APSTA-GE 2352

Statistical Computing: Lecture 11

Klint Kanopka

New York University

Table of Contents

Check-In

- PollEv.com/klintkanopka

Announcements

- PS6 is posted!
  - It's your 2nd to last pset
  - It's shorter than PS5
- Lab on Wednesday before Thanksgiving is cancelled
- After today:
  - 3 more lectures
  - 3 more labs
  - 2 more psets
- Reminder about next week's PRIISM seminar talk with my friend Charlie Rahal from Oxford!

Advanced Debugging

The tools we have so far

The tools we have so far

- Googling error messages

The tools we have so far

- Googling error messages
- `print()` statements

The tools we have so far

- Googling error messages
- `print()` statements
- `traceback()`

The tools we have so far

- Googling error messages
- `print()` statements
- `traceback()`
- Any others?

What about generative AI?

What about generative AI?

- I honestly don't know much about debugging code with tools like ChatGPT

What about generative AI?

- I honestly don't know much about debugging code with tools like ChatGPT
- What's your experience been with using ChatGPT to help you debug or write code?

What about generative AI?

- I honestly don't know much about debugging code with tools like ChatGPT

- What's your experience been with using ChatGPT to help you debug or write code?

- Are there other models/tools that you use?

What about generative AI?

- I honestly don't know much about debugging code with tools like ChatGPT

- What's your experience been with using ChatGPT to help you debug or write code?

- Are there other models/tools that you use?

- What prompting strategies do you use?

What about generative AI?

- I honestly don't know much about debugging code with tools like ChatGPT

- What's your experience been with using ChatGPT to help you debug or write code?

- Are there other models/tools that you use?

- What prompting strategies do you use?

- Do you feel like using generative AI to help with coding tasks increases your understanding?

What about generative AI?

- I honestly don't know much about debugging code with tools like ChatGPT
- What's your experience been with using ChatGPT to help you debug or write code?
- Are there other models/tools that you use?
- What prompting strategies do you use?
- Do you feel like using generative AI to help with coding tasks increases your understanding?
- Any other notes?

A more powerful debugging tool: `browser()`

- `browser()` lets you step into the code and execute it line by line while you monitor what is loaded in each environment and the values of intermediate objects
- You can further step into functions you encounter along the way
- How does it work?
    - You insert the `browser()` function call into some code you want to debug
    - From here, the right pane shows you the values and variables for the environment you're currently executing code in
    - You get a special prompt in the console, `Browse[1]>` to let you know you're doing browser stuff
    - All execution halts and the next line to be run appears above the prompt

Using `browser()`

- First, you can use the `Browse[1]>` prompt like any other sort of console prompt and execute code from there

- Second, and more importantly, you get access to a bunch of new commands:

  1. `n` runs the **next** line of code (whatever is currently above the prompt)

  2. `s` is like next, but if the next line is a function, you'll **step into** it and run it interactively (line-by-line)

  3. `c` stops running code line-by-line and **continues** executing the current function you're currently in

  4. `Q` **quits** out of the browser

Why and how to use `browser()`?

- Essentially it's not too far off from inserting `print()` statements after every single line

- It allows you to see what changes after each line of code is run

- You can also insert code into a function to see if it fixes your problem

- A few other ways to use it:

  - Running `debug(FUN())` inserts `browser()` into the first line of `FUN()`, and so running `FUN()` will *always* open the browser. Stop this with `undebug(FUN())`

  - `debugonce(FUN(args))` runs `FUN(args)` immediately opening a browser, but doesn't modify `FUN()`

Why and how to use `browser()`?

- Essentially it's not too far off from inserting `print()` statements after every single line

- It allows you to see what changes after each line of code is run

- You can also insert code into a function to see if it fixes your problem

- A few other ways to use it:

  - Running `debug(FUN())` inserts `browser()` into the first line of `FUN()`, and so running `FUN()` will *always* open the browser. Stop this with `undebug(FUN())`

  - `debugonce(FUN(args))` runs `FUN(args)` immediately opening a browser, but doesn't modify `FUN()`

- **Next let's use** `browser()` **with the activity from lab yesterday**

The `apply()` Family

Side Effects

- If a function or operation modifies things outside its local environment, it has "side effects"

- These side effects can be hard to observe, but cause serious errors in analysis!

- Side effects most often occur when we are trying to hack together a solution to a problem

  - Functions that take and modify information from the global environment

  - Janky loops

Avoiding Side Effects

- Writing things in functions is a good start!
  - Calls to functions generate new environments
  - New environments protect data in the global environment
- We can "hide loops"
  - The `apply()` family of functions does this really well
  - These nests the entire loop we want to carry out inside of a function call

The `apply()` Family

- `apply()` takes a function and applies it to each element of a data object

- Whole bunch of different `apply()` functions

  - Different functions take different data objects as input

  - Different functions spit out different data objects

- Note that `apply()` functions aren't really any faster than loops!

  - They are not vectorized

  - They are, however, very easy to *parallelize*

The main `apply()` functions

function	input	output	comment
`apply()`	matrix or array	vector or array or list	
`lapply()`	list or vector	list	
`sapply()`	list or vector	vector or matrix or list	simplify
`vapply()`	list or vector	vector or matrix or list	safer simplify
`tapply()`	data, categories	array or list	ragged
`mapply()`	lists and/or vectors	vector or matrix or list	multiple

Applying `apply()`

Uncertainty in mean estimates

- First we conduct a simulation study to observe how the uncertainty in estimating the mean of a normal distribution depends on the number of samples we draw from it

```
1  # TODO
```

Uncertainty in mean estimates

- First we conduct a simulation study to observe how the uncertainty in estimating the mean of a normal distribution depends on the number of samples we draw from it

```
1  Ns <- seq(from = 50, to = 1000, by = 50)
```

Uncertainty in mean estimates

- First we conduct a simulation study to observe how the uncertainty in estimating the mean of a normal distribution depends on the number of samples we draw from it

```
1  Ns <- seq(from = 50, to = 1000, by = 50)
2  M <- 1000
```

Uncertainty in mean estimates

- First we conduct a simulation study to observe how the uncertainty in estimating the mean of a normal distribution depends on the number of samples we draw from it

```
1  Ns <- seq(from = 50, to = 1000, by = 50)
2  M <- 1000
3  means <- list(length = length(Ns))
```

Uncertainty in mean estimates

- First we conduct a simulation study to observe how the uncertainty in estimating the mean of a normal distribution depends on the number of samples we draw from it

```
1   Ns <- seq(from = 50, to = 1000, by = 50)
2   M <- 1000
3   means <- list(length = length(Ns))
4
5   for (i in seq_along(Ns)) {
6
7   }
```

Uncertainty in mean estimates

- First we conduct a simulation study to observe how the uncertainty in estimating the mean of a normal distribution depends on the number of samples we draw from it

```
1  Ns <- seq(from = 50, to = 1000, by = 50)
2  M <- 1000
3  means <- list(length = length(Ns))
4
5  for (i in seq_along(Ns)) {
6    for (j in 1:M) {
7
8    }
9  }
```

Uncertainty in mean estimates

- First we conduct a simulation study to observe how the uncertainty in estimating the mean of a normal distribution depends on the number of samples we draw from it

```r
1  Ns <- seq(from = 50, to = 1000, by = 50)
2  M <- 1000
3  means <- list(length = length(Ns))
4
5  for (i in seq_along(Ns)) {
6    means[[i]] <- vector(length = M)
7    for (j in 1:M) {
8
9    }
10  }
```

Uncertainty in mean estimates

- First we conduct a simulation study to observe how the uncertainty in estimating the mean of a normal distribution depends on the number of samples we draw from it

```
1  Ns <- seq(from = 50, to = 1000, by = 50)
2  M <- 1000
3  means <- list(length = length(Ns))
4
5  for (i in seq_along(Ns)) {
6    means[[i]] <- vector(length = M)
7    for (j in 1:M) {
8      means[[i]][j] <- mean(rnorm(Ns[i]))
9    }
10 }
```

Uncertainty in mean estimates

- First we conduct a simulation study to observe how the uncertainty in estimating the mean of a normal distribution depends on the number of samples we draw from it

```
 1  Ns <- seq(from = 50, to = 1000, by = 50)
 2  M <- 1000
 3  means <- list(length = length(Ns))
 4
 5  for (i in seq_along(Ns)) {
 6    means[[i]] <- vector(length = M)
 7    for (j in 1:M) {
 8      means[[i]][j] <- mean(rnorm(Ns[i]))
 9    }
10  }
11
12  d_mean <- data.frame()
```

Uncertainty in mean estimates

- First we conduct a simulation study to observe how the uncertainty in estimating the mean of a normal distribution depends on the number of samples we draw from it

```
1   Ns <- seq(from = 50, to = 1000, by = 50)
2   M <- 1000
3   means <- list(length = length(Ns))
4
5   for (i in seq_along(Ns)) {
6     means[[i]] <- vector(length = M)
7     for (j in 1:M) {
8       means[[i]][j] <- mean(rnorm(Ns[i]))
9     }
10  }
11
12  d_mean <- data.frame(n = , mean = )
```

Uncertainty in mean estimates

- First we conduct a simulation study to observe how the uncertainty in estimating the mean of a normal distribution depends on the number of samples we draw from it

```
1   Ns <- seq(from = 50, to = 1000, by = 50)
2   M <- 1000
3   means <- list(length = length(Ns))
4
5   for (i in seq_along(Ns)) {
6     means[[i]] <- vector(length = M)
7     for (j in 1:M) {
8       means[[i]][j] <- mean(rnorm(Ns[i]))
9     }
10  }
11
12  d_mean <- data.frame(n = , mean = do.call(c, means))
```
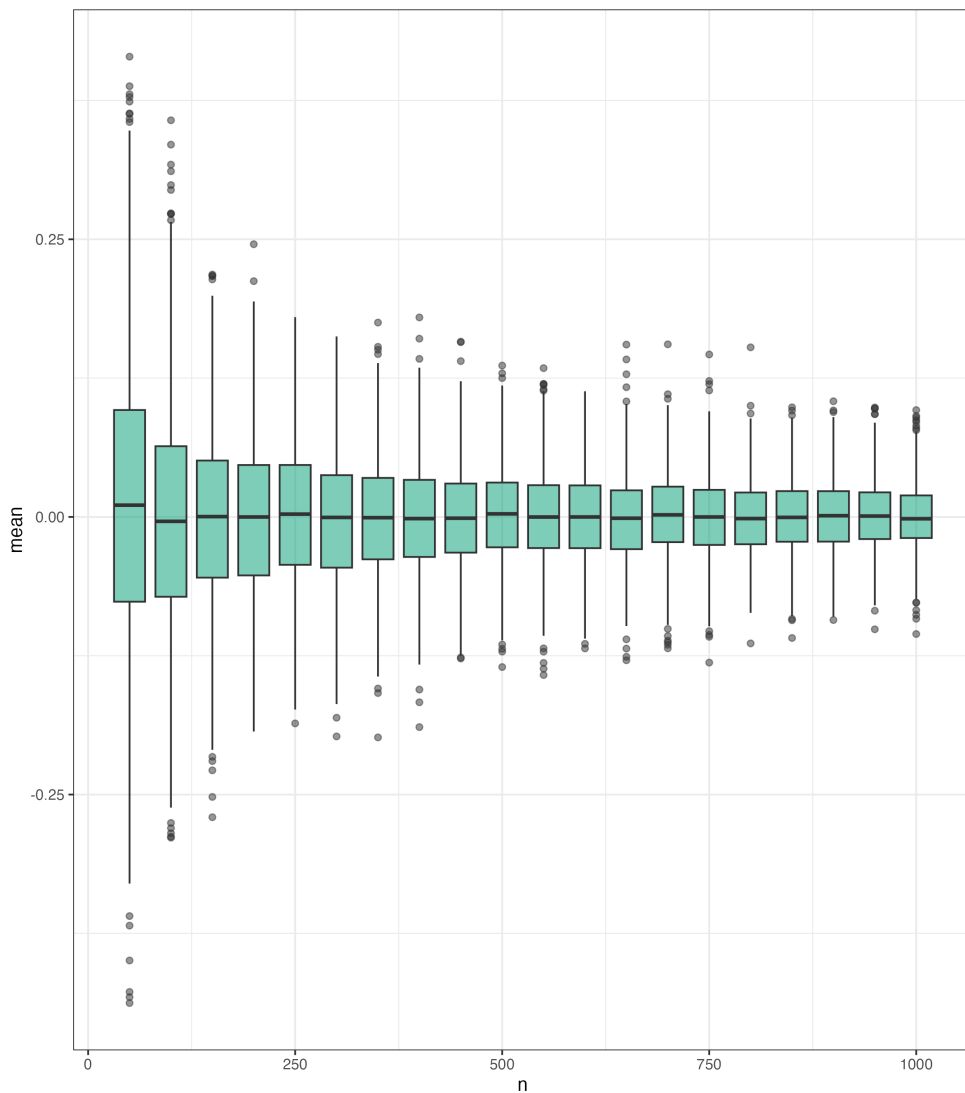
Uncertainty in mean estimates

- First we conduct a simulation study to observe how the uncertainty in estimating the mean of a normal distribution depends on the number of samples we draw from it

```r
 1  Ns <- seq(from = 50, to = 1000, by = 50)
 2  M <- 1000
 3  means <- list(length = length(Ns))
 4
 5  for (i in seq_along(Ns)) {
 6    means[[i]] <- vector(length = M)
 7    for (j in 1:M) {
 8      means[[i]][j] <- mean(rnorm(Ns[i]))
 9    }
10  }
11
12  d_mean <- data.frame(n = rep(Ns, each = M), mean = do.call(c, means))
```

Visualizing uncertainty in mean estimates

```
1   ggplot(d_mean, aes(x = n,
2                      y = mean,
3                      group = n)) +
4   geom_boxplot(alpha = 0.5,
5                fill = okabeito_colors(3)) +
6   theme_bw()
```

Estimating uncertainty directly

- Next we visualize our estimates of the standard errors of these sampling distributions at each sample size

```
1   # TODO
```

Estimating uncertainty directly

- Next we visualize our estimates of the standard errors of these sampling distributions at each sample size

```
1  sds <- vector(length = length(Ns))
```

Estimating uncertainty directly

- Next we visualize our estimates of the standard errors of these sampling distributions at each sample size

```r
1  sds <- vector(length = length(Ns))
2
3  for (i in seq_along(Ns)) {
4
5  }
```

Estimating uncertainty directly

- Next we visualize our estimates of the standard errors of these sampling distributions at each sample size

```r
1  sds <- vector(length = length(Ns))
2
3  for (i in seq_along(Ns)) {
4    sds[i] <- sd(means[[i]])
5  }
```

Estimating uncertainty directly

- Next we visualize our estimates of the standard errors of these sampling distributions at each sample size

```r
1  sds <- vector(length = length(Ns))
2
3  for (i in seq_along(Ns)) {
4    sds[i] <- sd(means[[i]])
5  }
6
7  d_sd <- data.frame()
```

Estimating uncertainty directly

- Next we visualize our estimates of the standard errors of these sampling distributions at each sample size

```r
1  sds <- vector(length = length(Ns))
2
3  for (i in seq_along(Ns)) {
4    sds[i] <- sd(means[[i]])
5  }
6
7  d_sd <- data.frame(n = , sd = )
```

Estimating uncertainty directly

- Next we visualize our estimates of the standard errors of these sampling distributions at each sample size

```
1  sds <- vector(length = length(Ns))
2
3  for (i in seq_along(Ns)) {
4    sds[i] <- sd(means[[i]])
5  }
6
7  d_sd <- data.frame(n = , sd = sds)
```

Estimating uncertainty directly

- Next we visualize our estimates of the standard errors of these sampling distributions at each sample size

```r
1  sds <- vector(length = length(Ns))
2
3  for (i in seq_along(Ns)) {
4    sds[i] <- sd(means[[i]])
5  }
6
7  d_sd <- data.frame(n = Ns, sd = sds)
```

Visualizing uncertainty estimates

```r
1   ggplot(d_sd, aes(x = n, y = sd)) +
2     geom_point(color = okabeito_colors(2)) +
3     geom_line(color = okabeito_colors(2)) +
4     theme_bw()
```
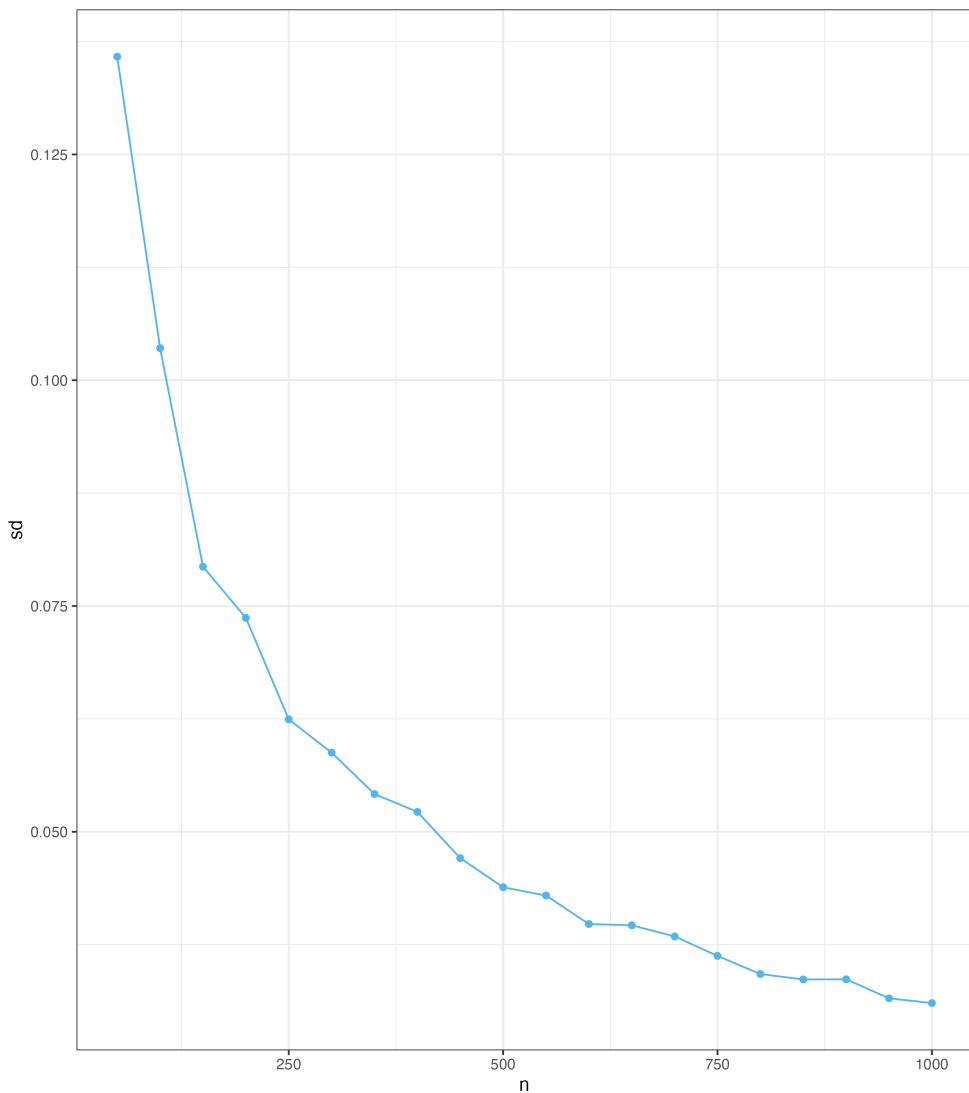
Uncertainty in mean estimates with `apply()`

- Now we redo the first simulation with something from the `apply()` family

```
1  # TODO: Write a simulation function
2
3  # TODO: Construct the object to apply() over
4
5  # TODO: Construct the output
```

Uncertainty in mean estimates with `apply()`

- Now we redo the first simulation with something from the `apply()` family

```r
1  # TODO: Write a simulation function
2
3  SimFun1 <- function() {
4
5  }
6
7  # TODO: Construct the object to apply() over
8
9  # TODO: Construct the output
```

Uncertainty in mean estimates with `apply()`

- Now we redo the first simulation with something from the `apply()` family

```
1   # TODO: Write a simulation function
2
3   SimFun1 <- function() {
4     return(m)
5   }
6
7   # TODO: Construct the object to apply() over
8
9   # TODO: Construct the output
```

Uncertainty in mean estimates with `apply()`

- Now we redo the first simulation with something from the `apply()` family

```r
1   # TODO: Write a simulation function
2
3   SimFun1 <- function(n_draws) {
4      draws <- rnorm(n_draws)
5      return(m)
6   }
7
8   # TODO: Construct the object to apply() over
9
10  # TODO: Construct the output
```

Uncertainty in mean estimates with `apply()`

- Now we redo the first simulation with something from the `apply()` family

```
1   # TODO: Write a simulation function
2
3   SimFun1 <- function(n_draws) {
4     draws <- rnorm(n_draws)
5     m <- mean(draws)
6     return(m)
7   }
8
9   # TODO: Construct the object to apply() over
10
11  # TODO: Construct the output
```

Uncertainty in mean estimates with `apply()`

- Now we redo the first simulation with something from the `apply()` family

```r
1   # TODO: Write a simulation function
2
3   SimFun1 <- function(n_draws) {
4     draws <- rnorm(n_draws)
5     m <- mean(draws)
6     return(m)
7   }
8
9   # TODO: Construct the object to apply() over
10
11  n <- rep(Ns, each = M)
12
13  # TODO: Construct the output
```

Uncertainty in mean estimates with `apply()`

- Now we redo the first simulation with something from the `apply()` family

```r
1   # TODO: Write a simulation function
2
3   SimFun1 <- function(n_draws) {
4     draws <- rnorm(n_draws)
5     m <- mean(draws)
6     return(m)
7   }
8
9   # TODO: Construct the object to apply() over
10
11  n <- rep(Ns, each = M)
12
13  # TODO: Construct the output
14
15  d_mean_apply <- data.frame()
```

Uncertainty in mean estimates with `apply()`

- Now we redo the first simulation with something from the `apply()` family

```r
1   # TODO: Write a simulation function
2
3   SimFun1 <- function(n_draws) {
4     draws <- rnorm(n_draws)
5     m <- mean(draws)
6     return(m)
7   }
8
9   # TODO: Construct the object to apply() over
10
11  n <- rep(Ns, each = M)
12
13  # TODO: Construct the output
14
15  d_mean_apply <- data.frame(n = , mean = )
```

Uncertainty in mean estimates with `apply()`

- Now we redo the first simulation with something from the `apply()` family

```r
1   # TODO: Write a simulation function
2
3   SimFun1 <- function(n_draws) {
4     draws <- rnorm(n_draws)
5     m <- mean(draws)
6     return(m)
7   }
8
9   # TODO: Construct the object to apply() over
10
11  n <- rep(Ns, each = M)
12
13  # TODO: Construct the output
14
15  d_mean_apply <- data.frame(n = n, mean = )
```

Uncertainty in mean estimates with `apply()`

- Now we redo the first simulation with something from the `apply()` family

```r
1   # TODO: Write a simulation function
2
3   SimFun1 <- function(n_draws) {
4     draws <- rnorm(n_draws)
5     m <- mean(draws)
6     return(m)
7   }
8
9   # TODO: Construct the object to apply() over
10
11  n <- rep(Ns, each = M)
12
13  # TODO: Construct the output
14
15  d_mean_apply <- data.frame(n = n, mean = sapply())
```

Uncertainty in mean estimates with `apply()`

- Now we redo the first simulation with something from the `apply()` family

```r
1   # TODO: Write a simulation function
2
3   SimFun1 <- function(n_draws) {
4     draws <- rnorm(n_draws)
5     m <- mean(draws)
6     return(m)
7   }
8
9   # TODO: Construct the object to apply() over
10
11  n <- rep(Ns, each = M)
12
13  # TODO: Construct the output
14
15  d_mean_apply <- data.frame(n = n, mean = sapply(X = , FUN = ))
```

Uncertainty in mean estimates with `apply()`

- Now we redo the first simulation with something from the `apply()` family

```r
1   # TODO: Write a simulation function
2
3   SimFun1 <- function(n_draws) {
4     draws <- rnorm(n_draws)
5     m <- mean(draws)
6     return(m)
7   }
8
9   # TODO: Construct the object to apply() over
10
11  n <- rep(Ns, each = M)
12
13  # TODO: Construct the output
14
15  d_mean_apply <- data.frame(n = n, mean = sapply(X = , FUN = SimFun1))
```

Uncertainty in mean estimates with `apply()`

- Now we redo the first simulation with something from the `apply()` family

```r
# TODO: Write a simulation function

SimFun1 <- function(n_draws) {
  draws <- rnorm(n_draws)
  m <- mean(draws)
  return(m)
}

# TODO: Construct the object to apply() over

n <- rep(Ns, each = M)

# TODO: Construct the output

d_mean_apply <- data.frame(n = n, mean = sapply(X = n, FUN = SimFun1))
```

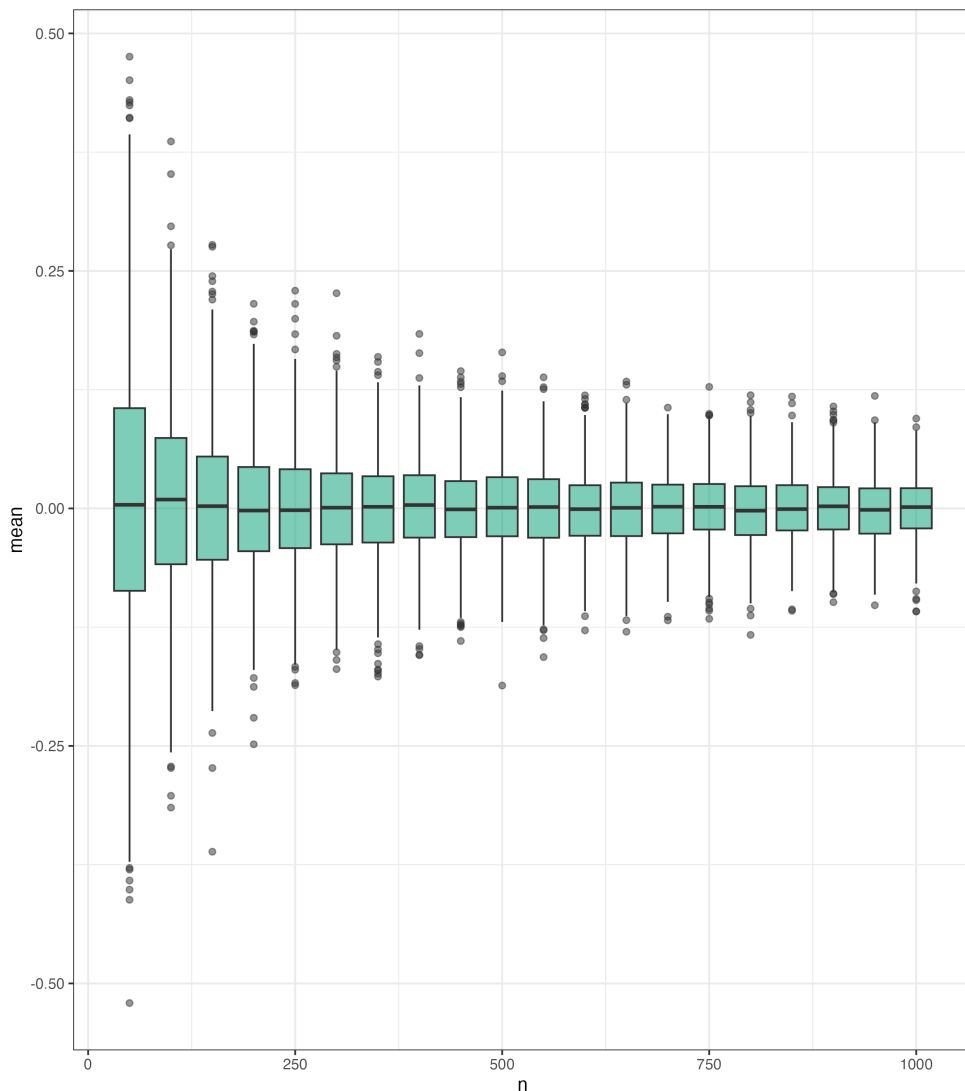# Visualizing uncertainty in mean estimates with `apply()`

```
1    ggplot(d_mean_apply, aes(x = n,
2                             y = mean,
3                             group = n)) +
4      geom_boxplot(alpha = 0.5,
5                   fill = okabeito_colors(3)) +
6      theme_bw()
```

Estimating uncertainty directly with `apply()`

```
1   # TODO: Write a simulation function
2
3   # TODO: Construct the object to apply() over
4
5   # TODO: Construct the output
```

Estimating uncertainty directly with `apply()`

```r
1   # TODO: Write a simulation function
2
3   SimFun2 <- function() {
4
5   }
6
7   # TODO: Construct the object to apply() over
8
9   # TODO: Construct the output
```

Estimating uncertainty directly with `apply()`

```r
1  # TODO: Write a simulation function
2
3  SimFun2 <- function() {
4    return(sdev)
5  }
6
7  # TODO: Construct the object to apply() over
8
9  # TODO: Construct the output
```

Estimating uncertainty directly with `apply()`

```
 1   # TODO: Write a simulation function
 2
 3   SimFun2 <- function() {
 4     draws <- matrix(rnorm(n = n_draws * n_reps),
 5                     nrow = n_draws,
 6                     ncol = n_reps)
 7     return(sdev)
 8   }
 9
10   # TODO: Construct the object to apply() over
11
12   # TODO: Construct the output
```

Estimating uncertainty directly with `apply()`

```r
1   # TODO: Write a simulation function
2
3   SimFun2 <- function(n_draws, n_reps) {
4     draws <- matrix(rnorm(n = n_draws * n_reps),
5                     nrow = n_draws,
6                     ncol = n_reps)
7     return(sdev)
8   }
9
10  # TODO: Construct the object to apply() over
11
12  # TODO: Construct the output
```

Estimating uncertainty directly with `apply()`

```r
1   # TODO: Write a simulation function
2
3   SimFun2 <- function(n_draws, n_reps) {
4     draws <- matrix(rnorm(n = n_draws * n_reps),
5                     nrow = n_draws,
6                     ncol = n_reps)
7     means <- apply(X = , MARGIN = , FUN = )
8     return(sdev)
9   }
10
11  # TODO: Construct the object to apply() over
12
13  # TODO: Construct the output
```

Estimating uncertainty directly with `apply()`

```r
1   # TODO: Write a simulation function
2
3   SimFun2 <- function(n_draws, n_reps) {
4     draws <- matrix(rnorm(n = n_draws * n_reps),
5                     nrow = n_draws,
6                     ncol = n_reps)
7     means <- apply(X = , MARGIN = , FUN = mean)
8     return(sdev)
9   }
10
11  # TODO: Construct the object to apply() over
12
13  # TODO: Construct the output
```

Estimating uncertainty directly with `apply()`

```r
1   # TODO: Write a simulation function
2
3   SimFun2 <- function(n_draws, n_reps) {
4     draws <- matrix(rnorm(n = n_draws * n_reps),
5                     nrow = n_draws,
6                     ncol = n_reps)
7     means <- apply(X = draws, MARGIN = , FUN = mean)
8     return(sdev)
9   }
10
11  # TODO: Construct the object to apply() over
12
13  # TODO: Construct the output
```

Estimating uncertainty directly with `apply()`

```r
1   # TODO: Write a simulation function
2
3   SimFun2 <- function(n_draws, n_reps) {
4     draws <- matrix(rnorm(n = n_draws * n_reps),
5                     nrow = n_draws,
6                     ncol = n_reps)
7     means <- apply(X = draws, MARGIN = 2, FUN = mean)
8     return(sdev)
9   }
10
11  # TODO: Construct the object to apply() over
12
13  # TODO: Construct the output
```

Estimating uncertainty directly with `apply()`

```r
# TODO: Write a simulation function

SimFun2 <- function(n_draws, n_reps) {
  draws <- matrix(rnorm(n = n_draws * n_reps),
                  nrow = n_draws,
                  ncol = n_reps)
  means <- apply(X = draws, MARGIN = 2, FUN = mean)
  sdev <- sd(means)
  return(sdev)
}

# TODO: Construct the object to apply() over

# TODO: Construct the output
```

Estimating uncertainty directly with `apply()`

```r
1   # TODO: Write a simulation function
2
3   SimFun2 <- function(n_draws, n_reps) {
4     draws <- matrix(rnorm(n = n_draws * n_reps),
5                     nrow = n_draws,
6                     ncol = n_reps)
7     means <- apply(X = draws, MARGIN = 2, FUN = mean)
8     sdev <- sd(means)
9     return(sdev)
10  }
11
12  # TODO: Construct the object to apply() over
13
14  sim_control_2 <- data.frame()
15
16  # TODO: Construct the output
```

Estimating uncertainty directly with `apply()`

```r
1   # TODO: Write a simulation function
2
3   SimFun2 <- function(n_draws, n_reps) {
4     draws <- matrix(rnorm(n = n_draws * n_reps),
5                     nrow = n_draws,
6                     ncol = n_reps)
7     means <- apply(X = draws, MARGIN = 2, FUN = mean)
8     sdev <- sd(means)
9     return(sdev)
10  }
11
12  # TODO: Construct the object to apply() over
13
14  sim_control_2 <- data.frame(n_draws = , n_reps = )
15
16  # TODO: Construct the output
```

Estimating uncertainty directly with `apply()`

```r
1   # TODO: Write a simulation function
2
3   SimFun2 <- function(n_draws, n_reps) {
4     draws <- matrix(rnorm(n = n_draws * n_reps),
5                     nrow = n_draws,
6                     ncol = n_reps)
7     means <- apply(X = draws, MARGIN = 2, FUN = mean)
8     sdev <- sd(means)
9     return(sdev)
10  }
11
12  # TODO: Construct the object to apply() over
13
14  sim_control_2 <- data.frame(n_draws = Ns, n_reps = M)
15
16  # TODO: Construct the output
```

Estimating uncertainty directly with `apply()`

```r
1   # TODO: Write a simulation function
2
3   SimFun2 <- function(n_draws, n_reps) {
4     draws <- matrix(rnorm(n = n_draws * n_reps),
5                     nrow = n_draws,
6                     ncol = n_reps)
7     means <- apply(X = draws, MARGIN = 2, FUN = mean)
8     sdev <- sd(means)
9     return(sdev)
10  }
11
12  # TODO: Construct the object to apply() over
13
14  sim_control_2 <- data.frame(n_draws = Ns, n_reps = M)
15
16  # TODO: Construct the output
17
18  sim_control_2$sd <- mapply(FUN = , ...)
```

Estimating uncertainty directly with `apply()`

```r
1   # TODO: Write a simulation function
2
3   SimFun2 <- function(n_draws, n_reps) {
4     draws <- matrix(rnorm(n = n_draws * n_reps),
5                     nrow = n_draws,
6                     ncol = n_reps)
7     means <- apply(X = draws, MARGIN = 2, FUN = mean)
8     sdev <- sd(means)
9     return(sdev)
10  }
11
12  # TODO: Construct the object to apply() over
13
14  sim_control_2 <- data.frame(n_draws = Ns, n_reps = M)
15
16  # TODO: Construct the output
17
18  sim_control_2$sd <- mapply(FUN = SimFun2, ...)
```

Estimating uncertainty directly with `apply()`

```r
1   # TODO: Write a simulation function
2
3   SimFun2 <- function(n_draws, n_reps) {
4     draws <- matrix(rnorm(n = n_draws * n_reps),
5                     nrow = n_draws,
6                     ncol = n_reps)
7     means <- apply(X = draws, MARGIN = 2, FUN = mean)
8     sdev <- sd(means)
9     return(sdev)
10  }
11
12  # TODO: Construct the object to apply() over
13
14  sim_control_2 <- data.frame(n_draws = Ns, n_reps = M)
15
16  # TODO: Construct the output
17
18  sim_control_2$sd <- mapply(FUN = SimFun2,
19                             sim_control_2$n_draws,
20                             ...)
```

Estimating uncertainty directly with `apply()`
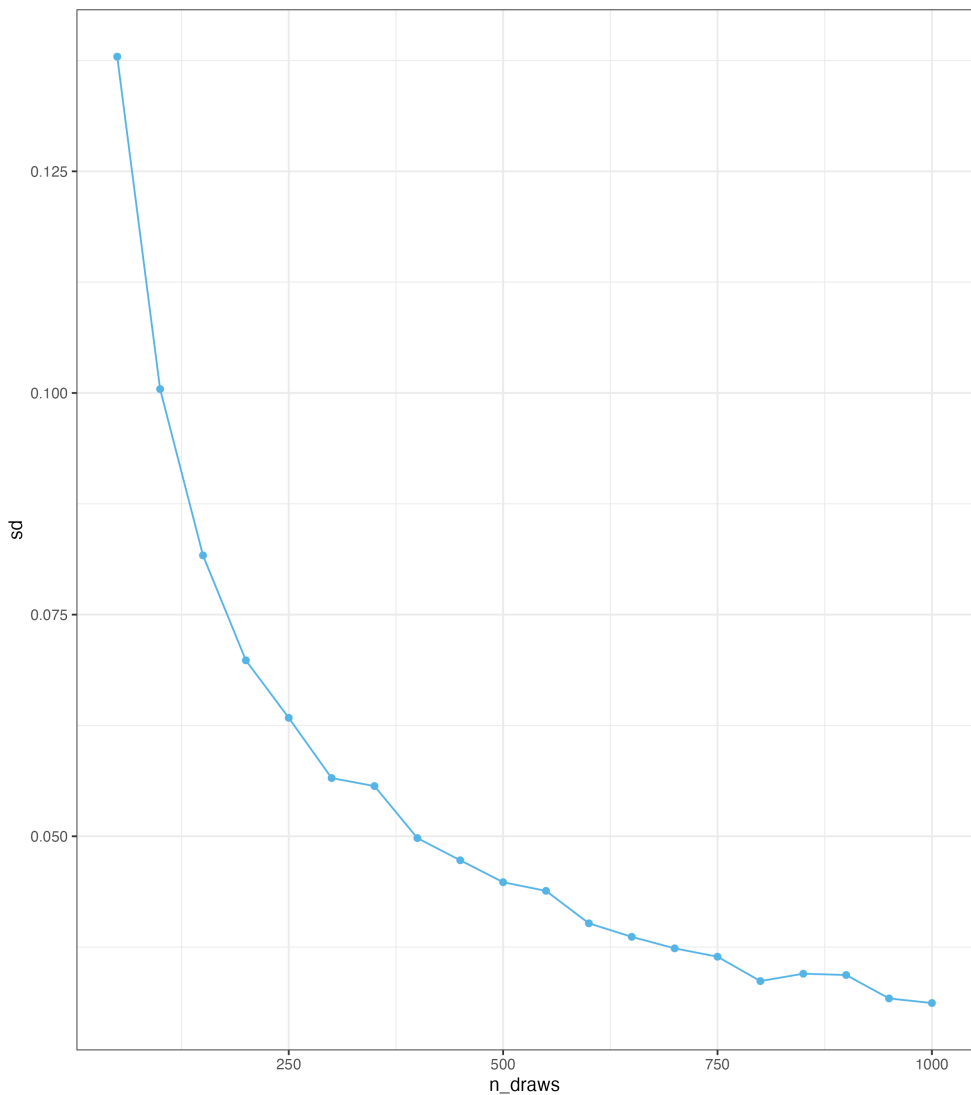
```r
1   # TODO: Write a simulation function
2
3   SimFun2 <- function(n_draws, n_reps) {
4     draws <- matrix(rnorm(n = n_draws * n_reps),
5                     nrow = n_draws,
6                     ncol = n_reps)
7     means <- apply(X = draws, MARGIN = 2, FUN = mean)
8     sdev <- sd(means)
9     return(sdev)
10  }
11
12  # TODO: Construct the object to apply() over
13
14  sim_control_2 <- data.frame(n_draws = Ns, n_reps = M)
15
16  # TODO: Construct the output
17
18  sim_control_2$sd <- mapply(FUN = SimFun2,
19                             sim_control_2$n_draws,
20                             sim_control_2$n_reps)
```

Visualizing uncertainty estimates with `apply()`

```
1    ggplot(sim_control_2, aes(x = n_draws,
2                              y = sd)) +
3      geom_point(color = okabeito_colors(2)) +
4      geom_line(color = okabeito_colors(2)) +
5      theme_bw()
```

Extending to more distributions

```
1  # TODO: Write a simulation function
2
3  # TODO: Construct the object to apply() over
4
5  # TODO: Construct the output
```

Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun2 <- function(n_draws, n_reps) {
4     draws <- matrix(rnorm(n = n_draws * n_reps),
5                     nrow = n_draws,
6                     ncol = n_reps)
7     means <- apply(X = draws, MARGIN = 2, FUN = mean)
8     sdev <- sd(means)
9     return(sdev)
10  }
11
12  # TODO: Construct the object to apply() over
13
14  # TODO: Construct the output
```

Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps) {
4     draws <- matrix(rnorm(n = n_draws * n_reps),
5                     nrow = n_draws,
6                     ncol = n_reps)
7     means <- apply(X = draws, MARGIN = 2, FUN = mean)
8     sdev <- sd(means)
9     return(sdev)
10  }
11
12  # TODO: Construct the object to apply() over
13
14  # TODO: Construct the output
```

Extending to more distributions

```
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     draws <- matrix(rnorm(n = n_draws * n_reps),
5                     nrow = n_draws,
6                     ncol = n_reps)
7     means <- apply(X = draws, MARGIN = 2, FUN = mean)
8     sdev <- sd(means)
9     return(sdev)
10  }
11
12  # TODO: Construct the object to apply() over
13
14  # TODO: Construct the output
```

Extending to more distributions

```r
1  # TODO: Write a simulation function
2
3  SimFun3 <- function(n_draws, n_reps, dist) {
4    # TODO: draw from distribution
5    # TODO: reshape into matrix
6    means <- apply(X = draws, MARGIN = 2, FUN = mean)
7    sdev <- sd(means)
8    return(sdev)
9  }
10
11 # TODO: Construct the object to apply() over
12
13 # TODO: Construct the output
```

Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     # TODO: draw from distribution
5     draws <- do.call()
6     # TODO: reshape into matrix
7     means <- apply(X = draws, MARGIN = 2, FUN = mean)
8     sdev <- sd(means)
9     return(sdev)
10  }
11
12  # TODO: Construct the object to apply() over
13
14  # TODO: Construct the output
```

Extending to more distributions

```
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     # TODO: draw from distribution
5     draws <- do.call(what = , args = )
6     # TODO: reshape into matrix
7     means <- apply(X = draws, MARGIN = 2, FUN = mean)
8     sdev <- sd(means)
9     return(sdev)
10  }
11
12  # TODO: Construct the object to apply() over
13
14  # TODO: Construct the output
```

Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     # TODO: draw from distribution
5     draws <- do.call(what = dist, args = )
6     # TODO: reshape into matrix
7     means <- apply(X = draws, MARGIN = 2, FUN = mean)
8     sdev <- sd(means)
9     return(sdev)
10  }
11
12  # TODO: Construct the object to apply() over
13
14  # TODO: Construct the output
```

Extending to more distributions

```r
 1   # TODO: Write a simulation function
 2
 3   SimFun3 <- function(n_draws, n_reps, dist) {
 4     # TODO: draw from distribution
 5     draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
 6     # TODO: reshape into matrix
 7     means <- apply(X = draws, MARGIN = 2, FUN = mean)
 8     sdev <- sd(means)
 9     return(sdev)
10   }
11
12   # TODO: Construct the object to apply() over
13
14   # TODO: Construct the output
```

Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     # TODO: draw from distribution
5     draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
6     # TODO: reshape into matrix
7     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
8     means <- apply(X = draws, MARGIN = 2, FUN = mean)
9     sdev <- sd(means)
10    return(sdev)
11  }
12
13  # TODO: Construct the object to apply() over
14
15  # TODO: Construct the output
```

Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     means <- apply(X = draws, MARGIN = 2, FUN = mean)
7     sdev <- sd(means)
8     return(sdev)
9   }
10
11  # TODO: Construct the object to apply() over
12
13  # TODO: Construct the output
```

Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     means <- apply(X = draws, MARGIN = 2, FUN = mean)
7     sdev <- sd(means)
8     return(sdev)
9   }
10
11  # TODO: Construct the object to apply() over
12
13  sim_control_3 <- data.frame()
14
15  # TODO: Construct the output
```

Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     means <- apply(X = draws, MARGIN = 2, FUN = mean)
7     sdev <- sd(means)
8     return(sdev)
9   }
10
11  # TODO: Construct the object to apply() over
12
13  distributions <- c('rnorm', 'runif', 'rexp')
14  sim_control_3 <- data.frame()
15
16  # TODO: Construct the output
```

Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     means <- apply(X = draws, MARGIN = 2, FUN = mean)
7     sdev <- sd(means)
8     return(sdev)
9   }
10
11  # TODO: Construct the object to apply() over
12
13  distributions <- c('rnorm', 'runif', 'rexp')
14  sim_control_3 <- data.frame(n_draws = ,
15                              n_reps = ,
16                              distribution = )
17
18  # TODO: Construct the output
```

Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     means <- apply(X = draws, MARGIN = 2, FUN = mean)
7     sdev <- sd(means)
8     return(sdev)
9   }
10
11  # TODO: Construct the object to apply() over
12
13  distributions <- c('rnorm', 'runif', 'rexp')
14  sim_control_3 <- data.frame(n_draws = rep(Ns, times = length(distributions)),
15                              n_reps = ,
16                              distribution = )
17
18  # TODO: Construct the output
```

Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     means <- apply(X = draws, MARGIN = 2, FUN = mean)
7     sdev <- sd(means)
8     return(sdev)
9   }
10
11  # TODO: Construct the object to apply() over
12
13  distributions <- c('rnorm', 'runif', 'rexp')
14  sim_control_3 <- data.frame(n_draws = rep(Ns, times = length(distributions)),
15                              n_reps = M,
16                              distribution = )
17
18  # TODO: Construct the output
```

Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     means <- apply(X = draws, MARGIN = 2, FUN = mean)
7     sdev <- sd(means)
8     return(sdev)
9   }
10
11  # TODO: Construct the object to apply() over
12
13  distributions <- c('rnorm', 'runif', 'rexp')
14  sim_control_3 <- data.frame(n_draws = rep(Ns, times = length(distributions)),
15                              n_reps = M,
16                              distribution = rep(distributions, each = length(Ns)))
17
18  # TODO: Construct the output
```

Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     means <- apply(X = draws, MARGIN = 2, FUN = mean)
7     sdev <- sd(means)
8     return(sdev)
9   }
10
11  # TODO: Construct the object to apply() over
12
13  distributions <- c('rnorm', 'runif', 'rexp')
14  sim_control_3 <- data.frame(n_draws = rep(Ns, times = length(distributions)),
15                              n_reps = M,
16                              distribution = rep(distributions, each = length(Ns)))
17
18  # TODO: Construct the output
19
20  sim_control_3$sd <- mapply(FUN = , ...)
```

Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     means <- apply(X = draws, MARGIN = 2, FUN = mean)
7     sdev <- sd(means)
8     return(sdev)
9   }
10
11  # TODO: Construct the object to apply() over
12
13  distributions <- c('rnorm', 'runif', 'rexp')
14  sim_control_3 <- data.frame(n_draws = rep(Ns, times = length(distributions)),
15                              n_reps = M,
16                              distribution = rep(distributions, each = length(Ns)))
17
18  # TODO: Construct the output
19
20  sim_control_3$sd <- mapply(FUN = SimFun3,
21                             ...)
```

Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     means <- apply(X = draws, MARGIN = 2, FUN = mean)
7     sdev <- sd(means)
8     return(sdev)
9   }
10
11  # TODO: Construct the object to apply() over
12
13  distributions <- c('rnorm', 'runif', 'rexp')
14  sim_control_3 <- data.frame(n_draws = rep(Ns, times = length(distributions)),
15                              n_reps = M,
16                              distribution = rep(distributions, each = length(Ns)))
17
18  # TODO: Construct the output
19
20  sim_control_3$sd <- mapply(FUN = SimFun3,
21                             sim_control_3$n_draws,
22                             ...)
```
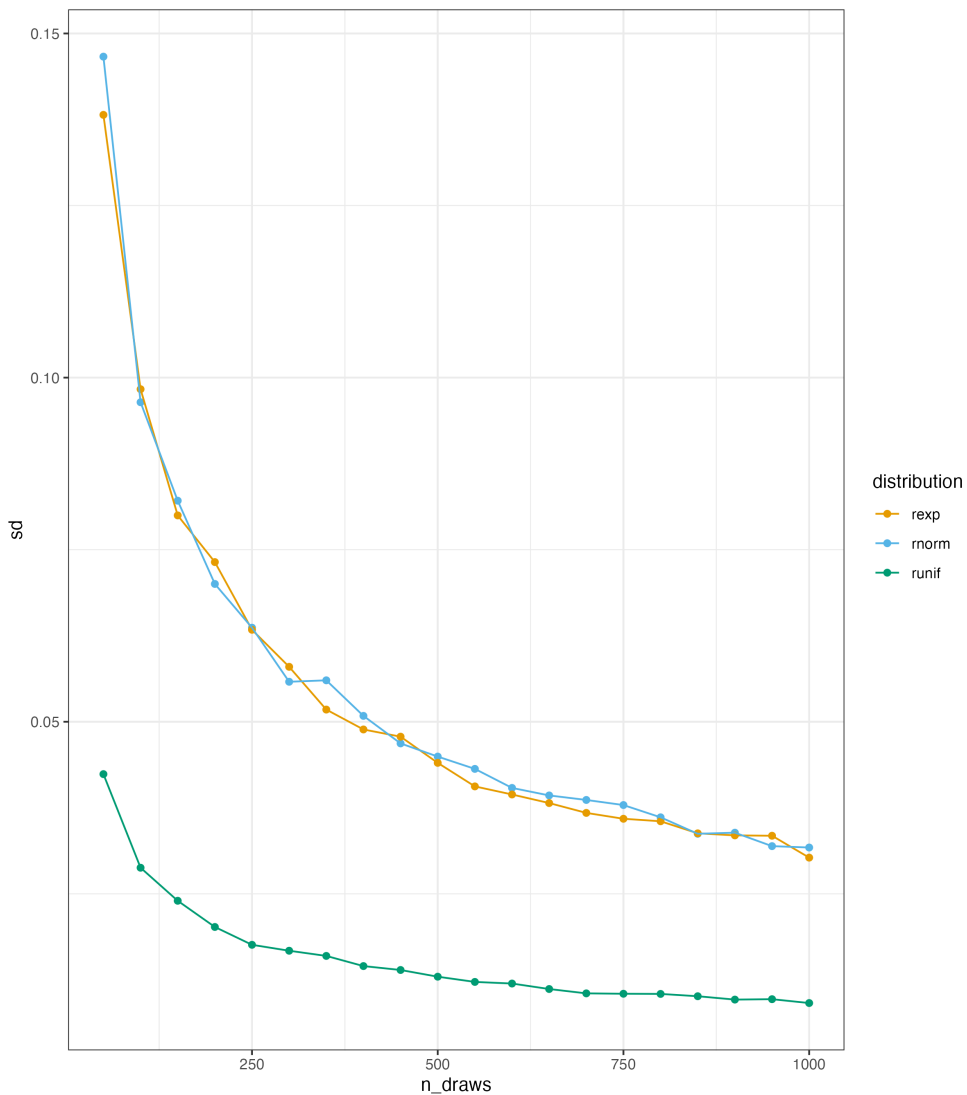
Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     means <- apply(X = draws, MARGIN = 2, FUN = mean)
7     sdev <- sd(means)
8     return(sdev)
9   }
10
11  # TODO: Construct the object to apply() over
12
13  distributions <- c('rnorm', 'runif', 'rexp')
14  sim_control_3 <- data.frame(n_draws = rep(Ns, times = length(distributions)),
15                              n_reps = M,
16                              distribution = rep(distributions, each = length(Ns)))
17
18  # TODO: Construct the output
19
20  sim_control_3$sd <- mapply(FUN = SimFun3,
21                             sim_control_3$n_draws,
22                             sim_control_3$n_reps,
23                             ...)
```

Extending to more distributions

```r
1   # TODO: Write a simulation function
2
3   SimFun3 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     means <- apply(X = draws, MARGIN = 2, FUN = mean)
7     sdev <- sd(means)
8     return(sdev)
9   }
10
11  # TODO: Construct the object to apply() over
12
13  distributions <- c('rnorm', 'runif', 'rexp')
14  sim_control_3 <- data.frame(n_draws = rep(Ns, times = length(distributions)),
15                              n_reps = M,
16                              distribution = rep(distributions, each = length(Ns)))
17
18  # TODO: Construct the output
19
20  sim_control_3$sd <- mapply(FUN = SimFun3,
21                             sim_control_3$n_draws,
22                             sim_control_3$n_reps,
23                             sim_control_3$distribution)
```

Visualizing uncertainty across distributions

```
1   ggplot(sim_control_3,
2          aes(x = n_draws,
3              y = sd,
4              color = distribution)) +
5     geom_point() +
6     geom_line() +
7     scale_color_okabeito() +
8     theme_bw()
```

Constructing more complex data frame output

```
1  # TODO: Write a simulation function
2
3  # TODO: Construct the output
```

Constructing more complex data frame output

```r
# TODO: Write a simulation function

SimFun3 <- function(n_draws, n_reps, dist) {
  draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
  draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
  means <- apply(X = draws, MARGIN = 2, FUN = mean)
  sdev <- sd(means)
  return(sdev)
}

# TODO: Construct the output
```

Constructing more complex data frame output

```r
1   # TODO: Write a simulation function
2
3   SimFun4 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     means <- apply(X = draws, MARGIN = 2, FUN = mean)
7     out <- data.frame()
8     return(out)
9   }
10
11  # TODO: Construct the output
```

Constructing more complex data frame output

```r
# TODO: Write a simulation function

SimFun4 <- function(n_draws, n_reps, dist) {
  draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
  draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
  means <- apply(X = draws, MARGIN = 2, FUN = mean)
  out <- data.frame(n = , dist = , mean = )
  return(out)
}

# TODO: Construct the output
```

Constructing more complex data frame output

```r
# TODO: Write a simulation function

SimFun4 <- function(n_draws, n_reps, dist) {
  draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
  draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
  means <- apply(X = draws, MARGIN = 2, FUN = mean)
  out <- data.frame(n = n_draws, dist = , mean = )
  return(out)
}

# TODO: Construct the output
```

Constructing more complex data frame output

```r
1   # TODO: Write a simulation function
2
3   SimFun4 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     means <- apply(X = draws, MARGIN = 2, FUN = mean)
7     out <- data.frame(n = n_draws, dist = dist, mean = )
8     return(out)
9   }
10
11  # TODO: Construct the output
```

Constructing more complex data frame output

```r
# TODO: Write a simulation function

SimFun4 <- function(n_draws, n_reps, dist) {
  draws <- do.call(what = dist, args = list(n = n_draws * n_reps))
  draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
  means <- apply(X = draws, MARGIN = 2, FUN = mean)
  out <- data.frame(n = n_draws, dist = dist, mean = means)
  return(out)
}

# TODO: Construct the output
```

Constructing more complex data frame output

```r
1   # TODO: Write a simulation function
2
3   SimFun4 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     out <- data.frame(n = n_draws, dist = dist, mean = apply(X = draws, MARGIN = 2, FUN = mean))
7     return(out)
8   }
9
10  # TODO: Construct the output
```

Constructing more complex data frame output

```r
1   # TODO: Write a simulation function
2
3   SimFun4 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     out <- data.frame(n = n_draws, dist = dist, mean = apply(X = draws, MARGIN = 2, FUN = mean))
7     return(out)
8   }
9
10  # TODO: Construct the output
11
12  out <- mapply(FUN = , ...)
```

Constructing more complex data frame output

```r
1   # TODO: Write a simulation function
2
3   SimFun4 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     out <- data.frame(n = n_draws, dist = dist, mean = apply(X = draws, MARGIN = 2, FUN = mean))
7     return(out)
8   }
9
10  # TODO: Construct the output
11
12  out <- mapply(
13    FUN = SimFun4,
14    ...
15  )
```

Constructing more complex data frame output

```
1   # TODO: Write a simulation function
2
3   SimFun4 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     out <- data.frame(n = n_draws, dist = dist, mean = apply(X = draws, MARGIN = 2, FUN = mean))
7     return(out)
8   }
9
10  # TODO: Construct the output
11
12  out <- mapply(
13    FUN = SimFun4,
14    sim_control_3$n_draws,
15    ...
16  )
```

Constructing more complex data frame output

```r
1   # TODO: Write a simulation function
2
3   SimFun4 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     out <- data.frame(n = n_draws, dist = dist, mean = apply(X = draws, MARGIN = 2, FUN = mean))
7     return(out)
8   }
9
10  # TODO: Construct the output
11
12  out <- mapply(
13    FUN = SimFun4,
14    sim_control_3$n_draws,
15    sim_control_3$n_reps,
16    ...
17  )
```

Constructing more complex data frame output

```r
1   # TODO: Write a simulation function
2
3   SimFun4 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     out <- data.frame(n = n_draws, dist = dist, mean = apply(X = draws, MARGIN = 2, FUN = mean))
7     return(out)
8   }
9
10  # TODO: Construct the output
11
12  out <- mapply(
13    FUN = SimFun4,
14    sim_control_3$n_draws,
15    sim_control_3$n_reps,
16    sim_control_3$distribution
17  )
```

# Constructing more complex data frame output

```r
1   # TODO: Write a simulation function
2
3   SimFun4 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     out <- data.frame(n = n_draws, dist = dist, mean = apply(X = draws, MARGIN = 2, FUN = mean))
7     return(out)
8   }
9
10  # TODO: Construct the output
11
12  out <- mapply(
13    FUN = SimFun4,
14    sim_control_3$n_draws,
15    sim_control_3$n_reps,
16    sim_control_3$distribution,
17    SIMPLIFY = FALSE
18  )
```

Constructing more complex data frame output

```r
1   # TODO: Write a simulation function
2
3   SimFun4 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     out <- data.frame(n = n_draws, dist = dist, mean = apply(X = draws, MARGIN = 2, FUN = mean))
7     return(out)
8   }
9
10  # TODO: Construct the output
11
12  out <- mapply(
13    FUN = SimFun4,
14    sim_control_3$n_draws,
15    sim_control_3$n_reps,
16    sim_control_3$distribution,
17    SIMPLIFY = FALSE
18  )
19
20  d_sim_4 <- do.call('rbind', out)
```

Constructing more complex data frame output

```r
1   # TODO: Write a simulation function
2
3   SimFun4 <- function(n_draws, n_reps, dist) {
4     draws <- do.call(dist, args = list(n = n_draws * n_reps))
5     draws <- matrix(draws, nrow = n_draws, ncol = n_reps)
6     out <- data.frame(n = n_draws, dist = dist, mean = apply(X = draws, MARGIN = 2, FUN = mean))
7     return(out)
8   }
9
10  # TODO: Construct the output
11
12  out <- mapply(FUN = SimFun4,
13                sim_control_3$n_draws,
14                sim_control_3$n_reps,
15                sim_control_3$distribution,
16                SIMPLIFY = FALSE)
17
18  d_sim_4 <- do.call('rbind', out)
19
20  ggplot(d_sim_4, aes(x = n, y = mean, group = n, fill = dist)) +
21    geom_boxplot(alpha = 0.5) +
22    facet_grid(. ~ dist) +
23    scale_fill_okabeito() +
24    theme_bw()
```
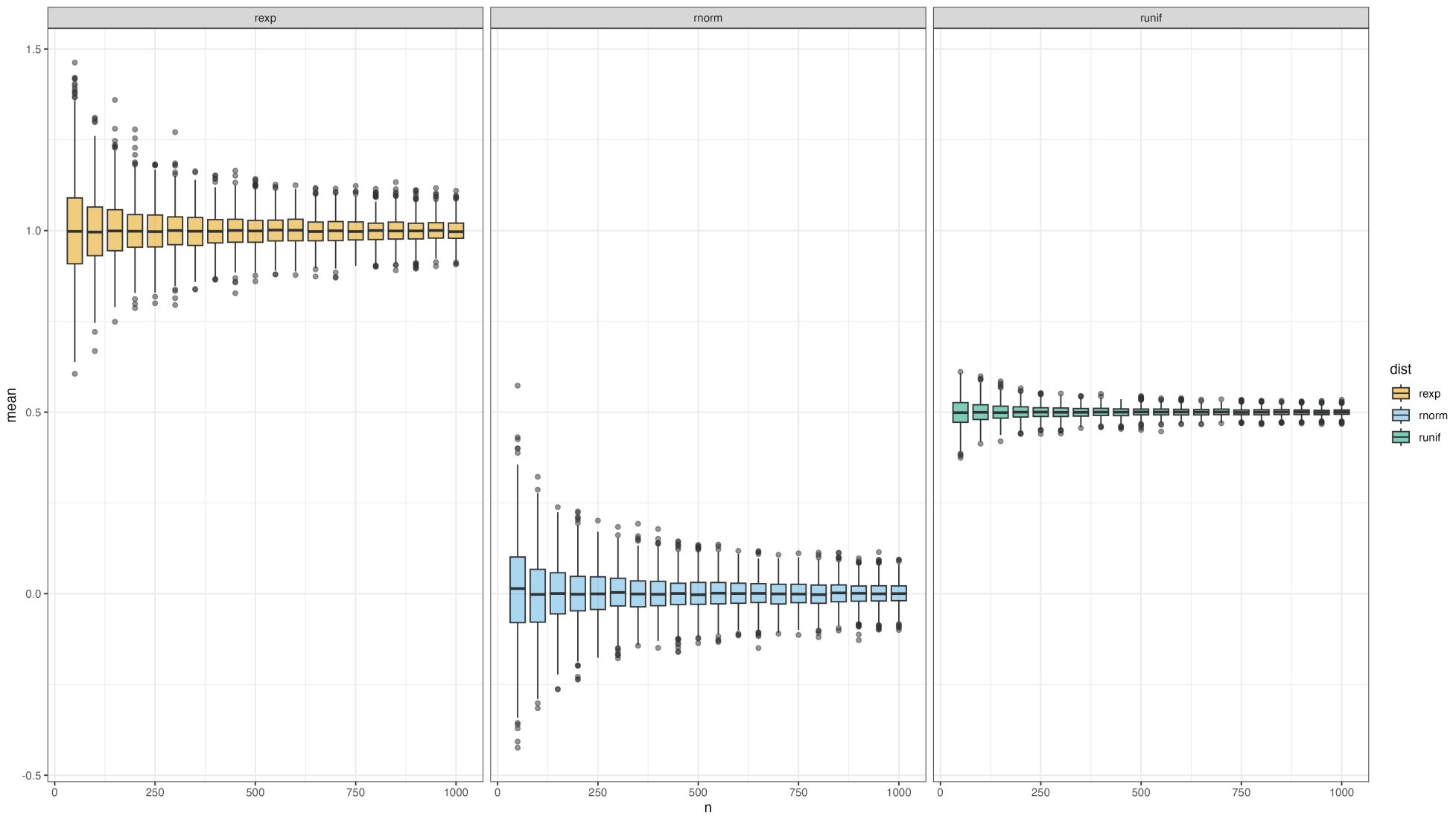
Wrap Up

Recap

- Debugging is challenging, but we have four (five?) main tools:
  - Googling error messages
  - `print()` statements
  - `traceback()`
  - `browser()`
  - Maybe ChatGPT if you like it?
- The `apply()` family of functions allows you to hide loops
  - Not more efficient, but *safer* than loops
  - Encourages functionalization of code
  - Makes code easier to read

Final Thoughts

- PollEv.com/klintkanopka