Week 8: Tidy Data

{tidyr}

Joe Nese

University of Oregon

Fall 2021

Tidy Data

Week 8

Assignment Updates

Change

- Final Project: Draft Data Script: due 11/24 (Week 9)
- Final Project: Peer Review of Script: due 12/1 (Week 10)

Same

- Final Project: Presentation: due 12/1 (Week 10)
- Final Project: Paper: due 12/8 (Week 11)

How I feel about my R code sometimes

Agenda

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

Learning Objectives

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


tidyverse

Providing grammar for:

- Graphics
  - `{ggplot2}`
- Data manipulations
  - `{dplyr}`
  - `{tidyr}`
- Expanding area of specialized topics
  - `{lubridate}`
  - `{glue}`
  - `{tidymodels}`
- Many more...

Providing grammar for:

- Graphics
  - `{ggplot2}`
- Data manipulations
  - `{dplyr}`
  - `{tidyr}`
- Expanding area of specialized topics
  - `{lubridate}`
  - `{glue}`
  - `{tidymodels}`
- Many more...

{dplyr} vs {tidyr}

{dplyr}

Helps you manipulate your data

- `select()`
- `filter()`
- `mutate()`
- `summarize()`
- more...

{tidyr}

Helps you get your data into a tidy format

- `pivot_longer()`
- `pivot_wider()`
- `separate()` and `extract()`
- `unite()`
- `nest()`

Talk with a neighbor about what you think `pivot_longer()` and `pivot_wider()` do

01:30

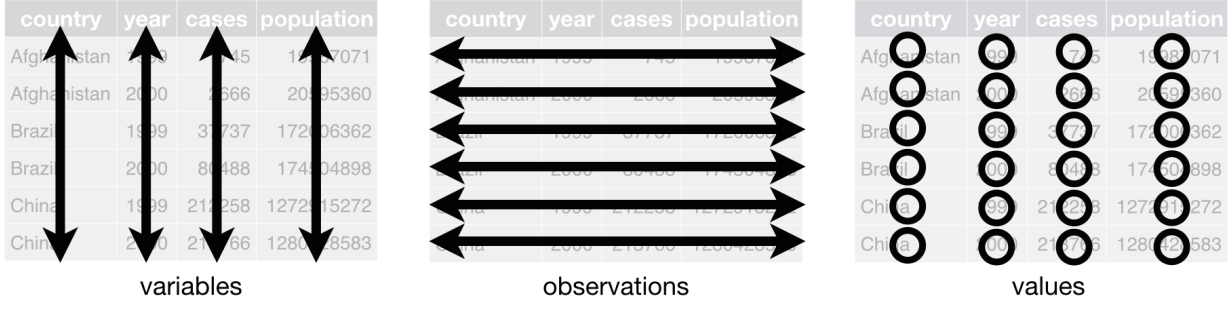
Data cleaning

> “It is often said that 80% of data analysis is spent on the process of cleaning and preparing the data (Dasu and Johnson 2003).”
>
> - Hadley Wickham (Tidy Data)

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

Tidy Data

- Each variable is a column
- Each observation is a row
- Each type of observational unit forms a table
    - We won't talk much about this point

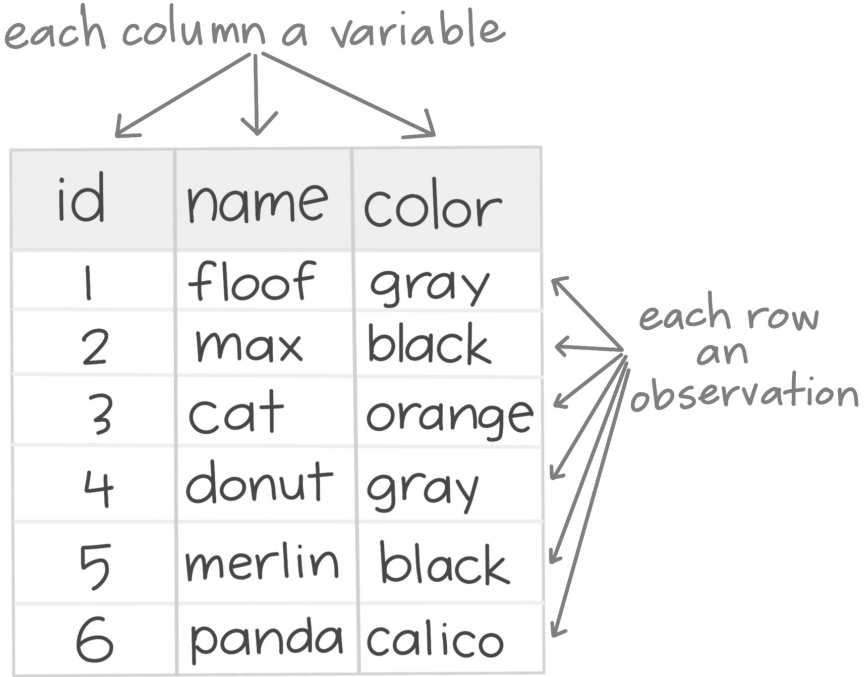


"TIDY DATA is a standard way of mapping the meaning of a dataset to its structure."

—HADLEY WICKHAM

In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement



Allison Horst

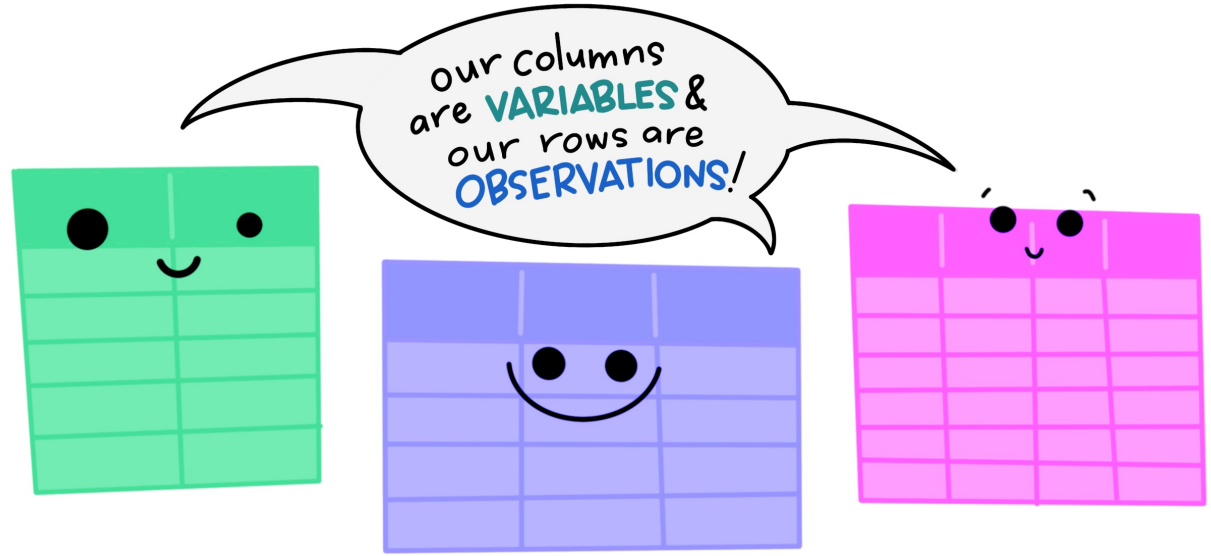
Wickham, H. (2014). Tidy Data. Journal of Statistical Software *59* (10). DOI: 10.18637/jss.v059.i10

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

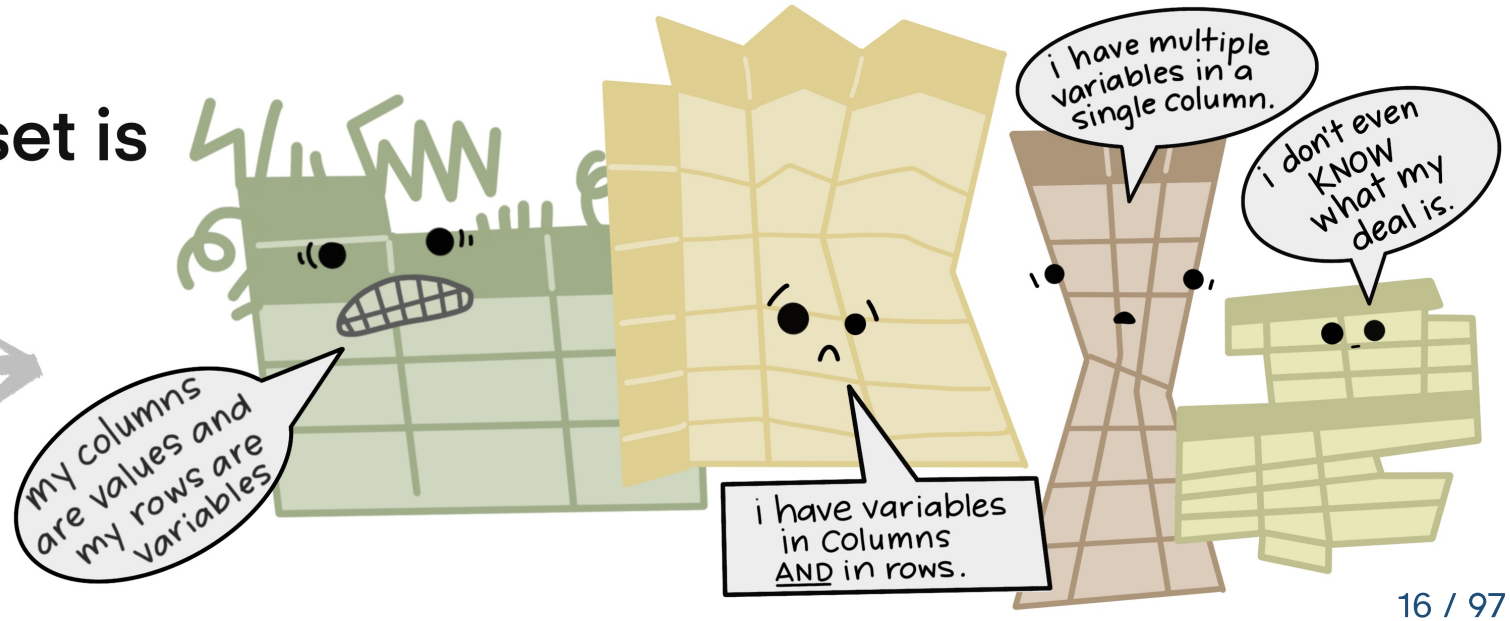
- See the pivoting vignette for examples and solutions

The standard structure of tidy data means that "tidy datasets are all alike..."



"...but every messy dataset is messy in its own way."

–HADLEY WICKHAM



Allison Horst

Defining tidy data

Essentially, two rules 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

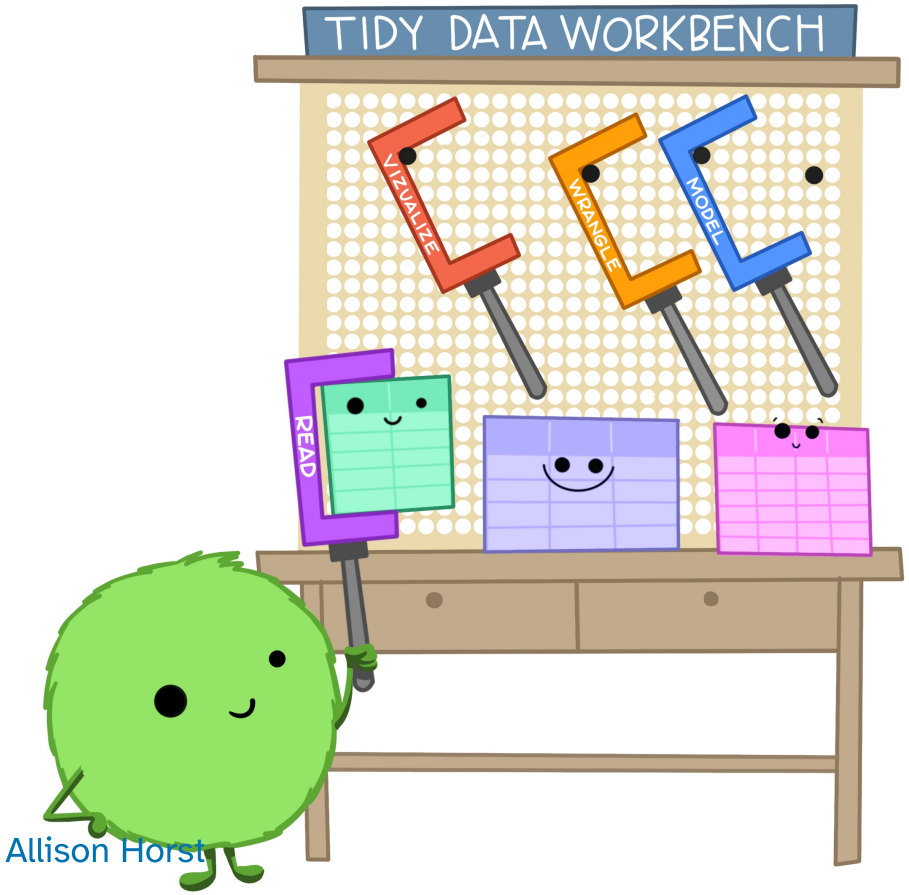
The combination of rows and columns make each observational unit (row) unique, even though cells may be repeated many times (e.g., student identifier)

[1] Modern Data Science with R

Why tidy?

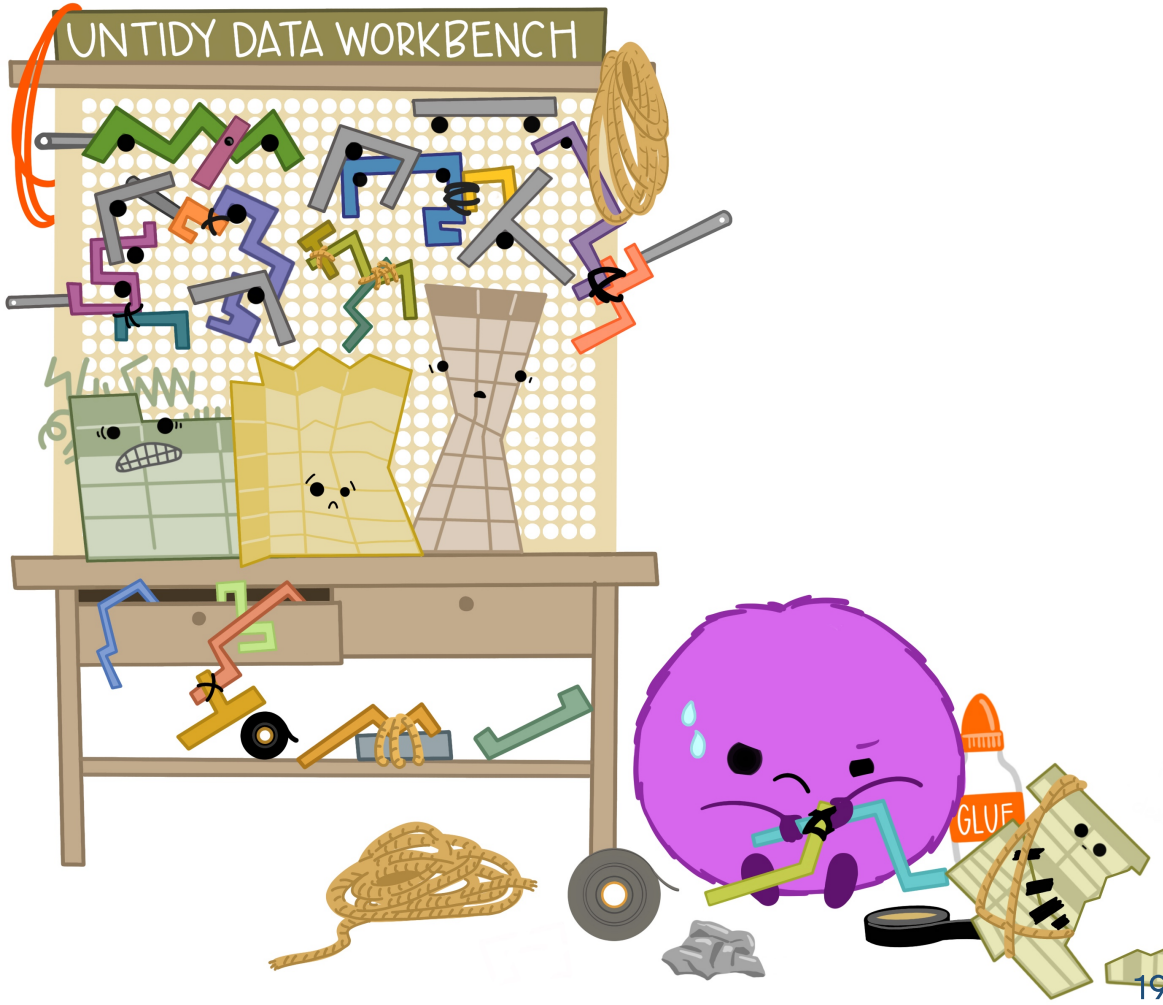
- There are many reasons why you might want to have "messy" data
    - e.g., for other software or specific modeling purposes
- However, tidy data is an extremely useful format generally, and particularly useful when applying tools within the `{tidyverse}`
- All packages within the `{tidyverse}` are 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

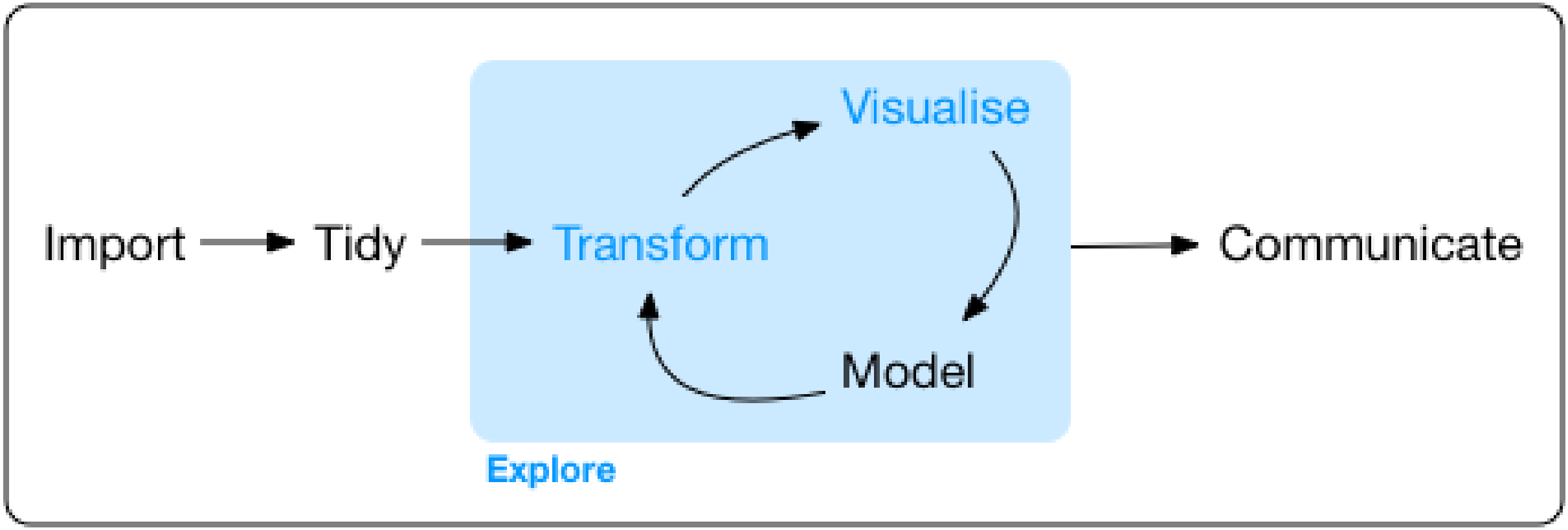
When working with tidy data, we can use the same tools in similar ways for different datasets...



Allison Horst

...but working with untidy data often means reinventing the wheel with one-time approaches that are hard to iterate or reuse.




Visualise
Import
Tidy
Transform
Communicate
Model
Explore
Program

{tidyr} verbs

`pivot_longer()`: "lengthens" data, increasing the number of rows and decreasing the number of columns

`pivot_wider()`: "widens" data, increasing the number of columns and decreasing the number of rows

`separate()`: turns a single character column into multiple columns

`extract()`: given a regular expression for capturing groups, turns each group into a new column

`unite()`: paste together multiple columns into one

`nest()`: creates a list-column of data frames; you get one row for each group defined by the non-nested columns. This is useful in conjunction with other summaries that work with whole datasets, like models

- **so** powerful

pivot_longer()

There are different ways to `pivot_longer()` that will depend on what your data look like

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  names_prefix = NULL,
  names_sep = NULL,
  names_pattern = NULL,
  names_ptypes = list(),
  names_transform = list(),
  names_repair = "check_unique",
  values_to = "value",
  values_drop_na = FALSE,
  values_ptypes = list(),
  values_transform = list(),
  ...
)
```

pivot_longer()

There are different ways to `pivot_longer()` that will depend on what your data look like

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  names_prefix = NULL,
  names_sep = NULL,
  names_pattern = NULL,
  names_ptypes = list(),
  names_transform = list(),
  names_repair = "check_unique",
  values_to = "value",
  values_drop_na = FALSE,
  values_ptypes = list(),
  values_transform = list(),
  ...
)
```

pivot_longer()

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  values_to = "value",
)
```

pivot_longer()

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  values_to = "value",
)
```

a data frame to pivot

```
df %>%
  pivot_wider()
```

or

```
pivot_wider(df)
```

pivot_longer()

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  values_to = "value",
)
```

The names of the `col`umns you want to pivot into longer format

pivot_longer()

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  values_to = "value",
)
```

A string specifying the name of the **new** column to create from the data stored in the `column` names of data

Can be a character vector, creating multiple columns, if `names_sep` or `names_pattern` is provided

pivot_longer()

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  values_to = "value",
)
```

A string specifying the name of the **new** column to create from the data stored in cell values of the `col`umns

Messy data

What makes this data messy? Or, why is this data not tidy?

```
tidyr::relig_income
```

```
## # A tibble: 18 x 11
##    religion       `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k` `$50-75k` `$75-100k`
##    <chr>            <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>      <dbl>
##  1 Agnostic            27        34        60        81        76       137        122
##  2 Atheist             12        27        37        52        35        70         73
##  3 Buddhist            27        21        30        34        33        58         62
##  4 Catholic           418       617       732       670       638      1116        949
##  5 Don't know/re~      15        14        15        11        10        35         21
##  6 Evangelical P~     575       869      1064       982       881      1486        949
##  7 Hindu                1         9         7         9        11        34         47
##  8 Historically ~     228       244       236       238       197       223        131
##  9 Jehovah's Wit~      20        27        24        24        21        30         15
## 10 Jewish              19        19        25        25        30        95         69
## 11 Mainline Prot      289       495       619       655       651      1107        939
## 12 Mormon              29        40        48        51        56       112         85
## 13 Muslim               6         7         9        10         9        23         16
## 14 Orthodox            13        17        23        32        32        47         38
## 15 Other Christi~       9         7        11        13        13        14         18
```

Messy data

Column headers are values (data!), not variable names

```
tidyr::relig_income
```

```
## # A tibble: 18 x 11
##    religion        `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k` `$50-75k` `$75-100k`
##    <chr>             <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>       <dbl>
##  1 Agnostic             27        34        60        81        76       137         122
##  2 Atheist              12        27        37        52        35        70          73
##  3 Buddhist             27        21        30        34        33        58          62
##  4 Catholic            418       617       732       670       638      1116         949
##  5 Don't know/re~       15        14        15        11        10        35          21
##  6 Evangelical P~      575       869      1064       982       881      1486         949
##  7 Hindu                 1         9         7         9        11        34          47
##  8 Historically ~      228       244       236       238       197       223         131
##  9 Jehovah's Wit~       20        27        24        24        21        30          15
## 10 Jewish               19        19        25        25        30        95          69
## 11 Mainline Prot       289       495       619       655       651      1107         939
## 12 Mormon               29        40        48        51        56       112          85
## 13 Muslim                6         7         9        10         9        23          16
## 14 Orthodox             13        17        23        32        32        47          38
## 15 Other Christi~        9         7        11        13        13        14          18
```

Let's `pivot_longer()`

Necessary arguments

- `data` - of course
- `cols` – which columns you will pivot longer
- `names_to` – new variable name (in quotes) for those `cols`
- `values_to` – new variable name (in quotes) for the cell values

Let's pivot_longer()

```
relig_income %>%
 pivot_longer(
  cols = _____,
  names_to = "_____
  values_to = "____
 )
```

```
## # A tibble: 18 x 11
##    religion       `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k` `$50-75k` `$
##    <chr>            <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
##  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/re~      15        14        15        11        10        35
##  6 Evangelical P~     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 Wit~      20        27        24        24        21        30
## 10 Jewish              19        19        25        25        30        95
## 11 Mainline Prot      289       495       619       655       651      1107
## 12 Mormon              29        40        48        51        56       112
## 13 Muslim               6         7         9        10         9        23
## 14 Orthodox            13        17        23        32        32        47
## 15 Other Christi~       9         7        11        13        13        14
## 16 Other Faiths        20        33        40        46        49        63
## 17 Other World R~       5         2         3         4         2         7
```

Let's pivot_longer()

```
relig_income %>%
  pivot_longer(
    cols = -religion,
    names_to = "income",
    values_to = "frequency"
  )
```

```
## # A tibble: 180 x 3
##    religion income             frequency
##    <chr>    <chr>                  <dbl>
##  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
## # ... with 170 more rows
```

Why are tidy data useful?

When used with {dplyr}, tidy data can result in large gains in efficiency

For example, suppose we want to calculate the percent of respondents in each income category *by* religion

First let's save our pivoted data

```
relig_long <- relig_income %>%
  pivot_longer(
    cols = -religion,
    names_to = "income",
    values_to = "frequency"
  )
```

Why are tidy data useful?

When used with `{dplyr}`, tidy data can result in large gains in efficiency

For example, suppose we want to calculate the percent of respondents in each income category *by* religion

Then let's relevel the `income` variable so it is ordered in a meaningful way

```
relig_long <- relig_long %>%
  mutate(income = fct_relevel(income,
                              "Don't know/refused",
                              "<$10k",
                              "$10-20k",
                              "$20-30k",
                              "$30-40k",
                              "$40-50k",
                              "$50-75k",
                              "$75-100k",
                              "$100-150k",
                              ">150k"))
```

Why are tidy data useful?

When used with {dplyr}, tidy data can result in large gains in efficiency

For example, suppose we want to calculate the percent of respondents in each income category *by* religion

Now let's group by religion to create a pct variable that indicates the percent of respondents in each income category *within* each religion category

```
relig_long %>%
  group_by(religion) %>%
  mutate(pct = frequency/sum(frequency) * 100)
```

```
## # A tibble: 180 x 4
## # Groups:   religion [18]
##    religion income             frequency   pct
##    <chr>    <fct>                  <dbl> <dbl>
##  1 Agnostic <$10k                     27  3.27
##  2 Agnostic $10-20k                   34  4.12
##  3 Agnostic $20-30k                   60  7.26
##  4 Agnostic $30-40k                   81  9.81
##  5 Agnostic $40-50k                   76  9.20
##  6 Agnostic $50-75k                  137 16.6
##  7 Agnostic $75-100k                 122 14.8
```

Why are tidy data useful?

When used with {ggplot2}, tidy data can result in large gains in efficiency

Now we can facet by religion and see all data at once!

```
relig_long %>%
  group_by(religion) %>%
  mutate(pct = frequency/sum(frequency) * 100) %>%
  ggplot(aes(income, pct)) +
  geom_point() +
  geom_line(group = 1) +
  facet_wrap(~religion)
```

Messy data

Many variables in column names

`tidyr::who` (modified slightly)

```
## # A tibble: 7,240 x 60
##    country     iso2  iso3   year new_sp_m_014 new_sp_m_1524 new_sp_m_2534 new_sp_m_3544
##    <chr>       <chr> <chr> <int>        <int>          <int>          <int>          <int>
##  1 Afghanistan AF    AFG    1980           NA             NA             NA             NA
##  2 Afghanistan AF    AFG    1981           NA             NA             NA             NA
##  3 Afghanistan AF    AFG    1982           NA             NA             NA             NA
##  4 Afghanistan AF    AFG    1983           NA             NA             NA             NA
##  5 Afghanistan AF    AFG    1984           NA             NA             NA             NA
##  6 Afghanistan AF    AFG    1985           NA             NA             NA             NA
##  7 Afghanistan AF    AFG    1986           NA             NA             NA             NA
##  8 Afghanistan AF    AFG    1987           NA             NA             NA             NA
##  9 Afghanistan AF    AFG    1988           NA             NA             NA             NA
## 10 Afghanistan AF    AFG    1989           NA             NA             NA             NA
## # ... with 7,230 more rows, and 52 more variables: new_sp_m_4554 <int>,
## #   new_sp_m_5564 <int>, new_sp_m_65 <int>, new_sp_f_014 <int>, new_sp_f_1524 <int>,
## #   new_sp_f_2534 <int>, new_sp_f_3544 <int>, new_sp_f_4554 <int>,
## #   new_sp_f_5564 <int>, new_sp_f_65 <int>, new_sn_m_014 <int>, new_sn_m_1524 <int>,
## #   new_sn_m_2534 <int>, new_sn_m_3544 <int>, new_sn_m_4554 <int>,
```

Messy data

Many variables in column names

- Example column names

`new_sp_m_014`

`new_rel_f_65`

Messy data

Many variables in column names

- Example column names

new_sp_m_014

new_rel_f_65

That's a lot of information in the column names!

all col names start with "new_", which we don't need

followed by a diagnosis category: "sp", "*sn*", "ep", "*rel*"

then sex, which in these data is limited to only "m" *or* "*f*"

and finally an age range

Messy data

Many variables in column names

Necessary arguments

- `data` - of course
- `cols` – which columns you will pivot longer
- `names_to` – new variable name**s** (in quotes) for those `cols`
- `values_to` – new variable name (in quotes) for the cell values
- `names_separate` – how to *separate* those `cols` names

Let's apply it

```
who_r %>%
 pivot_longer(
  cols = new_sp_m_014:new_rel_f_65,
  names_to = c("new", "diagnosis", "sex", "age"),
  names_sep = "_",
  values_to = "count"
)
```

```
## # A tibble: 405,440 x 9
##    country     iso2  iso3   year new   diagnosis sex   age   count
##    <chr>       <chr> <chr> <int> <chr> <chr>     <chr> <chr> <int>
##  1 Afghanistan AF    AFG    1980 new   sp        m     014      NA
##  2 Afghanistan AF    AFG    1980 new   sp        m     1524     NA
##  3 Afghanistan AF    AFG    1980 new   sp        m     2534     NA
##  4 Afghanistan AF    AFG    1980 new   sp        m     3544     NA
##  5 Afghanistan AF    AFG    1980 new   sp        m     4554     NA
##  6 Afghanistan AF    AFG    1980 new   sp        m     5564     NA
##  7 Afghanistan AF    AFG    1980 new   sp        m     65       NA
##  8 Afghanistan AF    AFG    1980 new   sp        f     014      NA
##  9 Afghanistan AF    AFG    1980 new   sp        f     1524     NA
## 10 Afghanistan AF    AFG    1980 new   sp        f     2534     NA
## # ... with 405,430 more rows
```

Don’t get discouraged

- This is tricky!
- Takes practice
- Takes trial and error
- Takes looking at the help documentation and the vignette - a lot
- But you are rarely (never) gifted with tidy data

Declaring `cols` to `pivot_longer`

We can declare the columns to pivot other ways

The important part is just being clear which columns should be part of the gather

All of the below are equivalent

```
relig_income %>%
  pivot_longer(
    cols = -religion,
    ...
```

```
relig_income %>%
  pivot_longer(
    cols = -1,
    ...
```

```
names(relig_income) %>%
  pivot_longer(
    cols = c(`<$10k`:`Don't know/refused`),
    ...
```

```
names(who) %>%
  pivot_longer(
    cols = new_sp_m_014:new_rel_f_65,
    ...
```

```
names(who) %>%
  pivot_longer(
    cols = starts_with("new_"),
    ...
```

Let's look at some education data

Load the *exam1.csv* data from Canvas or from your cloned course repo

- You should start a new R script or Rmd

```
library(tidyverse)
library(here)
exam <- read_csv(here("data", "exam1.csv"))
```

exam1 data

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

Talk with a neighbor

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

03:00

exam1 data

What do we want?

Let's change all *item* variables into two variables: *item* and *score*

```
## # A tibble: 630 x 4
##    stu_name gender item    score
##    <chr>    <chr>  <chr>   <dbl>
##  1 Adam     M      item_1      1
##  2 Adam     M      item_2      1
##  3 Adam     M      item_3      1
##  4 Adam     M      item_4      1
##  5 Adam     M      item_5      1
##  6 Adam     M      item_6      1
##  7 Adam     M      item_7      1
##  8 Adam     M      item_8      0
##  9 Adam     M      item_9      0
## 10 Adam     M      item_10     0
## # ... with 620 more rows
```

Try to tidy!

Necessary arguments

- `data` - of course
- `cols` – which columns you will pivot longer
- `names_to` – new variable name (in quotes) for those `cols`
  - "*item*"
- `values_to` – new variable name (in quotes) for the cell values
  - "*score*"

Don't look ahead!

05:00

exam1 tidy

```
exam %>%
  pivot_longer(
    cols = starts_with("item"),
    names_to = "item",
    values_to = "score"
  )
```

```
## # A tibble: 630 x 4
##    stu_name gender item    score
##    <chr>    <chr>  <chr>   <dbl>
##  1 Adam     M      item_1      1
##  2 Adam     M      item_2      1
##  3 Adam     M      item_3      1
##  4 Adam     M      item_4      1
##  5 Adam     M      item_5      1
##  6 Adam     M      item_6      1
##  7 Adam     M      item_7      1
##  8 Adam     M      item_8      0
##  9 Adam     M      item_9      0
## 10 Adam     M      item_10     0
## # ... with 620 more rows
```

exam1 tidy

Let's go one step further

```
exam %>%
  pivot_longer(
    cols = starts_with("item"),
    names_to = "item",
    names_prefix = "item_",
    values_to = "score"
  )
```

```
## # A tibble: 630 x 4
##    stu_name gender item  score
##    <chr>    <chr>  <chr> <dbl>
##  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
```

exam1 tidy

One more step

```
exam %>%
  pivot_longer(
    cols = starts_with("item"),
    names_to = "item",
    names_prefix = "item_",
    names_transform = list(item = as.integer),
    values_to = "score"
  )
```

```
## # A tibble: 630 x 4
##    stu_name gender  item score
##    <chr>    <chr>  <int> <dbl>
##  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
```

exam1 tidy

One more step

```
exam_tidy <- exam %>%
  pivot_longer(
    cols = starts_with("item"),
    names_to = "item",
    names_prefix = "item_",
    names_transform = list(item = as.integer),
    values_to = "score"
  )
```

Why not quotes around *item* here, but quotes as it appears in `names_to`?

One more look at names_prefix

```
who_r %>%
 pivot_longer(
  cols = new_sp_m_014:new_rel_f_65,
  names_to = c("new", "diagnosis", "sex", "age"),
  names_sep = "_",
  values_to = "count"
)
```

```
## # A tibble: 405,440 x 9
##    country     iso2  iso3   year new   diagnosis sex   age   count
##    <chr>       <chr> <chr> <int> <chr> <chr>     <chr> <chr> <int>
##  1 Afghanistan AF    AFG    1980 new   sp        m     014      NA
##  2 Afghanistan AF    AFG    1980 new   sp        m     1524     NA
##  3 Afghanistan AF    AFG    1980 new   sp        m     2534     NA
##  4 Afghanistan AF    AFG    1980 new   sp        m     3544     NA
##  5 Afghanistan AF    AFG    1980 new   sp        m     4554     NA
##  6 Afghanistan AF    AFG    1980 new   sp        m     5564     NA
##  7 Afghanistan AF    AFG    1980 new   sp        m     65       NA
##  8 Afghanistan AF    AFG    1980 new   sp        f     014      NA
##  9 Afghanistan AF    AFG    1980 new   sp        f     1524     NA
## 10 Afghanistan AF    AFG    1980 new   sp        f     2534     NA
```

One more look at names_prefix

```
who_r %>%
 pivot_longer(
  cols = new_sp_m_014:new_rel_f_65,
  names_to = c("diagnosis", "sex", "age"),
  names_prefix = "new_",
  names_sep = "_",
  values_to = "count"
)
```

```
## # A tibble: 405,440 x 8
##    country     iso2  iso3   year diagnosis sex   age   count
##    <chr>       <chr> <chr> <int> <chr>     <chr> <chr> <int>
##  1 Afghanistan AF    AFG    1980 sp        m     014      NA
##  2 Afghanistan AF    AFG    1980 sp        m     1524     NA
##  3 Afghanistan AF    AFG    1980 sp        m     2534     NA
##  4 Afghanistan AF    AFG    1980 sp        m     3544     NA
##  5 Afghanistan AF    AFG    1980 sp        m     4554     NA
##  6 Afghanistan AF    AFG    1980 sp        m     5564     NA
##  7 Afghanistan AF    AFG    1980 sp        m     65       NA
##  8 Afghanistan AF    AFG    1980 sp        f     014      NA
##  9 Afghanistan AF    AFG    1980 sp        f     1524     NA
```

Why are tidy data useful?

Question: How would we calculate the proportion of students responding correctly to each item?

Why are tidy data useful?

Question: How would we calculate the proportion of students responding correctly to each item?

```
## # A tibble: 18 x 2
##     item   prop
##    <int>  <dbl>
##  1     1 1
##  2     2 1
##  3     3 1
##  4     4 0.914
##  5     5 0.886
##  6     6 0.857
##  7     7 0.886
##  8     8 0.771
##  9     9 0.857
## 10    10 0.686
## 11    11 0.343
## 12    12 0.171
## 13    13 0.2
## 14    14 0.0857
## 15    15 0.0286
## 16    16 0.0286
## 17    17 0.0286
```

Why are tidy data useful?

Question: How would we calculate the proportion of students responding correctly to each item?

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

```
## # A tibble: 18 x 2
##     item   prop
##    <int>  <dbl>
##  1     1 1
##  2     2 1
##  3     3 1
##  4     4 0.914
##  5     5 0.886
##  6     6 0.857
##  7     7 0.886
##  8     8 0.771
##  9     9 0.857
## 10    10 0.686
## 11    11 0.343
## 12    12 0.171
## 13    13 0.2
```

Why are tidy data useful?

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

Why are tidy data useful?

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

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

```
## # A tibble: 36 x 3
## # Groups:   item [18]
##     item gender  prop
##    <int> <chr>  <dbl>
##  1     1 F      1
##  2     1 M      1
##  3     2 F      1
##  4     2 M      1
##  5     3 F      1
##  6     3 M      1
##  7     4 F      0.889
##  8     4 M      0.941
##  9     5 F      0.833
## 10     5 M      0.941
## # ... with 26 more rows
```

More data

Load the *year-end_scores.csv* data from Canvas or from your cloned course repo

- Name the data *scores*

```
scores <- read_csv(here("data", "year-end_scores.csv"))
```

Talk as a class

- Are these data tidy?
    - If you wanted to summarize scores, or plot and use facet, what would you `group_by` or `facet` by?
- What do we need to do to make these tidy?
    - What `pivot_longer()` arguments do we need?

Work as a class to tidy, name new data *scores_tidy*

03:00

Let's try another

Load the *longitudinal_sim.csv* data from Canvas or the course repo

- Name the data `sim`

```
sim <- read_csv(here("data", "longitudinal_sim.csv"))
```

Talk with a neighbor

- Are these data tidy?
  - If you wanted to summarize scores, or plot and use facet, what would you `group_by` or `facet` by?
- What do we need to do to make these tidy?
  - What `pivot_longer()` arguments do we need?

Work with your neighbor to tidy, name new data *tidy_sim*

Tell me your solutions [demo a couple]

05:00

Other possible issues

The *hiv* data

```
hiv <- read_csv(here("data", "hiv.csv"))
```

Other possible issues

Let's make some manipulations (just `{dplyr}` variety `filter()` and `select()`) to get a sample of the data

```
(hiv_r <- hiv %>%
  filter(country == "France" |
           country == "South Africa" |
             country == "United States") %>%
  select(country, `1979`, `1989`, `1999`, `2009`))
```

```
## # 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
```

- Are these data tidy?
- How would you add a new variable - say, *percent HIV*?
- Discuss how you would transform these data, if at all

Tidy the data

```
(hiv_tidy <- hiv_r %>%
  pivot_longer(
    cols = -1,
    names_to = "year",
    values_to = "percentage"
  ))
```

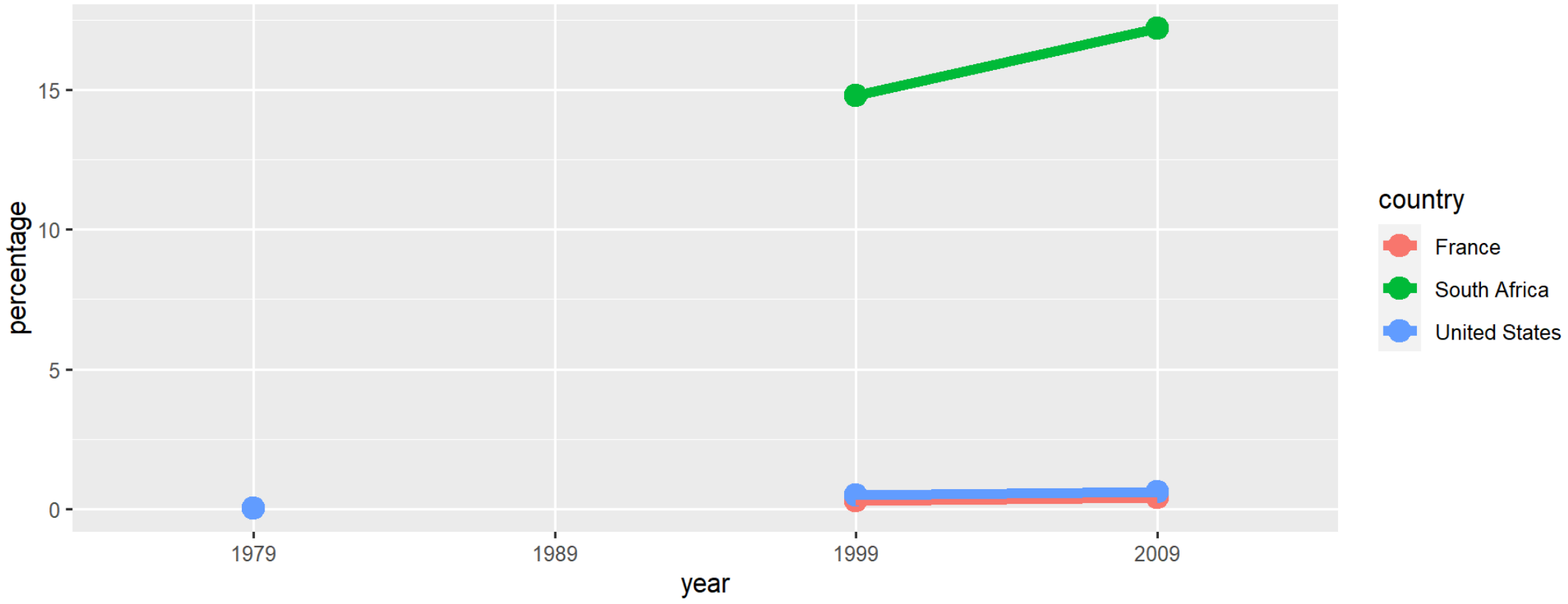
```
## # A tibble: 12 x 3
##    country       year  percentage
##    <chr>         <chr>      <dbl>
##  1 France        1979      NA
##  2 France        1989      NA
##  3 France        1999       0.3
##  4 France        2009       0.4
##  5 South Africa  1979      NA
##  6 South Africa  1989      NA
##  7 South Africa  1999      14.8
##  8 South Africa  2009      17.2
##  9 United States 1979       0.0318
## 10 United States 1989      NA
## 11 United States 1999       0.5
## 12 United States 2009       0.6
```

Let's plot it

```
hiv_tidy %>%
  ggplot(aes(year, percentage, color = country, group = country)) +
  geom_point(size = 4) +
  geom_line(size = 2)
```

Let's plot it

There is a problem with the data – what is it?



Why aren’t lines connecting for the United States?

Missing data

```
hiv_tidy
```

```
## # A tibble: 12 x 3
##    country       year  percentage
##    <chr>         <chr>      <dbl>
##  1 France        1979      NA
##  2 France        1989      NA
##  3 France        1999       0.3
##  4 France        2009       0.4
##  5 South Africa  1979      NA
##  6 South Africa  1989      NA
##  7 South Africa  1999      14.8
##  8 South Africa  2009      17.2
##  9 United States 1979       0.0318
## 10 United States 1989      NA
## 11 United States 1999       0.5
## 12 United States 2009       0.6
```

Drop NAs

```
hiv_tidy %>%
  drop_na(percentage)
```

OR

(not generally ideal unless you are **certain** you can remove these missing data)

```
hiv_tidy <- hiv_r %>%
  pivot_longer(
    cols = -1,
    names_to = "year",
    values_to = "percentage",
    names_transform = list(year = as.integer),
    values_drop_na = TRUE
  )
```

Plot again

```
hiv_tidy %>%
  ggplot(aes(year, percentage, color = country, group = country)) +
  geom_point(size = 4) +
  geom_line(size = 2)
```

pivot_wider()

Tidy data are excellent when conducting preliminary descriptive analyses, plotting data, linear models

- Occasionally, you need the data in a different format
  - For example, if you're using other packages for analysis

pivot_wider()

```
pivot_wider(
  data,
  id_cols = NULL,
  names_from = name,
  names_prefix = "",
  names_sep = "_",
  names_glue = NULL,
  names_sort = FALSE,
  names_repair = "check_unique",
  values_from = value,
  values_fill = NULL,
  values_fn = NULL,
  ...
)
```

pivot_wider()

```
pivot_wider(
  data,
  id_cols = NULL,
  names_from = name,
  names_prefix = "",
  names_sep = "_",
  names_glue = NULL,
  names_sort = FALSE,
  names_repair = "check_unique",
  values_from = value,
  values_fill = NULL,
  values_fn = NULL,
  ...
)
```

pivot_wider()

```
pivot_wider(
  data,
  names_from = name,
  values_from = value
)
```

pivot_wider()

```
pivot_wider(
  data,
  names_from = name,
  values_from = value
)
```

The names of the (tidy) column (or columns) from which to get the name of the new data columns

pivot_wider()

```
pivot_wider(
  data,
  names_from = name,
  values_from = value
)
```

The names of the (tidy) column (or columns) from which to get the the cell values

An example

fish_encounters: when fish swimming down a river are detected by automatic monitoring stations

```
tidyr::fish_encounters
```

```
## # A tibble: 114 x 3
##    fish  station  seen
##    <fct> <fct>   <int>
##  1 4842  Release     1
##  2 4842  I80_1       1
##  3 4842  Lisbon      1
##  4 4842  Rstr        1
##  5 4842  Base_TD     1
##  6 4842  BCE         1
##  7 4842  BCW         1
##  8 4842  BCE2        1
##  9 4842  BCW2        1
## 10 4842  MAE         1
## # ... with 104 more rows
```

An example

So tidy, so nice...let's muck it up

```
fish_encounters %>%
  pivot_wider(
    names_from = station,
    values_from = seen,
)
```

```
## # A tibble: 19 x 12
##    fish  Release I80_1 Lisbon  Rstr Base_TD   BCE   BCW  BCE2  BCW2   MAE   MAW
##    <fct>   <int> <int>  <int> <int>   <int> <int> <int> <int> <int> <int> <int>
##  1 4842        1     1      1     1       1     1     1     1     1     1     1
##  2 4843        1     1      1     1       1     1     1     1     1     1     1
##  3 4844        1     1      1     1       1     1     1     1     1     1     1
##  4 4845        1     1      1     1       1    NA    NA    NA    NA    NA    NA
##  5 4847        1     1      1    NA      NA    NA    NA    NA    NA    NA    NA
##  6 4848        1     1      1     1      NA    NA    NA    NA    NA    NA    NA
##  7 4849        1     1     NA    NA      NA    NA    NA    NA    NA    NA    NA
##  8 4850        1     1     NA     1       1     1     1    NA    NA    NA    NA
##  9 4851        1     1     NA    NA      NA    NA    NA    NA    NA    NA    NA
## 10 4854        1     1     NA    NA      NA    NA    NA    NA    NA    NA    NA
## 11 4855        1     1      1     1       1    NA    NA    NA    NA    NA    NA
```

An example

So tidy, so nice...let's muck it up

```
fish_encounters %>%
  pivot_wider(
    names_from = station,
    values_from = seen,
    values_fill = list(seen = 0)
)
```

```
## # A tibble: 19 x 12
##    fish  Release I80_1 Lisbon  Rstr Base_TD   BCE   BCW  BCE2  BCW2   MAE   MAW
##    <fct>   <int> <int>  <int> <int>   <int> <int> <int> <int> <int> <int> <int>
##  1 4842        1     1      1     1       1     1     1     1     1     1     1
##  2 4843        1     1      1     1       1     1     1     1     1     1     1
##  3 4844        1     1      1     1       1     1     1     1     1     1     1
##  4 4845        1     1      1     1       1     0     0     0     0     0     0
##  5 4847        1     1      1     0       0     0     0     0     0     0     0
##  6 4848        1     1      1     1       0     0     0     0     0     0     0
##  7 4849        1     1      0     0       0     0     0     0     0     0     0
##  8 4850        1     1      0     1       1     1     1     0     0     0     0
##  9 4851        1     1      0     0       0     0     0     0     0     0     0
## 10 4854        1     1      0     0       0     0     0     0     0     0     0
```

Remember scores_tidy?

```
scores_tidy
```

```
## # A tibble: 20 x 7
##       id   sex ethnicity  sped frl   content score
##    <dbl> <dbl>     <dbl> <dbl> <lgl> <chr>   <dbl>
##  1     1     0         5     0 NA    math      101
##  2     1     0         5     0 NA    read       99
##  3     2     1         2     0 NA    math       96
##  4     2     1         2     0 NA    read       NA
##  5     3     1         5     0 NA    math      131
##  6     3     1         5     0 NA    read      119
##  7     4     1         4     0 NA    math       84
##  8     4     1         4     0 NA    read       93
##  9     5     0         1     0 NA    math      110
## 10     5     0         1     0 NA    read      118
## 11     6     0         5     1 NA    math       NA
## 12     6     0         5     1 NA    read      105
## 13     7     1         5     1 NA    math       93
## 14     7     1         5     1 NA    read       99
## 15     8     0         3     0 NA    math       81
## 16     8     0         3     0 NA    read       77
```

Untidy `scores_tidy`

```
scores_tidy %>%
  pivot_wider(
    names_from = content,
    values_from = score
  )
```

```
## # A tibble: 10 x 7
##       id   sex ethnicity  sped frl    math  read
##    <dbl> <dbl>     <dbl> <dbl> <lgl> <dbl> <dbl>
##  1     1     0         5     0 NA      101    99
##  2     2     1         2     0 NA       96    NA
##  3     3     1         5     0 NA      131   119
##  4     4     1         4     0 NA       84    93
##  5     5     0         1     0 NA      110   118
##  6     6     0         5     1 NA       NA   105
##  7     7     1         5     1 NA       93    99
##  8     8     0         3     0 NA       81    77
##  9     9     1         2     0 NA      129   135
## 10    10     0         1     0 NA      117   122
```

Same as the original data

```
read_csv(here("data", "year-end_scores.csv"))
```

```
## # A tibble: 10 x 7
##       id   sex ethnicity  sped frl    math  read
##    <dbl> <dbl>     <dbl> <dbl> <lgl> <dbl> <dbl>
##  1     1     0         5     0 NA      101    99
##  2     2     1         2     0 NA       96    NA
##  3     3     1         5     0 NA      131   119
##  4     4     1         4     0 NA       84    93
##  5     5     0         1     0 NA      110   118
##  6     6     0         5     1 NA       NA   105
##  7     7     1         5     1 NA       93    99
##  8     8     0         3     0 NA       81    77
##  9     9     1         2     0 NA      129   135
## 10    10     0         1     0 NA      117   122
```

Let's do this together

Use `pivot_wider()` to change *sim_tidy* back to *sim*

```
sim_tidy
```

```
## # A tibble: 40 x 4
##      sid wave  content score
##    <dbl> <chr> <chr>   <dbl>
##  1     1 1     math       95
##  2     1 2     math       98
##  3     1 3     math      102
##  4     1 4     math      105
##  5     1 1     rdg        96
##  6     1 2     rdg        98
##  7     1 3     rdg       101
##  8     1 4     rdg       103
##  9     2 1     math      101
## 10     2 2     math      103
## # ... with 30 more rows
```

```
sim
```

```
## # A tibble: 5 x 9
##     sid wave_1_math wave_2_math wave_3_math wave_4
##   <dbl>       <dbl>       <dbl>       <dbl>
## 1     1          95          98         102
## 2     2         101         103         107
## 3     3          99         103         106
## 4     4         109         111         115
## 5     5         101         104         107
## # ... with 2 more variables: wave_3_rdg <dbl>, wav
```

Longer then wider

Some problems can't be solved by pivoting in a single direction

Here is an example of how you might combine `pivot_longer()` and `pivot_wider()` to solve more complex problems

```
sim2 <- read_csv(here("data", "longitudinal_sim2.csv"))
head(sim2)
```

```
## # A tibble: 6 x 19
##     SID male_g6 male_g7 male_g8 ell_g6 ell_g7 ell_g8 sped_g6 sped_g7 sped_g8
##   <dbl>   <dbl>   <dbl>   <dbl>  <dbl>  <dbl>  <dbl>   <dbl>   <dbl>   <dbl>
## 1     1       1       1       1      0      0      0       0       0       0
## 2     2       0       0       0      0      0      0       0       0       0
## 3     3       1       1       1      0      0      0       0       0       0
## 4     4       1       1       1      0      0      0       0       0       0
## 5     5       0       0       0      0      0      0       1       1       1
## 6     6       0       0       0      0      0      0       0       0       0
## # ... with 9 more variables: pullouts_g6 <dbl>, pullouts_g7 <dbl>, pullouts_g8 <dbl>,
## #   disability_g6 <chr>, disability_g7 <chr>, disability_g8 <chr>, score_g6 <dbl>,
## #   score_g7 <dbl>, score_g8 <dbl>
```

[View() data]

First `pivot_longer()`

```
sim2 %>%
  pivot_longer(
    cols = -1,
    names_to = c("variable", "grade"),
    values_to = "values",
    names_sep = "_")
```

```
## Error: Can't combine `male_g6` <double> and `disability_g6` <character>.
```

First pivot_longer()

```
sim2 %>%
  pivot_longer(
    cols = -1,
    names_to = c("variable", "grade"),
    values_to = "values",
    names_sep = "_",
#to combine numeric & character
    values_transform = list(values = as.character))
```

```
## # A tibble: 1,800 x 4
##       SID variable grade values
##     <dbl> <chr>    <chr> <chr>
##  1      1 male     g6    1
##  2      1 male     g7    1
##  3      1 male     g8    1
##  4      1 ell      g6    0
##  5      1 ell      g7    0
##  6      1 ell      g8    0
##  7      1 sped     g6    0
##  8      1 sped     g7    0
##  9      1 sped     g8    0
```

Let's refine a column using `mutate()`

```
sim2 %>%
  pivot_longer(
    cols = -1,
    names_to = c("variable", "grade"),
    values_to = "values",
    names_sep = "_",
    values_transform = list(values = as.character)) %>%
  mutate(grade = parse_number(grade))
```

```
## # A tibble: 1,800 x 4
##       SID variable grade values
##     <dbl> <chr>    <dbl> <chr>
##  1      1 male         6 1
##  2      1 male         7 1
##  3      1 male         8 1
##  4      1 ell          6 0
##  5      1 ell          7 0
##  6      1 ell          8 0
##  7      1 sped         6 0
##  8      1 sped         7 0
##  9      1 sped         8 0
```

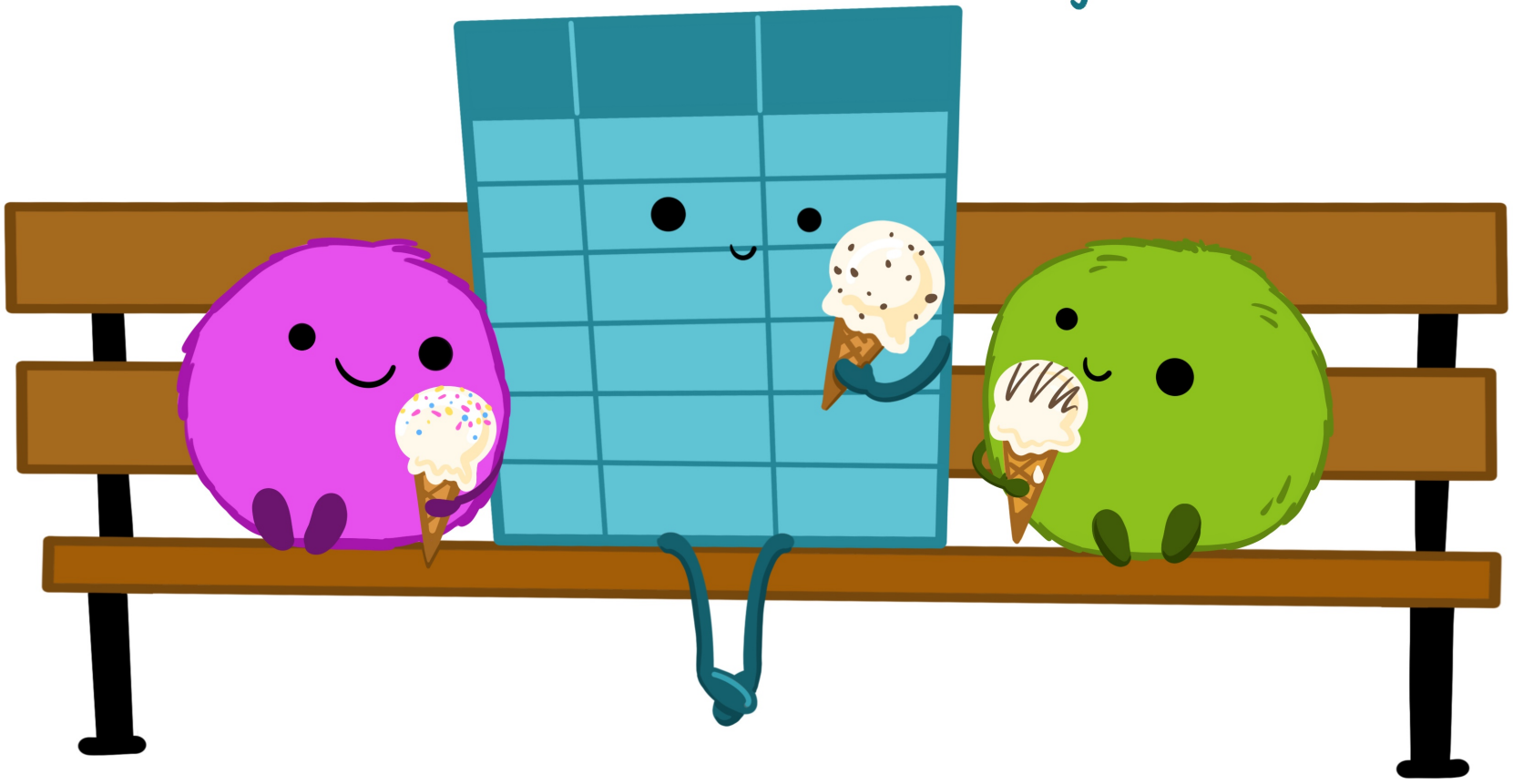
Then pivot_wider()

```
sim2 %>%
  pivot_longer(
    cols = -1,
    names_to = c("variable", "grade"),
    values_to = "values",
    names_sep = "_",
    values_transform = list(values = as.character)) %>%
  mutate(grade = parse_number(grade)) %>%
  pivot_wider(
    names_from = variable,
    values_from = values
  )
```

```
## # A tibble: 300 x 8
##       SID grade male  ell   sped  pullouts disability score
##     <dbl> <dbl> <chr> <chr> <chr> <chr>    <chr>      <chr>
##  1     1     6 1     0     0     0        none       204.711224219096
##  2     1     7 1     0     0     0        none       207.641655929138
##  3     1     8 1     0     0     0        none       211.468672710232
##  4     2     6 0     0     0     1        none       212.343498534668
##  5     2     7 0     0     0     1        none       217.565910013073
```

```
## # A tibble: 300 x 8
##      SID grade male  ell   sped  pullouts disability score
##    <dbl> <dbl> <chr> <chr> <chr> <chr>    <chr>      <chr>
##  1     1     6 1     0     0     0        none       204.711224219096
##  2     1     7 1     0     0     0        none       207.641655929138
##  3     1     8 1     0     0     0        none       211.468672710232
##  4     2     6 0     0     0     1        none       212.343498534668
##  5     2     7 0     0     0     1        none       217.565910013073
##  6     2     8 0     0     0     1        none       227.036910872597
##  7     3     6 1     0     0     0        none       215.436668173264
##  8     3     7 1     0     0     0        none       217.029498660224
##  9     3     8 1     0     0     0        none       222.232373013961
## 10     4     6 1     0     0     0        none       219.182457533654
## # ... with 290 more rows
```

make friends with tidy data.



Next time

Before next class

- Final Project
  - Final Project: Draft Data Script
  - *for sure this time*
- Reading
  - R4DS 15
- Homework
  - **Homework 9**