

STAT — Computing in Statistics

Permutation Test Quiz

Yen-Yi Ho

University of South Carolina

2026-09-14

Section 1

The Questions

Q1. Tidy Data

In a **tidy** dataset, what does each **row** represent?

- A. A variable
- B. A dataset
- C. A single **observation**
- D. A summary statistic

Q1. Tidy Data

In a **tidy** dataset, what does each **row** represent?

- A. A variable
- B. A dataset
- C. A single **observation**
- D. A summary statistic

Answer: C

The three rules of tidy data: each **variable** has its own column, each **observation** has its own row, each **value** has its own cell. Getting the data into this shape is what makes `filter()`, `mutate()` and `summarize()` behave predictably — the verbs are written assuming rows are observations.

Q2. The Native Pipe

What value does this pipe chain produce?

```
c(1, 4, 9) |> sqrt() |> sum()
```

- A. 14
- B. 6
- C. 3.74
- D. Error — you cannot pipe into base R functions

Q2. The Native Pipe

What value does this pipe chain produce?

```
c(1, 4, 9) |> sqrt() |> sum()
```

- A. 14
- B. 6
- C. 3.74
- D. Error — you cannot pipe into base R functions

Answer: B

Work left to right: `sqrt(c(1, 4, 9))` gives `c(1, 2, 3)`, and `sum(c(1, 2, 3))` gives **6**. The native `|>` (R 4.1+) inserts the left-hand result as the **first argument** of the right-hand function, and works with *any* function — base R included, so **D** is a common but wrong instinct. Distractor **A** is `sum(c(1,4,9))`; **C** is `sqrt(14)`.

Q3. dplyr Verbs

Which dplyr verb **adds new columns** to a data frame without removing the existing ones?

- A. `filter()`
- B. `select()`
- C. `arrange()`
- D. `mutate()`

Q3. dplyr Verbs

Which dplyr verb **adds new columns** to a data frame without removing the existing ones?

- A. filter()
- B. select()
- C. arrange()
- D. mutate()

Answer: D

Verb	What it changes
mutate()	adds or transforms columns
filter()	keeps or drops rows (by condition)
select()	keeps or drops columns (by name)
arrange()	reorders rows

The usual mix-up is filter() vs select(): one acts on rows, the other on columns.

Q4. `summarize()` vs `reframe()`

True or false? `summarize()` can return multiple rows per group — one row per quantile, say — just like `reframe()`.

- A. True
- B. False

Q4. `summarize()` vs `reframe()`

True or false? `summarize()` can return multiple rows per group — one row per quantile, say — just like `reframe()`.

A. True

B. False

Answer: B — False

`summarize()` requires each expression to return **exactly one value** per group. `quantile(x, c(0, .25, .5, .75, 1))` returns five, so it errors. `reframe()` exists precisely to lift that restriction: use it whenever one group should produce several rows.

Q4 Follow-up: The Code

```
# Errors: summarize() wants ONE value per group
heights |>
  group_by(sex) |>
  summarize(q = quantile(height, c(0, .25, .5, .75, 1)))

# Works: reframe() allows MANY rows per group
heights |>
  group_by(sex) |>
  reframe(q = quantile(height, c(0, .25, .5, .75, 1)))
```

Rule of thumb: **one number out per group** → `summarize()`. **A vector out per group** → `reframe()`. Before dplyr 1.1 the second form “worked” with a warning; it is now a hard error, so older course code may need updating.

Q5. case_when()

```
mutate(safety = case_when(  
  rate < 1 ~ "Very Safe",  
  between(rate, 1, 2) ~ "Safe",  
  rate > 2 ~ "Risky"  
))
```

In `case_when()`, what happens when **more than one** condition is TRUE for a row?

- A. The **first** matching condition wins
- B. The **last** matching condition wins
- C. All matching labels are returned as a list
- D. An error is thrown

Q5. case_when()

```
mutate(safety = case_when(  
  rate < 1 ~ "Very Safe",  
  between(rate, 1, 2) ~ "Safe",  
  rate > 2 ~ "Risky"  
))
```

In `case_when()`, what happens when **more than one** condition is TRUE for a row?

- A. The **first** matching condition wins
- B. The **last** matching condition wins
- C. All matching labels are returned as a list
- D. An error is thrown

Answer: A

Conditions are checked **top to bottom**; the first TRUE wins, so order matters — put the most specific condition first. Here `rate = 1.5` matches arms two and three, and lands in “Safe”.

Answer Key

Q	Type	Topic	Answer
1	MC	tidy data	C
2	MC	native pipe	B
3	MC	dplyr verbs	D
4	T/F	summarize	False
5	MC	case_when()	A

The through-line

- Tidy shape first — rows are **observations**; every verb assumes it
- `|>` passes the left side as the **first argument**, to any function
- Know which verb touches **rows** and which touches **columns**
- One value per group → `summarize()`; many → `reframe()`
- `case_when()` is **ordered**: first match wins