APSTA-GE 2352

Statistical Computing: Lecture 5

Klint Kanopka

New York University

Table of Contents

Announcements

- PS2 is due tonight at 11.59p!
- PS3 is out!
  - It has six parts
  - They are not ordered by difficulty
  - You should be able to do them all using information from lecture today
  - Start soon!
- Really enjoying seeing people around the department, at office hours, and active in Slack

Check-In

- PollEv.com/klintkanopka

A Note on Searching

- Everything we really do is a search problem
- The challenge is that the search spaces are often infinite
- All of the models we pick and assumptions we make limit the size of the search space
- The algorithms we use define how we carry out the search
- Good choice of models/assumptions/algorithms allow us to take intractable problems and solve them relatively quickly and easily!

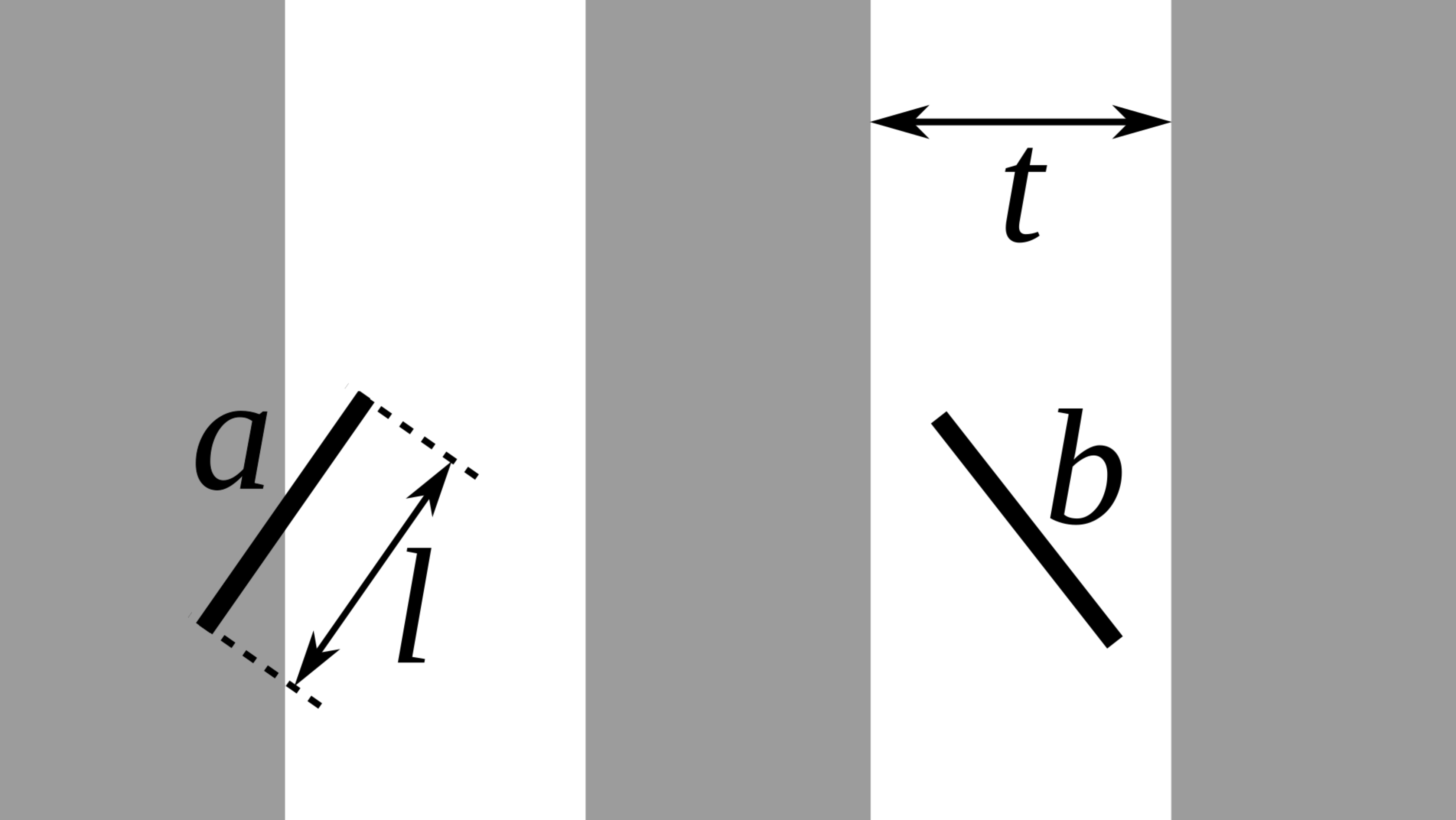
Motivating Problem

Buffon’s Needle

Suppose we have a floor made of parallel strips of wood, each with the same width, and we drop a needle onto the floor. What is the probability that the needle will lie across a line between two strips?

More Specifically:

Given a needle of length l dropped on a floor with parallel lines distance t apart, what is the probability that the needle will lie across a line when landing?



t
a
l
b

Tools

Randomization

Randomization

- Some problems are hard to solve in a straightforward way
  - The function to optimize could be really tricky
  - The search space could be really huge
  - There might be an easier solution, but you don’t know what it is
  - No clean closed form solution may exist!
- For these cases, we can leverage randomization to get *approximate* solutions
  - If we are willing to invest more time, we can get more precise solutions
- Are there downsides?
  - Can be slooooooooooooooooooooooooooooooooooooooooooow
  - It can be guaranteed that you will eventually find the right answer, but sometimes there is no guarantee on *when*
  - Sometimes having a nice closed form solution is what you actually need
- If you have a better choice, typically you want to use that!

Lady Tasting Tea Experiment

Content warning: This is the most British experiment ever

Muriel Bristol (1888-1950)

- British phycologist (studied algae)
- Claimed she can tell *by taste alone* if a cup of tea was made by pouring milk into tea or tea into milk

Ronald Fisher (1890-1962)

- British statistician, biologist, and geneticist
- Called the "greatest statistician of all time"
- Noted eugenicist
- Thought Bristol’s claim was nuts, so devised an experiment to test her tea-tasting ability

The Lady Tasting Tea Experiment

Step 1: Pour eight cups of tea, with four having milk added first and four having tea added first
Step 2: Cups are presented in a random order, and Muriel has to identify them
Question: Assuming she does not have any special ability and randomly guessed, what is the probability she would get all eight correct?

Quick Aside: Combinatorics

- *Combinatorics* is concerned with (among other things) counting
- Today, we need to think about counting *combinations* of items from a set

For a set of n items, the number of ways to select k elements is:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Lady Tasting Tea Experiment

- How many possible ways can you select four "tea before milk" cups from eight?

$$\binom{8}{4} = \frac{8!}{4!(8-4)!} = \frac{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7 \cdot 8}{(1 \cdot 2 \cdot 3 \cdot 4)(1 \cdot 2 \cdot 3 \cdot 4)} = 70$$

- There’s only one way to get them all right, so we can find the probability assuming she’s randomly guessing (the null hypothesis of no ability)

$$P(\text{All correct}) = \frac{1}{70} \approx 1.43\%$$

- *Fisher’s Exact Test* is a method to build the exact distribution of possible outcomes, so we get an exact *p*-value for any outcome

Fisher's Exact Test

- Enumerate all possible outcomes
- Compute the probability of each number of successes
- Look at the distribution and compute p-values

Successes	Combinations	p-value
0	$\binom{4}{0} \times \binom{4}{4} = 1$	1.0000
1	$\binom{4}{1} \times \binom{4}{3} = 16$	0.9857
2	$\binom{4}{2} \times \binom{4}{2} = 36$	0.4286
3	$\binom{4}{3} \times \binom{4}{1} = 16$	0.2429
4	$\binom{4}{4} \times \binom{4}{0} = 1$	0.0143

Permutation Tests

Permutation Tests

- Permutation tests are a form of exact test used when comparing two separate samples
- Often you want to know if two samples came from two different distributions (they have different means, or different variances, or something else)
- Permutation tests are similar to Fisher's exact test, but we look at the value we want to test against a distribution constructed from *every possible permutation of group assignments*

Permutation Tests

- Say I have test scores from two groups of ten students, and group one has a higher mean score
- I want to say if I really think group one is higher ability than group two
- I divide the pool of 20 students into two even groups in every possible way
  - 184,756 possible group assignments
- I compute the difference in mean score between the two groups
- I see if the original difference is extreme relative to the distribution of possible differences!

Permutation Tests

1. Figure out every possible group assignment in your data
2. Compute the summary statistic of interest for each possible group assignment
3. Find the probability of the statistic being that value (or more extreme) from this distribution

Permutation Tests

- *Upside*: The answer is correct!
- *Downside*: If you have 40 objects evenly divided into two groups, there are well over 2 billion possible group assignments, so this usually doesn't work well at all
- So, what if we just… didn't?
- Big idea: Permutation test and other types of exact tests are just way too much work for datasets of any reasonable size
- The numbers of permutations are often too large for computers to deal with
- So, what if we started doing a permutation test, and then just stopped part way through?
- We can instead *sample* from possible permutations
- Randomly sampling from our data will eventually converge to the true answer
- Note that *eventually* is doing some work here

Monte Carlo Methods


CASINO
MONTE-CARLO

Monte Carlo Methods

- Leverages the fact that randomly sampling from your data will converge to the true answer
- Works in *tons* of situations
- General task:
  1. Specify the possible inputs
  2. Randomly sample from the possible inputs
  3. Perform some calculation on each sample
  4. Repeat a bunch of times
  5. Aggregate the results

Monte Carlo Permutation Test

Remember the test scores from two groups of students, where group one had a higher mean score?

1. Randomly sample from the possible group assignments
2. Compute mean differences
3. Repeat a bunch of times
4. Look at the distribution

Monte Carlo Permutation Test

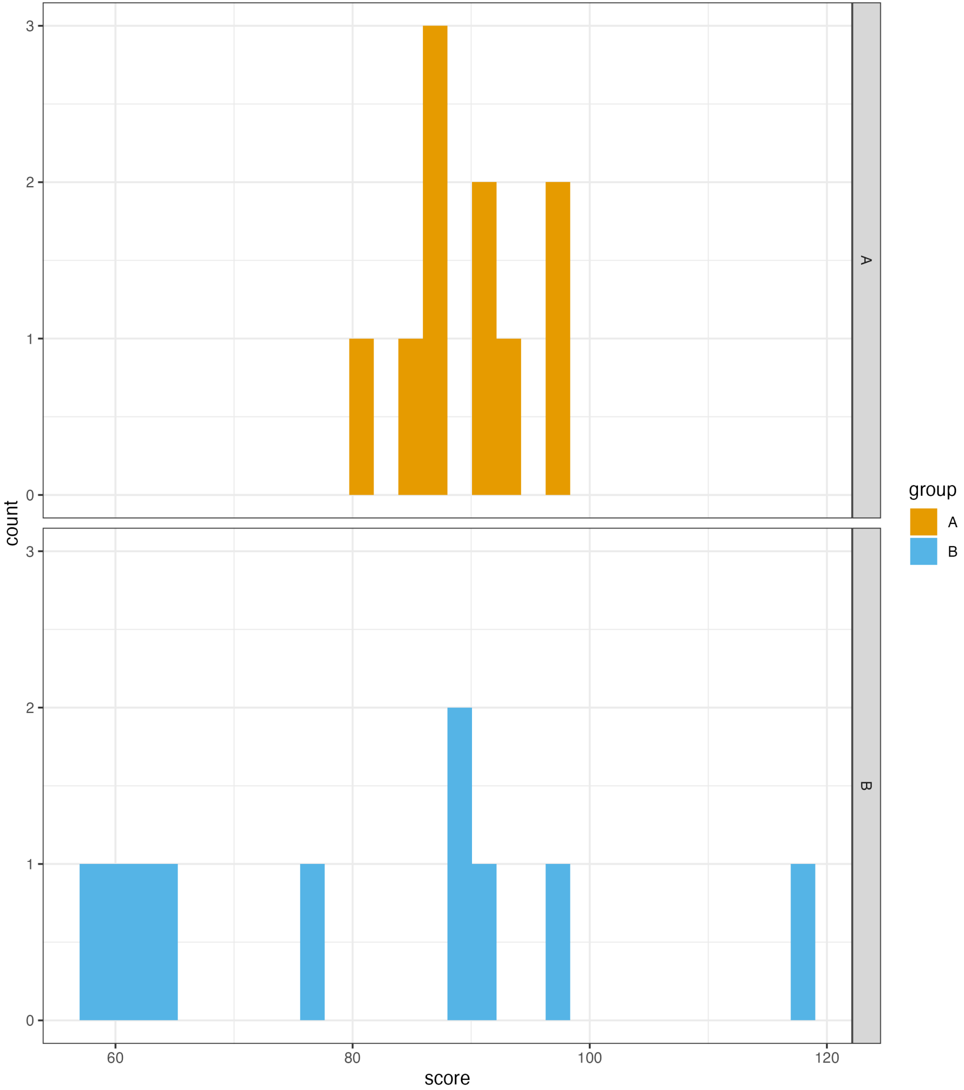
```
set.seed(215)
test_data <- data.frame(group = rep(c('A', 'B'), each=10),
                        score = c(round(rnorm(10, mean=90, sd=5)),
                                  round(rnorm(10, mean=75, sd=15))))
true_diff <- mean(test_data$score[test_data=='A']) - mean(test_data$score[test_data=='B'])

true_diff

# [1] 9
```

Monte Carlo Permutation Test

```r
ggplot(test_data,
       aes(x = score, fill = group)) +
  geom_histogram() +
  facet_grid(group ~ .) +
  scale_fill_okabeito() +
  theme_bw()
```



Monte Carlo Permutation Test

```r
ReassignScores <- function(data){

  # TODO: Reassign groups
  new_groups <- sample(data[['group']])

  # TODO: Compute means
  mean_A <- mean(data[['score']][new_groups == 'A'])
  mean_B <- mean(data[['score']][new_groups == 'B'])

  # TODO: Return difference in means
  mean_diff <- mean_A - mean_B
  return(mean_diff)
}

ReassignScores(test_data)

# [1] -0.2
```

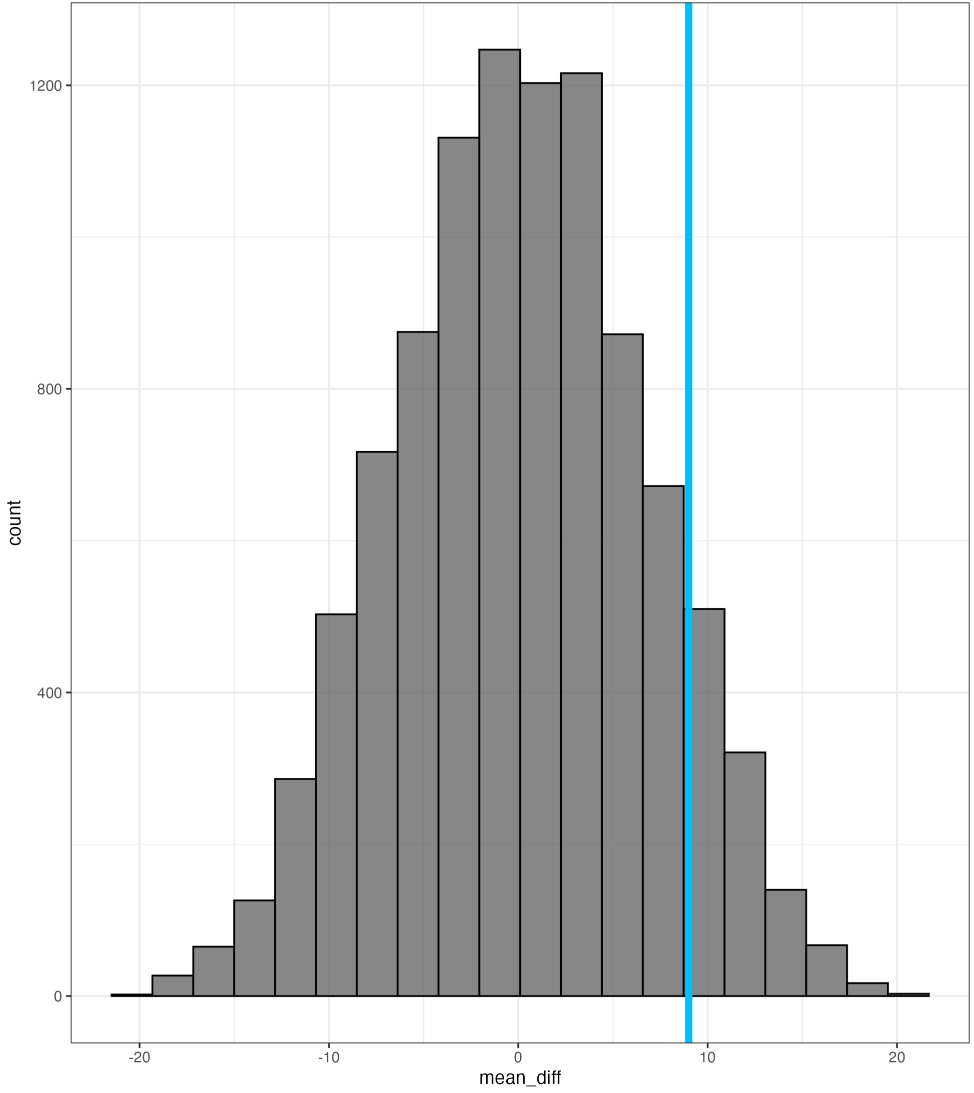
Monte Carlo Permutation Test

```r
mean_diff <-
  replicate(1e4, ReassignScores(test_data))

mean(mean_diff >= true_diff)

# [1] 0.1015

data.frame(mean_diff=mean_diff) |>
  ggplot(aes(x=mean_diff)) +
  geom_histogram(alpha = 0.7,
                 color = 'black',
                 bins = 20) +
  geom_vline(aes(xintercept=true_diff),
                 linewidth = 2,
                 color='deepskyblue') +
  theme_bw()
```



Monte Carlo Permutation Test

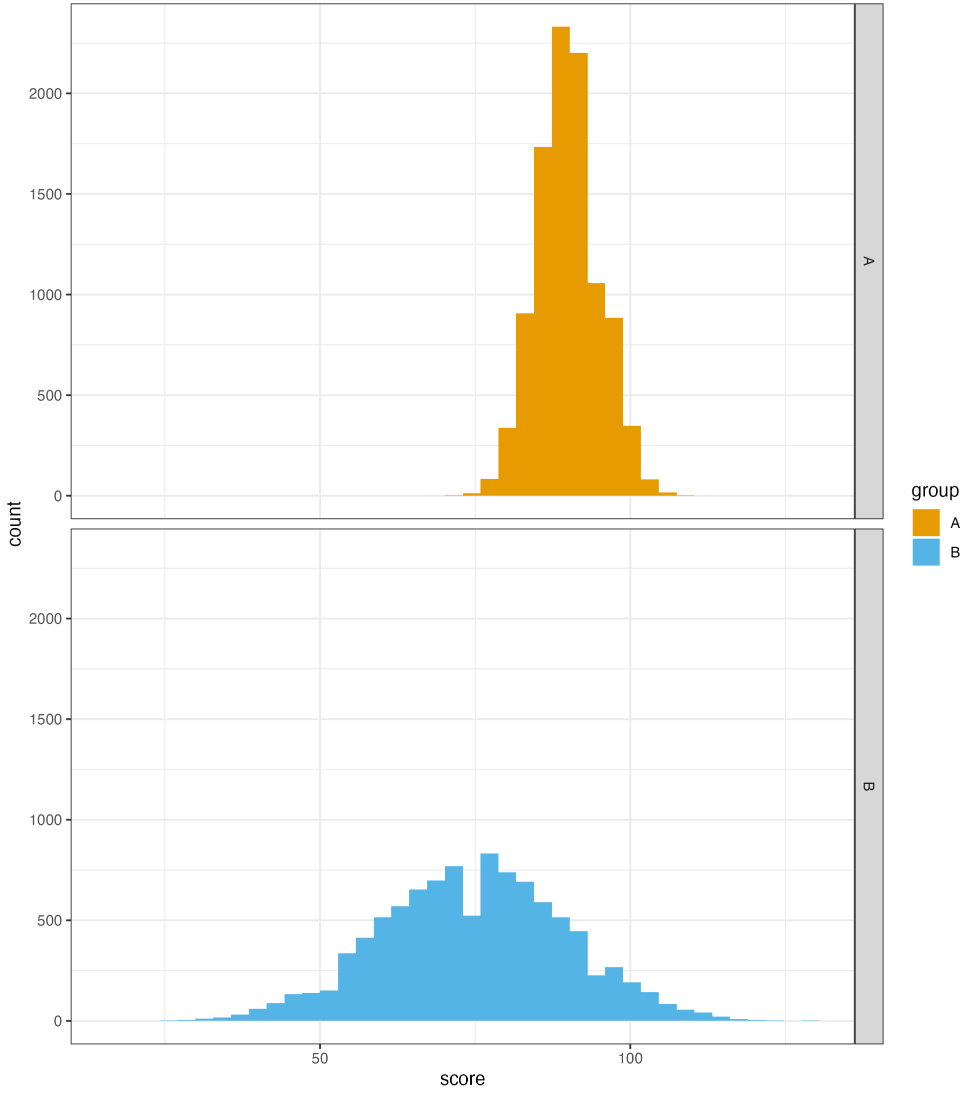
```r
test_data <- data.frame(group = rep(c('A', 'B'), each=1e4),
                        score = c(round(rnorm(1e4, mean=90, sd=5)),
                                  round(rnorm(1e4, mean=75, sd=15))))
true_diff <- mean(test_data$score[test_data=='A']) - mean(test_data$score[test_data=='B'])

true_diff

# [1] 15.1591
```

Monte Carlo Permutation Test

```
ggplot(test_data,
        aes(x = score, fill = group)) +
  geom_histogram(bins = 40) +
  facet_grid(group ~ .) +
  scale_fill_okabeito() +
  theme_bw()
```



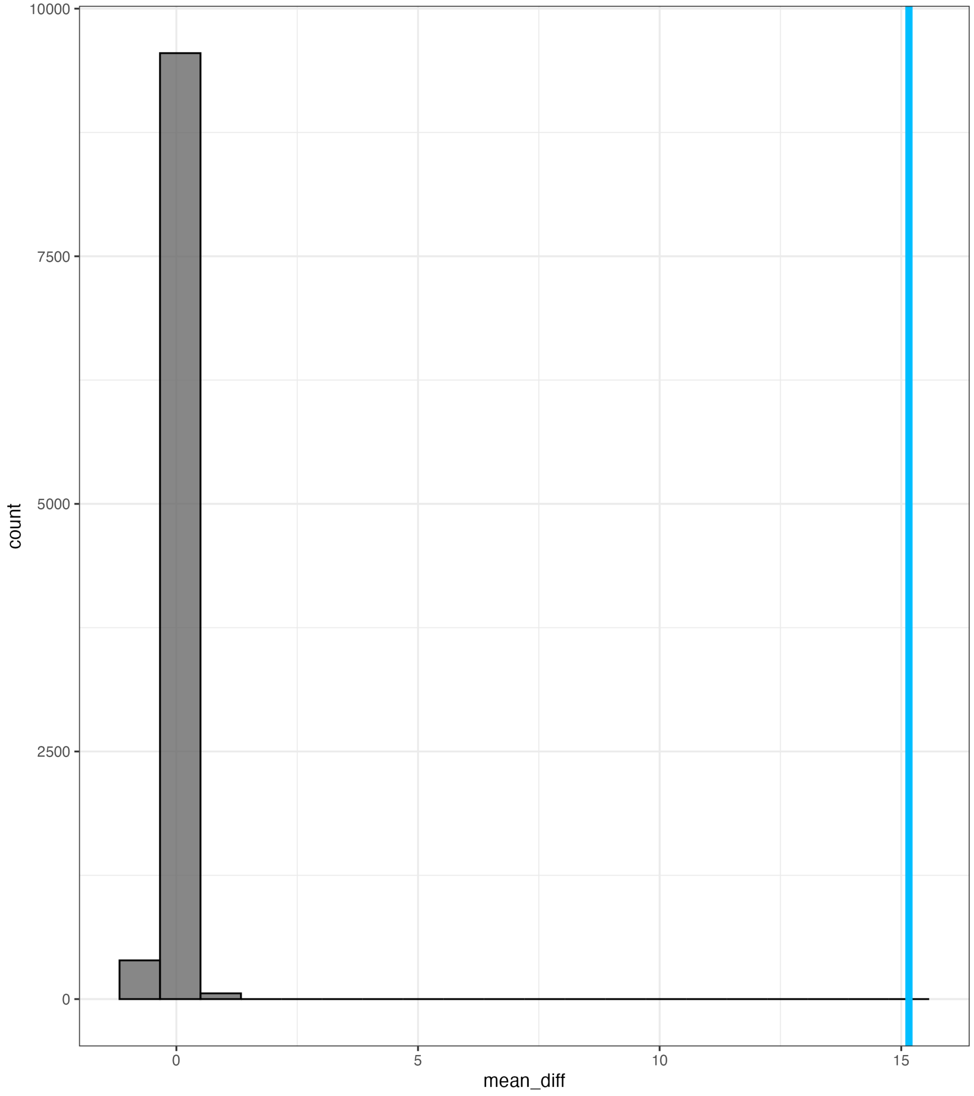
Monte Carlo Permutation Test

```r
mean_diff <-
  replicate(1e4, ReassignScores(test_data))

mean(mean_diff >= true_diff)

# [1] 0

data.frame(mean_diff=mean_diff) |>
  ggplot(aes(x=mean_diff)) +
  geom_histogram(alpha = 0.7,
                 color = 'black',
                 bins = 20) +
  geom_vline(aes(xintercept=true_diff),
                 linewidth = 2,
                 color='deepskyblue') +
  theme_bw()
```



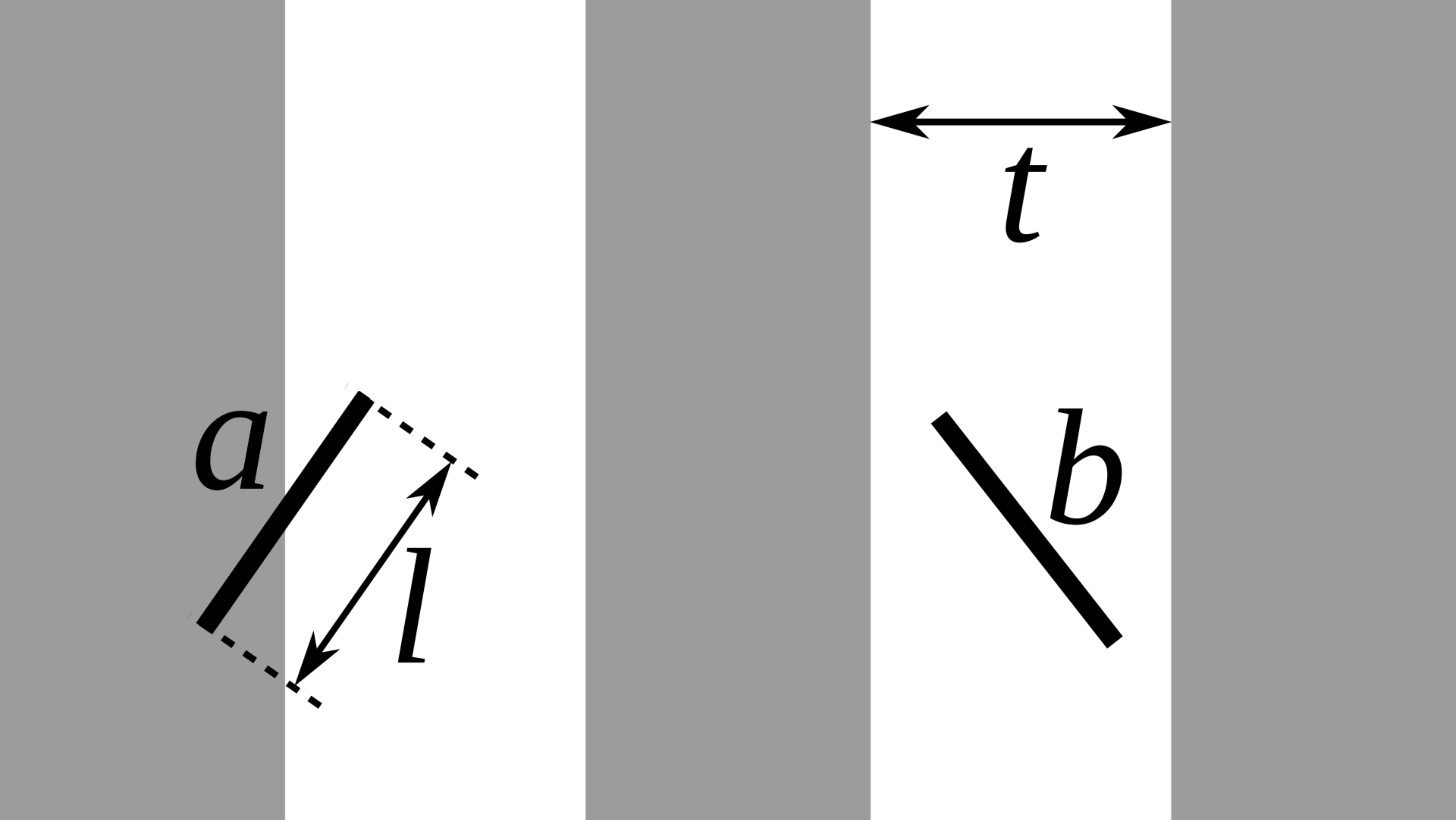
Buffon’s Needle

The Problem

Given a needle of length l dropped on a floor with parallel lines distance t apart, what is the probability that the needle will lie across a line when landing?

Plan:

1. Write a function that simulates tossing one needle on the floor and returns `TRUE` if it crosses a line and `FALSE` otherwise
2. Run the function 10^5 times
3. Compute the proportion of trials that cross a line


t
a
l
b

Tossing a Needle

```r
TossNeedle <- function(l, t){
  # TODO: Simulate a needle toss on the floor
  left_edge <- runif(1, min=0, max=t)
  theta <- runif(1, min=-pi/2, max=pi/2)
  # TODO: Decide if it crosses a threshold
  right_edge <- left_edge + l * cos(theta)
  result <- right_edge > t
  # TODO: Return TRUE or FALSE depending
  return(result)
}

TossNeedle(1, 2)

# [1] FALSE

replicate(3, TossNeedle(1, 2))

# [1]  TRUE FALSE FALSE
```

Tossing Lots of Needles

```
mean(replicate(1e5, TossNeedle(1, 2)))

# [1] 0.3185

mean(replicate(1e5, TossNeedle(1, 5)))

# [1] 0.12809

mean(replicate(1e5, TossNeedle(1, 10)))

# [1] 0.06178
```

Wrap Up

Recap: Monte Carlo Simulations

- Often a much faster way to solve problems than doing it analytically
- The general setup:
  - Write a function to do it once
  - Run the function a lot of times
  - Look at the distribution of the results
- PS3 is just six questions where you do this over and over in different ways
- I think it's very fun, but maybe I'm a sociopath?

Final Thoughts

- PollEv.com/klintkanopka