Bootstrapping and Confidence Intervals

El Mex

Contents


```
library(tidyverse)
library(moderndive)
library(infer)
```

1 Bootstrapping and Confidence Intervals

1.1 Pennies activity

1.1.1 What is the average year on US pennies in 2019?

Try to imagine all the pennies being used in the United States in 2019. That's a lot of pennies! Now say we're interested in the average year of minting of all these pennies. Instead of collecting all the pennies in the USA (near impossible), let's collect a sample of 50 pennies from a local bank.

The `moderndive` package contains this data on our 50 sampled pennies in the `pennies_sample` data frame:

```
# check it out
glimpse(pennies_sample)
```

```
## Rows: 50
## Columns: 2
## $ ID   <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,...
## $ year <dbl> 2002, 1986, 2017, 1988, 2008, 1983, 2008, 1996, 2004, 2000, 19...
```

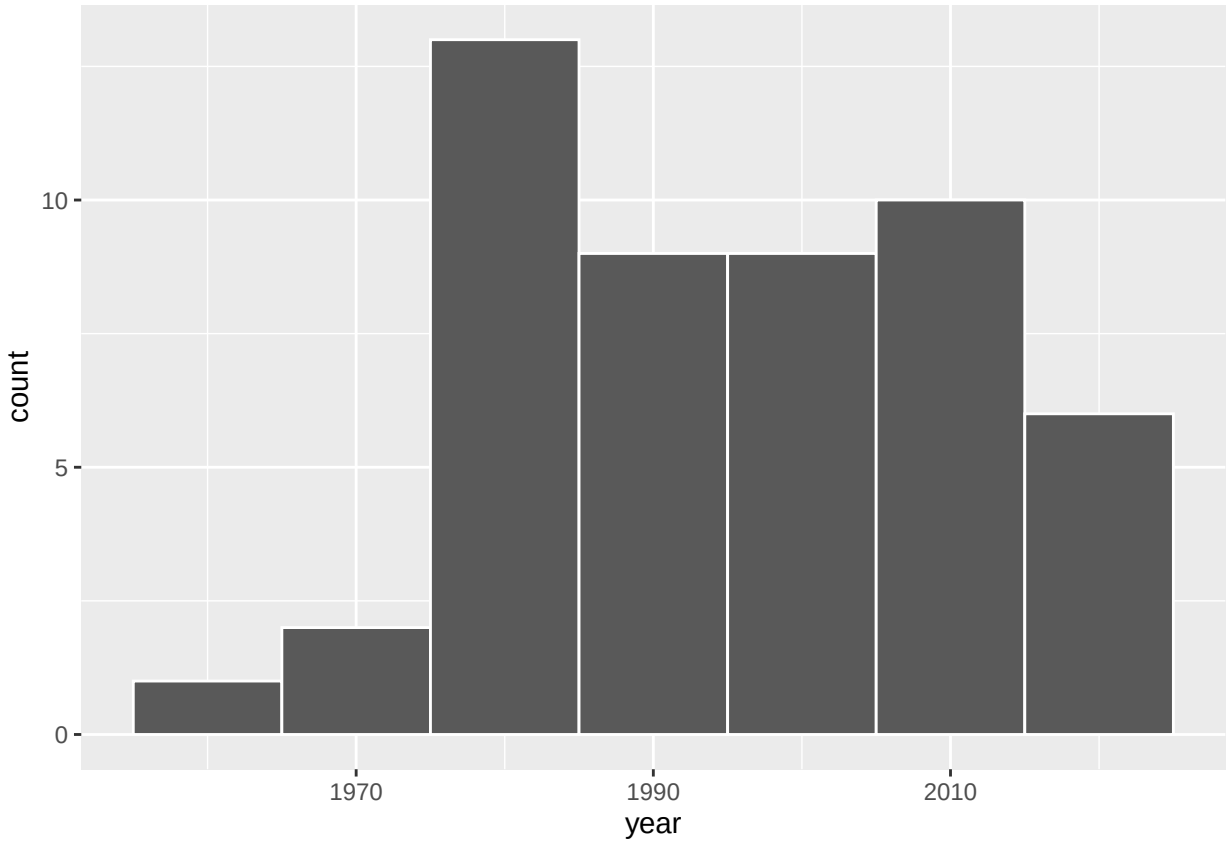
```
head(pennies_sample, 10)
```

```
## # A tibble: 10 x 2
##       ID  year
##    <int> <dbl>
##  1     1  2002
##  2     2  1986
##  3     3  2017
##  4     4  1988
##  5     5  2008
##  6     6  1983
##  7     7  2008
##  8     8  1996
##  9     9  2004
## 10    10  2000
```

The first variable `ID` corresponds to the ID labels, whereas the second variable `year` corresponds to the year of minting saved as a numeric variable, also known as a double (`dbl`).

Let's first visualize the distribution of the year of these 50 pennies. Since `year` is a numerical variable, we use a histogram.

```
ggplot(pennies_sample, aes(x = year)) +
  geom_histogram(binwidth = 10, color = "white")
```



Observe a slightly left-skewed distribution, since most pennies fall between 1980 and 2010 with only a few pennies older than 1970. What is the average year for the 50 sampled pennies? Eyeballing the histogram it appears to be around 1990. Let's now compute this value exactly.

```r
# use summarize() to get the mean
x_bar <- pennies_sample %>%
  summarize(mean_year = mean(year))

x_bar
```

```
## # A tibble: 1 x 1
##   mean_year
##       <dbl>
## 1     1995.
```

If we're willing to assume that pennies_sample is a representative sample from all US pennies, a "good guess" of the average year of minting of all US pennies would be 1995.44. This quantity is an estimate of the population mean year of all US pennies . Such estimates are prone to sampling variation and to study that we need too many samples. Say we're feeling lazy, however, and don't want to go back to the bank for another sample of pennies. How can we study the effects of sampling variation using our single sample?

1.1.2 Resampling once

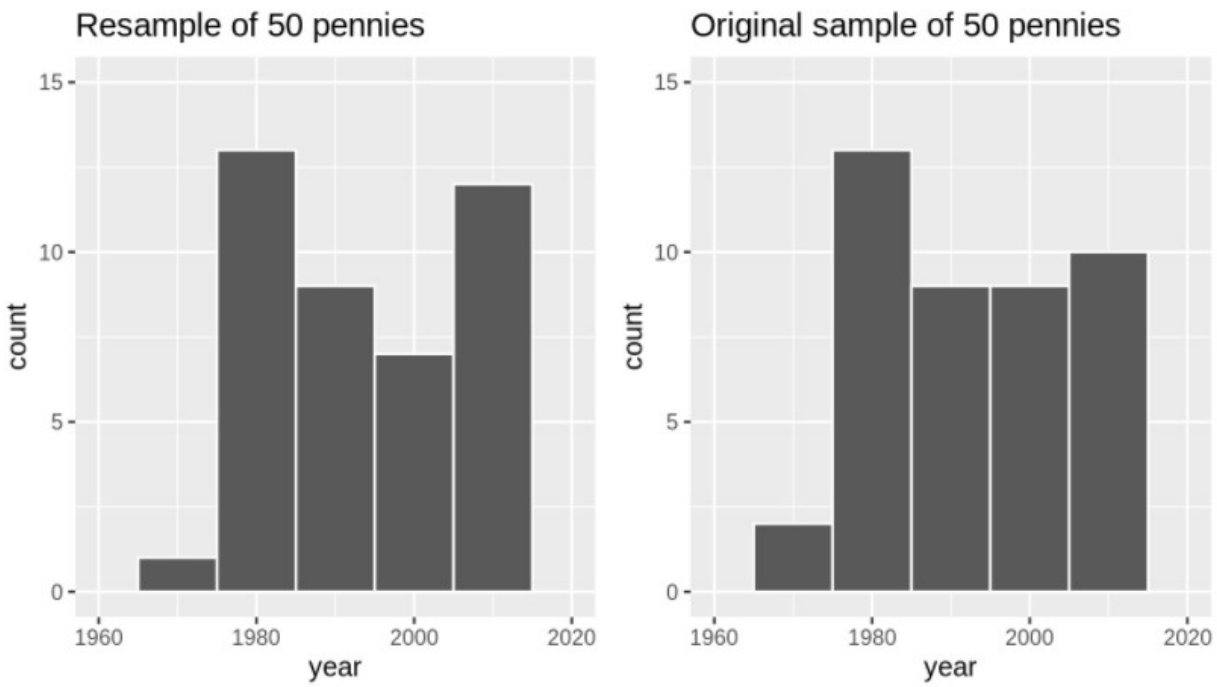
First, let's load the data into R by manually creating a data frame `pennies_resample` of our 50 resampled values. We'll do this using the `tibble()` command from the `dplyr` package. Note that the 50 values you resample will almost certainly not be the same as ours given the inherent randomness.

```r
pennies_resample <- tibble(
  year = c(1976, 1962, 1976, 1983, 2017, 2015, 2015, 1962, 2016, 1976,
           2006, 1997, 1988, 2015, 2015, 1988, 2016, 1978, 1979, 1997,
           1974, 2013, 1978, 2015, 2008, 1982, 1986, 1979, 1981, 2004,
           2000, 1995, 1999, 2006, 1979, 2015, 1979, 1998, 1981, 2015,
           2000, 1999, 1988, 2017, 1992, 1997, 1990, 1988, 2006, 2000)
)
```

The 50 values of `year` in `pennies_resample` represent a resample of size 50 from the original sample of 50 pennies. Let's compare the distribution of the numerical variable `year` of our 50 resampled pennies with the distribution of the numerical variable `year` of our original sample of 50 pennies.

```r
ggplot(pennies_resample, aes(x = year)) +
  geom_histogram(binwidth = 10, color = "white") +
  labs(title = "Resample of 50 pennies")

ggplot(pennies_sample, aes(x = year)) +
  geom_histogram(binwidth = 10, color = "white") +
  labs(title = "Original sample of 50 pennies")
```



While the general shapes of both distributions of year are roughly similar, they are not identical.

What about the sample mean for our resample?

```
pennies_resample %>%
  summarize(mean_year = mean(year))
```

```
## # A tibble: 1 x 1
##   mean_year
##       <dbl>
## 1     1995.
```

We obtained a different mean year of 1994.82. This variation is induced by the resampling with replacement we performed earlier.

What if we repeated this resampling exercise many times? Would we obtain the same mean year each time? let's perform this resampling activity with the help of some of our friends: 35 friends in total.

1.1.3 Resampling 35 times

Each of our 35 friends will get 50 numbers, 50 times. For your convenience, we've taken these 35 x 50 = 1750 values and saved them in `pennies_resamples`, a "tidy" data frame included in the `moderndive` package.

```
# explore it
glimpse(pennies_resamples)
```

```
## Rows: 1,750
```

```
## Columns: 3
## Groups: name [35]
## $ replicate <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ name      <chr> "Arianna", "Arianna", "Arianna", "Arianna", "Arianna", "A...
## $ year      <dbl> 1988, 2002, 2015, 1998, 1979, 1971, 1971, 2015, 1988, 197...
```

```
head(pennies_resamples, 10)
```

```
## # A tibble: 10 x 3
## # Groups:   name [1]
##    replicate name     year
##        <int> <chr>   <dbl>
##  1         1 Arianna  1988
##  2         1 Arianna  2002
##  3         1 Arianna  2015
##  4         1 Arianna  1998
##  5         1 Arianna  1979
##  6         1 Arianna  1971
##  7         1 Arianna  1971
##  8         1 Arianna  2015
##  9         1 Arianna  1988
## 10         1 Arianna  1979
```

What did each of our 35 friends obtain as the mean year? group by `name`and `summarize` each group of 50 rows by their mean `year`:

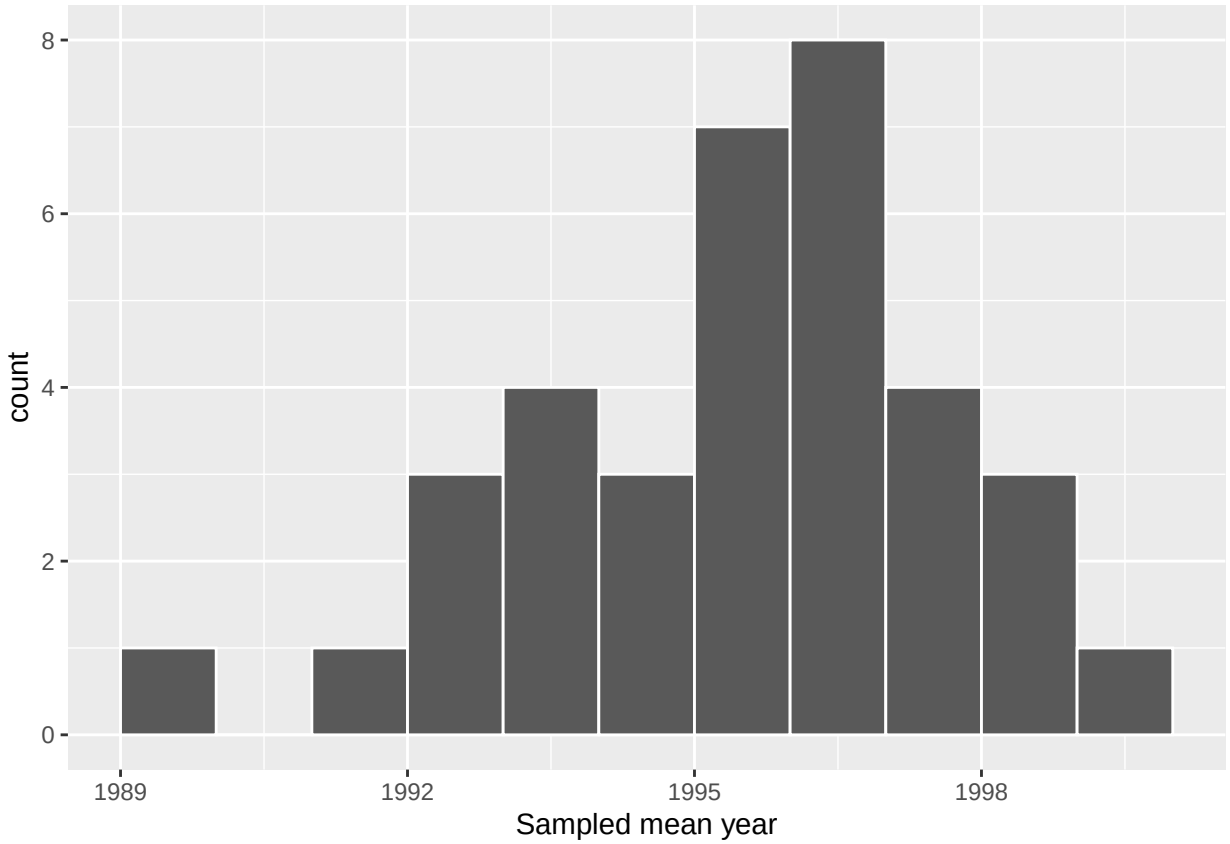
```
resampled_means <- pennies_resamples %>%
  group_by(name) %>%
  summarize(mean_year = mean(year))

head(resampled_means, 10)
```

```
## # A tibble: 10 x 2
##    name      mean_year
##    <chr>         <dbl>
##  1 Arianna       1992.
##  2 Artemis       1996.
##  3 Bea           1996.
##  4 Camryn        1997.
##  5 Cassandra     1991.
##  6 Cindy         1995.
##  7 Claire        1996.
##  8 Dahlia        1998.
##  9 Dan           1994.
## 10 Eindra        1994.
```

Let's visualize this variation using a histogram. Recall that adding the argument `boundary = 1990` to the `geom_histogram()` sets the binning structure so that one of the bin boundaries is at 1990 exactly.

```
ggplot(resampled_means, aes(x = mean_year)) +
  geom_histogram(binwidth = 1, boundary = 1990, color = "white") +
  labs(x = "Sampled mean year")
```



The distribution looks roughly normal and that we rarely observe sample mean years less than 1992 or greater than 2000. Also observe how the distribution is roughly centered at 1995, which is close to the sample mean of 1995.44 of the original sample of 50 pennies from the bank.

1.2 Computer simulation of resampling

1.2.1 Virtually resampling once

```
# use rep_sample_n() to perform the resampling with replacement of the 50 slips of paper representing o
virtual_resample <- pennies_sample %>%
  rep_sample_n(size = 50, replace = TRUE)

glimpse(virtual_resample)
```

```
## Rows: 50
## Columns: 3
## Groups: replicate [1]
## $ replicate <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
```

```
## $ ID        <int> 42, 19, 10, 42, 8, 29, 40, 25, 38, 4, 4, 13, 19, 21, 36, ...
## $ year      <dbl> 1997, 1983, 2000, 1997, 1996, 1988, 1990, 1979, 1999, 198...
```

```
head(virtual_resample, 10)
```

```
## # A tibble: 10 x 3
## # Groups:   replicate [1]
##    replicate    ID  year
##        <int> <int> <dbl>
##  1         1    42  1997
##  2         1    19  1983
##  3         1    10  2000
##  4         1    42  1997
##  5         1     8  1996
##  6         1    29  1988
##  7         1    40  1990
##  8         1    25  1979
##  9         1    38  1999
## 10         1     4  1988
```

Observe how we explicitly set the `replace` argument to `TRUE` in order to tell `rep_sample_n()` that we would like to sample pennies with replacement. Since we didn't specify the number of replicates via the `reps` argument, the function assumes the default of one replicate `reps = 1`.

Let's now compute the mean year in our virtual resample:

```
virtual_resample %>%
  summarize(resample_mean = mean(year))
```

```
## # A tibble: 1 x 2
##   replicate resample_mean
##       <int>         <dbl>
## 1         1         1997.
```

1.2.2 Virtually resampling 35 times

Let's first add a `reps = 35` argument to `rep_sample_n()` to indicate we would like 35 replicates.

```
virtual_resamples <- pennies_sample %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 35)

glimpse(virtual_resamples)
```

```
## Rows: 1,750
## Columns: 3
## Groups: replicate [35]
## $ replicate <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ ID        <int> 15, 49, 34, 20, 15, 17, 23, 36, 10, 29, 9, 31, 35, 43, 10...
## $ year      <dbl> 1974, 2006, 1985, 1971, 1974, 2016, 1998, 2015, 2000, 198...
```

```
head(virtual_resamples, 10)
```

```
## # A tibble: 10 x 3
## # Groups:   replicate [1]
##    replicate    ID  year
##        <int> <int> <dbl>
##  1         1    15  1974
##  2         1    49  2006
##  3         1    34  1985
##  4         1    20  1971
##  5         1    15  1974
##  6         1    17  2016
##  7         1    23  1998
##  8         1    36  2015
##  9         1    10  2000
## 10         1    29  1988
```

Let's now compute the resulting 35 sample means, but this time adding a `group_by(replicate)`

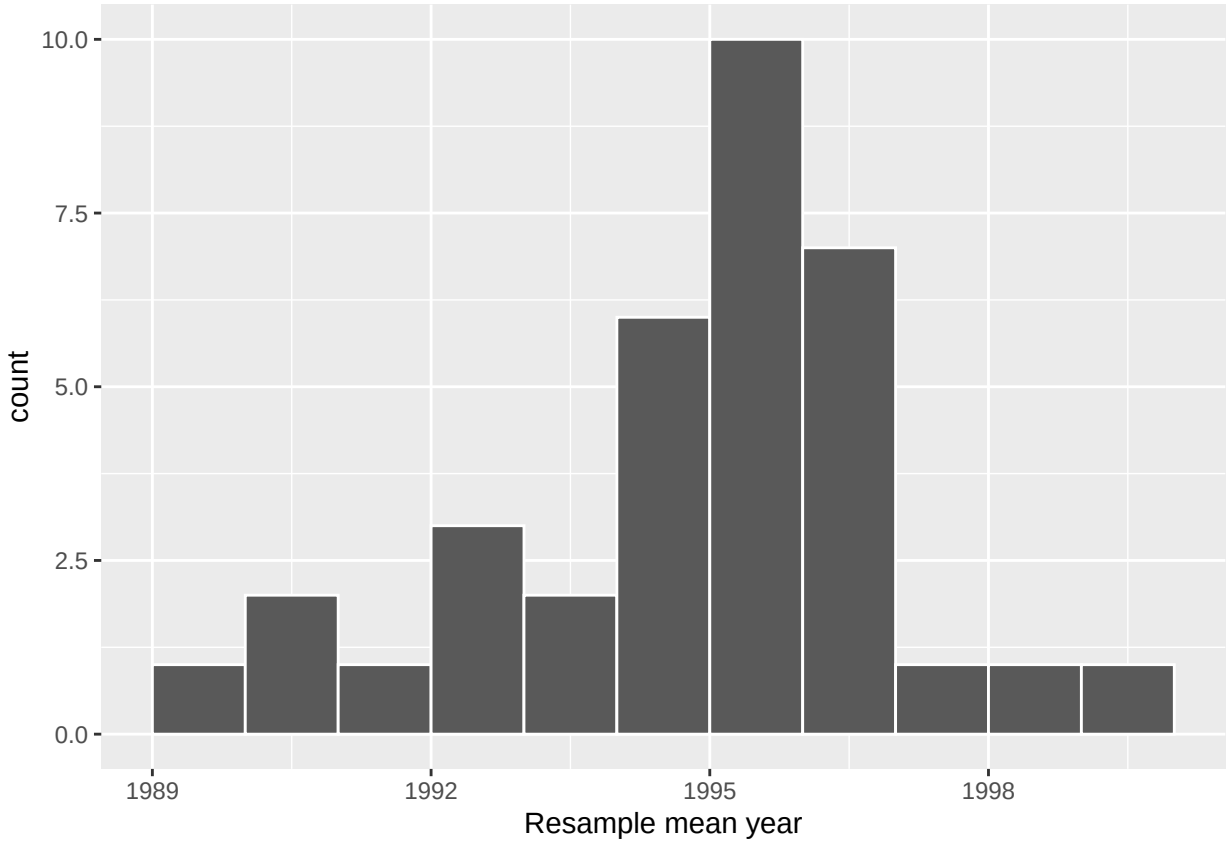
```
virtual_resampled_means <- virtual_resamples %>%
  group_by(replicate) %>%
  summarize(mean_year = mean(year))

head(virtual_resampled_means, 10)
```

```
## # A tibble: 10 x 2
##    replicate mean_year
##        <int>     <dbl>
##  1         1     1994.
##  2         2     1995.
##  3         3     1996.
##  4         4     1997.
##  5         5     1994.
##  6         6     1992.
##  7         7     1996.
##  8         8     1989.
##  9         9     1996.
## 10        10     1997.
```

Let's visualize this variation using a histogram.

```
ggplot(virtual_resampled_means, aes(x = mean_year)) +
  geom_histogram(binwidth = 1, boundary = 1990, color = "white") +
  labs(x = "Resample mean year")
```



Compare this virtually constructed bootstrap distribution with the one our 35 friends constructed via our tactile resampling exercise. Observe how they are somewhat similar, but not identical.

1.2.3 Virtually resampling 1000 times

Let's increase the number of resamples to 1000

```
# Repeat resampling 1000 times
virtual_resamples <- pennies_sample %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 1000)

# Compute 1000 sample means
virtual_resampled_means <- virtual_resamples %>%
  group_by(replicate) %>%
  summarize(mean_year = mean(year))
```

For brevity, let's combine these two operations into a single chain of pipe (%>%) operators:

```
virtual_resampled_means <- pennies_sample %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 1000) %>%
  group_by(replicate) %>%
  summarize(mean_year = mean(year))

glimpse(virtual_resampled_means)
```

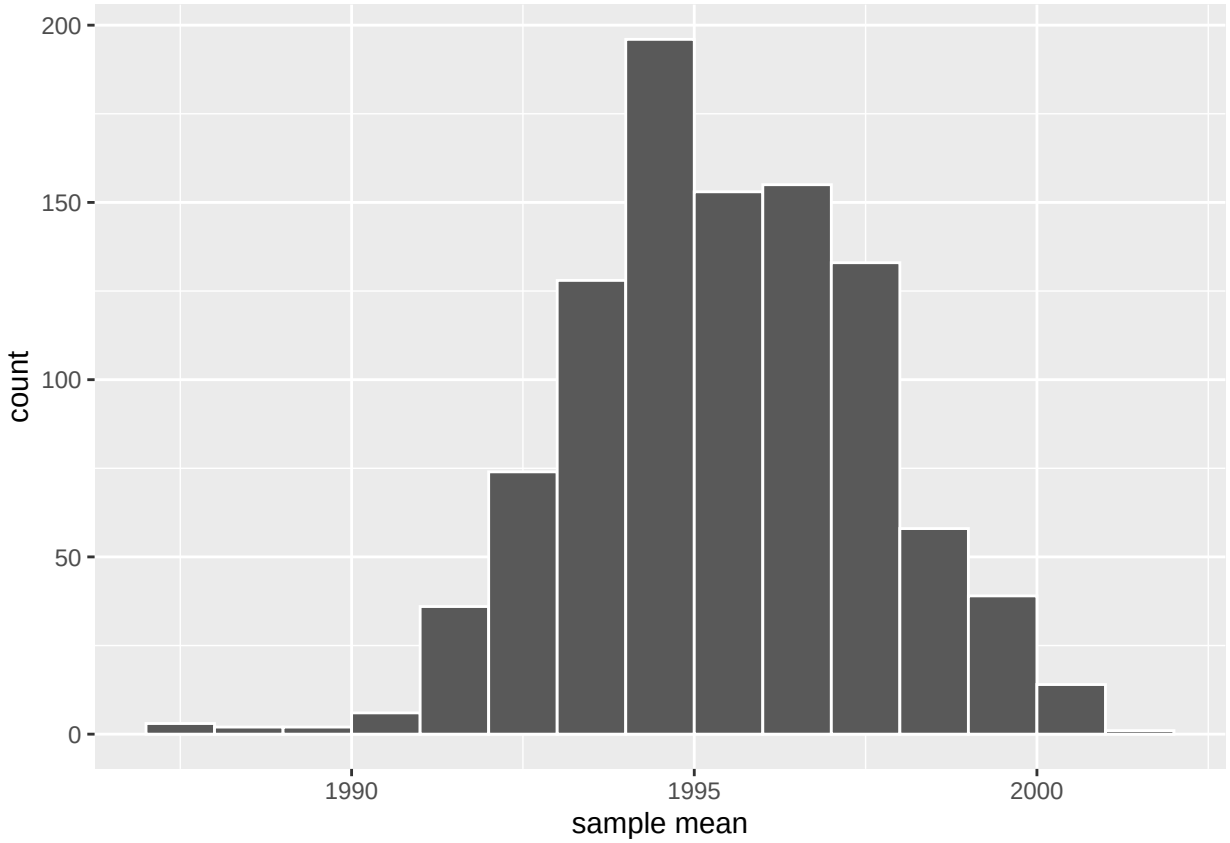
```
## Rows: 1,000
## Columns: 2
## $ replicate <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17...
## $ mean_year <dbl> 1995.36, 1997.12, 1995.34, 1993.86, 1996.16, 1995.40, 199...
```

```r
head(virtual_resampled_means, 10)
```

```
## # A tibble: 10 x 2
##    replicate mean_year
##        <int>     <dbl>
##  1         1     1995.
##  2         2     1997.
##  3         3     1995.
##  4         4     1994.
##  5         5     1996.
##  6         6     1995.
##  7         7     1996.
##  8         8     1994.
##  9         9     1998.
## 10        10     1996.
```

Let's visualize the bootstrap distribution of these 1000 means based on 1000 virtual resamples:

```r
ggplot(virtual_resampled_means, aes(x = mean_year)) +
  geom_histogram(binwidth = 1, boundary = 1990, color = "white") +
  labs(x = "sample mean")
```



Note here that the bell shape is starting to become much more apparent. We now have a general sense for the range of values that the sample mean may take on. But where is this histogram centered? Let's compute the mean of the 1000 resample means:

```
virtual_resampled_means %>%
  summarize(mean_of_means = mean(mean_year))
```

```
## # A tibble: 1 x 1
##   mean_of_means
##           <dbl>
## 1         1995.
```

The mean of these 1000 means is 1995.38, which is quite close to the mean of our original sample of 50 pennies of 1995.44.

1.3 Understanding confidence intervals

We now introduce two methods for constructing such intervals in a more exact fashion: the percentile method and the standard error method.

1.3.1 Percentile method

One method to construct a confidence interval is to use the middle 95% of values of the bootstrap distribution. We can do this by computing the 2.5th and 97.5th percentiles, which are 1991.059 and 1999.283, respectively. This is known as the percentile method for constructing confidence intervals.

Let's mark these percentiles on the bootstrap distribution with vertical lines in Figure 8.16. About 95% of the `mean_year` variable values in `virtual_resampled_means` fall between 1991.059 and 1999.283, with 2.5% to the left of the leftmost line and 2.5% to the right of the rightmost line.

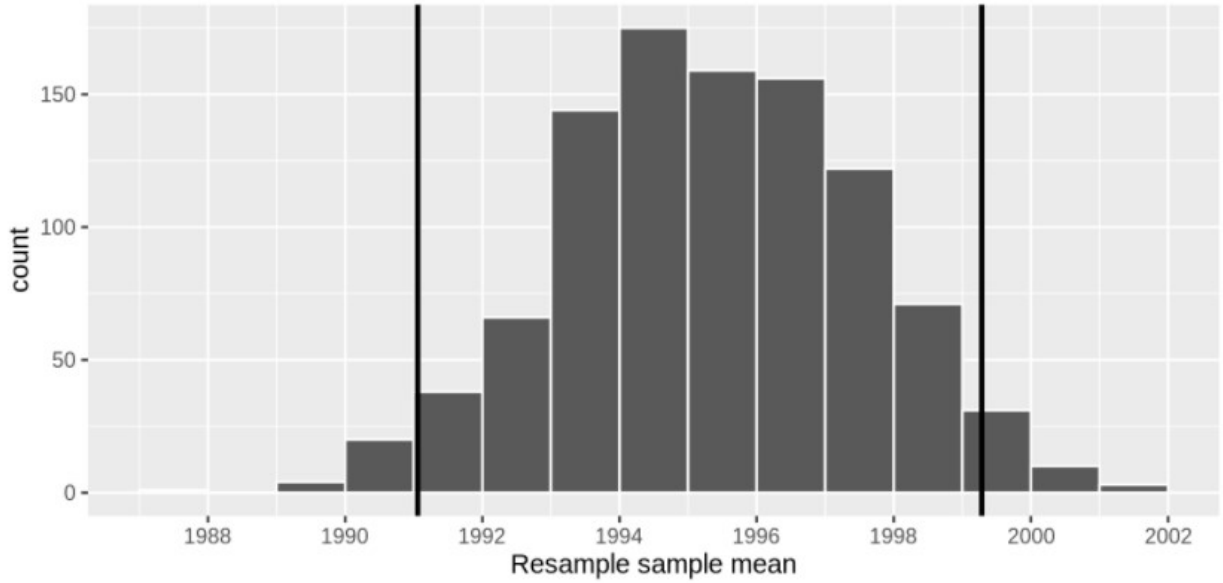


FIGURE 8.16: Percentile method 95% confidence interval. Interval endpoints marked by vertical lines.

1.3.2 Standard error method

We saw that if a numerical variable follows a normal distribution, or, in other words, the histogram of this variable is bell-shaped, then roughly 95% of values fall between ±1.96 standard deviations of the mean. Given that our bootstrap distribution based on 1000 resamples with replacement is normally shaped, let's use this fact about normal distributions to construct a confidence interval in a different way.

Let's compute the standard deviation of the bootstrap distribution using the values of `mean_year` in the `virtual_resampled_means` data frame:

```
virtual_resampled_means %>%
  summarize(SE = sd(mean_year))
```

```
## # A tibble: 1 x 1
##      SE
##   <dbl>
## 1  2.15
```

We can say that 2.164 is an approximation of the standard error of x.

Using our 95% rule of thumb about normal distributions, we can use the following formula to determine the lower and upper endpoints of a 95% confidence interval for :

$$\begin{aligned}\overline{x} \pm 1.96 \cdot SE &= (\overline{x} - 1.96 \cdot SE, \overline{x} + 1.96 \cdot SE) \\ &= (1995.44 - 1.96 \cdot 2.15, 1995.44 + 1.96 \cdot 2.15) \\ &= (1991.15, 1999.73)\end{aligned}$$

Let's now add the SE method confidence interval with dashed lines in Figure 8.17.

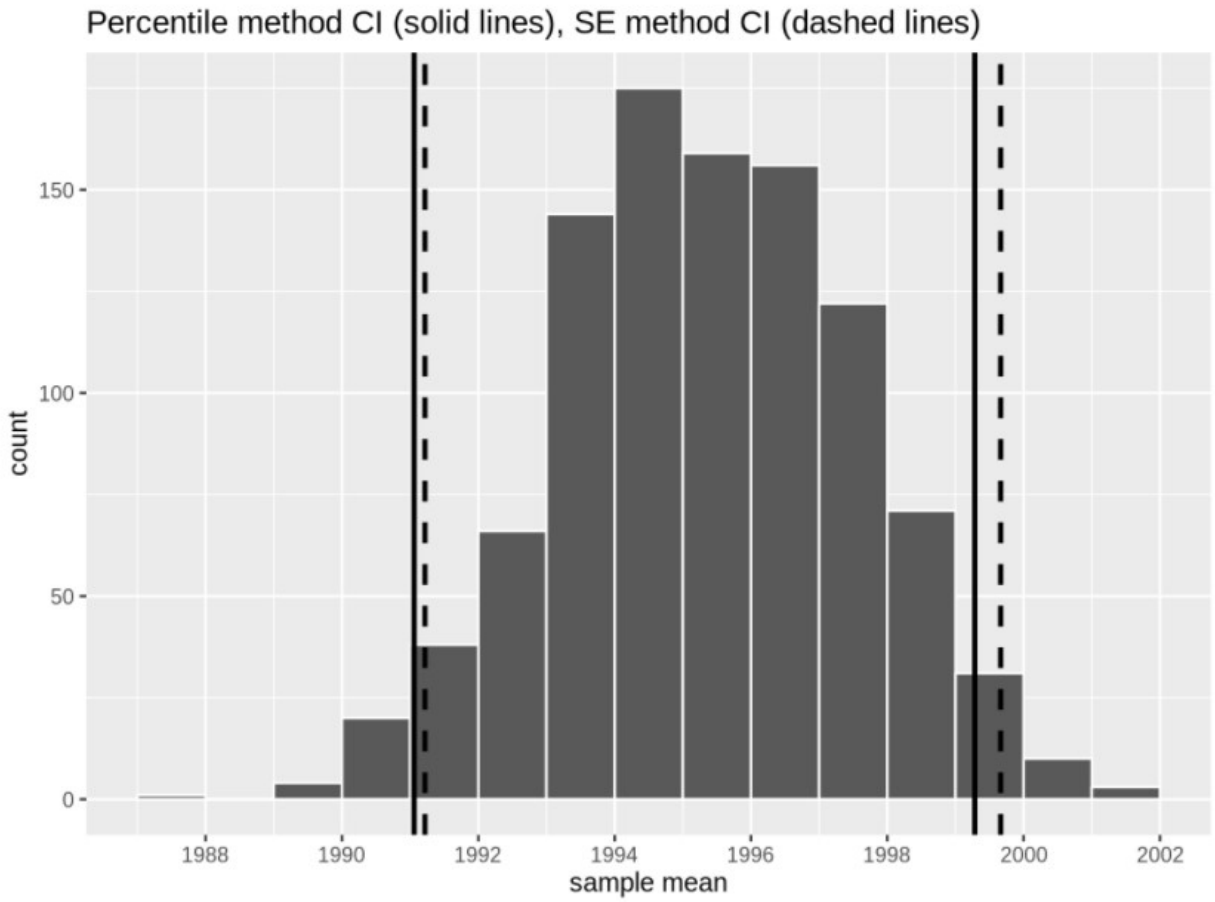


FIGURE 8.17: Comparing two 95% confidence interval methods.

We see that both methods produce nearly identical 95% confidence intervals for with the percentile method yielding (1991.06, 1999.28) while the standard error method produces (1991.22, 1999.66). However, recall that we can only use the standard error rule when the bootstrap distribution is roughly normally shaped.

1.4 Constructing confidence intervals

1.4.1 Original workflow

We virtually performed bootstrap resampling with replacement to construct bootstrap distributions. Such distributions are approximations to the sampling distributions we saw in Chapter 7, but are constructed using only a single sample.

First, we used the `rep_sample_n()` function to resample `size = 50` pennies with replacement from the original sample of 50 pennies in `pennies_sample` by setting `replace = TRUE`. Furthermore, we repeated this resampling 1000 times by setting `reps = 1000`:

```
pennies_sample %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 1000)
```

Second, since for each of our 1000 resamples of size 50, we wanted to compute a separate sample mean, we used the `dplyr` verb `group_by()` to group observations/rows together by the `replicate` variable...

```
pennies_sample %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 1000) %>%
  group_by(replicate)
```

... followed by using `summarize()` to compute the sample `mean()` year for each `replicate` group:

```
pennies_sample %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 1000) %>%
  group_by(replicate) %>%
  summarize(mean_year = mean(year))
```

We can get by with using the `rep_sample_n()` function and a couple of `dplyr` verbs to construct the bootstrap distribution. For more complicated situations, we'll need a little more firepower.

1.4.2 `infer` package workflow

Let's go back to our pennies. Previously, we computed the value of the sample mean x using the dplyr function `summarize()`

```
pennies_sample %>%
  summarize(stats = mean(year))
```

```
## # A tibble: 1 x 1
##   stats
##   <dbl>
## 1 1995.
```

We can also do this using `infer` functions `specify()` and `calculate()`:

```
pennies_sample %>%
  specify(response = year) %>%
  calculate(stat = "mean")
```

```
## # A tibble: 1 x 1
##    stat
##   <dbl>
## 1 1995.
```

Let's now illustrate the sequence of verbs necessary to construct a confidence interval for , the population mean year of minting of all US pennies in 2019.

1. `specify` variables

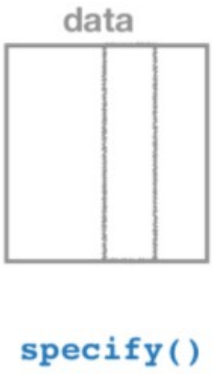


FIGURE 8.18: Diagram of the specify() verb.

As shown in Figure 8.18, the `specify()` function is used to choose which variables in a data frame will be the focus of our statistical inference. We do this by `specify`ing the `response` argument. For example, in our `pennies_sample` data frame of the 50 pennies sampled from the bank, the variable of interest is `year`:

```
pennies_sample %>%
  specify(response = year)
```

```
Response: year (numeric)
# A tibble: 50 x 1
    year
   <dbl>
 1  2002
 2  1986
 3  2017
 4  1988
 5  2008
 6  1983
 7  2008
 8  1996
 9  2004
10  2000
# … with 40 more rows
```

Notice how the data itself doesn't change, but the `Response: year (numeric)` meta-data does. This is similar to how the `group_by()` verb from `dplyr` doesn't change the data, but only adds "grouping" meta-data

The following use of specify() with the formula argument yields the same result seen previously:

```
pennies_sample %>%
  specify(formula = year ~ NULL)
```

Since in the case of pennies we only have a response variable and no explanatory variable of interest, we set the `x` on the right-hand side of the `~` to be `NULL`.

2. `generate` replicates

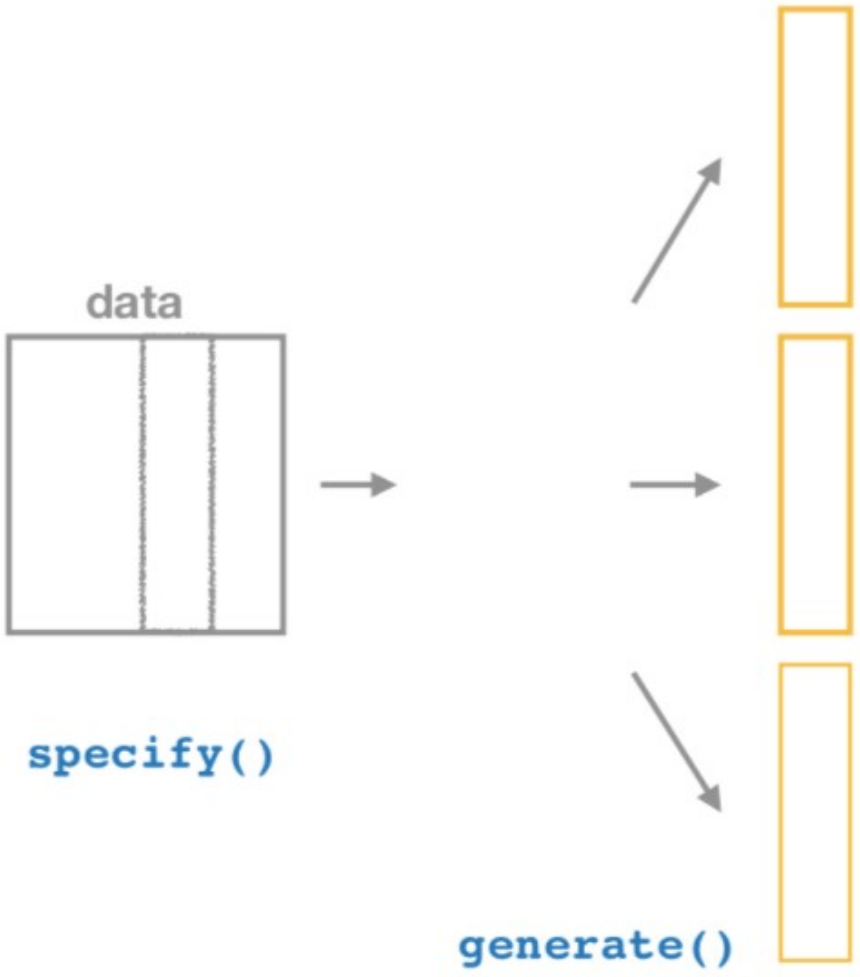


FIGURE 8.19: Diagram of generate() replicates.

The `generate()` function's first argument is `reps`, which sets the number of replicates we would like to generate. Since we want to resample the 50 pennies in `pennies_sample` with replacement 1000 times, we set `reps = 1000`. The second argument type determines the type of computer simulation we'd like to perform. We set this to `type = "bootstrap"` indicating that we want to perform bootstrap resampling.

```r
pennies_sample %>%
  specify(response = year) %>%
  generate(reps = 1000, type = "bootstrap")
```

```
Response: year (numeric)
# A tibble: 50,000 x 2
# Groups:   replicate [1,000]
   replicate  year
       <int> <dbl>
 1         1  1981
 2         1  1988
 3         1  2006
 4         1  2016
 5         1  2002
 6         1  1985
 7         1  1979
 8         1  2000
 9         1  2006
10         1  2016
# … with 49,990 more rows
```

We performed resampling of 50 pennies with replacement 1000 times and 50,000 = 50 x 1000. The variable `replicate` indicates which resample each row belongs to. So it has the value `1` 50 times, the value `2` 50 times, all the way through to the value `1000` 50 times.

The default value of the type argument is `"bootstrap"` in this scenario, so if the last line was written as `generate(reps = 1000)`, we'd obtain the same results.

So far comparing infer workflow to the original, the following two code chunks produce similar results:

```r
# infer workflow:
pennies_sample %>%
  specify(response = year) %>%
  generate(reps = 1000)
```

```r
# Original workflow:
pennies_sample %>%
  rep_sample_n(size = 50, replace = TRUE,
               reps = 1000)
```

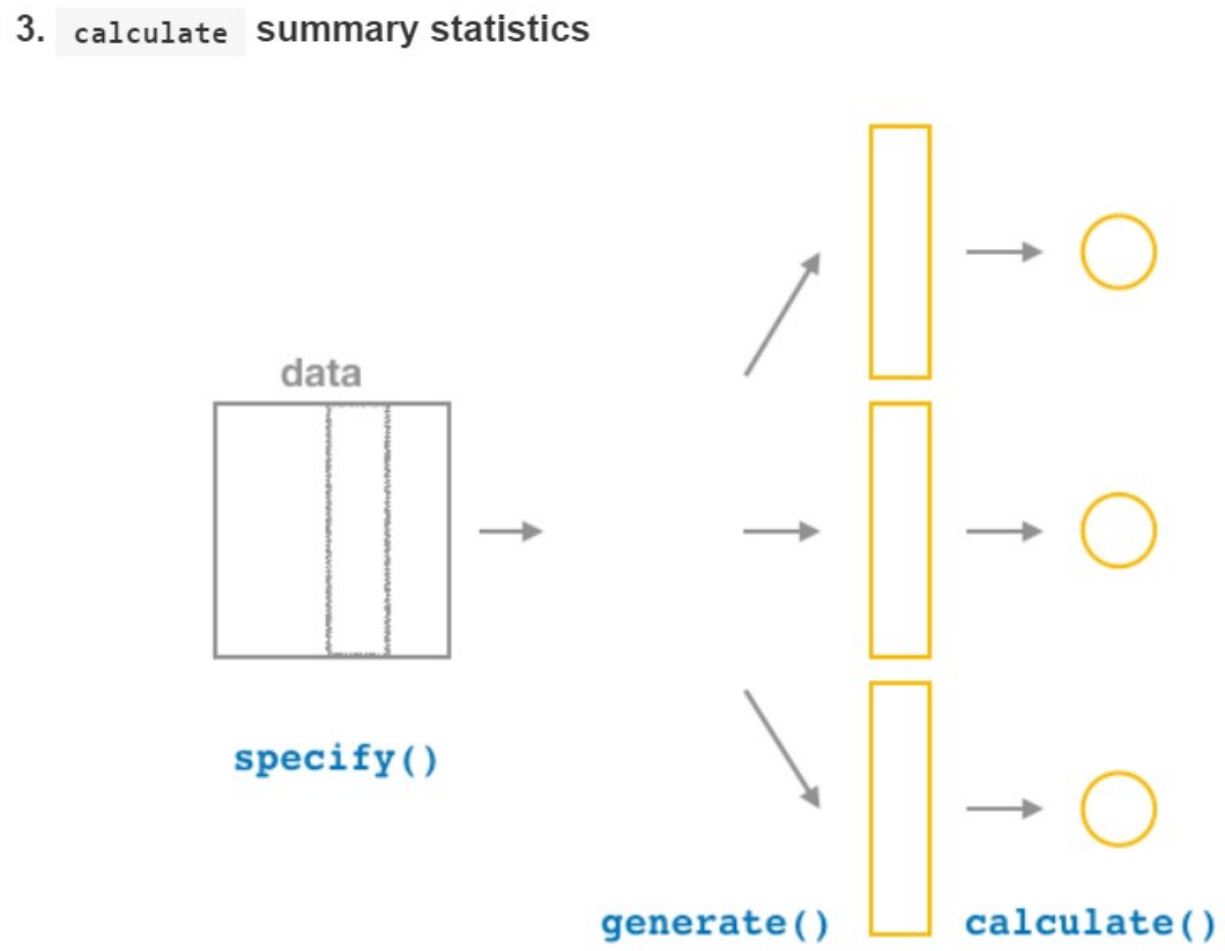


FIGURE 8.20: Diagram of calculate() summary statistics.

We want to calculate the mean `year` for each bootstrap resample of size 50. To do so, we set the `stat` argument to `"mean"`. You can also set the `stat` argument to a variety of other common summary statistics, like `"median"`, `"sum"`, `"sd"` (standard deviation), and `"prop"` (proportion).

Let's save the result in a data frame called `bootstrap_distribution` and explore its contents:

```
bootstrap_distribution <- pennies_sample %>%
  specify(response = year) %>%
  generate(reps = 1000) %>%
  calculate(stat = "mean")

glimpse(bootstrap_distribution)
```

```
## Rows: 1,000
## Columns: 2
## $ replicate <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17...
## $ stat      <dbl> 1997.20, 1993.82, 1990.50, 1994.78, 1996.58, 1996.66, 199...
```

```
head(bootstrap_distribution, 10)
```

```
## # A tibble: 10 x 2
##    replicate  stat
##        <int> <dbl>
##  1         1 1997.
##  2         2 1994.
##  3         3 1990.
##  4         4 1995.
##  5         5 1997.
##  6         6 1997.
##  7         7 1998.
##  8         8 1994.
##  9         9 1996.
## 10        10 1994.
```

Compare both worflows for similarities:

```
# infer workflow:
pennies_sample %>%
  specify(response = year) %>%
  generate(reps = 1000) %>%
  calculate(stat = "mean")
```

```
# Original workflow:
pennies_sample %>%
  rep_sample_n(size = 50, replace = TRUE,
               reps = 1000) %>%
  group_by(replicate) %>%
  summarize(stat = mean(year))
```

4. `visualize` the results

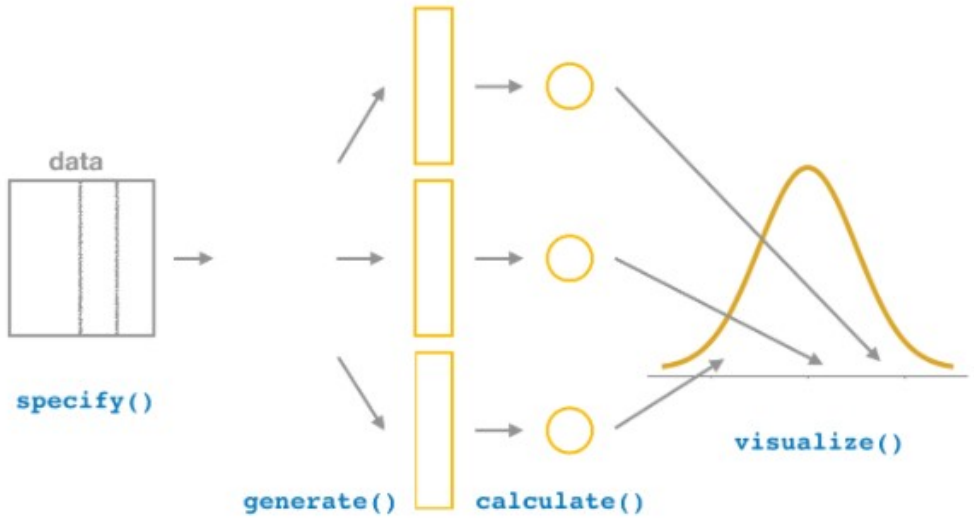
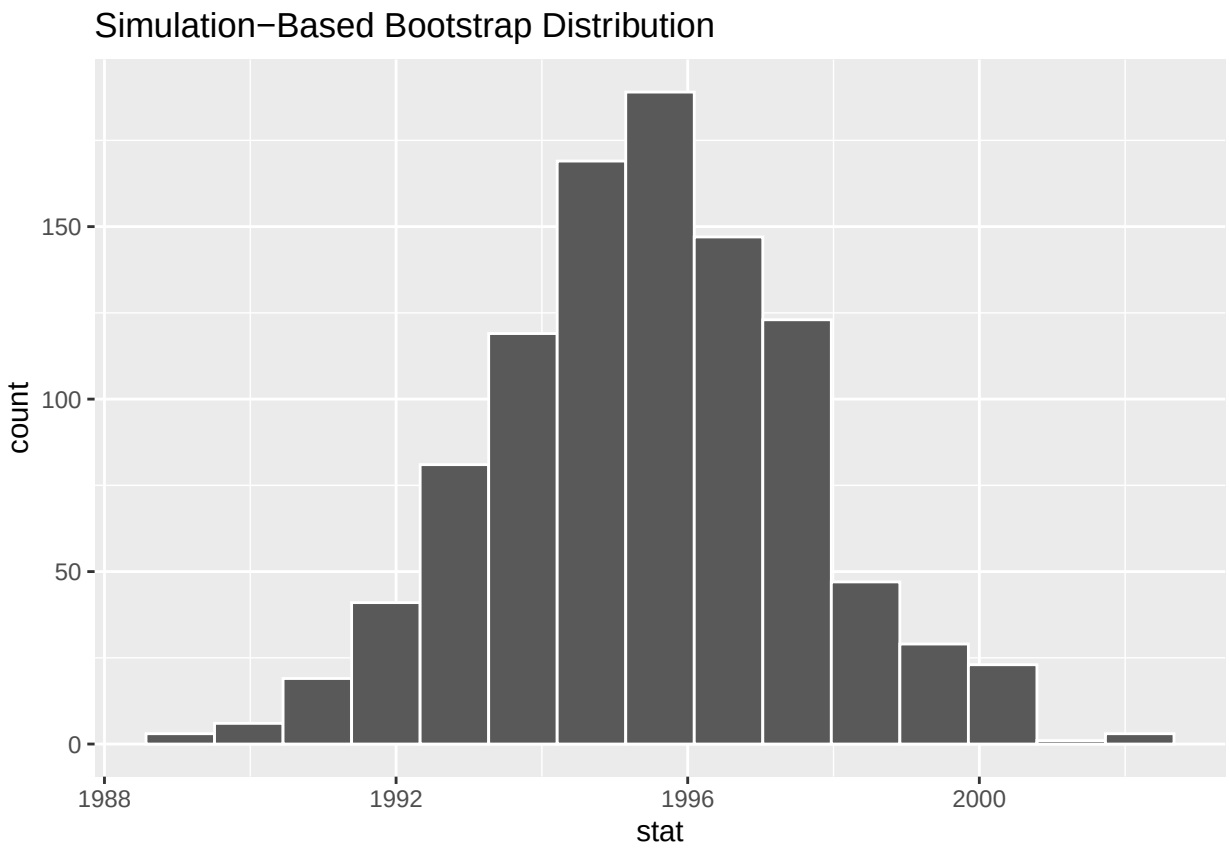


FIGURE 8.21: Diagram of visualize() results.

The `visualize()` verb provides a quick way to visualize the bootstrap distribution as a histogram of the numerical stat variable's values.

```
visualize(bootstrap_distribution)
```



Comparing to original workflow:

```
# infer workflow:
visualize(bootstrap_distribution)
```

```
# Original workflow:
ggplot(bootstrap_distribution,
       aes(x = stat)) +
  geom_histogram()
```

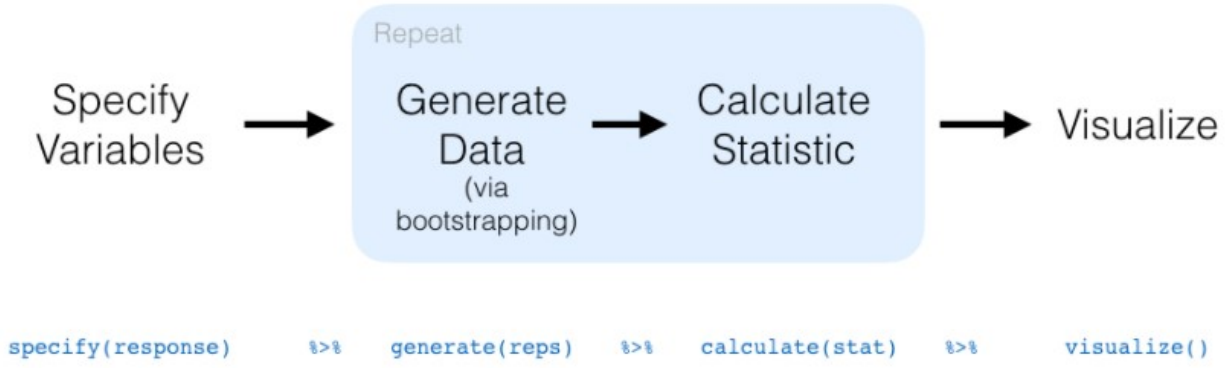


FIGURE 8.23: infer package workflow for confidence intervals.

1.4.3 Percentile method with `infer`

We can compute the 95% confidence interval by piping `bootstrap_distribution` into the `get_confidence_interval()` function from the `infer` package, with the confidence `level` set to 0.95 and the confidence interval `type` to be `"percentile"`. Let's save the results in `percentile_ci`.

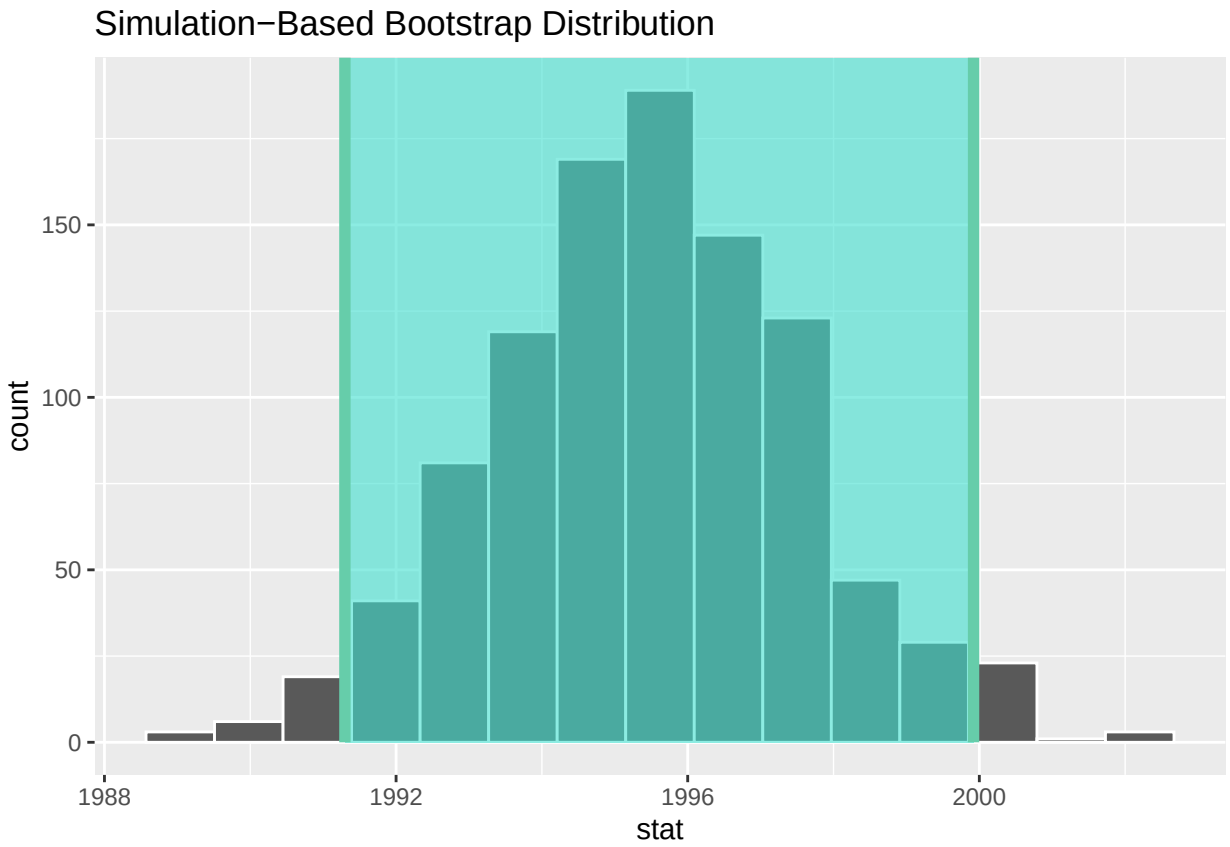
```
percentile_ci <- bootstrap_distribution %>%
  get_confidence_interval(level = 0.95, type = "percentile")

percentile_ci
```

```
## # A tibble: 1 x 2
##   lower_ci upper_ci
##      <dbl>    <dbl>
## 1    1991.    2000.
```

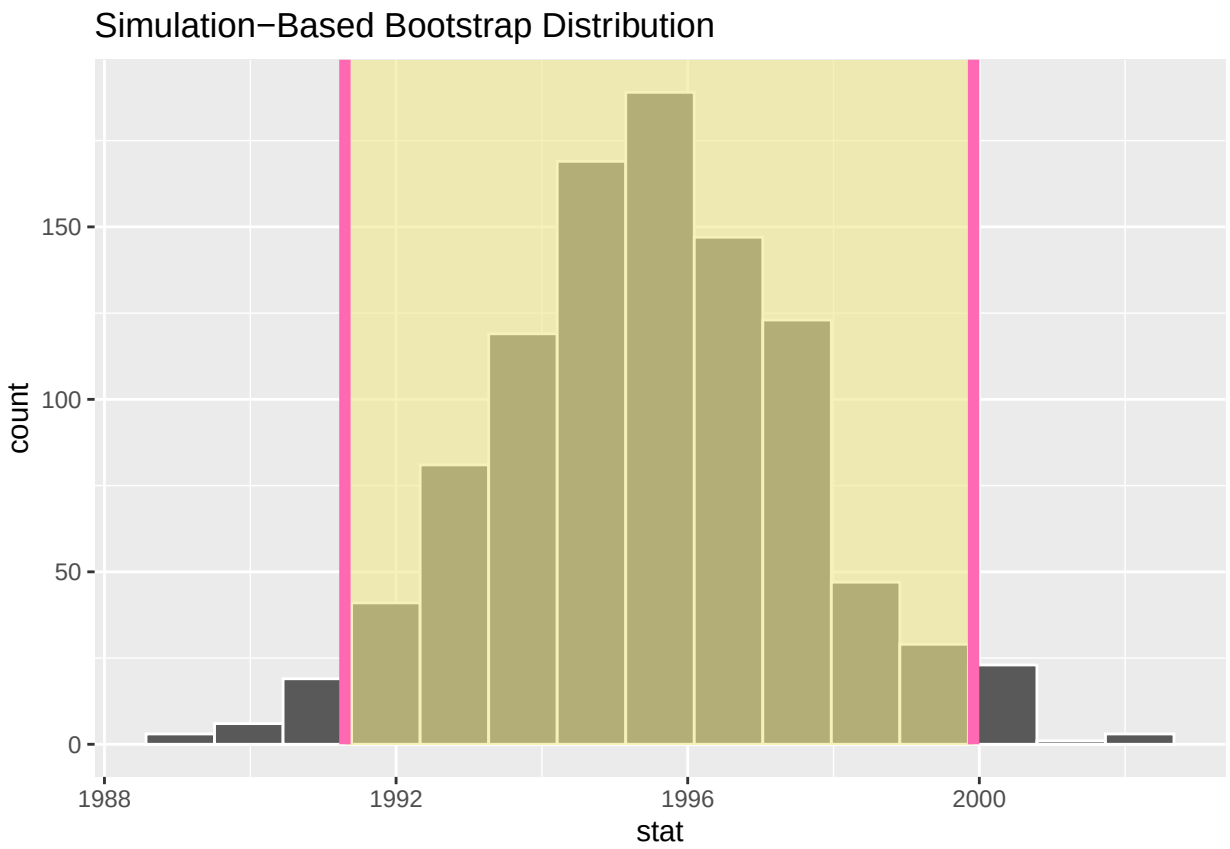
Alternatively, we can visualize the interval (1991.12, 1999.36) inside the graph by using `visualize()` function and adding a `shade_confidence_interval()` layer. Set the `endpoints` argument to be `percentile_ci`.

```
visualize(bootstrap_distribution) +
  shade_confidence_interval(endpoints = percentile_ci)
```



A shorter version with `shade_ci` and modifying colors:

```
visualize(bootstrap_distribution) +
  shade_ci(endpoints = percentile_ci, color = "hotpink", fill = "khaki")
```



1.4.4 Standard error method with `infer`

This time we set the first type argument to be `"se"`. Second, we must specify the `point_estimate` argument in order to set the center of the confidence interval. We set this to be the sample mean of the original sample of 50 pennies of 1995.44 we saved in `x_bar` earlier.

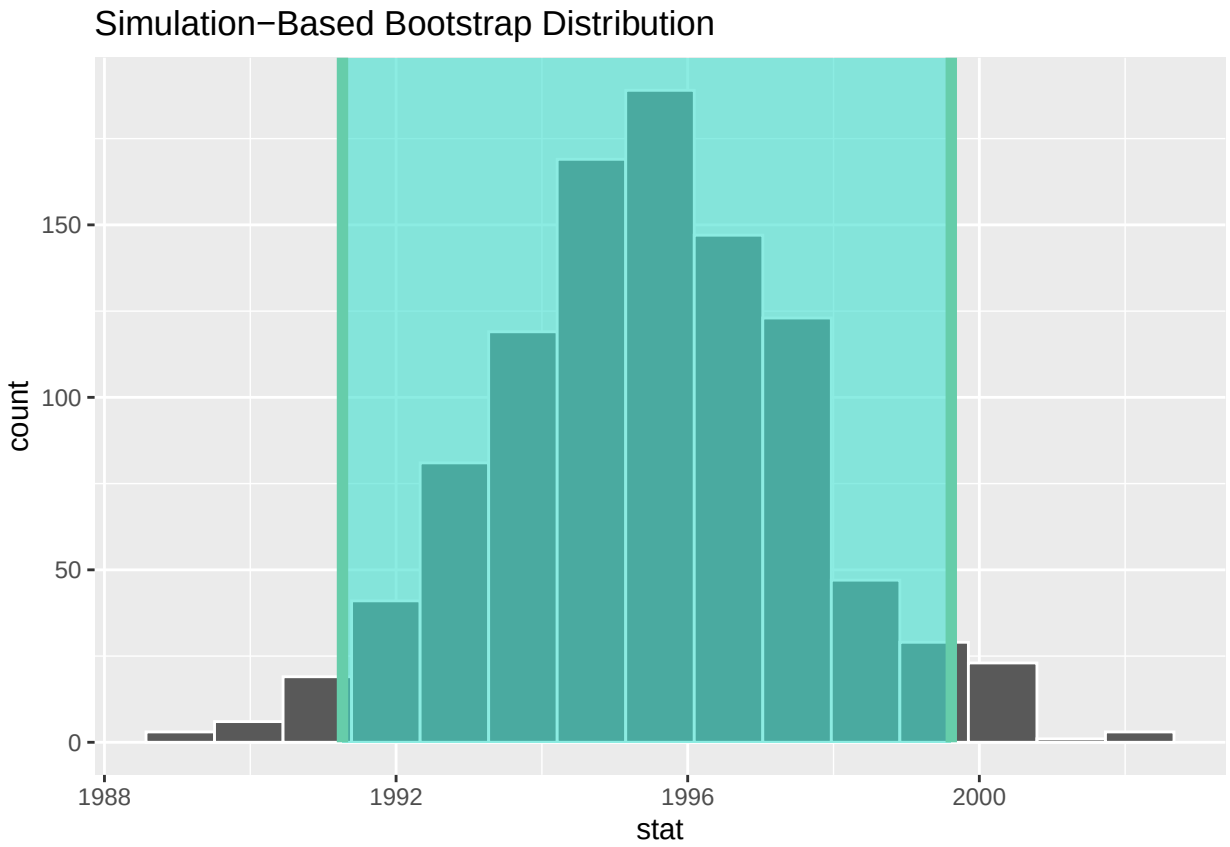
```
standard_error_ci <- bootstrap_distribution %>%
  get_confidence_interval(type = "se", point_estimate = x_bar)

standard_error_ci
```

```
## # A tibble: 1 x 2
##   lower_ci upper_ci
##      <dbl>    <dbl>
## 1    1991.    2000.
```

Visualize the confidence interval:

```
visualize(bootstrap_distribution) +
  shade_confidence_interval(endpoints = standard_error_ci) # remember shade_ci too
```



Both methods produce similar confidence intervals.

1.5 Interpreting confidence intervals

In order to interpret a confidence interval's effectiveness, we need to know what the value of the population parameter is. That way we can say whether or not a confidence interval "captured" this value.

What proportion of the bowl's 2400 balls are red?

```
bowl %>%
  summarize(p_red = mean(color == "red"))
```

```
## # A tibble: 1 x 1
##   p_red
##   <dbl>
## 1 0.375
```

In this case, we know what the value of the population parameter is: we know that the population proportion p is 0.375 (37.5% of the balls are red). In real life, we won't know what the true value of the population parameter is, hence the need for estimation.

1.5.1 Did the net capture the fish?

Recall that we had 33 groups of friends each take samples of size 50 from the bowl and then compute the sample proportion of red balls. Let's focus on Ilyas and Yohan's sample, which

is saved in the `bowl_sample_1` of `moderndive` package.

```
glimpse(bowl_sample_1)
```

```
## Rows: 50
## Columns: 1
## $ color <chr> "white", "white", "red", "red", "white", "white", "red", "whi...
```

```
head(bowl_sample_1, 10)
```

```
## # A tibble: 10 x 1
##    color
##    <chr>
##  1 white
##  2 white
##  3 red
##  4 red
##  5 white
##  6 white
##  7 red
##  8 white
##  9 white
## 10 white
```

They observed 21 red balls out of 50 and thus their sample proportion was $21/50 = 0.42 = 42\%$.

1. `specify` variables

```
# We need to define which event is of interest! red or white in the success argument.
bowl_sample_1 %>%
  specify(response = color, success = "red")
```

```
Response: color (factor)
# A tibble: 50 x 1
   color
   <fct>
 1 white
 2 white
 3 red
 4 red
 5 white
 6 white
 7 red
 8 white
 9 white
10 white
# … with 40 more rows
```

2. `generate` replicates

1000 replicates of bootstrap resampling with replacement from bowl_sample_1

```
bowl_sample_1 %>%
  specify(response = color, success = "red") %>%
  generate(reps = 1000, type = "bootstrap")
```

```
Response: color (factor)
# A tibble: 50,000 x 2
# Groups:   replicate [1,000]
   replicate color
       <int> <fct>
 1         1 white
 2         1 white
 3         1 white
 4         1 white
 5         1 red
 6         1 white
 7         1 white
 8         1 white
 9         1 white
10         1 red
# … with 49,990 more rows
```

The variable `replicate` indicates which resample each row belongs to. So it has the value `1` 50 times, the value `2` 50 times, all the way through to the value `1000` 50 times.

3. `calculate` summary statistics

We summarize each of the 1000 resamples of size 50 with the proportion of successes (red balls). We can set the summary statistic to be calculated as the proportion by setting the `stat` argument to be `"prop"`.

```r
#  Let's save the result as sample_1_bootstrap:
sample_1_bootstrap <- bowl_sample_1 %>%
  specify(response = color, success = "red") %>%
  generate(reps = 1000, type = "bootstrap") %>%
  calculate(stat = "prop")

glimpse(sample_1_bootstrap)
```

```
## Rows: 1,000
## Columns: 2
## $ replicate <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17...
## $ stat      <dbl> 0.46, 0.28, 0.36, 0.38, 0.40, 0.56, 0.46, 0.52, 0.36, 0.5...
```

```
head(sample_1_bootstrap, 10)
```

```
## # A tibble: 10 x 2
##    replicate  stat
##        <int> <dbl>
##  1         1  0.46
##  2         2  0.28
##  3         3  0.36
##  4         4  0.38
##  5         5  0.4
##  6         6  0.56
##  7         7  0.46
##  8         8  0.52
##  9         9  0.36
## 10        10  0.54
```

4. `visualize` the results

Let's compute the resulting 95% confidence interval:

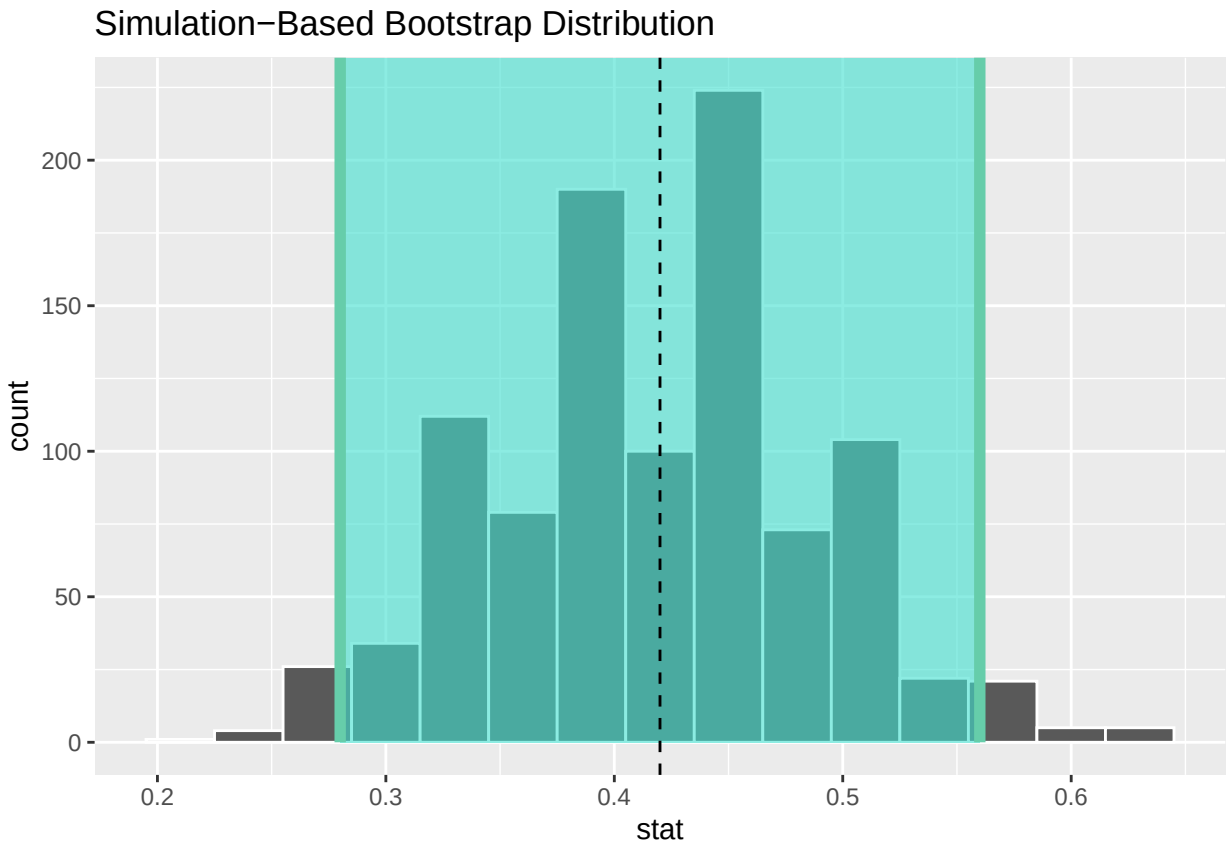
```
percentile_ci_1 <- sample_1_bootstrap %>%
  get_confidence_interval(level = 0.95, type = "percentile")

percentile_ci_1
```

```
## # A tibble: 1 x 2
##   lower_ci upper_ci
##      <dbl>    <dbl>
## 1     0.28     0.56
```

We'll use `geom_vline()` and add a dashed vertical line at Ilyas and Yohan's observed $\hat{p}$=21/50=0.42=42%.

```
sample_1_bootstrap %>%
  visualize(bins = 15) +
  shade_confidence_interval(endpoints = percentile_ci_1) +
  geom_vline(xintercept = 0.42, linetype = "dashed")
```



Did their 95% confidence interval for p based on their sample contain the true value of p of 0.375? Yes! 0.375 is between the endpoints of their confidence interval (0.28, 0.56).

However, will every 95% confidence interval for p capture this value? if we had a different sample of 50 balls and constructed a different confidence interval, would it necessarily contain $p = 0.375$ as well? Let's first take a different sample from the bowl

```
bowl_sample_2 <- bowl %>% rep_sample_n(size = 50)

glimpse(bowl_sample_2)
```

```
## Rows: 50
## Columns: 3
## Groups: replicate [1]
## $ replicate <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ ball_ID   <int> 2183, 606, 336, 696, 1388, 1046, 1734, 720, 1100, 1709, 1...
## $ color     <chr> "red", "red", "white", "white", "white", "red", "red", "w...
```

```
head(bowl_sample_2, 10)
```

```
## # A tibble: 10 x 3
## # Groups:   replicate [1]
##    replicate ball_ID color
##        <int>   <int> <chr>
##  1         1    2183 red
```

```
##  2         1     606 red
##  3         1     336 white
##  4         1     696 white
##  5         1    1388 white
##  6         1    1046 red
##  7         1    1734 red
##  8         1     720 white
##  9         1    1100 white
## 10         1    1709 red
```

Let's repeate the same steps:

```
sample_2_bootstrap <- bowl_sample_2 %>%
  specify(response = color,
          success = "red") %>%
  generate(reps = 1000,
           type = "bootstrap") %>%
  calculate(stat = "prop")

glimpse(sample_2_bootstrap)
```

```
## Rows: 1,000
## Columns: 2
## $ replicate <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17...
## $ stat      <dbl> 0.40, 0.38, 0.32, 0.38, 0.34, 0.44, 0.38, 0.34, 0.30, 0.3...
```

```
head(sample_2_bootstrap, 10)
```

```
## # A tibble: 10 x 2
##    replicate  stat
##        <int> <dbl>
##  1         1  0.4
##  2         2  0.38
##  3         3  0.32
##  4         4  0.38
##  5         5  0.34
##  6         6  0.44
##  7         7  0.38
##  8         8  0.34
##  9         9  0.3
## 10        10  0.32
```

Compute a percentile-based 95% confidence interval for p:

```
percentile_ci_2 <- sample_2_bootstrap %>%
  get_confidence_interval(level = 0.95, type = "percentile")

percentile_ci_2
```

```
## # A tibble: 1 x 2
##   lower_ci upper_ci
##      <dbl>    <dbl>
## 1     0.24      0.5
```

Again, the 95% confidence interval for p contain the true value of p of 0.375 (CI 0.3, 0.58).

1.5.2 Wrong, precise and shorthand interpretation

Wrong interpretation: "There is a 95% probability that the confidence interval contains p". Why wrong? because each confidence interval either does or does not contain p.

Precise interpretation: If we repeated our sampling procedure a large number of times, we expect about 95% of the resulting confidence intervals to capture the value of the population parameter.

Short-hand interpretation: We are 95% "confident" that a 95% confidence interval captures the value of the population parameter.

Back to our example, we are 95% "confident" that the true mean year of pennies in circulation in 2019 is somewhere between 1991.12 and 1999.36.

1.5.2.1 Extra key points:

The bootstrap distribution will likely not have the same center as the sampling distribution. In other words, bootstrapping cannot improve the quality of an estimate.

Even if the bootstrap distribution might not have the same center as the sampling distribution, it will likely have very similar shape and spread. In other words, bootstrapping will give you a good estimate of the standard error.