Tutorial 02: Descriptive Statistics

```
#| 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 — Five-number summary

The five-number summary (Min, Q1, Median, Q3, Max) + IQR quickly reveals center and spread. Return a five number summary for the dataset airquality's Ozone column. Further, return the IQR.

Info

The five-number summary (Min, Q1, Median, Q3, Max) quickly shows center and spread. The IQR = Q3 − Q1 is robust to outliers and underlies boxplot fences (Q1 − 1.5 · IQR, Q3 + 1.5 · IQR). You'll use base R functions that compute these directly for a numeric

Preview

Run the code below to see a preview of the dataset.

```
#| echo: true
air_data <- datasets::airquality
head(air_data, 10)
```

```
#| exercise: ex_ozone_fivenum
#| exercise.lines: 4
#| echo: false
```

Call the base function that prints the five key stats, then on the next line return the robust spread of that same column; remember missing values.

Solution.

```
#| exercise: ex_ozone_fivenum
#| solution: true
summary(airquality$Ozone)
IQR(airquality$Ozone, na.rm = TRUE)
```

```
#| exercise: ex_ozone_fivenum
#| check: true
gradethis::grade_this({
  code <- paste(.user_code, collapse = "\n")
  if (!grepl("summary\\s*\\(\\s*airquality\\$Ozone\\s*\\)", code))
    fail("First line should be `summary(airquality$Ozone)`.")
  out <- .result
  tgt <- IQR(airquality$Ozone, na.rm = TRUE)
  if (!is.numeric(out) || length(out) != 1 || !is.finite(out))
    fail("Second line should evaluate to a single numeric IQR (use `na.rm=TRUE`).")
  if (!isTRUE(all.equal(out, tgt, tol = 1e-8)))
    fail("IQR doesn't match `IQR(airquality$Ozone, na.rm=TRUE)`.")
  pass("  Summary then IQR returned correctly.")
})
```

Q2 — Mean, variance, and sd

Compute Mean, Variance and Standard Deviation of airquality dataset's Temp column.

Info

Mean, variance, and standard deviation summarize center and spread. Variance and SD are in squared and original units respectively. Real-world data often has NAs; make sure your summary ignores them appropriately.

Photo by Tim Witzdam on Unsplash

Preview

Run the code below to see a preview of the dataset.

```
#| echo: true
air_data <- datasets::airquality
head(air_data, 10)
```

```
#| exercise: ex_temp_stats
#| exercise.lines: 4
#| echo: false
# Return a named vector: c(mean=..., var=..., sd=...)
# using airquality$Temp and na.rm=TRUE.
```

Construct a single named vector with three entries; each entry calls the corresponding base summary function with missing-value handling.

Solution.

```
#| exercise: ex_temp_stats
#| solution: true
c(
  mean = mean(airquality$Temp, na.rm = TRUE),
  var  = var(airquality$Temp,  na.rm = TRUE),
  sd   = sd(airquality$Temp,   na.rm = TRUE)
)
```

```
#| exercise: ex_temp_stats
#| check: true
gradethis::grade_this({
  v <- .result
  if (!is.numeric(v) || !setequal(names(v), c("mean","var","sd")))
    fail("Return a **named** numeric vector with names exactly: mean, var, sd (use `na.rm=TRU
  pass("  mean/var/sd computed.")
})
```

Q3 — Grouped summary by Month

Compute the mean Ozone by Month using base R. Return a named numeric vector (names are months).

Preview

Run the code below to see a preview of the dataset.

```
#| echo: true
air_data <- datasets::airquality
head(air_data, 10)
```

```
#| exercise: ex_group_month
#| exercise.lines: 3
#| echo: false
# Return a named numeric vector: mean Ozone per Month
tapply(airquality$ , airquality$ , mean, na.rm =  )
```

Learn about the function tapply().

Solution.

```
#| exercise: ex_group_month
#| solution: true
tapply(airquality$Ozone, airquality$Month, mean, na.rm = TRUE)
```

```
#| exercise: ex_group_month
#| check: true
gradethis::grade_this({
  # expected via tapply
  target <- tapply(airquality$Ozone, airquality$Month, mean, na.rm = TRUE)

  res <- .result

  # Allow either a named numeric vector OR a 2-col data.frame (Month, Ozone)
  if (is.numeric(res) && !is.null(names(res))) {
    # named numeric vector path
    same_names <- setequal(names(res), names(target))
    close_vals <- all.equal(res[order(names(res))], target[order(names(target))], tol = 1e-8
    if (!same_names) fail("Return a **named** numeric vector with month names (e.g., 5, 6, 7
    if (!isTRUE(close_vals)) fail("Values don't match `tapply(airquality$Ozone, airquality$Mo
    pass("  Mean Ozone by month computed (named numeric vector).")
  } else if (is.data.frame(res) && all(c("Month","Ozone") %in% names(res))) {
    # accept aggregate-like result too
    comp <- setNames(res$Ozone, res$Month)
    close_vals <- all.equal(comp[order(names(comp))], target[order(names(target))], tol = 1e
    if (!isTRUE(close_vals)) fail("The numbers don't match the month-wise means with `na.rm=
    pass("  Mean Ozone by month (data frame) accepted.")
  } else {
    fail("Return either a **named numeric vector** (preferred) or a data frame with columns
  }
})
```

Q4 — ggplot Boxplot ordered by median (ChickWeight)

Now, we are moving on to the ChickWeight dataset which has data from an experiment on the effect of diet on early growth of chicks. Return a ggplot object that boxplots weight by Diet, with Diet reordered by the median weight. Fill in the blanks.

Photo by Karim MANJRA on Unsplash

Info

You can reorder categories by a statistic (like the median) right inside aes() to make comparisons meaningful. In ggplot2, this is commonly done with reorder() (or forcats helpers). The layer for boxplots is geom_boxplot().

```
#| exercise: ex_box_reorder
#| exercise.lines: 8
#| echo: false
library(ggplot2)
# Return a ggplot object that boxplots mpg by cyl, with cyl reordered by the median mpg.
  (   , aes(x = reorder(factor(   ),    , FUN = median), y =   )
) +   () #Boxplot command in ggplot
```

Map Diet to x and weight to y; reorder x by median(weight) inside aes(...). Use geom_boxplot() for the layer.

Solution.

```
#| exercise: ex_box_reorder
#| solution: true
library(ggplot2)
ggplot(
ChickWeight,
aes(x = reorder(factor(Diet), weight, FUN = median), y = weight)
) +
geom_boxplot()
```

```
#| exercise: ex_box_reorder
#| check: true

gradethis::grade_this({
  if (!requireNamespace("ggplot2", quietly = TRUE)) fail("Load ggplot2.")
  p <- .result

  is_box <- inherits(p, "ggplot") &&
    any(vapply(p$layers, function(L) inherits(L$geom, "GeomBoxplot"), logical(1)))
  if (!is_box) fail("Return a ggplot with geom_boxplot().")

  mx <- deparse(p$mapping$x)
  my <- deparse(p$mapping$y)
  # Accept reorder(...) or forcats::fct_reorder(...); require factor(Diet) and weight
  pat_x <- "^~\\s*(reorder|forcats::fct_reorder)\\s*\\(\\s*factor\\s*\\(\\s*Diet\\s*\\)\\s*,
  x_ok  <- grepl(pat_x, mx, perl = TRUE)
  y_ok  <- grepl("^~\\s*weight\\s*$", my, perl = TRUE)

  if (!(x_ok && y_ok))
    fail("Map x = reorder(factor(Diet), weight, FUN = median) (or forcats::fct_reorder) and

  pass("  Boxplot created and Diet reordered by median weight.")
})
```

Q5 — Histogram with Tukey fences

Show a histogram of ChickWeight$weight and add vertical lines at Q1, Q3, and the Tukey fences Q1−1.5 · IQR, Q3+1.5 · IQR

Info

Tukey fences are statistical boundaries, used to identify potential outliers in a dataset. Data points falling outside these fences are considered outliers. $Q1 - 1.5 \cdot IQR$ and $Q3 + 1.5 \cdot IQR$ are Tukey fences!

Preview

Here is a preview of the dataset:

```
#| echo: true
head(ChickWeight[, c("weight","Time","Diet")], 8)
```

```
#| setup: true
#| exercise: ex_weight_hist_fences
#| 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("ggplot2"))
```

```
#| exercise: ex_weight_hist_fences
#| exercise.lines: 14
#| echo: false
# Return a ggplot object that draws the histogram.
library(ggplot2)
qs <- quantile(ChickWeight$weight, c( ,  ) )
lo <- qs[1] - 1.5*diff(qs)
hi <-
ggplot(   , aes(weight)) +
geom_histogram(bins = 12) +
geom_vline(xintercept = c(qs[1],   , lo, hi))
```

geom_vline is used to add vertical lines in the plot at certain positions.

Solution.

```
#| exercise: ex_weight_hist_fences
#| solution: true
library(ggplot2)
qs <- quantile(ChickWeight$weight, c(.25,.75))
lo <- qs[1] - 1.5*diff(qs)
hi <- qs[2] + 1.5*diff(qs)
ggplot(ChickWeight, aes(weight)) +
geom_histogram(bins = 12) +
geom_vline(xintercept = c(qs[1], qs[2], lo, hi))
```

```
#| exercise: ex_weight_hist_fences
#| check: true
gradethis::grade_this({
e <- .envir_result
qs <- get0("qs", e); lo <- get0("lo", e); hi <- get0("hi", e)
if (is.null(qs) || is.null(lo) || is.null(hi) || length(qs) < 2)
fail("Define `qs`, `lo`, and `hi` as shown.")
r <- as.numeric(quantile(ChickWeight$weight, c(.25,.75)))
ok <- function(a,b) abs(as.numeric(a)-b) < 1e-8
if (ok(qs[1], r[1]) && ok(qs[2], r[2]) &&
ok(lo, r[1] - 1.5*diff(r)) && ok(hi, r[2] + 1.5*diff(r))) pass("  Correct Tukey fences.")
else fail("  Recheck quartiles and fence formulas.")
})
```

Q6 — Histogram of Weight (ggplot2)

Make a histogram with ~12 bins. Store in p_hist.

Info

Use the built-in ChickWeight dataset. Plot the distribution of weight as a histogram with about 12 bins and assign the plot to‘p_hist. Return a ggplot object (no printing required).

Preview

Here is a preview of the dataset:

```
#| echo: true
head(ChickWeight[, c("weight","Time","Diet")], 8)
```

```
#| exercise: ex_hist
#| exercise.lines: 5
#| echo: false
library(ggplot2)
p_hist <-  (ChickWeight, aes( )) +
 (bins =  )
p_hist
```

Create a ggplot using ChickWeight with weight mapped on x. Add a histogram layer with ~12 bins. Assign the plot object to p_hist.

Solution.

```
#| exercise: ex_hist
#| solution: true
library(ggplot2)
p_hist <- ggplot(ChickWeight, aes(weight)) +
geom_histogram(bins = 12)
p_hist
```

```
#| exercise: ex_hist
#| check: true
gradethis::grade_this({
  if (!requireNamespace("ggplot2", quietly = TRUE)) fail("Load **ggplot2**.")
  e <- get0(".envir_result", ifnotfound = parent.frame())
  if (!exists("p_hist", envir = e)) fail("Create `p_hist`.")
  p <- get("p_hist", envir = e)
  if (!inherits(p, "ggplot")) fail("`p_hist` must be a ggplot object.")

  # Must include a histogram layer (geom_histogram() uses GeomBar under the hood)
  has_hist <- any(vapply(p$layers, function(L) inherits(L$geom, "GeomBar"), logical(1)))
  if (!has_hist) fail("Use `geom_histogram()`.")

  # Accept x mapping as weight (handles weight / x=weight / ~weight etc.)
  if (!requireNamespace("rlang", quietly = TRUE)) fail("Missing `rlang`.")
  x_lab <- rlang::as_label(rlang::get_expr(p$mapping$x))
  x_lab <- sub("^~", "", x_lab)  # strip leading ~ if present
  if (x_lab != "weight") fail("Map `weight` to x.")

  # Optional: bins sanity (only if explicitly set)
  bins_vals <- vapply(
    p$layers,
    function(L) {
      if (!is.null(L$geom_params$bins)) L$geom_params$bins else NA_real_
    },
    numeric(1)
  )
  bins_vals <- bins_vals[!is.na(bins_vals)]
  if (length(bins_vals) > 0 && !any(bins_vals >= 10 & bins_vals <= 14)) {
    fail("Use about 12 bins (e.g., 10-14).")
  }

  pass("  Histogram done.")
})
```

Q7 — Boxplot of weight by Diet (ggplot2)

Make a boxplot of weight grouped by Diet (treat Diet as categorical). Store the plot in p_box

Info

Here is a preview of the dataset:

```
#| echo: true
head(ChickWeight[, c("weight","Diet","Time")], 8)
```

```
#| exercise: ex_box
#| exercise.lines: 5
#| echo: false
library(ggplot2)
p_box <- ggplot( , aes(x = factor( ), y =  )) +
   ()
p_box
```

Build a ggplot from ChickWeight. Map factor(Diet) to x and weight to y. Add a boxplot layer. Assign the result to p_box.

Solution.

```
#| exercise: ex_box
#| solution: true
library(ggplot2)
p_box <- ggplot(ChickWeight, aes(x = factor(Diet), y = weight)) +
geom_boxplot()
p_box
```

```
#| exercise: ex_box
#| check: true
gradethis::grade_this({
  if (!requireNamespace("ggplot2", quietly = TRUE)) fail("Load ggplot2.")

  e <- get0(".envir_result", ifnotfound = parent.frame())
  if (!exists("p_box", envir = e)) fail("Create `p_box`.")
  p <- get("p_box", envir = e)
  if (!inherits(p, "ggplot")) fail("`p_box` must be a ggplot object.")

  has_geom <- any(vapply(p$layers, function(L) inherits(L$geom, "GeomBoxplot"), logical(1)))
  if (!has_geom) fail("Use `geom_boxplot()`.")

  mx <- deparse(p$mapping$x)
  my <- deparse(p$mapping$y)

  # Allow factor(Diet) or ~factor(Diet)
  if (!grepl("^~?factor\\(Diet\\)$", mx)) fail("Map `factor(Diet)` to x.")
  # Allow weight or ~weight
  if (!grepl("^~?weight$", my)) fail("Map `weight` to y.")

  pass("  Boxplot good.")
})
```

Q8 — Scatter with smoother: Time vs weight (ggplot2)

Build a scatterplot from ChickWeight mapping Time → x and weight → y. Store in p_scatter.

Preview

```
#| echo: true
head(ChickWeight[, c("Time","weight","Diet")], 8)
```

```
#| exercise: ex_scatter
#| exercise.lines: 6
#| echo: false
library(ggplot2)
p_scatter <- ggplot( , aes( ,  )) + #order matters
geom_point() + #used for scatterplots
geom_smooth(se = FALSE)
p_scatter
```

Build a ggplot from ChickWeight mapping Time → x and weight → y. Add points, then a smoother (hide the SE ribbon). Assign the final plot to p_scatter.

Solution.

```
#| exercise: ex_scatter
#| solution: true
library(ggplot2)
p_scatter <- ggplot(ChickWeight, aes(Time, weight)) +
geom_point() +
geom_smooth(se = FALSE)
p_scatter
```

```
#| exercise: ex_scatter
#| check: true
```

```
gradethis::grade_this({
if (!requireNamespace("ggplot2", quietly = TRUE)) fail("Load ggplot2.")
e <- get0(".envir_result", ifnotfound = parent.frame())
if (!exists("p_scatter", envir = e)) fail("Create p_scatter.")
p <- get("p_scatter", envir = e)
if (!inherits(p, "ggplot")) fail("Return a ggplot.")
pts <- any(vapply(p$layers, function(L) inherits(L$geom, "GeomPoint"), logical(1)))
sm <- any(vapply(p$layers, function(L) inherits(L$geom, "GeomSmooth"), logical(1)))
if (!(pts && sm)) fail("Include both points and a smoother.")
mx <- deparse(p$mapping$x); my <- deparse(p$mapping$y)
if (!grepl("^~?Time$", mx) || !grepl("^~?weight$", my)) fail("Map Time to x and weight to y."
pass("  Scatter + smooth OK.")
})
```