Data Types: Logicals & Numbers

PSY 410: Data Science for Psychology

Dr. Sara Weston

2026-05-04

Setup

Why data types matter

A published error

A participant scores 4, 3, 2, 5, 1 on five items of the Rosenberg Self-Esteem Scale. Item 3 is reverse-coded.

. . .

What's their total score?

. . .

If you said 15, you forgot to reverse-code — `2` should become `(5 + 1) - 2 = 4`, making the real total **17**. This kind of mistake happens in published papers, and it lives in how you handle data types.

Everything in R is a type

In R, every value has a **type**:

- `"hello"` is a **string** (character)
- `42` is a **number** (double)
- `TRUE` is a **logical** (boolean)
- `"Female"` can be a **factor** (categorical)

Understanding types helps you choose the right functions, avoid cryptic errors, and transform data correctly.

Today's focus

We'll dive into two fundamental types:

1. **Logical vectors** — TRUE/FALSE values for filtering and conditional logic
2. **Numbers** — integers and doubles for calculations and summaries

. . .

Psychology application: Computing scale scores, recoding responses, handling missing data

Logical vectors

What are logicals?

Logical vectors contain only `TRUE`, `FALSE`, or `NA`:

```
x <- c(TRUE, FALSE, TRUE, NA, FALSE)
x
```

```
[1]  TRUE FALSE  TRUE    NA FALSE
```

. . .

They're the result of comparisons:

```
ages <- c(18, 22, 45, 17, 30)
ages >= 18  # Is each age 18 or older?
```

```
[1]  TRUE  TRUE  TRUE FALSE  TRUE
```

Operator	Meaning

Comparison operators (review)

Operator	Meaning
`==`	Equal to
`!=`	Not equal to
`<`	Less than
`>`	Greater than
`<=`	Less than or equal to
`>=`	Greater than or equal to
`%in%`	Is in a set

Logical operators

Combine comparisons with **Boolean operators**:

Operator	Meaning	Example
`&`	AND	`age >= 18 & age < 65`
`|`	OR	`diagnosis == "Depression" | diagnosis == "Anxiety"`
`!`	NOT	`!is.na(response)`

AND vs OR

```r
age <- c(17, 22, 45, 70)
diagnosis <- c("Depression", "Anxiety", "Other", "Depression")

# AND: Both conditions must be TRUE
age >= 18 & age < 65
```

```
[1] FALSE  TRUE  TRUE FALSE
```

. . .

```
# OR: At least one condition must be TRUE
diagnosis == "Depression" | diagnosis == "Anxiety"
```

```
[1]  TRUE  TRUE FALSE  TRUE
```

Using logicals in filter()

```
survey_data <- tibble(
  id = 1:5,
  age = c(17, 22, 45, 70, 30),
  consent = c(FALSE, TRUE, TRUE, TRUE, TRUE),
  depression = c(10, 25, 18, 12, 30)
)

# Keep only consenting adults with high depression
survey_data |>
  filter(consent & age >= 18 & depression >= 20)
```

```
# A tibble: 2 x 4
     id   age consent depression
  <int> <dbl> <lgl>        <dbl>
1     2    22 TRUE            25
2     5    30 TRUE            30
```

any() and all()

Check if **any** or **all** values are TRUE:

```
responses <- c(TRUE, FALSE, FALSE, TRUE)

any(responses)   # Is at least one TRUE?
```

```
[1] TRUE
```

```
all(responses)   # Are all TRUE?
```

```
[1] FALSE
```

. . .

Useful for checking data quality:

```
# Did anyone fail the attention check?
any(survey_data$consent == FALSE)
```

```
[1] TRUE
```

Counting with logicals

Remember: `TRUE = 1` and `FALSE = 0`

```
# How many adults?
sum(survey_data$age >= 18)
```

```
[1] 4
```

```
# What proportion are adults?
mean(survey_data$age >= 18)
```

```
[1] 0.8
```

Conditional values

if_else(): Two-way decisions

`if_else()` creates new values based on a condition:

```
survey_data |>
  mutate(
    age_group = if_else(age >= 18, "Adult", "Minor")
  )
```

```
# A tibble: 5 x 5
     id   age consent depression age_group
  <int> <dbl> <lgl>        <dbl> <chr>
1     1    17 FALSE           10 Minor
2     2    22 TRUE            25 Adult
3     3    45 TRUE            18 Adult
4     4    70 TRUE            12 Adult
5     5    30 TRUE            30 Adult
```

Syntax: `if_else(condition, value_if_true, value_if_false)`

Handling NA with if_else()

By default, `if_else()` keeps `NA` values:

```r
responses <- c(1, 2, NA, 4, 5)

if_else(responses >= 3, "High", "Low")
```

```
[1] "Low"  "Low"  NA     "High" "High"
```

. . .

You can specify what to do with `NA`:

```r
if_else(responses >= 3, "High", "Low", missing = "No response")
```

```
[1] "Low"         "Low"         "No response" "High"        "High"
```

case_when(): Multiple conditions

For more than two outcomes, use `case_when()`:

```r
survey_data |>
  mutate(
    depression_category = case_when(
      depression < 14 ~ "Minimal",
      depression < 20 ~ "Mild",
      depression < 29 ~ "Moderate",
```

```
      depression >= 29 ~ "Severe"
    )
  )
```

```
# A tibble: 5 x 5
     id   age consent depression depression_category
  <int> <dbl> <lgl>        <dbl> <chr>
1     1    17 FALSE           10 Minimal
2     2    22 TRUE            25 Moderate
3     3    45 TRUE            18 Mild
4     4    70 TRUE            12 Minimal
5     5    30 TRUE            30 Severe
```

case_when() rules

- Conditions are checked **in order**
- The first `TRUE` condition wins
- If no condition matches, you get `NA`

. . .

Always include a catch-all:

```
case_when(
  age < 18 ~ "Minor",
  age < 65 ~ "Adult",
  age >= 65 ~ "Senior",
  .default = NA  # Explicit about NAs
)
```

if_else() vs case_when() — when to use which

Situation	Use
Two outcomes (yes/no, pass/fail)	`if_else()`
Three or more categories	`case_when()`
Recoding a Likert scale into groups	`case_when()`
Flagging a single condition	`if_else()`

When in doubt, start with `if_else()`. Graduate to `case_when()` when you need more categories.

Psychology example: Reverse coding

Rosenberg Self-Esteem Scale
Please record the appropriate answer for each item, depending on whether you Strongly agree, agree, disagree, or strongly disagree with it.

1 = Strongly agree
2 = Agree
3 = Disagree
4 = Strongly disagree

____ 1. On the whole, I am satisfied with myself.
____ 2. At times I think I am no good at all.
____ 3. I feel that I have a number of good qualities.
____ 4. I am able to do things as well as most other people.
____ 5. I feel I do not have much to be proud of.
____ 6. I certainly feel useless at times.
____ 7. I feel that I'm a person of worth.
____ 8. I wish I could have more respect for myself.
____ 9. All in all, I am inclined to think that I am a failure.
____ 10. I take a positive attitude toward myself.

Psychology example: Reverse coding

Many scales have reverse-coded items:

```r
# Original responses (1-5 scale)
rosenberg <- tibble(
  id = 1:3,
  item1 = c(5, 4, 3),  # Regular item
  item2 = c(2, 3, 4)   # Reverse-coded item
)

# Reverse code item2
rosenberg |>
  mutate(
    item2_reversed = case_when(
      item2 == 1 ~ 5,
      item2 == 2 ~ 4,
```

```
      item2 == 3 ~ 3,
      item2 == 4 ~ 2,
      item2 == 5 ~ 1
    )
  )
```

```
# A tibble: 3 x 4
     id item1 item2 item2_reversed
  <int> <dbl> <dbl>          <dbl>
1     1     5     2              4
2     2     4     3              3
3     3     3     4              2
```

Easier reverse coding

For scales, use arithmetic:

```
rosenberg |>
  mutate(
    item2_reversed = 6 - item2  # For 1-5 scale: 6 - x
  )
```

```
# A tibble: 3 x 4
     id item1 item2 item2_reversed
  <int> <dbl> <dbl>          <dbl>
1     1     5     2              4
2     2     4     3              3
3     3     3     4              2
```

. . .

General formula: `(max + min) - original_value`

- 1-5 scale: `6 - x` (because $1 + 5 = 6$)
- 1-7 scale: `8 - x` (because $1 + 7 = 8$)

Pair coding break

Your turn: Recode responses

You have survey data with a 1-7 attention check item where the correct answer is 4:

```
attention_data <- tibble(
  participant_id = 1:6,
  attention_check = c(4, 3, 4, 7, NA, 4)
)
```

1. Create a new column `passed` that is `TRUE` if they answered 4, `FALSE` otherwise
2. Create a column `status` with three values: "Passed", "Failed", or "No response" (for NA)
3. What proportion of participants passed?

Time: 10 minutes

Numbers

Types of numbers

R distinguishes two numeric types:

- **Integers:** whole numbers (1, 2, 3)
- **Doubles:** numbers with decimals (1.5, 2.718, 3.14159)

. . .

Most of the time, R uses doubles automatically:

```
typeof(42)
```

```
[1] "double"
```

```
typeof(42L)  # The L forces it to be an integer
```

```
[1] "integer"
```

. . .

You rarely need to worry about this distinction!

Rounding

```r
reaction_times <- c(245.678, 198.234, 312.891)

round(reaction_times)          # Round to nearest integer
```

```
[1] 246 198 313
```

```r
round(reaction_times, 1)       # Round to 1 decimal place
```

```
[1] 245.7 198.2 312.9
```

```r
floor(reaction_times)          # Round down
```

```
[1] 245 198 312
```

```r
ceiling(reaction_times)        # Round up
```

```
[1] 246 199 313
```

Summary functions (review)

Common calculations you've been using:

```r
scores <- c(12, 18, 25, 22, 30, 15, NA)

mean(scores, na.rm = TRUE)
```

```
[1] 20.33333
```

```r
median(scores, na.rm = TRUE)
```

```
[1] 20
```

```r
sd(scores, na.rm = TRUE)
```

```
[1] 6.65332
```

```r
min(scores, na.rm = TRUE)
```

```
[1] 12
```

```r
max(scores, na.rm = TRUE)
```

```
[1] 30
```

The na.rm argument

Most summary functions need `na.rm = TRUE` to handle missing data:

```r
scores <- c(12, 18, NA, 22, 30)

mean(scores)              # Returns NA
```

```
[1] NA
```

```r
mean(scores, na.rm = TRUE)  # Ignores NA
```

```
[1] 20.5
```

. . .

> Warning
>
> **Think carefully** — Should you exclude missing values? Or is missingness meaningful?

Counting non-missing values

```r
scores <- c(12, 18, NA, 22, 30, NA)

sum(!is.na(scores))  # Count non-missing
```

```
[1] 4
```

. . .

Or in a summary:

```r
tibble(scores) |>
  summarize(
    n = sum(!is.na(scores)),
    mean_score = mean(scores, na.rm = TRUE)
  )
```

```
# A tibble: 1 x 2
      n mean_score
  <int>      <dbl>
1     4       20.5
```

Computing scale scores

Real psychology task: Scale scoring

You've collected survey data with multiple items per scale:

```r
scale_data <- tibble(
  id = 1:4,
  anxiety1 = c(3, 2, 4, NA),
  anxiety2 = c(4, 3, 5, 2),
  anxiety3 = c(3, 2, 4, 1),
  anxiety4 = c(4, 3, NA, 2)
)

scale_data
```

```
# A tibble: 4 x 5
     id anxiety1 anxiety2 anxiety3 anxiety4
  <int>    <dbl>    <dbl>    <dbl>    <dbl>
1     1        3        4        3        4
2     2        2        3        2        3
3     3        4        5        4       NA
4     4       NA        2        1        2
```

How do you compute a total or mean score?

Option 1: Manual addition

```
scale_data |>
  mutate(
    anxiety_total = anxiety1 + anxiety2 + anxiety3 + anxiety4
  )
```

```
# A tibble: 4 x 6
     id anxiety1 anxiety2 anxiety3 anxiety4 anxiety_total
  <int>    <dbl>    <dbl>    <dbl>    <dbl>         <dbl>
1     1        3        4        3        4            14
2     2        2        3        2        3            10
3     3        4        5        4       NA            NA
4     4       NA        2        1        2            NA
```

. . .

Problem: If any item is `NA`, the whole sum is `NA`!

Option 2: Sum with na.rm

We can't use `na.rm` directly in `mutate()` with `+`, but we can use `sum()`:

```
scale_data |>
  rowwise() |>  # Work row-by-row
  mutate(
    anxiety_total = sum(c(anxiety1, anxiety2, anxiety3, anxiety4),
                        na.rm = TRUE)
  ) |>
  ungroup()
```

```
# A tibble: 4 x 6
     id anxiety1 anxiety2 anxiety3 anxiety4 anxiety_total
  <int>    <dbl>    <dbl>    <dbl>    <dbl>         <dbl>
1     1        3        4        3        4            14
2     2        2        3        2        3            10
3     3        4        5        4       NA            13
4     4       NA        2        1        2             5
```

Computing mean scores

```
scale_data |>
  rowwise() |>
  mutate(
    anxiety_mean = mean(c(anxiety1, anxiety2, anxiety3, anxiety4),
                        na.rm = TRUE)
  ) |>
  ungroup()
```

```
# A tibble: 4 x 6
     id anxiety1 anxiety2 anxiety3 anxiety4 anxiety_mean
  <int>    <dbl>    <dbl>    <dbl>    <dbl>        <dbl>
1     1        3        4        3        4         3.5
2     2        2        3        2        3         2.5
3     3        4        5        4       NA         4.33
4     4       NA        2        1        2         1.67
```

. . .

> Tip
>
> **Mean vs Total:** Use means when participants might have different numbers of items answered.

Cleaner approach: pivot then summarize

```
scale_data |>
  pivot_longer(
    cols = starts_with("anxiety"),
```

```
    names_to = "item",
    values_to = "response"
  ) |>
  group_by(id) |>
  summarize(
    anxiety_mean = mean(response, na.rm = TRUE),
    anxiety_total = sum(response, na.rm = TRUE),
    n_items = sum(!is.na(response))
  )
```

```
# A tibble: 4 x 4
     id anxiety_mean anxiety_total n_items
  <int>        <dbl>         <dbl>   <int>
1     1         3.5             14       4
2     2         2.5             10       4
3     3         4.33            13       3
4     4         1.67             5       3
```

When to worry about missing items

Should you compute a scale score if someone only answered 1 out of 4 items?

. . .

Common rules:

- **Require** **50%** of items answered
- **Require** **75%** for critical scales
- **Document your decision** clearly

Implementing a missingness rule

```
scale_data |>
  rowwise() |>
  mutate(
    n_answered = sum(!is.na(c(anxiety1, anxiety2, anxiety3, anxiety4))),
    anxiety_mean = if_else(
      n_answered >= 3,  # At least 3 of 4 items
      mean(c(anxiety1, anxiety2, anxiety3, anxiety4), na.rm = TRUE),
      NA_real_  # NA if too many missing
```

```r
    )
  ) |>
  ungroup()
```

```
# A tibble: 4 x 7
     id anxiety1 anxiety2 anxiety3 anxiety4 n_answered anxiety_mean
  <int>    <dbl>    <dbl>    <dbl>    <dbl>      <int>        <dbl>
1     1        3        4        3        4          4         3.5
2     2        2        3        2        3          4         2.5
3     3        4        5        4       NA          3         4.33
4     4       NA        2        1        2          3         1.67
```

More complex scales: Subscales

Some measures have multiple subscales:

```r
dass_data <- tibble(
  id = 1:3,
  # Depression items
  dass_d1 = c(2, 3, 1), dass_d2 = c(3, 3, 2),
  # Anxiety items
  dass_a1 = c(1, 4, 2), dass_a2 = c(2, 4, 3),
  # Stress items
  dass_s1 = c(3, 2, 4), dass_s2 = c(4, 3, 4)
)
```

Computing subscales

```r
dass_data |>
  rowwise() |>
  mutate(
    depression = mean(c(dass_d1, dass_d2), na.rm = TRUE),
    anxiety = mean(c(dass_a1, dass_a2), na.rm = TRUE),
    stress = mean(c(dass_s1, dass_s2), na.rm = TRUE)
  ) |>
  ungroup() |>
  select(id, depression, anxiety, stress)
```

```
# A tibble: 3 x 4
     id depression anxiety stress
  <int>      <dbl>   <dbl>  <dbl>
1     1        2.5     1.5    3.5
2     2        3       4      2.5
3     3        1.5     2.5    4
```

End-of-deck exercise

Practice: Score the PHQ-9

The PHQ-9 is a 9-item depression screener (0-3 scale):

```r
phq_data <- tibble(
  participant = 1:5,
  phq1 = c(2, 1, 3, 0, 2),
  phq2 = c(2, 0, 3, 1, NA),
  phq3 = c(1, 1, 2, 0, 2),
  phq4 = c(2, NA, 3, 0, 1),
  phq5 = c(1, 0, 3, 1, 2),
  phq6 = c(2, 1, 2, 0, 3),
  phq7 = c(1, 1, 3, 1, 2),
  phq8 = c(2, 0, 2, 0, 1),
  phq9 = c(1, 1, 3, 0, 2)
)
```

1. Compute a **total score** (sum of all 9 items)
2. Only compute the score if **at least 7 items** are answered
3. Create a **severity category**: Minimal (0-4), Mild (5-9), Moderate (10-14), Moderately Severe (15-19), Severe (20-27)

Wrapping up

Key takeaways

1. **Logical vectors** (TRUE/FALSE) are powerful for filtering and conditions
2. **if_else()** for two-way decisions, **case_when()** for multiple outcomes
3. **Reverse coding:** `(max + min) - value`
4. **Scale scoring:**

    - Use `rowwise()` + `sum()/mean()` with `na.rm = TRUE`
    - Or pivot longer then group and summarize
    - Decide how to handle missing items

5. **Always document** your scoring decisions

Before next class

Read:

- R4DS Ch 14: Strings (sections 14.1–14.3 only)
- R4DS Ch 16: Factors

Do:

- Submit Assignment 5
- Practice computing scale scores with your own data
- Think about categorical variables in your final project

The one thing to remember

Scoring a scale correctly is the most common data task in psychology — and the easiest place to introduce errors.

See you Wednesday for strings and factors!