Pivoting and Measuring Confidence

Week 8

Packages needed and a Note about Icons

Please load up the following packages. Remember to first install the ones you don't have.

```
library(tidyverse)
library(mosaic)
library(ggplot2movies)
```

You may come across the following icons. The table below lists what each means.

Icon	Description
	Indicates that an example continues on the following slide.
	Indicates that a section using common syntax has ended.
	Indicates that there is an active hyperlink on the slide.
	Indicates that a section covering a concept has ended.

What is a confidence interval?

A *confidence interval* (CI) gives a range of possible values for a parameter. It depends on a specified *confidence level* with

- higher confidence levels corresponding to wider confidence intervals
- lower confidence levels corresponding to narrower confidence intervals.

The most common confidence levels include 90%, 95%, and 99%.

Problems with how confidence intervals are taught

You were just taught about the confidence interval in an bad way!

- Finding confidence intervals for some mean is to first assume a normal curve for a population and then magic
- But assuming normality is a BIG assumption!

Bootstrapping

Hypothesis testing

> - We simply want to if our H_0 or H_1 is correct.
> - First step in being able to generalize

Typically we have a sample of a population's data so we can

1. take repeated samples from a sample data of size whatever
2. calculate the mean for each of these samples
3. created a new distribution of these means
4. estimate the population distribution

aka *bootstrapping*

5. calculate the confidence interval (CI)

ggplot2movies

We'll look at CIs, but first let's look at the `ggplot2movies` data set...

```
head(movies)
```

```
## # A tibble: 3 × 4
##   Decision Delicious  Disgusting Totals
##   <chr>    <chr>      <chr>      <chr>
## 1 Yes      40.00% (4) 43.75% (7) 42.30% (11)
## 2 No       60.00% (6) 56.25% (9) 57.69% (15)
## 3 <i>N</i> (10)       (16)       (26)
```

...its size...

```
dim(movies)
```

```
## [1] 3 4
```

That's 58,788 rows by 24 columns!

... and the names of its columns.

```
names(movies)
```

```
## [1] "Decision"   "Delicious"  "Disgusting" "Totals"
```

You can see more about the functionality by looking at its documentation. For now, here's what the variables mean:

- **title**. Title of the movie.
- **year**. Year of release.
- **budget**. Total budget (if known) in US dollars
- **length**. Length in minutes.
- **rating**. Average IMDB user rating.
- **votes**. Number of IMDB users who rated this movie.
- **r1-10**. Multiplying by ten gives percentile (to nearest 10%) of users who rated this movie a 1.
- **mpaa**. MPAA rating.
- **Action**, **Animation**, **Comedy**, **Drama**, **Documentary**, **Romance**, **Short**. Binary variables representing if movie was classified as belonging to that genre.

```
ggplot2movies::movies %>%
  select(Action, Animation, Comedy,
         Drama, Documentary, Romance,
         Short) %>%
  pivot_longer(
    everything(),
    names_to = "genre"
  )
```

```
## # A tibble: 411,516 × 2
##    genre       value
##    <chr>       <int>
##  1 Action          0
##  2 Animation       0
##  3 Comedy          1
##  4 Drama           1
##  5 Documentary     0
##  6 Romance         0
##  7 Short           0
##  8 Action          0
##  9 Animation       0
## 10 Comedy          1
## # … with 411,506 more rows
```

In instances where we have to go from a

- long to wide data set, we'd use a command called `pivot_wider`
- wide to long data set, we use a command called `pivot_longer`

For more information, take a look at this fantastic overview courtesy of R-Ladies Sydney. For an advanced walkthrough, the Data Wrangling site over at Stanford is a great resource.

pivot_longer

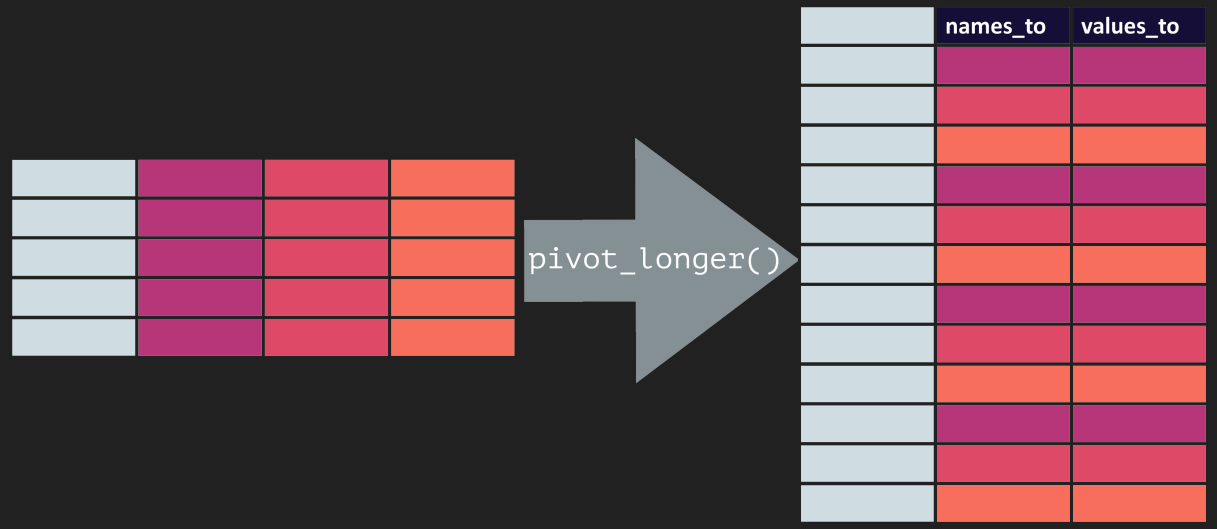
It is pretty rare that at this stage in your academic development that you need to go from long to wide so we'll be concentrating on the converse with `pivot_longer`.

OK let's begin!

The original graphics here were created by RStudio's Allison Hill. I have amended them for aesthetic purposes

An overview of `pivot_longer`

We'll concentrate one two options in `pivot_longer`: `names_to` and `values_to`.



Remember you can always run `?` in front of any command in the Console to get more information about it. For `pivot_longer`, we would simply type in

```
?pivot_longer
```

to see other options.

If you want to follow along with the fake data set we'll be using, run the following command to build the tibble


```
juniors_multiple <-
  tribble(
  ~ "baker", ~"cinnamon_1", ~"cardamom_2", ~"nutmeg_3",
  "Emma", 1L,   0L, 1L,
  "Harry", 1L,   1L, 1L,
  "Ruby", 1L,   0L, 1L,
  "Zainab", 0L, NA, 0L
)
```

and check it just to make sure

```
juniors_multiple
```

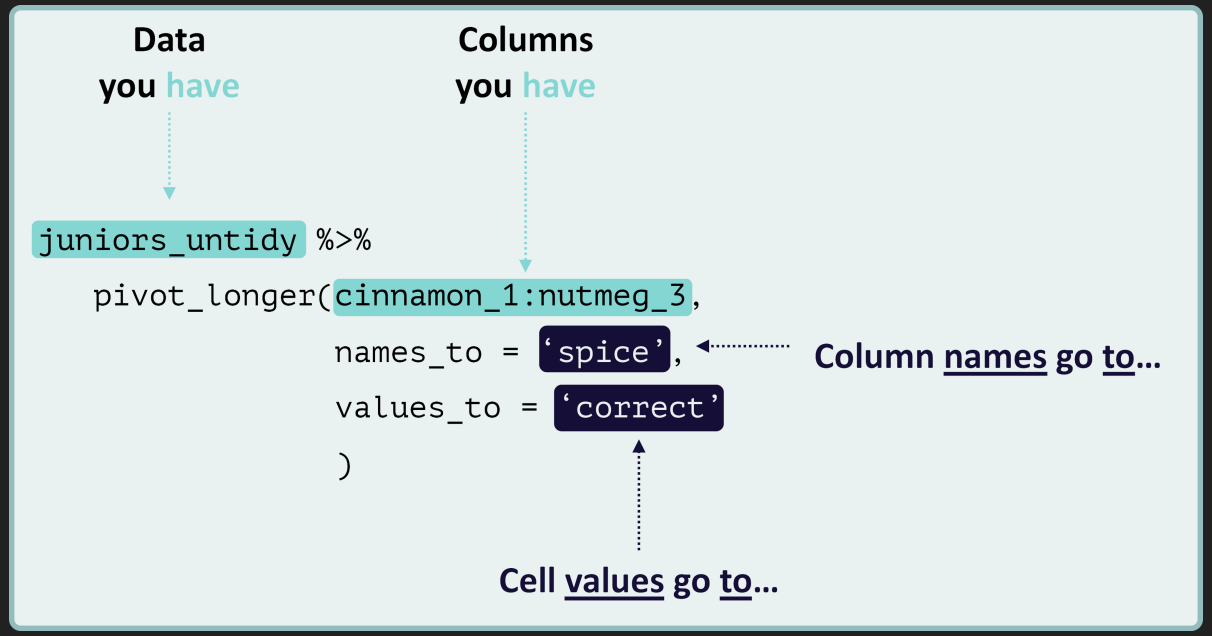
```
## # A tibble: 4 × 4
##   baker  cinnamon_1 cardamom_2 nutmeg_3
##   <chr>       <int>      <int>    <int>
## 1 Emma            1          0        1
## 2 Harry           1          1        1
## 3 Ruby            1          0        1
## 4 Zainab          0         NA        0
```

Looks good! Let's convert this!

To remind you of what the `juniors_multiple` data frame looks like, we have

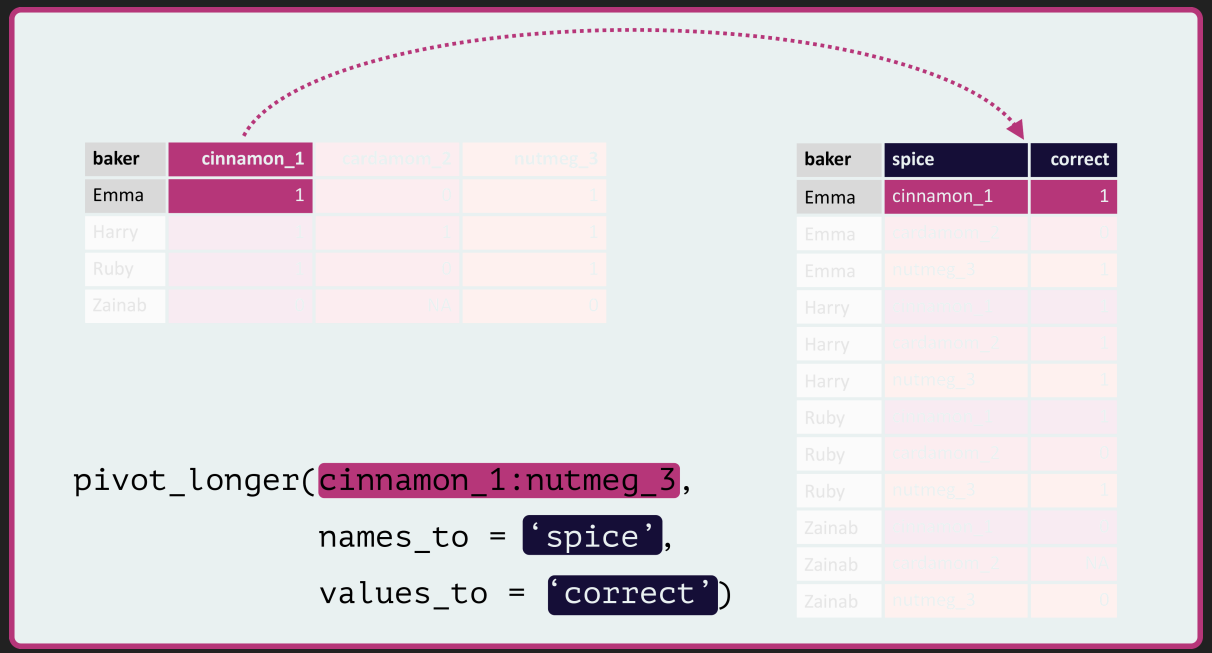
baker	cinnamon_1	cardamom_2	nutmeg_3
Emma	1	0	1
Harry	1	1	1
Ruby	1	0	1
Zainab	0	NA	0

We can assign names to the eventual columns using `names_to` and `values_to`.



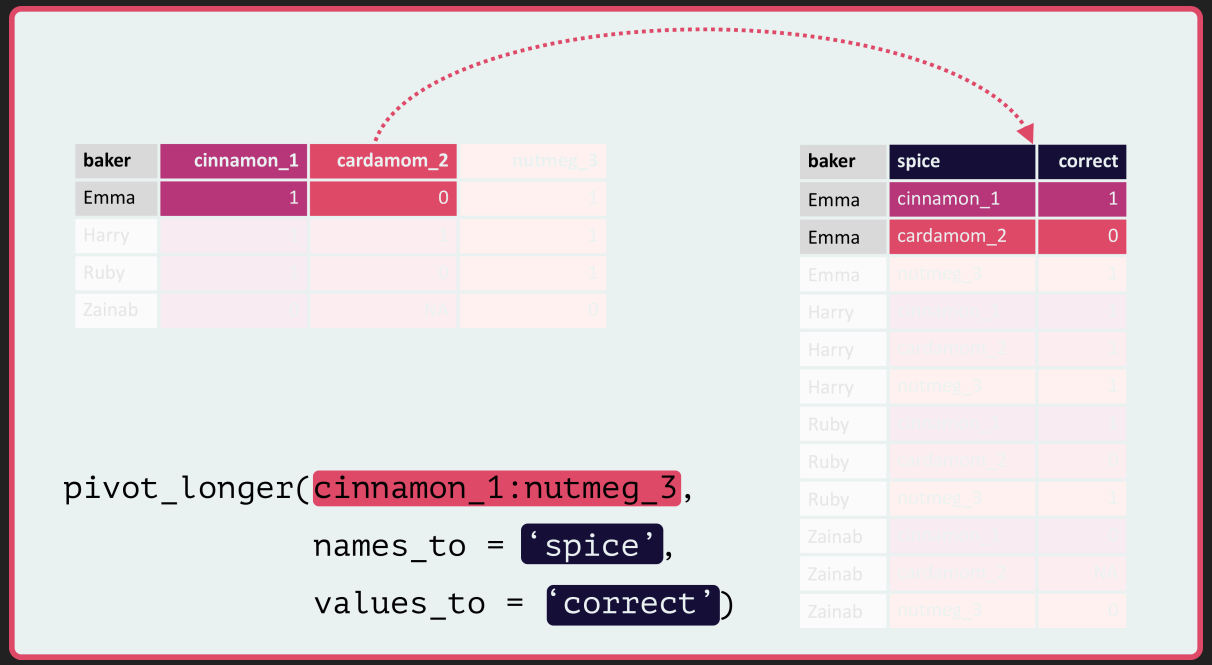
baker	cinnamon_1	cardamom_2	nutmeg_3
Emma	1	0	1
Harry	1	1	1
Ruby	1	0	1
Zainab	0	NA	0

We can assign names to the eventual columns using `names_to` and `values_to`.

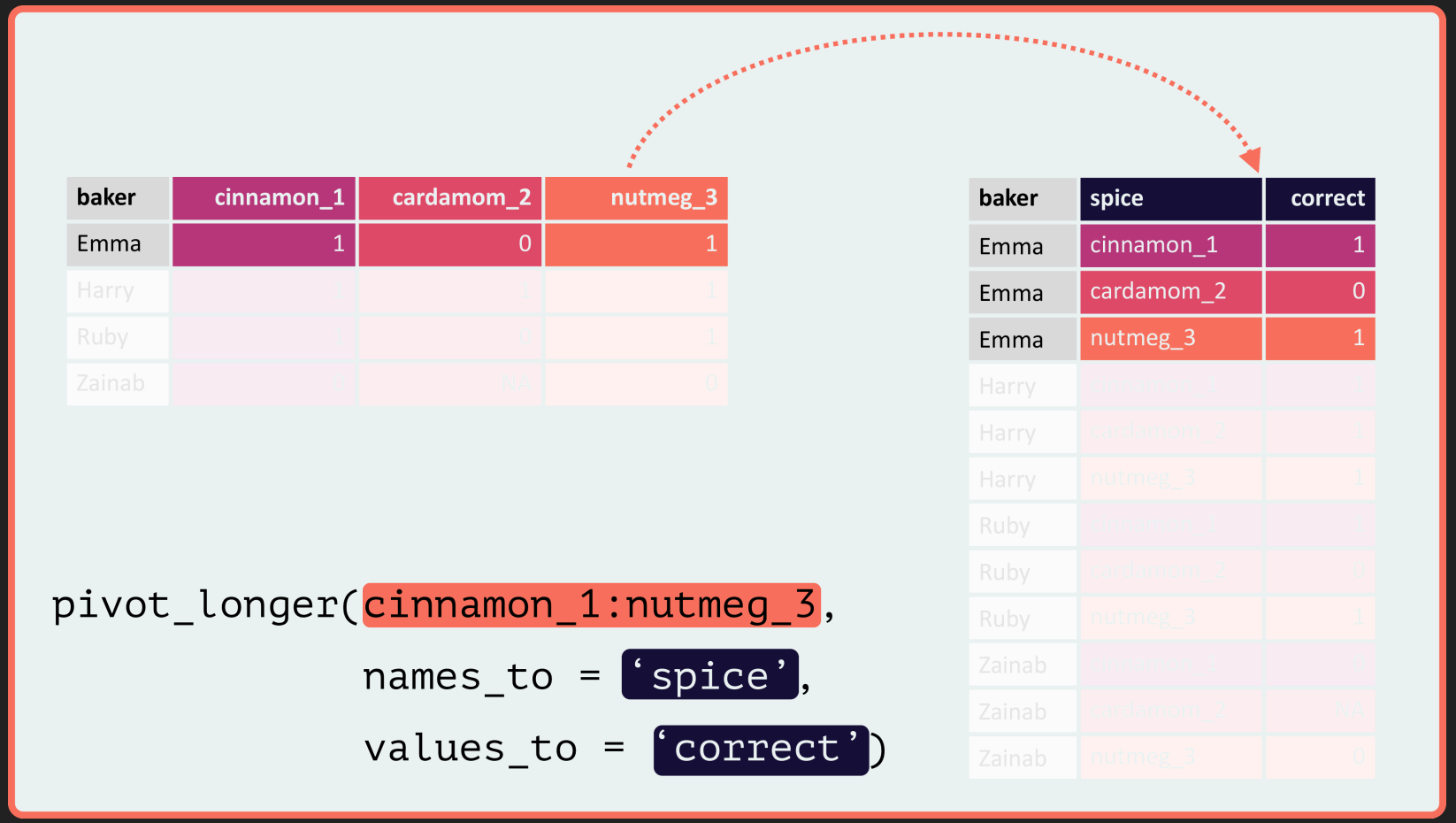


baker	cinnamon_1	cardamom_2	nutmeg_3
Emma	1	0	1
Harry	1	1	1
Ruby	1	0	1
Zainab	0	NA	0

Here you can see the first column `cinnamon_1` and its value `1` associated with the first row `Emma` becomes our first two values under the two columns `spice` and `correct` for our pivoted data frame.



This pattern continues until a whole row is used up.

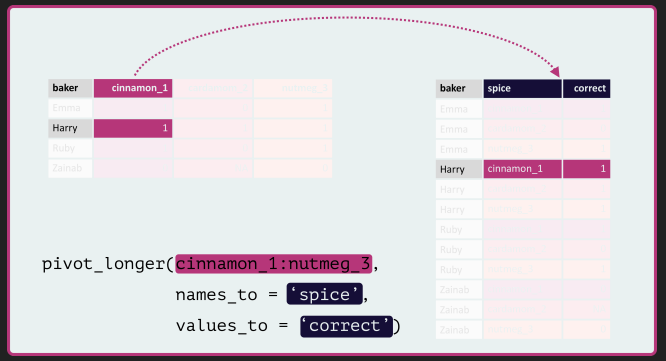


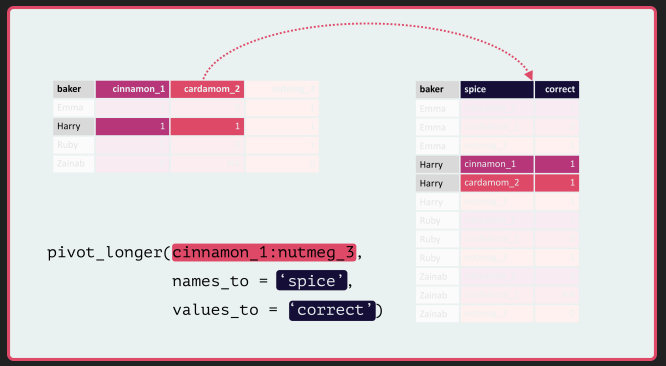
baker	cinnamon_1	cardamom_2	nutmeg_3
Emma	1	0	1
Harry	1	1	1
Ruby	1	0	1
Zainab	0	NA	0

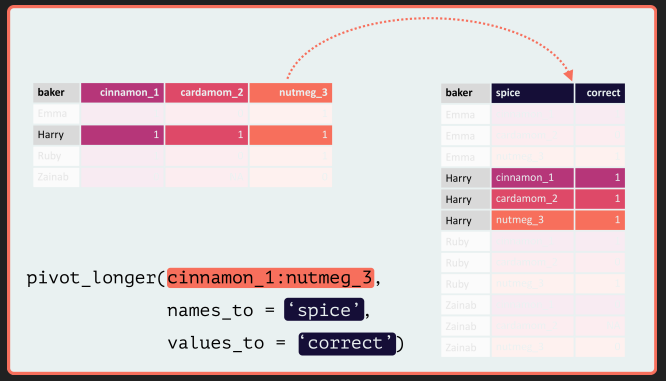
baker	spice	correct
Emma	cinnamon_1	1
Emma	cardamom_2	0
Emma	nutmeg_3	1
Harry	cinnamon_1	1
Harry	cardamom_2	1
Harry	nutmeg_3	1
Ruby	cinnamon_1	1
Ruby	cardamom_2	0
Ruby	nutmeg_3	1
Zainab	cinnamon_1	0
Zainab	cardamom_2	NA
Zainab	nutmeg_3	0

```
pivot_longer(cinnamon_1:nutmeg_3,
             names_to = 'spice',
             values_to = 'correct')
```

Then it repeats for the next row of values...

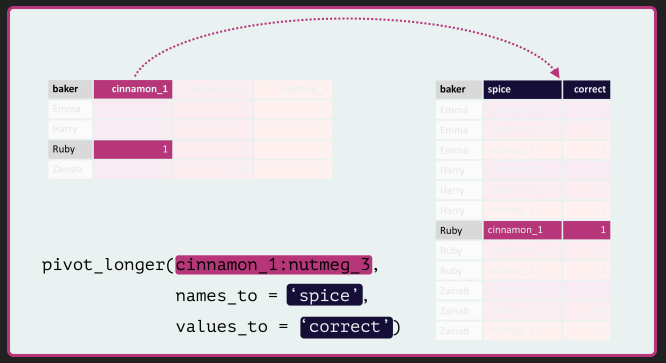


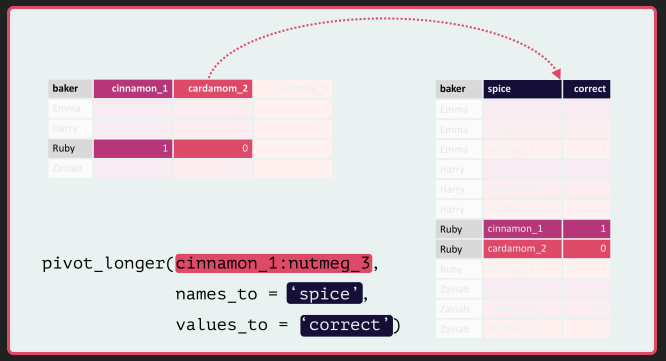


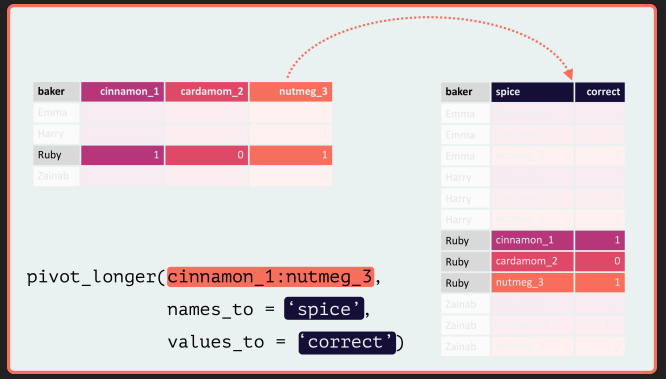


...and so forth...

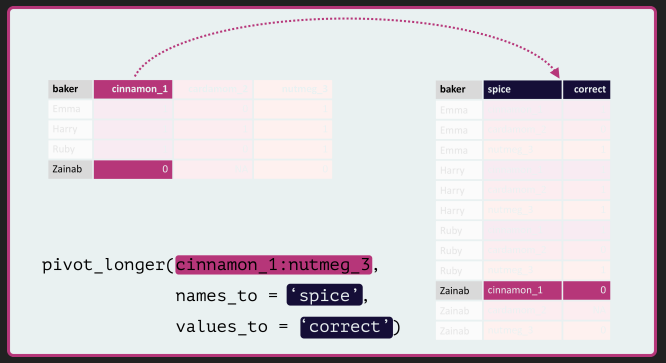

Statistical Methods I

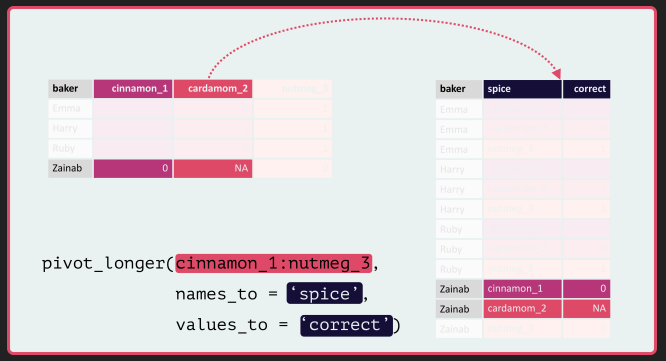

baker cinnamon_1
Ruby 1
baker spice correct
Ruby cinnamon_1 1
pivot_longer(cinnamon_1:nutmeg_3,
names_to = 'spice',
values_to = 'correct')

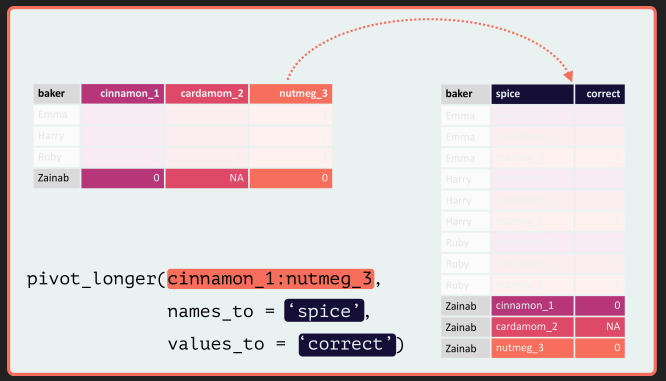

baker cinnamon_1 cardamom_2
Ruby 1 0
baker spice correct
Ruby cinnamon_1 1
Ruby cardamom_2 0
pivot_longer(cinnamon_1:nutmeg_3,
names_to = 'spice',
values_to = 'correct')


baker cinnamon_1 cardamom_2 nutmeg_3
Ruby 1 0 1
baker spice correct
Ruby cinnamon_1 1
Ruby cardamom_2 0
Ruby nutmeg_3 1
pivot_longer(cinnamon_1:nutmeg_3,
names_to = 'spice',
values_to = 'correct')

...until we run out of rows...







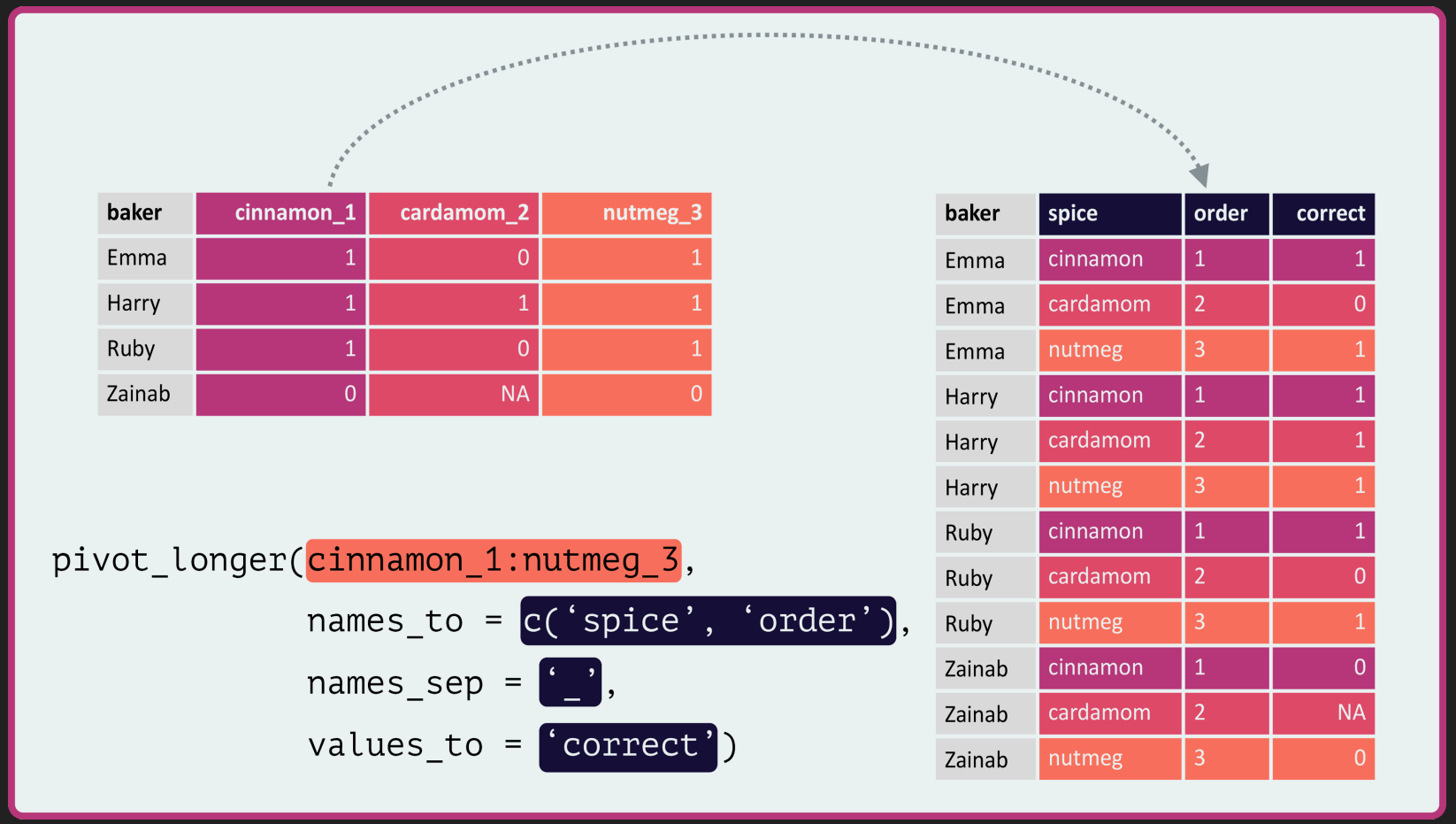
...and get the final table of pivoted values.

baker	cinnamon_1	cardamom_2	nutmeg_3
Emma	1	0	1
Harry	1	1	1
Ruby	1	0	1
Zainab	0	NA	0

```
pivot_longer(cinnamon_1:nutmeg_3,
             names_to = 'spice',
             values_to = 'correct')
```

baker	spice	correct
Emma	cinnamon_1	1
Emma	cardamom_2	0
Emma	nutmeg_3	1
Harry	cinnamon_1	1
Harry	cardamom_2	1
Harry	nutmeg_3	1
Ruby	cinnamon_1	1
Ruby	cardamom_2	0
Ruby	nutmeg_3	1
Zainab	cinnamon_1	0
Zainab	cardamom_2	NA
Zainab	nutmeg_3	0

We can even amend the current command to include things like `order`!



baker	cinnamon_1	cardamom_2	nutmeg_3
Emma	1	0	1
Harry	1	1	1
Ruby	1	0	1
Zainab	0	NA	0

```
pivot_longer(cinnamon_1:nutmeg_3,
             names_to = c('spice', 'order'),
             names_sep = '_',
             values_to = 'correct')
```

baker	spice	order	correct
Emma	cinnamon	1	1
Emma	cardamom	2	0
Emma	nutmeg	3	1
Harry	cinnamon	1	1
Harry	cardamom	2	1
Harry	nutmeg	3	1
Ruby	cinnamon	1	1
Ruby	cardamom	2	0
Ruby	nutmeg	3	1
Zainab	cinnamon	1	0
Zainab	cardamom	2	NA
Zainab	nutmeg	3	0

Shortcut

Rather than accounting for every column, you can just tell R not to account for columns

```
juniors_multiple %>%
  pivot_longer(-baker,
               names_to = c('spice', 'order'),
               names_sep = '_',
               values_to = 'correct')
```

```
## # A tibble: 12 × 4
##    baker  spice    order correct
##    <chr>  <chr>    <chr>   <int>
##  1 Emma   cinnamon 1           1
##  2 Emma   cardamom 2           0
##  3 Emma   nutmeg   3           1
##  4 Harry  cinnamon 1           1
##  5 Harry  cardamom 2           1
##  6 Harry  nutmeg   3           1
##  7 Ruby   cinnamon 1           1
##  8 Ruby   cardamom 2           0
##  9 Ruby   nutmeg   3           1
## 10 Zainab cinnamon 1           0
## 11 Zainab cardamom 2          NA
## 12 Zainab nutmeg   3           0
```

Single column types

`pivot_wider` is great for columns of the same type. For example, if we run

```
glimpse(juniors_multiple)
```

```
## Rows: 4
## Columns: 4
## $ baker      <chr> "Emma", "Harry", "Ruby", "Zainab"
## $ cinnamon_1 <int> 1, 1, 1, 0
## $ cardamom_2 <int> 0, 1, 0, NA
## $ nutmeg_3   <int> 1, 1, 1, 0
```

all we have are integers...

Multiple column types

... but for the following

```
juniors_multiple_full <-
  tribble(
  ~ "baker", ~"score_1", ~"score_2", ~"score_3",
  ~ "guess_1", ~"guess_2", ~"guess_3",
    "Emma", 1L, 0L, 1L, "cinnamon", "cloves", "nutmeg",
    "Harry", 1L,  1L, 1L, "cinnamon", "cardamom", "nutmeg",
    "Ruby", 1L, 0L, 1L, "cinnamon", "cumin", "nutmeg",
    "Zainab", 0L, NA, 0L, "cardamom", NA_character_, "cinnamon"
  )
```

```
juniors_multiple_full
```

```
## # A tibble: 4 × 7
##   baker  score_1 score_2 score_3 guess_1  guess_2  guess_3
##   <chr>    <int>   <int>   <int> <chr>    <chr>    <chr>
## 1 Emma         1       0       1 cinnamon cloves   nutmeg
## 2 Harry        1       1       1 cinnamon cardamom nutmeg
## 3 Ruby         1       0       1 cinnamon cumin    nutmeg
## 4 Zainab       0      NA       0 cardamom <NA>     cinnamon
```

```
glimpse(juniors_multiple_full)
```

```
## Rows: 4
## Columns: 7
## $ baker   <chr> "Emma", "Harry", "Ruby", "Zainab"
## $ score_1 <int> 1, 1, 1, 0
## $ score_2 <int> 0, 1, 0, NA
## $ score_3 <int> 1, 1, 1, 0
## $ guess_1 <chr> "cinnamon", "cinnamon", "cinnamon", "cardamom"
## $ guess_2 <chr> "cloves", "cardamom", "cumin", NA
## $ guess_3 <chr> "nutmeg", "nutmeg", "nutmeg", "cinnamon"
```

...we have both character and numeric vectors.

Try running the following


```
juniors_multiple_full %>%
  pivot_longer(score_1:guess_3,
               names_to = c('score', 'guess'),
               names_sep = "_",
               values_to = 'correct')
```

Do you get `Error: Can't combine score_1 <integer> and guess_1 <character>.` ? So what can you do?

Well since computers are stupid, you have to tell R what to look for.

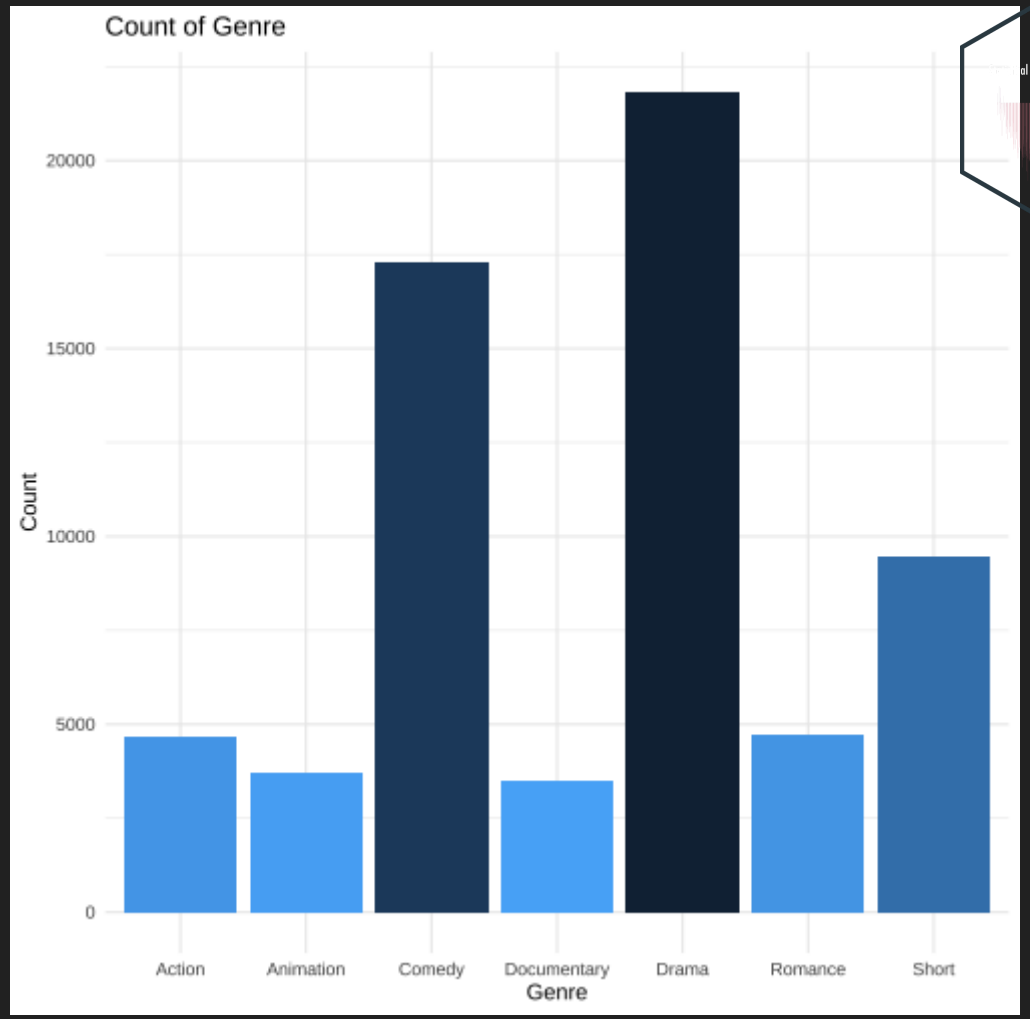
```r
juniors_multiple_full %>%
# Don't do anything with the baker column
    pivot_longer(-baker,
# Treat all columns the same and order them
                 names_to = c(".value", "order"),
# Control how the column names are broken up
                 names_sep = "_")
```

```
## # A tibble: 12 × 4
##    baker  order score guess
##    <chr>  <chr> <int> <chr>
##  1 Emma   1         1 cinnamon
##  2 Emma   2         0 cloves
##  3 Emma   3         1 nutmeg
##  4 Harry  1         1 cinnamon
##  5 Harry  2         1 cardamom
##  6 Harry  3         1 nutmeg
##  7 Ruby   1         1 cinnamon
##  8 Ruby   2         0 cumin
##  9 Ruby   3         1 nutmeg
## 10 Zainab 1         0 cardamom
## 11 Zainab 2        NA <NA>
## 12 Zainab 3         0 cinnamon
```

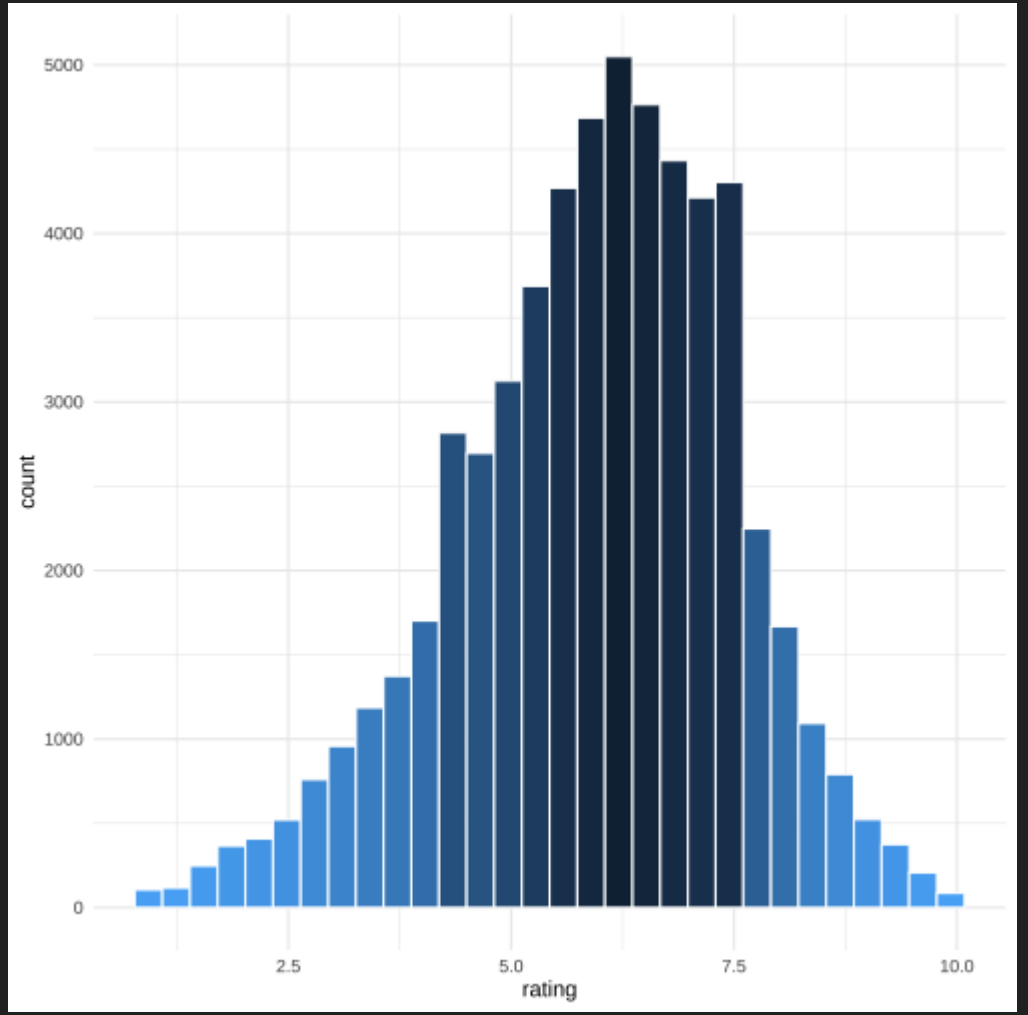
```r
ggplot2movies::movies %>%
  select(Action, Animation, Comedy,
         Drama, Documentary, Romance,
         Short) %>%
  pivot_longer(everything(),
               names_to = "genre") %>%
  group_by(genre) %>%
  dplyr::tally(value)
```

```
## # A tibble: 7 × 2
##   genre           n
##   <chr>       <int>
## 1 Action       4688
## 2 Animation    3690
## 3 Comedy      17271
## 4 Documentary  3472
## 5 Drama       21811
## 6 Romance      4744
## 7 Short        9458
```

```r
ggplot(movies_by_genre,
       aes(x = genre,
           y = n,
           fill = -n)) +
  geom_bar(stat='identity',
           show.legend = FALSE) +
  labs(title = "Count of Genre",
       x = "Genre",
       y = "Count") +
  theme_minimal()
```



```r
ggplot2movies::movies %>%
  ggplot(aes(x = rating)) +
  geom_histogram(aes(fill = -..count..),
                 color = "white",
                 bins = 30,
                 show.legend = FALSE) +
  theme_minimal()
```



```r
pop <-
  ggplot2movies::movies %>%
  ggplot(aes(x = rating)) +
  geom_histogram(aes(fill = -..count..),
                 color = "white",
                 bins = 30,
                 show.legend = FALSE) +
  theme_minimal() +
  ggtitle("Population")
```

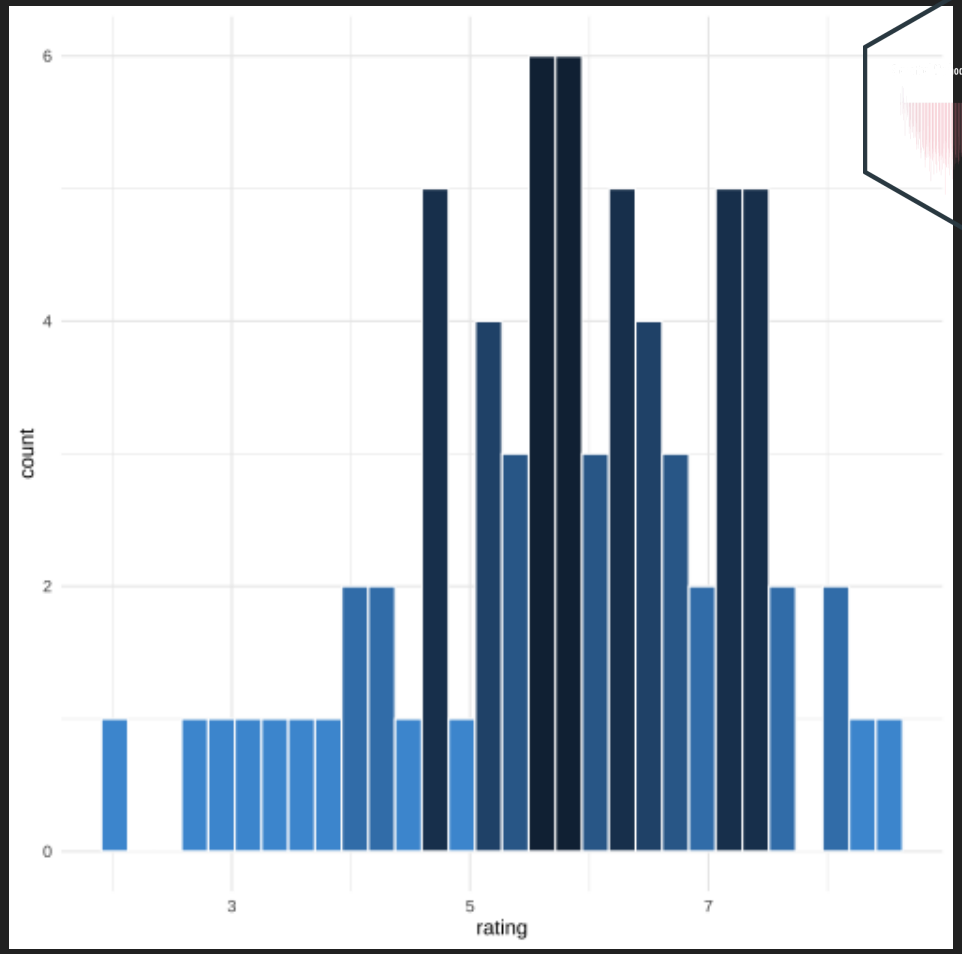
Purpose

We would like to produce a confidence interval for the population mean rating. Let's first pretend we had to take a sample of $n = 70$ from the $N = 58788$ movies. To do this, we'll use the `sample_n` command from the `dplyr` package.

```
set.seed(999) # Random number generator

movies_sample <-
  ggplot2movies::movies %>%
  sample_n(70)
```

```
ggplot(movies_sample,
       aes(x = rating)) +
       geom_histogram(aes(fill = -..count..),
                      color = "white",
                      bins = 30,
                      show.legend = FALSE) +
       theme_minimal()
```



Population Estimation

- The histogram is an estimate of our population distribution histogram

To estimate a range of values, we use the mean of the sample

```
 (movies_sample_mean <-
    movies_sample %>%
    summarize(mean = mean(rating)))
```

```
## # A tibble: 1 × 1
##    mean
##   <dbl>
## 1  5.81
```

- This is a single estimation.
- Earlier you sampled from the population - aka **sampling with replacement**.

A good way to to this is to add parentheses around a variable

```
resample(movies_sample) %>%
  arrange(orig.id) %>%
  summarize(mean = mean(rating))
```

```
## # A tibble: 1 × 1
##    mean
##   <dbl>
## 1  6.09
```

This is only one sample mean!

```
do(10) *
  (resample(movies_sample) %>%
     summarize(mean = mean(rating)))
```

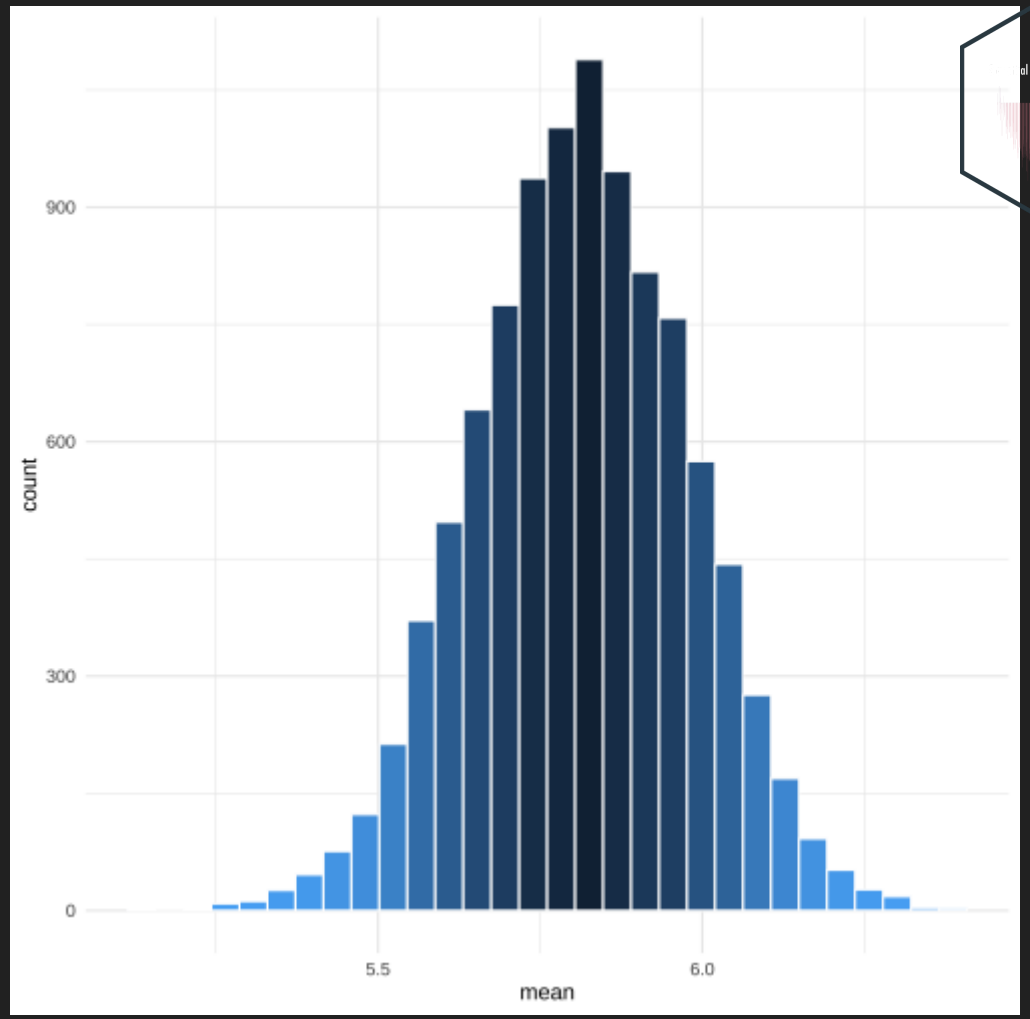
```
##        mean
## 1  5.537143
## 2  5.815714
## 3  5.804286
## 4  5.837143
## 5  5.920000
## 6  5.850000
## 7  5.628571
## 8  5.955714
## 9  5.755714
## 10 5.778571
```

But a sample of 10 is so lame. Let's think big and try 10000!

```
not_lame <-
  do(10000) * summarize(resample(movies_sample),
                        mean = mean(rating))
```

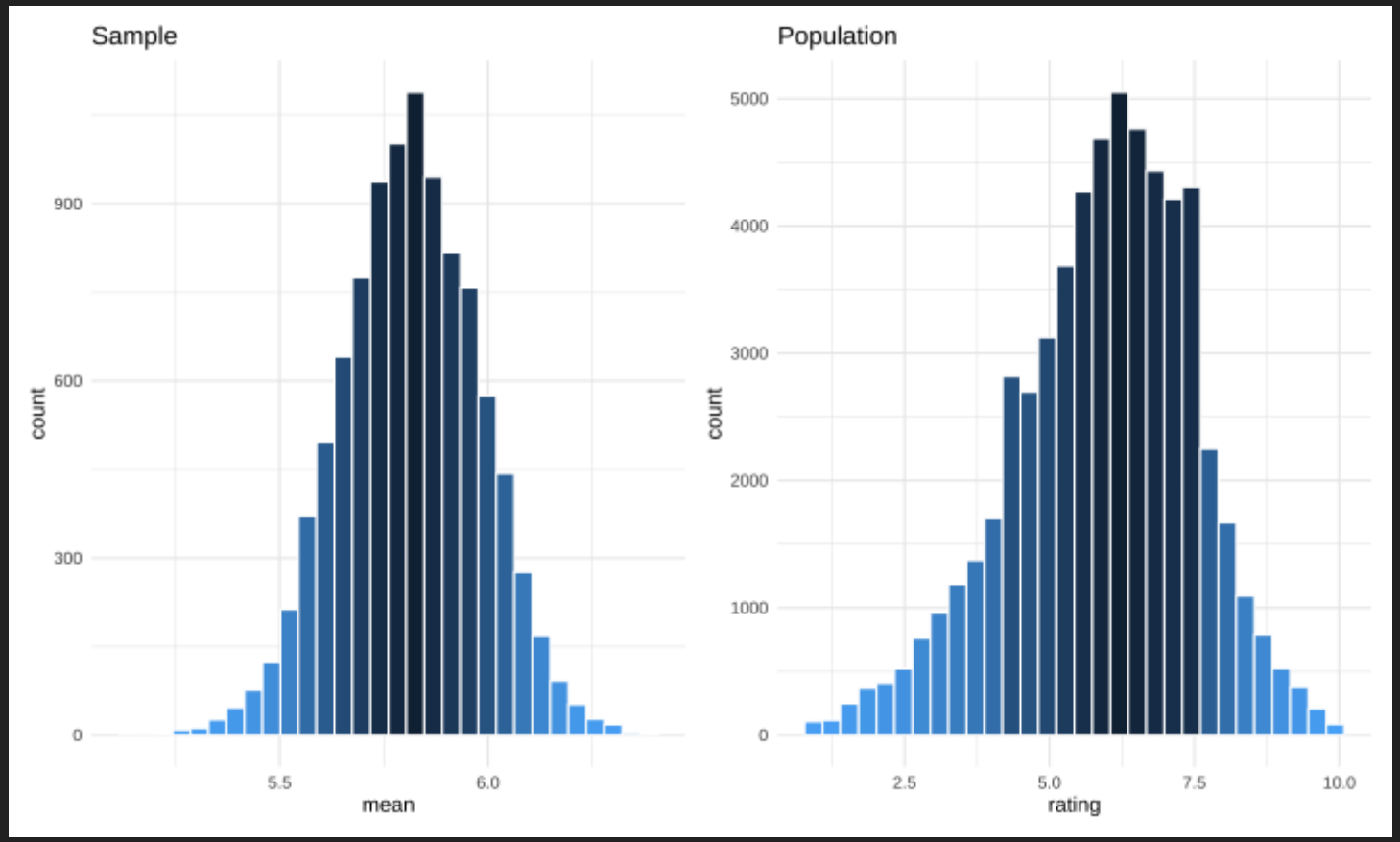
...wait a bit

```r
ggplot(data = not_lame ,
       mapping = aes(x = mean)) +
  geom_histogram(aes(fill = -..count..),
                 color = "white",
                 bins = 30,
                 show.legend = FALSE) +
  theme_minimal()
```



```r
samp <-
  ggplot(data = not_lame ,
       mapping = aes(x = mean)) +
  geom_histogram(aes(fill = -..count..),
                 color = "white",
                 bins = 30,
                 show.legend = FALSE) +
  theme_minimal() +
  ggtitle("Sample")
```

Comparison


Sample
900
600
300
0
count
5.5
6.0
mean
Population
5000
4000
3000
2000
1000
0
count
2.5
5.0
7.5
10.0
rating

Confidence using quantiles

quantiles are

- cut points dividing the range of a probability distribution into continuous intervals with equal probabilities
- found by isolating the middle 95% of values which corresponds to a 95% confidence interval for the population mean rating

```
(ci95_mean <- confint(not_lame,
                      level = 0.95,
                      method = "quantile"))
```

```
##   name lower    upper level     method estimate
## 1 mean  5.49 6.134286  0.95 percentile 5.814286
```

- we can be 95% confident that the true mean rating of ALL IMDB ratings is between 5.49 and about 6.13

Confidence using standard error

standard error is

- the standard deviation of the sampling distribution
- approximated by the bootstrap distribution or the null distribution depending on the context.

```
(ci95_mean <- confint(not_lame,
                      level = 0.95,
                      method = "stderr"))
```

```
## Warning: confint: Using df = Inf.
```

```
##   name    lower    upper level method estimate margin.of.error
## 1 mean 5.488206 6.139446  0.95 stderr 5.814286         0.32562
```

- we can be 95% confident that the true mean rating of ALL IMDB ratings is between 5.49 and about 6.13

Thats it!

> Yeah I just taught pivot_longer() in a class of 100 and the reaction was just a collective "OK cool whatever" while I'm up there like pic.twitter.com/gLuOuzoCmf
>
> — Allison Horst (@allison_horst) October 31, 2019