Tutorial 03: Distributions

```
#| edit: false
#| output: false
webr::install("gradethis", quiet = TRUE)
library(gradethis)
options(webr.exercise.checker = function(
  label, user_code, solution_code, check_code, envir_result, evaluate_result,
  envir_prep, last_value, engine, stage, ...
) {
  if (is.null(check_code)) {
    # No grading code, so just skip grading
    invisible(NULL)
  } else if (is.null(label)) {
    list(
      correct = FALSE,
      type = "warning",
      message = "All exercises must have a label."
    )
  } else if (is.null(solution_code)) {
    list(
      correct = FALSE,
      type = "warning",
      message = htmltools::tags$div(
        htmltools::tags$p("A problem occurred grading this exercise."),
        htmltools::tags$p(
          "No solution code was found. Note that grading exercises using the ",
          htmltools::tags$code("gradethis"),
          "package requires a model solution to be included in the document."
        )
      )
    )
  } else {
    gradethis::gradethis_exercise_checker(
```

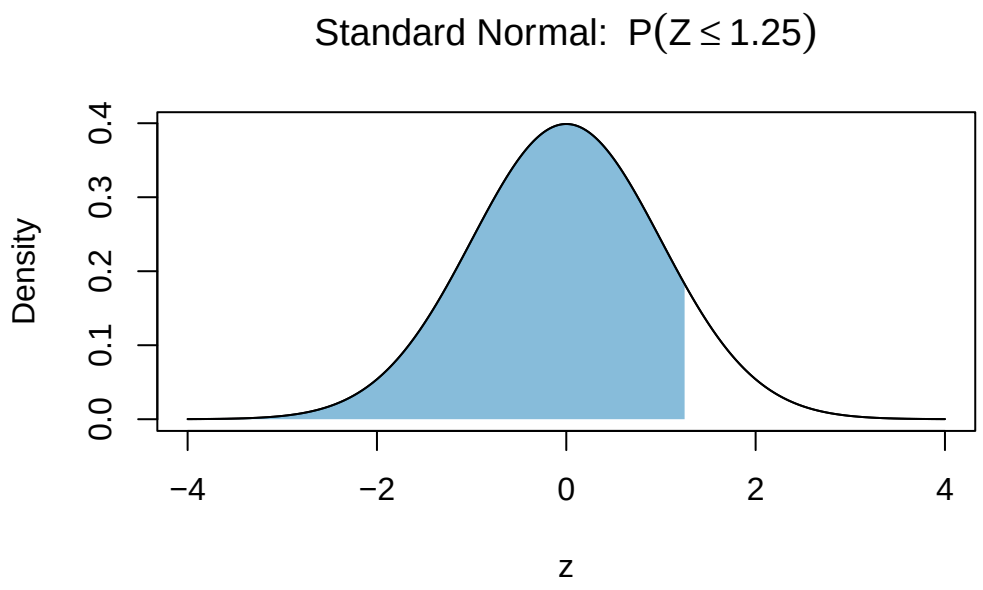
```
      label = label, solution_code = solution_code, user_code = user_code,
      check_code = check_code, envir_result = envir_result,
      evaluate_result = evaluate_result, envir_prep = envir_prep,
      last_value = last_value, stage = stage, engine = engine)
  }
})
```

Q1 — Standard Normal - Left Tail Probability

Compute $P(Z <= 1.25)$, where $Z \sim N(0, 1)$.

> **Info**
>
> Remember: R calculates area to the left unlike probability tables.

> **Info**
>
> This is the graphical representation of the question:




```
#| exercise: norm_left
#| exercise.lines: 1
#| echo: false
```

```
#your answer here
```

Use pnorm.

Solution.

```
#| exercise: norm_left
#| solution: true
pnorm(1.25)
```

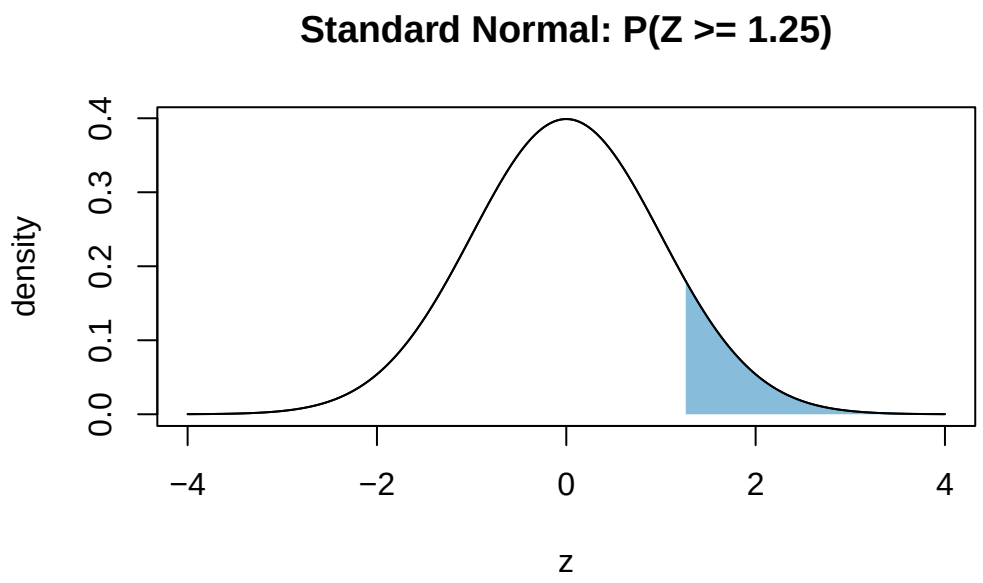
```
#| exercise: norm_left
#| check: true
gradethis::grade_this({
  exp <- pnorm(1.25)
  x <- .result
  if (!is.numeric(x) || length(x) != 1L || !is.finite(x)) {
    fail("Your last line must evaluate to a single numeric value.")
  }
  if (abs(x - exp) < 1e-6) pass("  Correct.")
  else fail("Not quite - check the function/arguments and try again.")
})
```

Q2 — Standard Normal - Right Tail Probability

Compute $P(Z >= 1.25)$, where $Z \sim N(0, 1)$.

> **Info**
>
> Remember: R calculates area to the left unlike probability tables.

> **Info**
>
> This is the graphical representation of the question:




```
#| exercise: norm_right
#| exercise.lines: 1
#| echo: false
#your answer here
```

pnorm gives area to the left. Remember, we need area to the right.

Solution.

```
#| exercise: norm_right
#| solution: true
1 - pnorm(1.25)
```

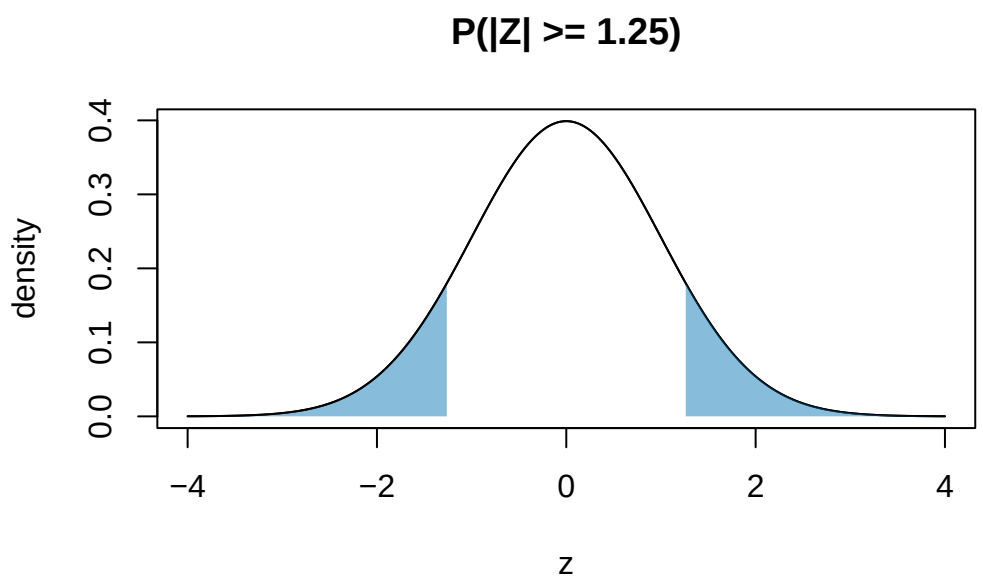
```
#| exercise: norm_right
#| check: true
gradethis::grade_this({
  x <- .result
  if (!is.numeric(x) || length(x)!=1L || !is.finite(x)) fail("`Your answer should be a single
  if (abs(x - (1 - pnorm(1.25))) < 1e-6) pass("  Correct.")
  else fail("Not quite. There's still something missing.")
})
```

Q3 — Standard Normal - Two-Sided Probability

Compute $P(|\ Z\ |>= 1.25)$, where $Z \sim N(0, 1)$.

> **Info**
>
> Remember: R calculates area to the left unlike probability tables.

> **Info**
>
> This is the graphical representation of the question:




```
#| exercise: norm_two_sided
#| exercise.lines: 1
#| echo: false
#your answer here
```

Effectively use pnorm.

Solution.

```
#| exercise: norm_two_sided
#| solution: true
p_two <- 2 * (1 - pnorm(1.25))
```

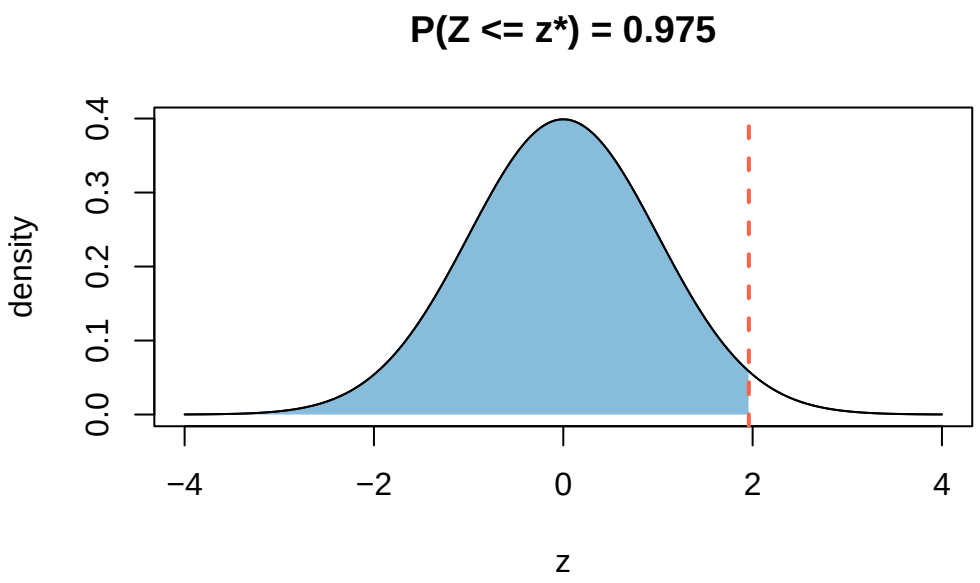
```
#| exercise: norm_two_sided
#| check: true
gradethis::grade_this({
  x <- .result
  if (!is.numeric(x) || length(x)!=1L || !is.finite(x)) fail("Your answer should be a single
  target <- 2 * (1 - pnorm(1.25))
  if (abs(x - target) < 1e-6) pass("  Correct: two tails summed.")
  else fail("Something is not quite right. Try again.")
})
```

Q4 — Normal Quantile (Critical Value)

Compute $Z_{0.975}$ where $Z \sim N(0,1)$.

Info

This is the graphical representation of the question:




```
#| exercise: qnorm
#| exercise.lines: 1
#| echo: false
#your answer here
```

Use qnorm.

Solution.

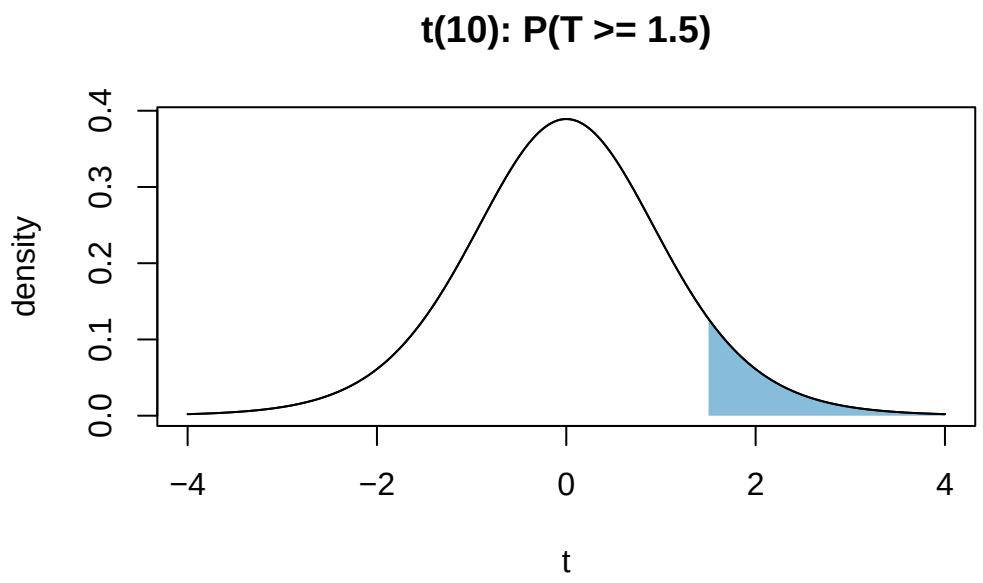
```
#| exercise: qnorm
#| solution: true
qnorm(0.975)
```

```
#| exercise: qnorm
#| check: true
gradethis::grade_this({
  x <- .result
  if (!is.numeric(x) || length(x)!=1L || !is.finite(x)) fail("Your answer should be a single
  if (abs(x - qnorm(0.975)) < 1e-6) pass("  1.96-ish is correct.")
  else fail("Not quite. Try again.")
})
```

Q5 — T-distribution - Right Tailed

Let $T \sim t_{10}$. Compute $P(T >= 1.5)$.

> **Info**
>
> This is the graphical representation of the question:




```
#| exercise: t_right
#| exercise.lines: 1
#| echo: false
#your answer here
```

Use pt.

Solution.

```
#| exercise: t_right
#| solution: true
1 - pt(1.5, df = 10)
```

```
#| exercise: t_right
#| check: true
gradethis::grade_this({
  msgs <- c()
  code <- paste(.user_code, collapse = "\n")

  # ---------- 1) Parse the code & find a pt(...) call (anywhere) ----------
  parsed <- try(parse(text = code), silent = TRUE)
```

```
find_pt_call <- function(exprs) {
  last <- NULL
  walk <- function(e) {
    if (is.call(e)) {
      fn <- as.character(e[[1]])
      if (identical(fn, "pt")) last <<- e
      # walk all children
      kids <- as.list(e)[-1]
      for (k in kids) if (is.language(k)) walk(k)
    }
  }
  for (e in exprs) walk(e)
  last
}
pt_call <- if (inherits(parsed, "try-error")) NULL else find_pt_call(parsed)

# ---------- 2) Partial-credit messages about how they computed ----------
used_pt   <- !is.null(pt_call) || grepl("\\bpt\\s*\\(", code)
used_rt1m <- grepl("1\\s*-\\s*pt\\s*\\(", code)
used_rtlt <- grepl("lower\\.tail\\s*=\\s*(FALSE|F)\\b", code)

msgs <- c(msgs, if (used_pt)  "  used `pt()`." else "  did not call `pt()`.")
if (used_rt1m || used_rtlt) {
  msgs <- c(msgs, "  computed the **right tail** (either `1 - pt(...)` or `lower.tail = FAL
} else {
  msgs <- c(msgs, "  For the **right tail**, use `1 - pt(q, df)` or `pt(q, df, lower.tail =
}

# Check q and df if we have a pt(...) call we can inspect
if (!is.null(pt_call)) {
  args <- as.list(pt_call)[-1]
  # q
  q_expr <- if (length(args) >= 1) args[[1]] else NULL
  q_val  <- try(eval(q_expr, envir = .envir_result), silent = TRUE)
  msgs <- c(msgs,
    if (!inherits(q_val, "try-error") && is.numeric(q_val) && length(q_val) == 1 && isTRUE
      "  first argument `q = 1.5`."
    else
      "  first argument should be 1.5."
  )
  # df (named or positional second)
  df_expr <- if (!is.null(names(args)) && "df" %in% names(args)) args[["df"]]
```

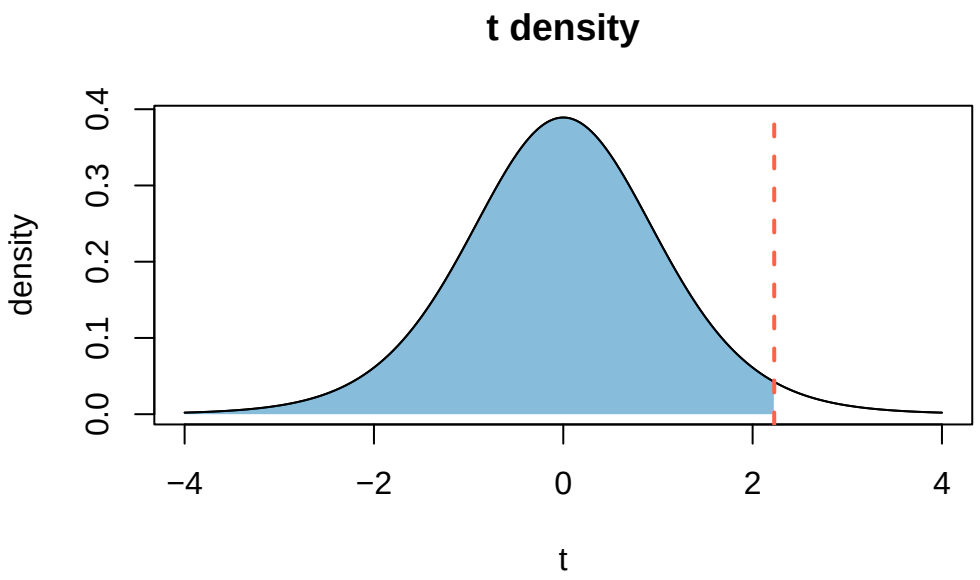
```
              else if (length(args) >= 2) args[[2]] else NULL
    df_val  <- try(eval(df_expr, envir = .envir_result), silent = TRUE)
    msgs <- c(msgs,
      if (!is.null(df_expr) && !inherits(df_val, "try-error") && is.numeric(df_val) && lengt
        "  degrees of freedom `df = 10`."
      else
        "  `df` should be 10."
    )
  }

  # ---------- 3) Numeric correctness from the LAST expression ----------
  target <- 1 - pt(1.5, df = 10)
  val <- .result

  # If last expression isn't numeric, try to re-evaluate the last non-comment line
  if (!is.numeric(val) || length(val) != 1L || !is.finite(val)) {
    lines <- strsplit(code, "\n", fixed = TRUE)[[1]]
    # drop blanks and full-line comments
    lines <- lines[grepl("^\\s*[^#]", lines)]
    if (length(lines) > 0) {
      exprs <- try(parse(text = paste(lines, collapse = "\n")), silent = TRUE)
      if (!inherits(exprs, "try-error") && length(exprs) > 0) {
        val <- try(eval(exprs[[length(exprs)]], envir = .envir_result), silent = TRUE)
      }
    }
  }

  if (!is.numeric(val) || length(val) != 1L || !is.finite(val)) {
    fail(paste(c(msgs, "  Your **last line** must evaluate to a single numeric value (e.g., `
  }

  if (abs(val - target) < 1e-6) {
    pass(paste(c(msgs, "  Numeric answer is correct."), collapse = "\n"))
  } else {
    fail(paste(c(msgs, "  Numeric value is not correct yet."), collapse = "\n"))
  }
})
```

Q6 — T Quantile - Left Probability

Find t^* such that $P(T <= t^*) = 0.975$, where $T \sim t_{10}$.

Info

This is the graphical representation of the question:




```
#| exercise: t_q
#| exercise.lines: 1
#| echo: false
#your answer here
```

Use qt.

Solution.

```
#| exercise: t_q
#| solution: true
qt(0.975, df = 10)
```

```
#| exercise: t_q
#| check: true
gradethis::grade_this({
  msgs <- c()
  code <- paste(.user_code, collapse = "\n")
  target <- qt(0.975, df = 10)
```

```
# ---- parse & find qt(...) anywhere (RHS, nested, etc.) ----
parsed <- try(parse(text = code), silent = TRUE)
find_call <- function(exprs, fname) {
  last <- NULL
  walk <- function(e) {
    if (is.call(e)) {
      fn <- as.character(e[[1]])
      if (identical(fn, fname)) last <<- e
      for (k in as.list(e)[-1]) if (is.language(k)) walk(k)
    }
  }
  for (e in exprs) walk(e)
  last
}
qt_call <- if (inherits(parsed, "try-error")) NULL else find_call(parsed, "qt")

# ---- partial-credit messages ----
msgs <- c(msgs, if (!is.null(qt_call) || grepl("\\bqt\\s*\\(", code)) "  used `qt()`." else

if (!is.null(qt_call)) {
  args <- as.list(qt_call)[-1]

  # p (first arg)
  p_expr <- if (length(args) >= 1) args[[1]] else NULL
  p_val  <- try(eval(p_expr, envir = .envir_result), silent = TRUE)
  msgs <- c(msgs,
    if (!inherits(p_val, "try-error") && is.numeric(p_val) && length(p_val) == 1 && abs(p_v
      "  probability p = 0.975."
    else
      "  first argument should be p = 0.975."
  )

  # df (named or positional 2nd)
  df_expr <- if (!is.null(names(args)) && "df" %in% names(args)) args[["df"]]
             else if (length(args) >= 2) args[[2]] else NULL
  df_val  <- try(eval(df_expr, envir = .envir_result), silent = TRUE)
  msgs <- c(msgs,
    if (!is.null(df_expr) && !inherits(df_val, "try-error") && is.numeric(df_val) && lengt
      "  df = 10."
    else
      "  `df` should be 10."
  )
```

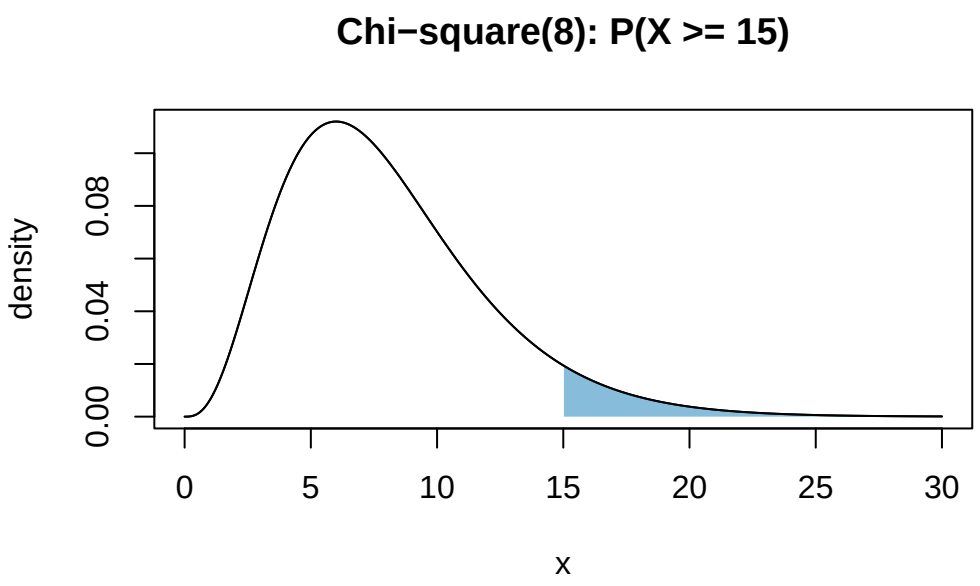
```
    # lower.tail note (optional)
    if (!is.null(names(args)) && "lower.tail" %in% names(args)) {
      lt <- try(eval(args[["lower.tail"]], envir = .envir_result), silent = TRUE)
      if (identical(lt, TRUE))  msgs <- c(msgs, "  left tail (default).")
      if (identical(lt, FALSE)) msgs <- c(msgs, "  right tail requested; then p should be 0.0
    }
  }

  # ---- numeric correctness from the LAST expression ----
  val <- .result

  # If last expression isn't numeric, try re-evaluating the last non-comment line
  if (!is.numeric(val) || length(val) != 1L || !is.finite(val)) {
    lines <- strsplit(code, "\n", fixed = TRUE)[[1]]
    lines <- lines[grepl("^\\s*[^#]", lines)]  # drop blanks & full-line comments
    if (length(lines) > 0) {
      exprs <- try(parse(text = paste(lines, collapse = "\n")), silent = TRUE)
      if (!inherits(exprs, "try-error") && length(exprs) > 0) {
        val <- try(eval(exprs[[length(exprs)]], envir = .envir_result), silent = TRUE)
      }
    }
  }

  if (!is.numeric(val) || length(val) != 1L || !is.finite(val)) {
    fail(paste(c(msgs, "  Your **last line** must evaluate to a single numeric value (e.g., `
  }

  if (abs(val - target) < 1e-6) {
    pass(paste(c(msgs, "  Numeric answer is correct."), collapse = "\n"))
  } else {
    fail(paste(c(msgs, "  Numeric value is not correct yet."), collapse = "\n"))
  }
})
```

Q7 — Chi-Square Distribution - Right Tailed

Let $X \sim \chi^2_8$. Compute $P(X >= 15)$.

Info

This is the graphical representation of the question:




```
#| exercise: chi_right
#| exercise.lines: 1
#| echo: false
#your answer here
```

Use pchisq.

Solution.

```
#| exercise: chi_right
#| solution: true
1 - pchisq(15, df = 8)
```

```
#| exercise: chi_right
#| check: true
gradethis::grade_this({
  msgs <- c()
  code <- paste(.user_code, collapse = "\n")
  target <- 1 - pchisq(15, df = 8)
```

```
# ---- parse & find pchisq(...) anywhere (RHS, nested, etc.) ----
parsed <- try(parse(text = code), silent = TRUE)
find_call <- function(exprs, fname){
  last <- NULL
  walk <- function(e){
    if (is.call(e)) {
      fn <- as.character(e[[1]])
      if (identical(fn, fname)) last <<- e
      for (k in as.list(e)[-1]) if (is.language(k)) walk(k)
    }
  }
  for (e in exprs) walk(e)
  last
}
pc <- if (inherits(parsed,"try-error")) NULL else find_call(parsed, "pchisq")

# ---- partial-credit about how it was computed ----
used_pchisq <- !is.null(pc) || grepl("\\bpchisq\\s*\\(", code)
msgs <- c(msgs, if (used_pchisq) "  used `pchisq()`." else "  did not call `pchisq()`.")

# detect right-tail approach
used_1_minus <- grepl("1\\s*-\\s*pchisq\\s*\\(", code)
used_lower_false <- grepl("lower\\.tail\\s*=\\s*(FALSE|F)\\b", code)
if (used_1_minus || used_lower_false) {
  msgs <- c(msgs, "  computed the **right tail** (either `1 - pchisq(...)` or `lower.tail =
} else {
  msgs <- c(msgs, "  For the **right tail**, use `1 - pchisq(x, df)` or `pchisq(x, df, lowe
}

# check x and df if we can inspect a pchisq(...) call
if (!is.null(pc)) {
  args <- as.list(pc)[-1]
  # x (first arg)
  x_expr <- if (length(args) >= 1) args[[1]] else NULL
  x_val  <- try(eval(x_expr, envir = .envir_result), silent = TRUE)
  msgs <- c(msgs,
    if (!inherits(x_val,"try-error") && is.numeric(x_val) && length(x_val)==1 && isTRUE(al
      "  first argument x = 15."
    else
      "  first argument should be 15."
  )
  # df (named or positional second)
```

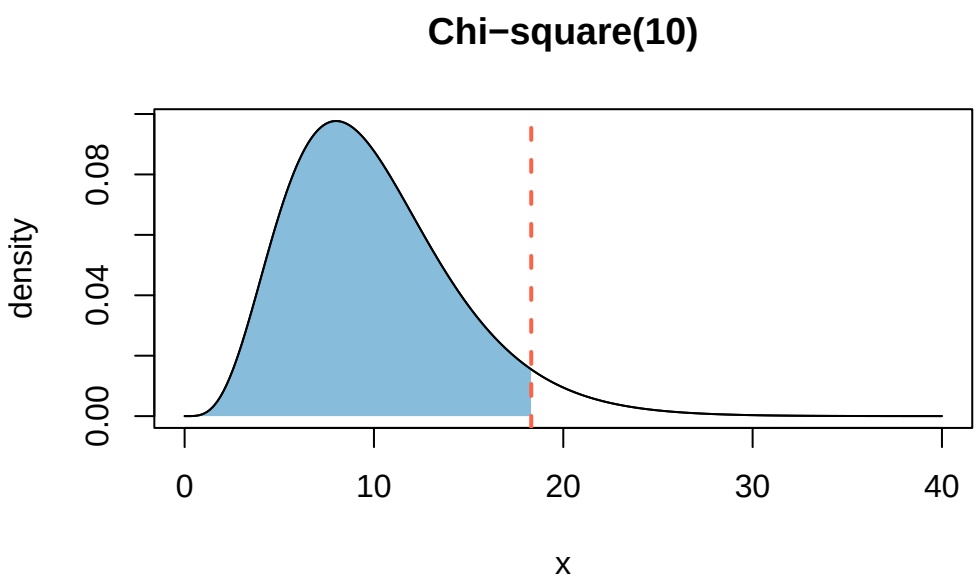
```
    df_expr <- if (!is.null(names(args)) && "df" %in% names(args)) args[["df"]]
               else if (length(args) >= 2) args[[2]] else NULL
    df_val  <- try(eval(df_expr, envir = .envir_result), silent = TRUE)
    msgs <- c(msgs,
      if (!is.null(df_expr) && !inherits(df_val,"try-error") && is.numeric(df_val) && length
        "  df = 8."
      else
        "  `df` should be 8."
    )
    # optional tail note if explicitly set
    if (!is.null(names(args)) && "lower.tail" %in% names(args)) {
      lt <- try(eval(args[["lower.tail"]], envir = .envir_result), silent = TRUE)
      if (identical(lt, TRUE))  msgs <- c(msgs, "  `lower.tail = TRUE` is left tail; for righ
      if (identical(lt, FALSE)) msgs <- c(msgs, "  explicitly used `lower.tail = FALSE` (righ
    }
  }

  # ---- numeric correctness from the LAST expression ----
  val <- .result

  # if last expression isn't numeric, try re-evaluating the last non-comment line
  if (!is.numeric(val) || length(val) != 1L || !is.finite(val)) {
    lines <- strsplit(code, "\n", fixed = TRUE)[[1]]
    lines <- lines[grepl("^\\s*[^#]", lines)]  # drop blanks & full-line comments
    if (length(lines) > 0) {
      exprs <- try(parse(text = paste(lines, collapse = "\n")), silent = TRUE)
      if (!inherits(exprs, "try-error") && length(exprs) > 0) {
        val <- try(eval(exprs[[length(exprs)]], envir = .envir_result), silent = TRUE)
      }
    }
  }

  if (!is.numeric(val) || length(val) != 1L || !is.finite(val)) {
    fail(paste(c(msgs, "  Your **last line** must evaluate to a single numeric value (e.g., `
  }

  if (abs(val - target) < 1e-6) {
    pass(paste(c(msgs, "  Numeric answer is correct."), collapse = "\n"))
  } else {
    fail(paste(c(msgs, "  Numeric value not correct yet."), collapse = "\n"))
  }
})
```

Q8 — Chi-Square Quantile - Left Probability

Find x^* such that $P(X <= x^*) = 0.95$, where $X \sim {\chi^2}_8$.

Info

This is the graphical representation of the question:




```
#| exercise: chi_q
#| exercise.lines: 1
#| echo: false
#your answer here
```

Use qchisq.

Solution.

```
#| exercise: chi_q
#| solution: true
qchisq(0.95, df = 10)
```

```
#| exercise: chi_q
#| check: true
```

```
gradethis::grade_this({
  msgs <- c()
  code   <- paste(.user_code, collapse = "\n")
  target <- qchisq(0.95, df = 10)

  # ---- parse & find qchisq(...) anywhere ----
  parsed <- try(parse(text = code), silent = TRUE)
  find_call <- function(exprs, fname){
    last <- NULL
    walk <- function(e){
      if (is.call(e)) {
        fn <- as.character(e[[1]])
        if (identical(fn, fname)) last <<- e
        for (k in as.list(e)[-1]) if (is.language(k)) walk(k)
      }
    }
    for (e in exprs) walk(e)
    last
  }
  qc <- if (inherits(parsed,"try-error")) NULL else find_call(parsed, "qchisq")

  # ---- partial-credit messages ----
  used_qchisq <- !is.null(qc) || grepl("\\bqchisq\\s*\\(", code)
  msgs <- c(msgs, if (used_qchisq) "  used `qchisq()`." else "  did not call `qchisq()`.")

  if (!is.null(qc)) {
    args <- as.list(qc)[-1]

    # p (first arg)
    p_expr <- if (length(args) >= 1) args[[1]] else NULL
    p_val  <- try(eval(p_expr, envir = .envir_result), silent = TRUE)
    msgs <- c(msgs,
      if (!inherits(p_val, "try-error") && is.numeric(p_val) && length(p_val) == 1 && abs(p_v
        "  probability p = 0.95."
      else
        "  first argument should be p = 0.95."
    )

    # df (named or positional second)
    df_expr <- if (!is.null(names(args)) && "df" %in% names(args)) args[["df"]]
               else if (length(args) >= 2) args[[2]] else NULL
    df_val  <- try(eval(df_expr, envir = .envir_result), silent = TRUE)
```

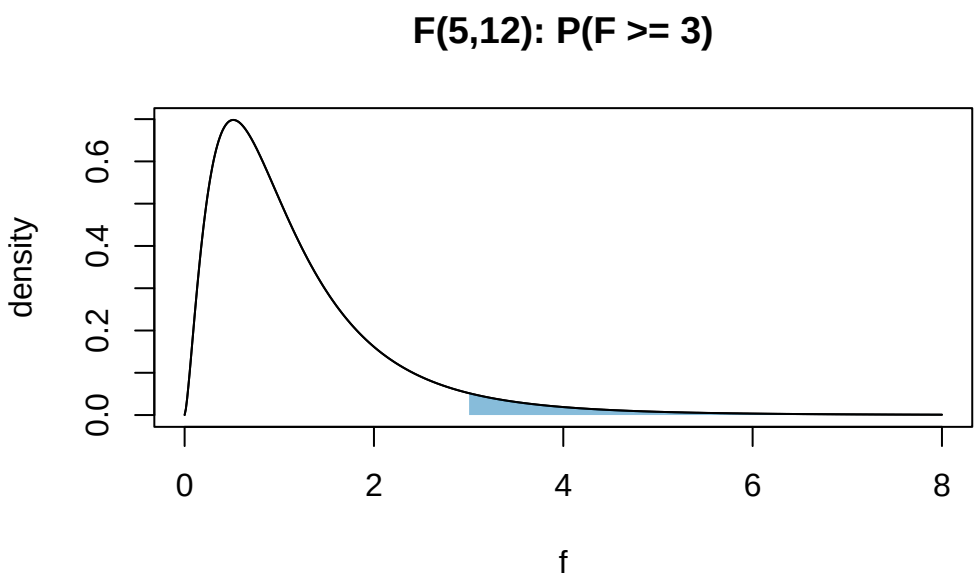
```
    msgs <- c(msgs,
      if (!is.null(df_expr) && !inherits(df_val,"try-error") && is.numeric(df_val) && length
        "  df = 10."
      else
        "  `df` should be 10."
    )

    # optional: tail note if explicitly supplied
    if (!is.null(names(args)) && "lower.tail" %in% names(args)) {
      lt <- try(eval(args[["lower.tail"]], envir = .envir_result), silent = TRUE)
      if (identical(lt, TRUE))  msgs <- c(msgs, "  left tail quantile (default).")
      if (identical(lt, FALSE)) msgs <- c(msgs, "  right-tail quantile: with `lower.tail=FALS
    }
  }

  # ---- numeric correctness from the LAST expression ----
  val <- .result

  # If the last expression isn't numeric, re-evaluate the last non-comment line
  if (!is.numeric(val) || length(val) != 1L || !is.finite(val)) {
    lines <- strsplit(code, "\n", fixed = TRUE)[[1]]
    lines <- lines[grepl("^\\s*[^#]", lines)]  # drop blanks & full-line comments
    if (length(lines) > 0) {
      exprs <- try(parse(text = paste(lines, collapse = "\n")), silent = TRUE)
      if (!inherits(exprs, "try-error") && length(exprs) > 0) {
        val <- try(eval(exprs[[length(exprs)]], envir = .envir_result), silent = TRUE)
      }
    }
  }

  if (!is.numeric(val) || length(val) != 1L || !is.finite(val)) {
    fail(paste(c(msgs, "  Your **last line** must evaluate to a single numeric value (e.g., `
  }

  if (abs(val - target) < 1e-6) {
    pass(paste(c(msgs, "  Numeric answer is correct."), collapse = "\n"))
  } else {
    fail(paste(c(msgs, "  Numeric value not correct yet."), collapse = "\n"))
  }
})
```

Q9 — F Distribution - Right Tail

Let $F \sim F_{5,12}$. Compute $P(F \geq 3)$.

Info

This is the graphical representation of the question:




```
#| exercise: f_right
#| exercise.lines: 1
#| echo: false
#your answer here
```

Use pf.

Solution.

```
#| exercise: f_right
#| solution: true
1 - pf(3, df1=5, df2=12)
```

```
#| exercise: f_right
#| check: true
```

```
gradethis::grade_this({
  msgs <- c()
  code   <- paste(.user_code, collapse = "\n")
  target <- 1 - pf(3, df1 = 5, df2 = 12)

  # ---- parse & find pf(...) anywhere ----
  parsed <- try(parse(text = code), silent = TRUE)
  find_call <- function(exprs, fname){
    last <- NULL
    walk <- function(e){
      if (is.call(e)) {
        fn <- as.character(e[[1]])
        if (identical(fn, fname)) last <<- e
        for (k in as.list(e)[-1]) if (is.language(k)) walk(k)
      }
    }
    for (e in exprs) walk(e)
    last
  }
  pc <- if (inherits(parsed,"try-error")) NULL else find_call(parsed, "pf")

  # ---- partial-credit messages ----
  used_pf <- !is.null(pc) || grepl("\\bpf\\s*\\(", code)
  msgs <- c(msgs, if (used_pf) "  used `pf()`." else "  did not call `pf()`.")

  # detect right-tail style
  used_1_minus     <- grepl("1\\s*-\\s*pf\\s*\\(", code)
  used_lower_false <- grepl("lower\\.tail\\s*=\\s*(FALSE|F)\\b", code)
  if (used_1_minus || used_lower_false) {
    msgs <- c(msgs, "  computed the **right tail** (either `1 - pf(...)` or `lower.tail = FAL
  } else {
    msgs <- c(msgs, "  For the **right tail**, use `1 - pf(x, df1, df2)` or `pf(x, df1, df2,
  }

  # check x, df1, df2 if we can inspect a pf(...) call
  if (!is.null(pc)) {
    args <- as.list(pc)[-1]
    # x (first)
    x_expr <- if (length(args) >= 1) args[[1]] else NULL
    x_val  <- try(eval(x_expr, envir = .envir_result), silent = TRUE)
    msgs <- c(msgs,
      if (!inherits(x_val,"try-error") && is.numeric(x_val) && length(x_val)==1 && isTRUE(al
```

```
      "  first argument x = 3."
    else
      "  first argument should be 3."
  )
  # df1 (named or 2nd)
  df1_expr <- if (!is.null(names(args)) && "df1" %in% names(args)) args[["df1"]]
              else if (length(args) >= 2) args[[2]] else NULL
  df1_val  <- try(eval(df1_expr, envir = .envir_result), silent = TRUE)
  msgs <- c(msgs,
    if (!is.null(df1_expr) && !inherits(df1_val,"try-error") && is.numeric(df1_val) && leng
      "  df1 = 5."
    else
      "  `df1` should be 5."
  )
  # df2 (named or 3rd)
  df2_expr <- if (!is.null(names(args)) && "df2" %in% names(args)) args[["df2"]]
              else if (length(args) >= 3) args[[3]] else NULL
  df2_val  <- try(eval(df2_expr, envir = .envir_result), silent = TRUE)
  msgs <- c(msgs,
    if (!is.null(df2_expr) && !inherits(df2_val,"try-error") && is.numeric(df2_val) && leng
      "  df2 = 12."
    else
      "  `df2` should be 12."
  )
}

# ---- numeric correctness from the LAST expression ----
val <- .result

# if last expression isn't numeric, try re-evaluating the last non-comment line
if (!is.numeric(val) || length(val) != 1L || !is.finite(val)) {
  lines <- strsplit(code, "\n", fixed = TRUE)[[1]]
  lines <- lines[grepl("^\\s*[^#]", lines)]  # drop blanks & full-line comments
  if (length(lines) > 0) {
    exprs <- try(parse(text = paste(lines, collapse = "\n")), silent = TRUE)
    if (!inherits(exprs, "try-error") && length(exprs) > 0) {
      val <- try(eval(exprs[[length(exprs)]], envir = .envir_result), silent = TRUE)
    }
  }
}

if (!is.numeric(val) || length(val) != 1L || !is.finite(val)) {
```

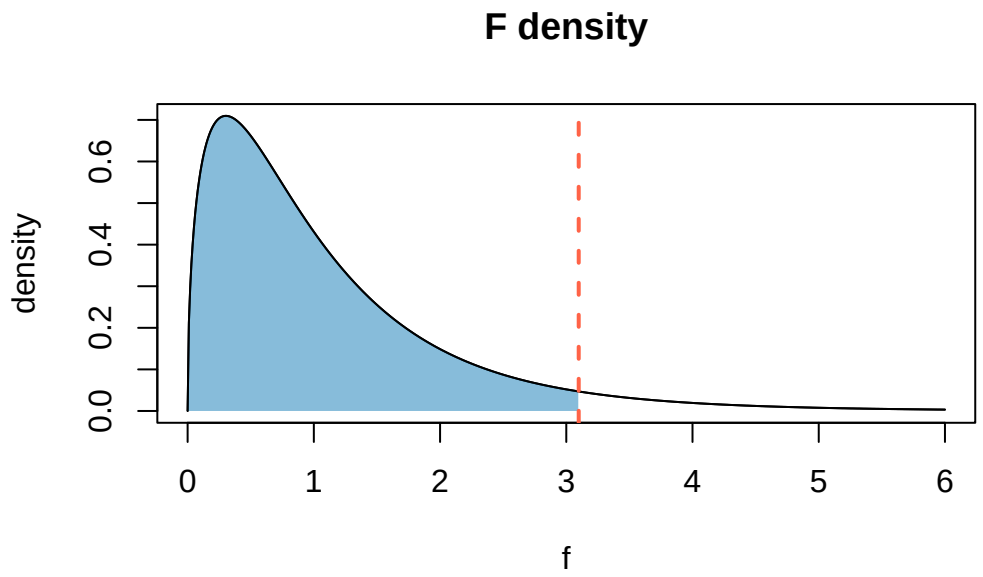
```
    fail(paste(c(msgs,
      "  Your **last line** must evaluate to a single numeric value (e.g., `1 - pf(3, df1 = 5
    ), collapse = "\n"))
  }

  if (abs(val - target) < 1e-6) {
    pass(paste(c(msgs, "  Numeric answer is correct."), collapse = "\n"))
  } else {
    fail(paste(c(msgs, "  Numeric value not correct yet."), collapse = "\n"))
  }
})
```

Q10 — F Quantile - Left Probability

Find f^* such that $P(F <= f^*) = 0.95$, where $F \sim F_{3,20}$.

Info

This is the graphical representation of the question:




```
#| exercise: f_q
#| exercise.lines: 1
```

```
#| echo: false
#your answer here
```

Use qf.

Solution.

```
#| exercise: f_q
#| solution: true
qf(0.95, df1=3, df2=20)
```

```
#| exercise: f_q
#| check: true
gradethis::grade_this({
  msgs <- c()
  code   <- paste(.user_code, collapse = "\n")
  target <- qf(0.95, df1 = 3, df2 = 20)

  # ---- parse & find qf(...) anywhere ----
  parsed <- try(parse(text = code), silent = TRUE)
  find_call <- function(exprs, fname){
    last <- NULL
    walk <- function(e){
      if (is.call(e)) {
        fn <- as.character(e[[1]])
        if (identical(fn, fname)) last <<- e
        for (k in as.list(e)[-1]) if (is.language(k)) walk(k)
      }
    }
    for (e in exprs) walk(e)
    last
  }
  qc <- if (inherits(parsed, "try-error")) NULL else find_call(parsed, "qf")

  # ---- partial-credit messages ----
  used_qf <- !is.null(qc) || grepl("\\bqf\\s*\\(", code)
  msgs <- c(msgs, if (used_qf) "  used `qf()`." else "  did not call `qf()`.")

  if (!is.null(qc)) {
    args <- as.list(qc)[-1]
    # p (first)
    p_expr <- if (length(args) >= 1) args[[1]] else NULL
```

```
  p_val  <- try(eval(p_expr, envir = .envir_result), silent = TRUE)
  msgs <- c(msgs,
    if (!inherits(p_val,"try-error") && is.numeric(p_val) && length(p_val)==1 && isTRUE(al
      "  probability p = 0.95."
    else
      "  first argument should be p = 0.95."
  )
  # df1 (named or 2nd)
  df1_expr <- if (!is.null(names(args)) && "df1" %in% names(args)) args[["df1"]]
              else if (length(args) >= 2) args[[2]] else NULL
  df1_val  <- try(eval(df1_expr, envir = .envir_result), silent = TRUE)
  msgs <- c(msgs,
    if (!is.null(df1_expr) && !inherits(df1_val,"try-error") && is.numeric(df1_val) && leng
      "  df1 = 3."
    else
      "  `df1` should be 3."
  )
  # df2 (named or 3rd)
  df2_expr <- if (!is.null(names(args)) && "df2" %in% names(args)) args[["df2"]]
              else if (length(args) >= 3) args[[3]] else NULL
  df2_val  <- try(eval(df2_expr, envir = .envir_result), silent = TRUE)
  msgs <- c(msgs,
    if (!is.null(df2_expr) && !inherits(df2_val,"try-error") && is.numeric(df2_val) && leng
      "  df2 = 20."
    else
      "  `df2` should be 20."
  )
  # optional: tail note if explicitly supplied
  if (!is.null(names(args)) && "lower.tail" %in% names(args)) {
    lt <- try(eval(args[["lower.tail"]], envir = .envir_result), silent = TRUE)
    if (identical(lt, TRUE))  msgs <- c(msgs, "  left-tail quantile (default).")
    if (identical(lt, FALSE)) msgs <- c(msgs, "  right-tail quantile requested; then use p
  }
}

# ---- numeric correctness from the LAST expression ----
val <- .result

# if last expression isn't numeric, try re-evaluating the last non-comment line
if (!is.numeric(val) || length(val) != 1L || !is.finite(val)) {
  lines <- strsplit(code, "\n", fixed = TRUE)[[1]]
  lines <- lines[grepl("^\\s*[^#]", lines)]  # drop blanks & full-line comments
```

```
    if (length(lines) > 0) {
      exprs <- try(parse(text = paste(lines, collapse = "\n")), silent = TRUE)
      if (!inherits(exprs, "try-error") && length(exprs) > 0) {
        val <- try(eval(exprs[[length(exprs)]], envir = .envir_result), silent = TRUE)
      }
    }
  }

  if (!is.numeric(val) || length(val) != 1L || !is.finite(val)) {
    fail(paste(c(msgs, "  Your **last line** must evaluate to a single numeric value (e.g., `
  }

  if (abs(val - target) < 1e-6) {
    pass(paste(c(msgs, "  Numeric answer is correct."), collapse = "\n"))
  } else {
    fail(paste(c(msgs, "  Numeric value not correct yet."), collapse = "\n"))
  }
})
```

Q11 — Overview with Skimr

The question below has a dataframe df created from the dataset msleep that contains data about mammal sleep traits. Make a skim summary of df and store it in ms_skim. Print it to see your result!

Photo by Mpho Mojapelo on Unsplash

> **Info**
>
> Info : skimr is an R package that provides summary statistics a user can quickly skim to understand their data.

```
#| setup: true
#| exercise: ms_skim
#| echo: false
ensure_pkgs <- function(pkgs){
  miss <- pkgs[!vapply(pkgs, requireNamespace, logical(1), quietly = TRUE)]
  if (length(miss)) webr::install(miss)
  invisible(lapply(pkgs, function(p) library(p, character.only = TRUE)))
}
ensure_pkgs(c("skimr","ggplot2"))
df <- ggplot2::msleep
```

```
#| exercise: ms_skim
#| exercise.lines: 2
#| echo: false
library(ggplot2)
library(skimr)
df <- msleep
ms_skim <-
ms_skim
```

Use the skim function from skimr.

Solution.

```
#| exercise: ms_skim
#| solution: true
library(ggplot2)
library(skimr)
df <- msleep
ms_skim <- skim(df)
ms_skim
```

```
#| exercise: ms_skim
#| check: true
gradethis::grade_this({
  if (!exists("ms_skim", envir=.envir_result)) fail("Assign `ms_skim <- skim(df)` and print i
  x <- get("ms_skim", envir=.envir_result)
  ok <- inherits(x, "skim_df") || ("skim_type" %in% names(x))
  if (ok) pass("  skim summary detected.")
  else fail("Something is not quite right here.")
})
```

Q12 — Histogram with ggplot

Build a ggplot histogram of sleep_total with binwidth = 1 by replacing the with the correct answer. Save the plot to p_hist. Fill the histogram with any color of your choice!

> **Info**
>
> Info : ggplot refers to the ggplot2 package in R, a powerful tool for creating statistical graphics.

```
#| setup: true
#| exercise: ms_hist
#| echo: false
ensure_pkgs <- function(pkgs){ miss <- pkgs[!vapply(pkgs, requireNamespace, logical(1), quie
  if (length(miss)) webr::install(miss)
  invisible(lapply(pkgs, function(p) library(p, character.only = TRUE))) }
ensure_pkgs(c("ggplot2"))
df <- ggplot2::msleep
```

```
#| exercise: ms_hist
#| exercise.lines: 4
#| echo: false
library(ggplot2)
p_hist <-   (df, aes(   )) +   (binwidth =   , fill="   ", color="black")
p_hist
```

Look up how to make plots with ggplot.

Solution.

```
#| exercise: ms_hist
#| solution: true
library(ggplot2)
p_hist <- ggplot(df, aes(sleep_total)) +
  geom_histogram(binwidth = 1)
p_hist
```

```
#| exercise: ms_hist
#| check: true
gradethis::grade_this({
  if (!exists("p_hist", envir=.envir_result)) fail("Create `p_hist` as a ggplot object.")
  p <- get("p_hist", envir=.envir_result)
  ok <- inherits(p, "ggplot")
  if (!ok) fail("`p_hist` must be a ggplot object.")
  code <- paste(.user_code, collapse="\n")
  if (!grepl("geom_histogram\\s*\\(", code)) fail("Add `geom_histogram(...)`.")
  if (!grepl("binwidth\\s*=\\s*1", code)) fail("Set `binwidth = 1`.")
  pass("  Histogram with binwidth 1 detected.")
})
```

Q13 — Scatterplot with ggplot

Make a scatterplot of bodywt (x) vs sleep_total (y), log_{10} scale on x, with a smooth trend (Read info to understand what these terms mean), by replacing the with the correct answer. Save the plot to p_scatter. Fill the scatterplot with any color of your choice!

> **Info**
>
> Info : ggplot refers to the ggplot2 package in R, a powerful tool for creating statistical graphics.

> **Info**
>
> Info : 1) A log_{10} scale on the x-axis transforms the x values before plotting.This spreads out small values and compresses very large ones. As an exercise, also try removing that argument and see how much harder it is to visualize the data without it.
>
> 2) A smooth trend line is added along the points by using geom_smooth(). This leads to better visualization of trends in the data.

```
#| setup: true
#| exercise: ms_scatter
#| echo: false
ensure_pkgs <- function(pkgs){ miss <- pkgs[!vapply(pkgs, requireNamespace, logical(1), quie
  if (length(miss)) webr::install(miss)
  invisible(lapply(pkgs, function(p) library(p, character.only = TRUE))) }
ensure_pkgs(c("ggplot2"))
df <- ggplot2::msleep
```

```
#| exercise: ms_scatter
#| exercise.lines: 6
#| echo: false
library(ggplot2)
p_scatter <-   (df, aes(x=   , y =   ,  color="   ")) +
     () +
  scale_x_log10() +
  geom_smooth(se = FALSE)
p_scatter
```

Look up how to make scatterplots with ggplot.

Solution.

```
#| exercise: ms_scatter
#| solution: true
library(ggplot2)
p_scatter <- ggplot(df, aes(bodywt, sleep_total)) +
  geom_point() +
  scale_x_log10() +
  geom_smooth(se = FALSE)
p_scatter
```

```
#| exercise: ms_scatter
#| check: true
gradethis::grade_this({
  if (!exists("p_scatter", envir=.envir_result)) fail("Create `p_scatter`.")
  p <- get("p_scatter", envir=.envir_result)
  if (!inherits(p, "ggplot")) fail("`p_scatter` must be a ggplot object.")
  code <- paste(.user_code, collapse="\n")
  if (!grepl("geom_point\\s*\\(", code)) fail("Add `geom_point()`.")
  if (!grepl("scale_x_log10\\s*\\(", code)) fail("Use `scale_x_log10()` for a log x-axis.")
  if (!grepl("geom_smooth\\s*\\(", code)) fail("Add `geom_smooth()` for a trend.")
  pass("  Scatter with log-x and smoother detected.")
})
```

Q14 — Mean and Standard Deviation

Compute the population mean and standard deviation of sleep_total. Save them in mu and sigma. Note that x is the vector containing sleep data pulled from the dataset.

> **Info**
>
> Info : This and the following examples illustrate the power of the Central Limit Theorem (CLT). CLT states that for a sufficiently large sample size, the sampling distribution of the sample mean will be approximately normal, regardless of the shape of the original population distribution.

```
# Q11 SETUP
#| setup: true
#| exercise: clt_baseline
#| echo: false
ensure_pkgs <- function(pkgs){
  miss <- pkgs[!vapply(pkgs, requireNamespace, logical(1), quietly = TRUE)]
  if (length(miss)) webr::install(miss)
```

```
  invisible(lapply(pkgs, function(p) library(p, character.only = TRUE)))
}
ensure_pkgs("ggplot2")
x <- stats::na.omit(ggplot2::msleep$sleep_total)
```

```
#| exercise: clt_baseline
#| exercise.lines: 4
#| echo: false
mu <-

mu

sigma <-

sigma
```

R has direct commands to compute mean and standard deviation. Look into those.

Solution.

```
#| exercise: clt_baseline
#| solution: true
mu <- mean(x)
sigma <- sd(x)
```

```
#| exercise: clt_baseline
#| check: true
gradethis::grade_this({
  if (!exists("mu", envir=.envir_result) || !exists("sigma", envir=.envir_result))
    fail("Create both mu and sigma.")
  a <- get("mu", envir=.envir_result); b <- get("sigma", envir=.envir_result)
  if (!is.numeric(a) || length(a)!=1) fail("`mu` must be a single number.")
  if (!is.numeric(b) || length(b)!=1) fail("`sigma` must be a single number.")
  pass("  Great - baseline mean and SD computed.")
})
```

Q15 — Sample Mean and Sample Standard Deviation

B = 200 samples of size n = 10 with replacement from x are stored in means_10. Compute the following: m_10 = Mean of means_n10, s_10 = Standard Deviation of means_n10

```
# Q12 SETUP
#| setup: true
#| exercise: clt_n10
#| echo: false
ensure_pkgs <- function(pkgs){
  miss <- pkgs[!vapply(pkgs, requireNamespace, logical(1), quietly = TRUE)]
  if (length(miss)) webr::install(miss)
  invisible(lapply(pkgs, function(p) library(p, character.only = TRUE)))
}
ensure_pkgs("ggplot2")
x <- stats::na.omit(ggplot2::msleep$sleep_total)
set.seed(123)
n <- 10
B <- 200
```

```
#| exercise: clt_n10
#| exercise.lines: 10
#| echo: false
means_n10 <- replicate(B, mean(sample(x, n, replace = TRUE)))
m_10  <-

m_10

s_10    <-

s_10
```

R has direct commands to compute mean and standard deviation. Look into those.

Solution.

```
#| exercise: clt_n10
#| solution: true
means_n10 <- replicate(B, mean(sample(x, n, replace = TRUE)))
m_10  <- mean(means_n10)
m_10
s_10    <- sd(means_n10)
s_10
```

```
#| exercise: clt_n10
#| check: true
```

```
gradethis::grade_this({

  # --- fetch student objects safely ---
  if (!exists("means_n10", envir = .envir_result))
    fail("Create `means_n10` as the vector of B sample means.")

  means_n10 <- get("means_n10", envir = .envir_result)
  if (!is.numeric(means_n10) || length(means_n10) < 50)
    fail("`means_n10` should be a numeric vector (length around B = 200).")

  if (!exists("m_10", envir = .envir_result) ||
      !exists("s_10",   envir = .envir_result))
    fail("Also compute `m_10 <- mean(means_n10)` and `s_10 <- sd(means_n10)`.")

  m_10 <- get("m_10", envir = .envir_result)
  s_10   <- get("s_10",   envir = .envir_result)

  # --- strict shape checks first ---
  if (!(is.numeric(m_10) && length(m_10) == 1 && is.finite(m_10)))
    fail("`m_10` must be a single numeric value.")
  if (!(is.numeric(s_10) && length(s_10) == 1 && is.finite(s_10)))
    fail("`s_10` must be a single numeric value.")

  # --- compute population refs locally (avoid relying on external `mu`, `sigma`) ---
  x_pop  <- stats::na.omit(ggplot2::msleep$sleep_total)
  mu     <- mean(x_pop)
  sigma  <- stats::sd(x_pop)
  theo_sd <- sigma / sqrt(10)

  msgs <- c()
  msgs <- c(msgs,
            if (abs(as.numeric(m_10) - mu) < 0.5)
              "  Note that the mean of sample means is close to  ."
            else
              "  The mean of sample means should be near   (try larger B).")
  msgs <- c(msgs,
            if (abs(as.numeric(s_10) - theo_sd) < 0.3)
              "  Note that the SD of sample means   /√n."
            else
              "  The SD of sample means should be about  /√n.")

  pass(paste(msgs, collapse = "\n"))
```

```
})
```

Q16 — Visualizing the Distribution

Make a histogram of means_n10 (binwidth = 0.5), and add a vertical dashed red line at the population mean mu. Save the plot in plot_clt and print it. Fill the plot with any color of your choice!

> **Info**
>
> Notice the distribution of the sample mean here and see if it resembles a normal distribution. Which theorem is in play here?

```
#| setup: true
#| exercise: clt_plot_n10
#| echo: false
ensure_pkgs <- function(pkgs){
  miss <- pkgs[!vapply(pkgs, requireNamespace, logical(1), quietly = TRUE)]
  if (length(miss)) webr::install(miss)
  invisible(lapply(pkgs, function(p) library(p, character.only = TRUE)))
}
ensure_pkgs("ggplot2")

# population vector and mu
x  <- stats::na.omit(ggplot2::msleep$sleep_total)
mu <- mean(x)

# simulation defaults
set.seed(123)
n <- 10
B <- 200

# safety net: ensure means_n10 exists for plotting/hints/checkers
if (!exists("means_n10", inherits = FALSE)) {
  means_n10 <- replicate(B, mean(sample(x, n, replace = TRUE)))
}
```

```
#| exercise: clt_plot_n10
#| exercise.lines: 8
#| echo: false
library(ggplot2)
```

```
means_n10 <- replicate(B, mean(sample(x, n, replace = TRUE)))

p_clt_n10 <-   (data.frame(m = means_n10), aes(m)) +
    (binwidth =   , fill = "   " , color = "white") +
     (xintercept =   , linetype = 2, color = "red") +
  labs(x = "sample mean (n=10)", y = "count",
               title = "Sampling distribution of the mean (n = 10)") +
  =theme_minimal()
p_clt_n10
```

Refer to previous exercises for learning how to plot a histogram. To draw a vertical line at a certain intercept, use the ggplot command- geom_vline .

Solution.

```
#| exercise: clt_plot_n10
#| solution: true
library(ggplot2)
p_clt_n10 <- ggplot(data.frame(m = means_n10), aes(m)) +
  geom_histogram(binwidth = 0.5, fill = "grey80", color = "white") +
  geom_vline(xintercept = mu, linetype = 2, color = "red") +
  labs(x = "sample mean (n=10)", y = "count",
               title = "Sampling distribution of the mean (n = 10)") +
  theme_minimal()
p_clt_n10
```

```
#| exercise: clt_plot_n10
#| check: true
gradethis::grade_this({
  if (!exists("p_clt_n10", envir=.envir_result)) fail("Create the ggplot object `p_clt_n10`."
  p <- get("p_clt_n10", envir=.envir_result)
  if (!inherits(p, "ggplot")) fail("`p_clt_n10` must be a ggplot object.")

  code <- paste(.user_code, collapse="\n")
  msgs <- c()
  msgs <- c(msgs, if (grepl("geom_histogram\\s*\\(", code)) "  Used a histogram." else "  Add
  msgs <- c(msgs, if (grepl("binwidth\\s*=\\s*0\\.?5", code)) "  binwidth = 0.5 set." else "
  msgs <- c(msgs, if (grepl("geom_vline\\s*\\(", code) && grepl("xintercept\\s*=\\s*mu", code
  pass(paste(msgs, collapse = "\n"))
})
```