

STAT 540 — Computing in Statistics

R Programming, Permutation & Bootstrap Quiz

Yen-Yi Ho

University of South Carolina

2026-09-21

Section 1

The Questions

Q1. Logical Indexing

```
library(dslabs)
murder_rate <- murders$total / murders$population * 100000
west <- murders$region == "West"
safe <- murder_rate <= 1
ind <- safe & west
```

What does `sum(ind)` return?

- A. The positions of the states that qualify
- B. The combined murder rate of the western states
- C. The number of western states with a rate of 1 or less
- D. A single TRUE or FALSE

Q1. Logical Indexing

```
library(dslabs)
murder_rate <- murders$total / murders$population * 100000
west <- murders$region == "West"
safe <- murder_rate <= 1
ind <- safe & west
```

What does `sum(ind)` return?

- A. The positions of the states that qualify
- B. The combined murder rate of the western states
- C. The number of western states with a rate of 1 or less
- D. A single TRUE or FALSE

Answer: C

`safe & west` compares **element by element**, so `ind` is a logical vector of length 51 and `sum()` counts the TRUEs. Use `murders$state[ind]` for the names, `which(ind)` for the positions. (`&&` takes single values, not vectors.)

Q2. HW3 — for Loop Inside a Function

HW3 Problem 5 asks for your own factorial. What does `myfac(0)` return?

```
myfac <- function(x){  
  ans <- 1  
  for (i in 1:x) ans <- ans * i  
  return(ans)  
}  
myfac(0)
```

- A.** 1 **B.** 0 **C.** NA **D.** Error — invalid loop range

Q2. HW3 — for Loop Inside a Function

HW3 Problem 5 asks for your own factorial. What does `myfac(0)` return?

```
myfac <- function(x){  
  ans <- 1  
  for (i in 1:x) ans <- ans * i  
  return(ans)  
}  
myfac(0)
```

A. 1 **B.** 0 **C.** NA **D.** Error — invalid loop range

Answer: B — 0

`1:0` counts **down**: it is `c(1, 0)`, not an empty sequence. So the loop runs twice and multiplies `ans` by 0. That is exactly why the solution wraps the loop in `if (x == 0) { ans <- 1 } else { ... }`. Use `seq_len(x)` — which *is* empty when `x = 0` — and add `if (x < 0) stop(...)` to guard negative input.

Q3. Permutation

A permutation test compares mean BMI between physically active and inactive adults. In each of the M iterations, **what is resampled?**

- A. The group labels, *without replacement*; the BMI values stay put
- B. The BMI values, *with replacement*; the labels stay fixed
- C. New BMI values simulated from a normal fitted to each group
- D. The labels, *with replacement*, so some adults are labeled twice

Q3. Permutation

A permutation test compares mean BMI between physically active and inactive adults. In each of the M iterations, **what is resampled?**

- A. The group labels, *without replacement*; the BMI values stay put
- B. The BMI values, *with replacement*; the labels stay fixed
- C. New BMI values simulated from a normal fitted to each group
- D. The labels, *with replacement*, so some adults are labeled twice

Answer: A — shuffle the labels, keep the values

Under H_0 the two groups come from the same distribution, so the labels carry no information and every assignment of n_1 of the n observations to group 1 is equally likely (**exchangeability**). In R that is `g_star <- sample(g)` — a permutation, with `no replace = TRUE`. The p -value is the proportion of shuffles at least as extreme as the observed statistic: $p = \#\{|t^{*b}| \geq |t|\} / M$.

Q4. Bootstrap CI

`bootstrap_ci()` has stored 2,000 resampled medians in `boot_meds`. Which line gives the 95% **percentile** CI for the median?

- A. `mean(boot_meds) + c(-1.96, 1.96) * sd(x)`
- B. `range(boot_meds)`
- C. `quantile(boot_meds, probs = c(0.025, 0.975))`
- D. `quantile(x, probs = c(0.025, 0.975))`

Q4. Bootstrap CI

`bootstrap_ci()` has stored 2,000 resampled medians in `boot_meds`. Which line gives the 95% **percentile** CI for the median?

- A. `mean(boot_meds) + c(-1.96, 1.96) * sd(x)`
- B. `range(boot_meds)`
- C. `quantile(boot_meds, probs = c(0.025, 0.975))`
- D. `quantile(x, probs = c(0.025, 0.975))`

Answer: C

The percentile method cuts the extreme 2.5% from each tail of the **bootstrap distribution of the statistic**. Why the others fail: **A** assumes normality and uses the SD of the raw data, not of the bootstrap medians; **B** returns the two most extreme replicates, which only get wider as *B* grows; **D** describes the spread of the BMI values themselves, not the uncertainty in their median.

Q5. `replace = TRUE` or not?

```
idx    <- sample(1:n, size = n, replace = TRUE)  # line a  
g_star <- sample(g)                               # line b
```

`x` is a data vector of length `n`; `g` is the vector of group labels.

- A. `a` is a permutation of the data; `b` is a bootstrap resample
- B. Both are bootstrap resamples — `sample()` always replaces
- C. Equivalent on average, so it does not matter which you use
- D. `a` is a bootstrap resample index; `b` is a permutation of the labels

Q5. `replace = TRUE` or not?

```
idx    <- sample(1:n, size = n, replace = TRUE)  # line a  
g_star <- sample(g)                             # line b
```

`x` is a data vector of length `n`; `g` is the vector of group labels.

- A. `a` is a permutation of the data; `b` is a bootstrap resample
- B. Both are bootstrap resamples — `sample()` always replaces
- C. Equivalent on average, so it does not matter which you use
- D. `a` is a bootstrap resample index; `b` is a permutation of the labels

Answer: D

`replace = TRUE` is what makes a bootstrap resample a *resample*: some values appear several times, others not at all — about **63.2%** of the original observations show up at least once. `sample(g)` with no other arguments just reorders; the default is `replace = FALSE`.

Q5 Follow-up: The Code

```
# Bootstrap: resample VALUES with replacement
idx      <- sample(1:n, size = n, replace = TRUE)
x_star    <- x[idx]
boot_med[b] <- median(x_star)

# Permutation: shuffle LABELS, no replacement
g_star    <- sample(g)
t_star[b]  <- mean(x[g_star == "Yes"]) - mean(x[g_star == "No"])
```

Forgetting `replace = TRUE` in the bootstrap loop is the classic bug: every “resample” is then a permutation of the same values, so **every bootstrap median equals the observed median** and the CI collapses to a point. Both loops should **pre-allocate** with `numeric(B)` rather than grow a vector with `c()`.

Answer Key

Q	Topic	Answer
1	logical indexing	C
2	HW3 P5 for	B
3	permutation	A
4	bootstrap CI	C
5	replace =	D

The through-line

- ▶ `sum()` on a logical vector counts the TRUEs
- ▶ `1:0` is `c(1, 0)` — use `seq_len()` instead
- ▶ Permutation shuffles **labels**, no replacement
- ▶ Percentile CI = quantiles of the **replicates**
- ▶ `replace = TRUE` → bootstrap; `FALSE` → permutation