      0
##  5 Bert     M           1      1      1      1      1      0      1      0
##  6 Betty    F           1      1      1      1      1      1      1      1
##  7 Blaise   M           1      1      1      1      1      1      1      1
##  8 Brenda   F           1      1      1      1      1      1      1      1
##  9 Britton  F           1      1      1      0      1      1      1      1
## 10 Carol    F           1      1      1      1      0      0      1      0
## # ... with 25 more rows, and 10 more variables: item_9 <int>,
## #   item_10 <int>, item_11 <int>, item_12 <int>, item_13 <int>,
## #   item_14 <int>, item_15 <int>, item_16 <int>, item_17 <int>,
## #   item_18 <int>
```

{dplyr} versus {tidyr}

{dplyr} versus {tidyr}

{dplyr}

Helps you manipulate your data
(create, remove, summarize, etc.)

{dplyr} versus {tidyr}

{dplyr}

Helps you manipulate your data
(create, remove, summarize, etc.)

{tidyr}

Helps you get your data into a tidy format

Verbs: `tidyr`

- `gather()`
- `spread()`
- `separate()` and `extract()`
- `unite()`
- `nest()`

What do you think each do?

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.

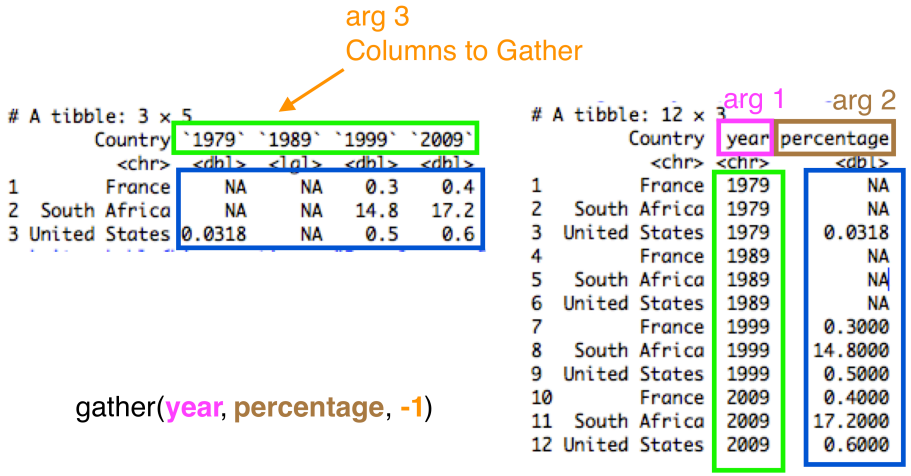
Usage

```
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?*




```
# 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
```

```
# 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))
```

```
## # A tibble: 18 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
##  7 15     0.02857143 0.1690309
##  8 16     0.02857143 0.1690309
##  9 17     0.02857143 0.1690309
## 10 18     0         0
## 11 2      1         0
## 12 3      1         0
## 13 4      0.9142857 0.2840286
## 14 5      0.8857143 0.3228029
## 15 6      0.8571429 0.3550358
## 16 7      0.8857143 0.3228029
## 17 8      0.7714286 0.4260430
```

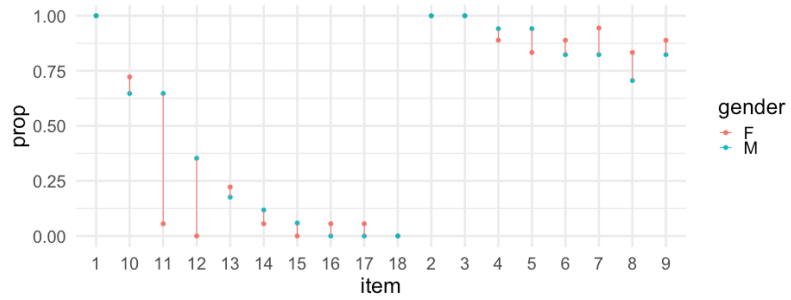

What if we wanted to know the proportion correct for each item by gender?

```
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_point(size = 2) +
  geom_line(aes(group = item))
```



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))
```

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))
```

```
## # A tibble: 35 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
##  7 Blaise          13
##  8 Brenda          10
##  9 Britton          8
## 10 Carol            6
```


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?

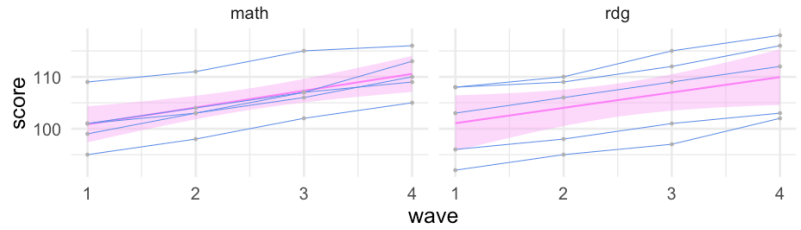
Quick summaries

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

```
## # A tibble: 8 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
## 7     3 rdg         5 106.8 7.563068
## 8     4 rdg         5 110.2 7.362065
```

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 `as.is = 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)
```

```
## # A tibble: 20 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
##  7      2     3   107   115
##  8      2     4   109   118
##  9      3     1    99   103
## 10      3     2   103   106
## 11      3     3   106   109
## 12      3     4   110   112
```


Lab