Class 33

DATA1220-55, Fall 2024

Sarah E. Grabinski

2024-11-25

Infer Package

- Functions for “tidy” statistical analysis
- Specify statistical models, calculate statistics
- Infer sampling distributions, test hypotheses
- Uses theoretical or permutation based null distributions

Tests in the Infer Package

- 1- or 2-sample proportion- or Z-tests
- 1- or 2-sample t-tests for means
- Chi-squared test of independence for categorical variables
- ANOVA test of independence for numeric variables
- Correlations and simple linear regression

Primary Functions

- `specify()`: set response variable (and explanatory, if needed)
- `calculate()`: calculate statistics
- `observe()`: combines `specify()` and `calculate()`
- `assume()`: sets a null distribution
- `hypothesize()`: sets a null hypothesis
- `get_ci()`: calculate a confidence interval from given distribution
- `visualize()`, `shade_p_value()`: visualize observed statistics vs null hypotheses
- `get_p_value()`: get p-value for observed statistic under null hypothesis

Packages for Today

We will be working with the `pennies` dataset from the `moderndive` package.

```
library(patchwork) # combining ggplot figures
library(GGally) # for ggpairs() plot matrix
library(Hmisc) # for describe function
library(moderndive) # contains pennies data
library(infer) # statistical functions
library(kableExtra) # for pretty tables
library(tidyverse) # always load last in list

theme_set(theme_bw()) # white background for ggplot2
```

Research Question

What is the average year of minting for pennies in circulation in the US in 2019?

Research Question

What is the average year of minting for pennies in circulation in the US in 2019?

Is it reasonable to gather up all the pennies in the US and get the average of the mint years?

Research Question

What is the average year of minting for pennies in circulation in the US in 2019?

Is it reasonable to gather up all the pennies in the US and get the average of the mint years?

No, we should probably take a sample of pennies and use it to draw inferences and test hypotheses regarding our study and target populations.

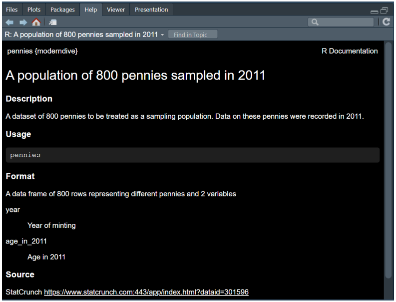
Sampling Methods

Figure 1: In 2011, Dr. Chester Ismay and Dr. Albert Y. Kim went to a local bank in Northampton, MA and requested all 800 pennies they had available.

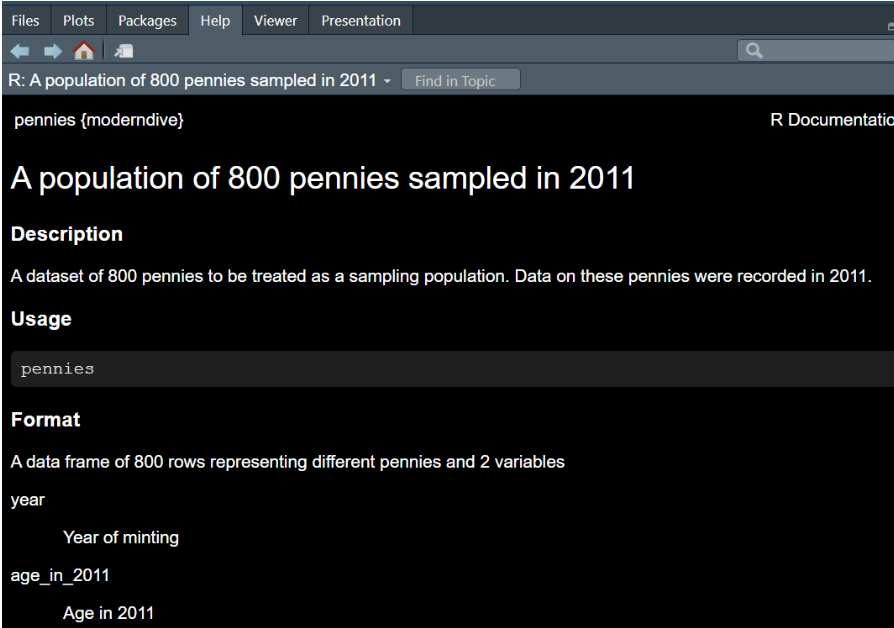
What's in the data?

Running a question mark before a dataset, function, or package name will do a search in RStudio for help pages on that topic.

```
?moderndive::pennies
```



What's in the data?



Files | Plots | Packages | Help | Viewer | Presentation

R: A population of 800 pennies sampled in 2011

Find in Topic

pennies {moderndive} R Documentatio

A population of 800 pennies sampled in 2011

Description

A dataset of 800 pennies to be treated as a sampling population. Data on these pennies were recorded in 2011.

Usage

```
pennies
```

Format

A data frame of 800 rows representing different pennies and 2 variables

year

Year of minting

age_in_2011

Age in 2011

A Peek at the Data

```r
# from the dplyr package
glimpse(pennies)
```

```
Rows: 800
Columns: 2
$ year        <int> 1986, 1996, 1994, 2008, 1999, 2010, 196
$ age_in_2011 <int> 25, 15, 17, 3, 12, 1, 47, 36, 16, 45, 1
```

Another Peek at the Data

```
# from base R
str(pennies)
```

```
tibble [800 x 2] (S3: tbl_df/tbl/data.frame)
 $ year       : int [1:800] 1986 1996 1994 2008 1999 2010 1
 $ age_in_2011: int [1:800] 25 15 17 3 12 1 47 36 16 45 ...
 - attr(*, "spec")=
  .. cols(
  ..   year = col_integer(),
  ..   age_in_2011 = col_integer()
  .. )
```

Codebook

- `year`: year that the penny was minted
- `age_in_2011`: the age of the penny in years in 2011

Exploratory Data Analysis

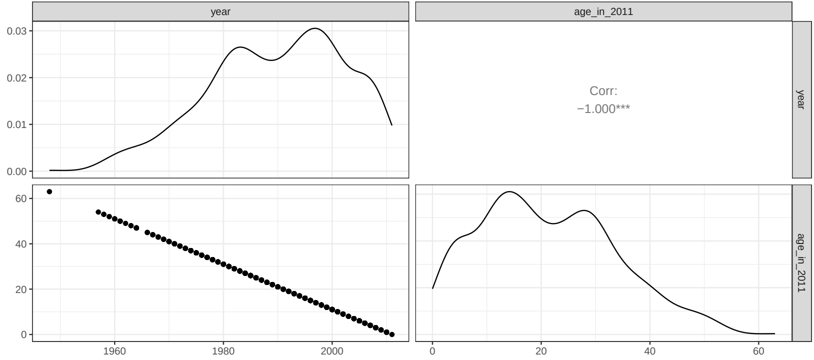
```
# from the Hmisc package
describe(pennies$year)
```

```
pennies$year
       n  missing distinct     Info     Mean      Gmd
     800        0       55    0.999     1990    14.16
     .25      .50      .75      .90      .95
    1981     1991     2000     2006     2008

lowest : 1948 1957 1958 1959 1960, highest: 2007 2008 2009
```

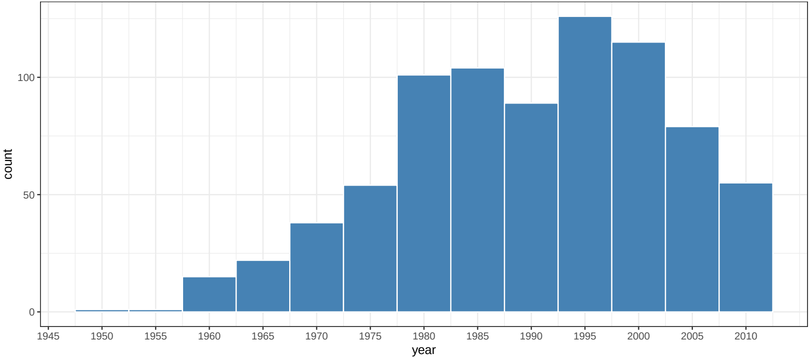
A Picture's Worth 1000 Words

```
# From the GGally package
ggpairs(pennies)
```



More Exploratory Data Analysis

What do you see in this population distribution?

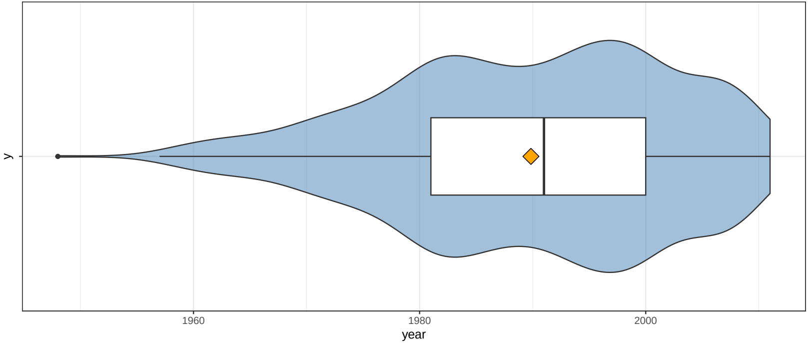


More Exploratory Data Analysis

```
pennies |>
  ggplot(aes(x = year)) +
  geom_histogram(binwidth = 5,
                 col = 'white',
                 fill = 'steelblue') +
  scale_x_continuous(breaks = seq(1945,
                                  2011,
                                  by = 5))
```

Keep it Going

Does this population seem normally distributed to you? Why or why not?



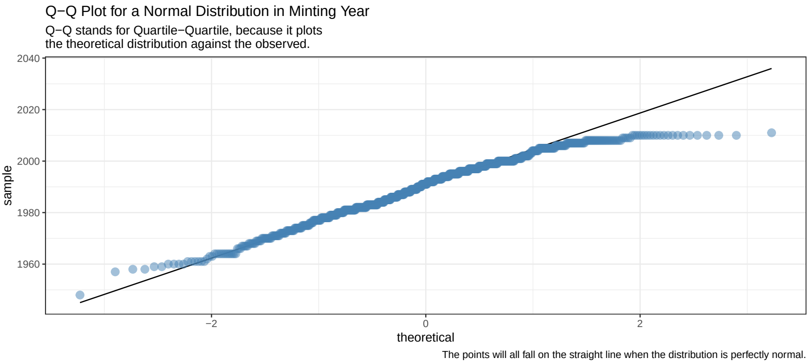
Orange diamond indicates the population mean.

Keep it Going

```
pennies |>
  ggplot(aes(x = year, y = '')) +
  geom_violin(fill = 'steelblue',
              alpha = 0.5) +
  geom_boxplot(width = 0.3) +
  stat_summary(fun = 'mean', geom = 'point',
               shape = 23, size = 5, fill = 'orange') +
  labs(caption = 'Orange diamond indicates the population r
```

Last One

If the points in your Q-Q plot don't follow the straight line closely, it may not be reasonable to assume that your population is normally distributed.



Last One

```
pennies |>
  ggplot(aes(sample = year)) +
  geom_qq_line() +
  geom_qq(col = 'steelblue', alpha = 0.5, size = 3) +
  labs(title = 'Q-Q Plot for a Normal Distribution in Mint
       subtitle = paste('Q-Q stands for Quartile-Quartile,
                        'the theoretical distribution agai
                        sep = '\n'),
       caption = 'The points will all fall on the straight
```

Calculating population parameters

You can use functions from the `kableExtra` package to turn your raw R outputs into attractive tables.

n	mean	SD
800	1989.8	12.4

Calculating population parameters

Find more info here: kableExtra Vignettes

```
# from the dplyr package
pennies |>
  summarize(
    n = n(),
    mean = mean(year),
    sd = sd(year)) |>
  kable(col.names = c('n', 'mean', 'SD'),
        digits = c(0, 1, 1)) |>
  kable_classic(full_width = F)
```

Storing population mean as a variable

Use the `pull()` function from `dplyr` to grab just the value(s) from a column, ditching the data frame component.

```
pop_mean <- pennies |>
  summarize(mean = mean(year)) |>
  pull(mean)

pop_mean
```

```
[1] 1989.848
```

Storing population standard deviation as a variable

Use the `pull()` function from `dplyr` to grab just the value(s) from a column, ditching the data frame component.

```
pop_sd <- pennies |>
  summarize(sd = sd(year)) |>
  pull(sd)

pop_sd
```

```
[1] 12.43956
```

Can we predict our sampling distribution?

```
# from the dplyr package
pennies |>
  summarize(
    n = 50,
    pop_mean = mean(year),
    sample_se = sd(year) / sqrt(50),
    lower95 = mean(year) - qt(0.975,
                              df = n()) * (sd(year) / sqrt(
    upper95 = mean(year) + qt(0.975,
                              df = n()) * (sd(year) / sqrt(
  kable(col.names = c('n', 'population mean',
                      'SE for n=50',
                      'Lower 95% CI', 'Upper 95% CI'),
        digits = c(0, 1, 1, 2, 1, 1)) |>
  kable_classic(full_width = F)
```

Can we predict our sampling distribution?

n	population mean	SE for n=50	Lower 95% CI	Upper 95% CI
50	1989.8	1.8	1986.39	1993.3

Can we predict our sampling distribution?

n	population mean	SE for n=50	Lower 95% CI	Upper 95% CI
50	1989.8	1.8	1986.39	1993.3

The sample means $\bar{x}$ from a sample of size $n = 50$ from this population should follow the distribution N$(1989.8, 1.76)$ if assumptions hold.

Storing sampling distribution estimates

Use the `pull()` function from `dplyr` to grab just the value(s) from a column, ditching the data frame component.

```
pop_low <- pennies |>
  summarize(lower95 = mean(year) - qt(0.975,
                                      df = n()) * (sd(year)
  pull(lower95)

pop_high <- pennies |>
  summarize(upper95 = mean(year) + qt(0.975,
                                      df = n()) * (sd(year)
  pull(upper95)
```

Sampling Method

The Sample


2002
1
1986
2
2017
3
1988
4
2008
5
1983
6
1994
11
1995
12
2015
13
1978
14
1974
15
2015
16
1981
21
1976
22
1998
23
2017
24
1979
25
1979
26
2013
31
1976
32
1979
33
1985
34
1985
35
2015
36

Taking a sample, pt. 1

When doing random processes in R, you need to use the `set.seed()` function and give it a number. This temporarily "fixes" the randomness so that the function generates the same set of numbers every time.

```
# store number of observations
n <- nrow(pennies)

# randomly sample row numbers/indexes
# set a seed to reproduce exact sample next time
set.seed(123); sample_rows <- sample(seq(1, n),
                                     size = 50)
sort(sample_rows)
```

```
 [1]  14  23  26  72  91 118 135 141 143 153 166 179 195 2
[20] 290 294 299 309 348 355 373 374 415 426 463 490 519 5
[39] 590 593 602 603 621 649 665 709 722 766 768 782
```

Taking a sample, pt. 1

When doing random processes in R, you need to use the `set.seed()` function and give it a number. This temporarily "fixes" the randomness so that the function generates the same set of numbers every time.

```
# store number of observations
n <- nrow(pennies)

# randomly sample row numbers/indexes
# set a seed to reproduce exact sample next time
set.seed(123); sample_rows <- sample(seq(1, n),
                                     size = 50)
sort(sample_rows)
```

```
 [1]  14  23  26  72  91 118 135 141 143 153 166 179 195 21
[20] 290 294 299 309 348 355 373 374 415 426 463 490 519 52
[39] 590 593 602 603 621 649 665 709 722 766 768 782
```

Taking a sample, pt. 2

Use the `filter()` function from the `dplyr` package to retain observations whose row number matches your generated sample.

```
# filter your table for rows whose
# number/index is in your sample list
sample1 <- pennies |>
  # function from dplyr package
  filter(row_number() %in% sample_rows)

nrow(sample1)
```

```
[1] 50
```

Or use base R to index the proper rows. The 1st parameter contains the row numbers to keep. The 2nd parameter contains the column numbers to keep. Leave blank if you want them all.

```
sample1 <- pennies[sample_rows, ]
```

What about another sample?

Set a different seed with the `set.seed()` function to generate a new (but reproducible) random sample.

```
# randomly sample row numbers/indexes
# set a seed to reproduce exact sample next time
set.seed(456); sample_rows2 <- sample(seq(1, n),
                                       size = 50)

# Select sampled rows through indexing
sample2 <- pennies[sample_rows2, ]
```

Resampling with Infer

You can use the `rep_sample_n()` function from the `infer` package to take `reps` number of repeated samples of a specified `size`, with (`replace = T`) or without (`replace = F`) replacement.

```
# don't forget to set a seed!
set.seed(789); samples <- pennies %>%
  rep_sample_n(size = 50,
               replace = TRUE,
               reps = 100)

head(samples)
```

```
# A tibble: 6 x 3
# Groups:   replicate [1]
  replicate  year age_in_2011
      <int> <int>       <int>
1         1  2007           4
2         1  1995          16
```

Summary Statistics & CIs

This function returns a grouped table, so you don't need to use the `.by =` parameter to group by replicate in the `summarize()` function.

replicate	n	point estimate	Lower 95% CI	Upper 95% CI
1	50	1993.9	1990.00	1997.8
2	50	1992.5	1989.33	1995.7
3	50	1988.3	1984.95	1991.7
4	50	1990.8	1987.71	1993.8
5	50	1989.9	1986.13	1993.6
6	50	1989.2	1985.90	1992.4

Summary Statistics & CIs

Complete the confidence interval calculations within the `summarize()` function.

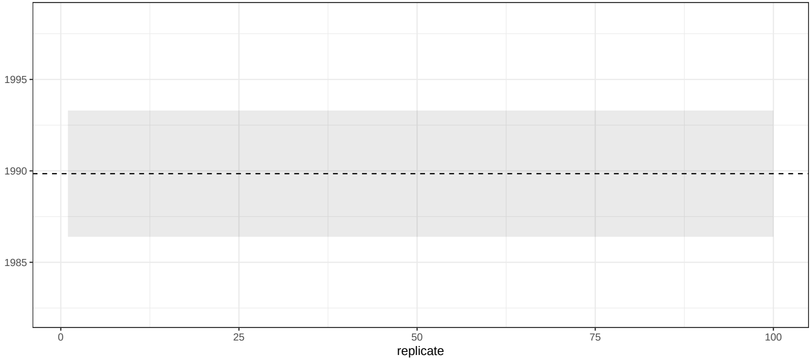
```
cis <- samples |>
  summarize(
    n = n(),
    mean = mean(year),
    s = sd(year),
    lower95 = mean(year) - qt(0.975,
                              df = n()) * (sd(year) / sqrt(
    upper95 = mean(year) + qt(0.975,
                              df = n()) * (sd(year) / sqrt(
```

Summary Statistics & CIs

Remember, the parameters were a mean year of 1989.8 and a standard deviation of 12.4 years in a population of 800. How do these compare?

```
cis |>
  head() |>
  kable(col.names = c('replicate', 'n', 'point estimate',
                      'Lower 95% CI', 'Upper 95% CI'),
        digits = c(0, 1, 1, 2, 1, 1), align = 'c') |>
  kable_classic(full_width = F)
```

Visualizing the CI's

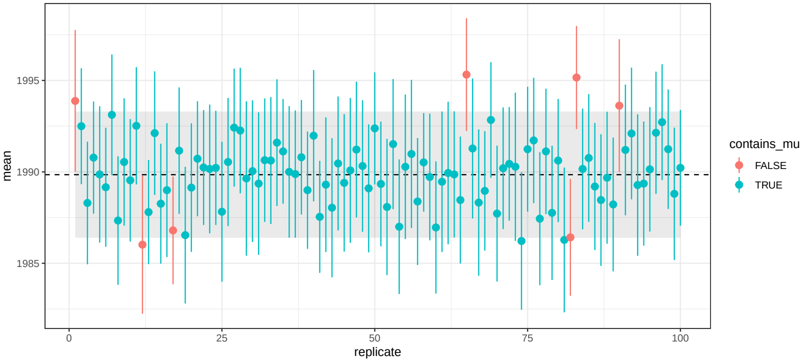


Visualizing the CI's

```
p1 <- cis |>
  # from the dplyr package, for modifying data
  mutate(contains_mu = ifelse(
    pop_mean >= lower95 &  #conditional statement
      pop_mean <= upper95,
    T, F)) |> # value to return if true, if false
  ggplot(aes(x = replicate)) +
  geom_ribbon(aes(ymin = pop_low,
                  ymax = pop_high),
              alpha = 0.1) +
  geom_hline(yintercept = pop_mean,
             linetype = 'dashed') +
  coord_cartesian(ylim = c(min(cis$lower95),
                           max(cis$upper95)))

p1
```

Visualizing the CI's

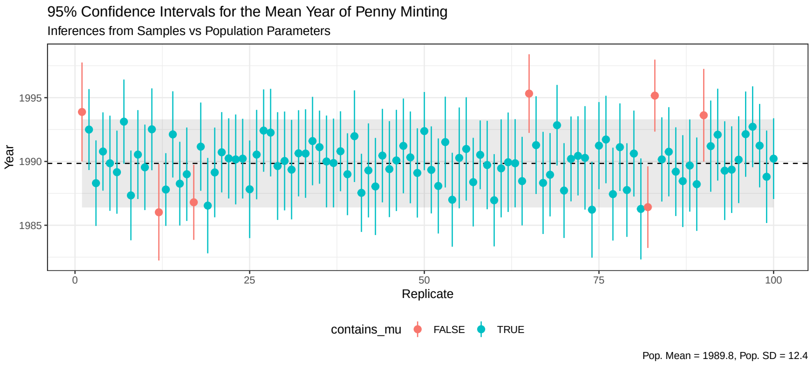


Visualizing the CI's

```
p2 <- p1 +
  geom_pointrange(aes(y = mean,
                      ymin = lower95,
                      ymax = upper95,
                      col = contains_mu,
                      fill = contains_mu))

p2
```

Visualizing the CI's

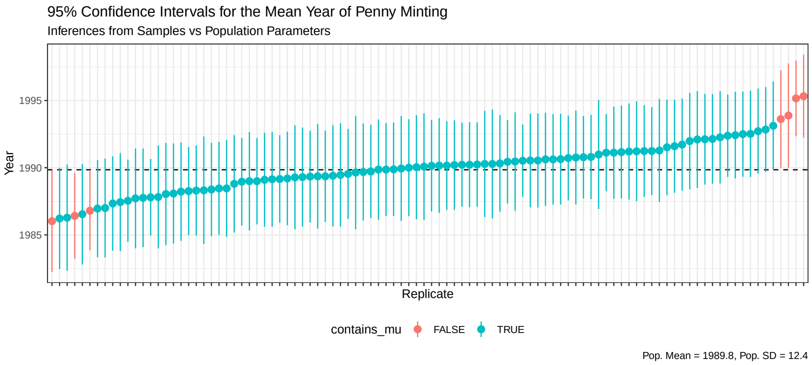


Visualizing the CI's

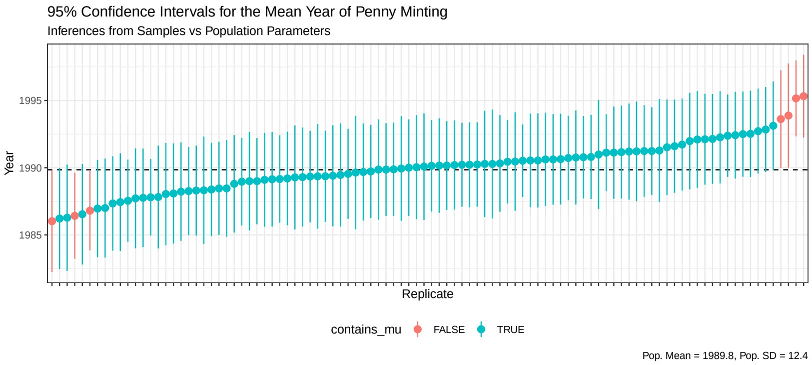
```
p3 <- p2 +
  theme(legend.position = 'bottom') +
  labs(title = '95% Confidence Intervals for the Mean Year
       subtitle = 'Inferences from Samples vs Population Pa
       caption = 'Pop. Mean = 1989.8, Pop. SD = 12.4',
       x = 'Replicate', y = 'Year')

p3
```

All Together



Reordered



95% Confidence Intervals for the Mean Year of Penny Minting
Inferences from Samples vs Population Parameters
Year
1995
1990
1985
Replicate
contains_mu
FALSE
TRUE
Pop. Mean = 1989.8, Pop. SD = 12.4

Reordered

```
cis |>
  mutate(contains_mu = ifelse(
    pop_mean >= lower95 &
      pop_mean <= upper95, T, F)) |>
  ggplot(aes(x = reorder(replicate, mean))) +
  geom_ribbon(aes(ymin = pop_low, ymax = pop_high),
              alpha = 0.1) +
  geom_hline(yintercept = pop_mean,
             linetype = 'dashed') +
  coord_cartesian(ylim = c(min(cis$lower95),
                           max(cis$upper95))) +
  geom_pointrange(aes(y = mean, ymin = lower95, ymax = uppe
                      col = contains_mu, fill = contains_mu
  theme(legend.position = 'bottom',
        axis.text.x = element_blank()) +
  labs(title = '95% Confidence Intervals for the Mean Year
       subtitle = 'Inferences from Samples vs Population Pa
       caption = 'Pop. Mean = 1989.8, Pop. SD = 12.4',
```

Misinterpreting Confidence Intervals

- Taking a sample and constructing a confidence interval are considered random processes

Misinterpreting Confidence Intervals

- Taking a sample and constructing a confidence interval are considered random processes
- Confidence is an expression of probability regarding the PROCESS, not the result

Misinterpreting Confidence Intervals

- Taking a sample and constructing a confidence interval are considered random processes
- Confidence is an expression of probability regarding the PROCESS, not the result
- The confidence level is a pre-analysis statement about the process, not a post-analysis statement about the result

Misinterpreting Confidence Intervals

- Taking a sample and constructing a confidence interval are considered random processes
- Confidence is an expression of probability regarding the PROCESS, not the result
- The confidence level is a pre-analysis statement about the process, not a post-analysis statement about the result
- 95% confidence means that 95% of the time, confidence intervals constructed using samples of size n from this population will contain the population parameter...

Misinterpreting Confidence Intervals

- Taking a sample and constructing a confidence interval are considered random processes
- Confidence is an expression of probability regarding the PROCESS, not the result
- The confidence level is a pre-analysis statement about the process, not a post-analysis statement about the result
- 95% confidence means that 95% of the time, confidence intervals constructed using samples of size n from this population will contain the population parameter...
- ***...meaning there's a 5% chance (alpha) that your confidence interval does NOT contain the population parameter no matter how good your data is!***

Misinterpreting Confidenc Intervals

- A “true” population parameter is unknownable. You should not be using this terminology to interpret your results.

Misinterpreting Confidenc Intervals

- A “true” population parameter is unknownable. You should not be using this terminology to interpret your results.
- If you have made reasonable assumptions, theory suggests (and one hopes) the point estimate or sample statistic will be close to that “true” population parameter we can’t know.

Misinterpreting Confidenc Intervals

- A "true" population parameter is unknownable. You should not be using this terminology to interpret your results.
- If you have made reasonable assumptions, theory suggests (and one hopes) the point estimate or sample statistic will be close to that "true" population parameter we can't know.
- However, if any assumptions are violated, the sample statistics may no longer be accurate estimators of the population parameters.

Misinterpreting Confidenc Intervals

- A "true" population parameter is unknownable. You should not be using this terminology to interpret your results.
- If you have made reasonable assumptions, theory suggests (and one hopes) the point estimate or sample statistic will be close to that "true" population parameter we can't know.
- However, if any assumptions are violated, the sample statistics may no longer be accurate estimators of the population parameters.
- The confidence level is not the probability that YOUR confidence interval contains the population parameter.

Activity

Interpreting confidence intervals: an interactive activity:
https://rpsychologist.com/d3/ci/

Questions

1. What happens to the intervals as you change the confidence level?
2. What happens to the intervals as you change the sample size?
3. What concept from Chapter 3 does the left-middle plot remind you of?
4. Is the confidence interval width normally distributed?
5. How has your understanding of confidence intervals and their interpretation changed?