```
# ======================================================================
#  SMARTstats Workshop Series | Multiple Imputation
#  02_workshop.R  --  THIRTY-MINUTE RSTUDIO DEMONSTRATION
# ----------------------------------------------------------------------
#  This script is the second half of the session. The first 50 minutes
#  are conceptual and code-free; everything here puts those concepts on
#  real data. It is written to be walked through live, top to bottom.
#
#  NEW TO R OR RSTUDIO? READ THIS BOX FIRST.
#    Run one line at a time with Ctrl+Enter (Cmd+Enter on a Mac). The
#    line runs in the Console below and you see the result immediately.
#
#    Symbols you will meet constantly:
#      <-      assignment. `x <- 5` means "put 5 into x".
#      #       a comment. R ignores everything after it on that line.
#      $       pulls one column out of a table. `obs$attend`.
#      c(...)  "combine" -- builds a vector (a list of values).
#      [ , ]   picks rows and columns. `d[1:5, ]` is the first 5 rows.
#      ~       reads as "explained by", used to write models.
#      NA      a missing value. Not zero, not blank -- absent.
#
#    Two comment styles are used below:
#      # A comment on its own line explains WHY -- the statistics.
#      x <- 1   # A comment at the end of a line explains WHAT that
#               # line of code does mechanically.
#
#  THE QUESTION
#    Does a school's victimisation climate predict spring internalizing
#    symptoms, over and above a student's own victimisation?
#
#  WHY SIMULATED DATA
#    Because we generated it, the true answer is known. Every method
#    below gets scored against a number we can actually check -- which
#    is impossible with real data.
#
#  SOFTWARE  R 4.3+ (tested on 4.3.3 and 4.4.2)
#  PACKAGES  mice, lme4, mitml   (plus `pan`, which mice loads itself)
#              install.packages(c("mice", "lme4", "mitml", "pan"))
#            `miceadds` is NOT required -- the multilevel imputers ship
#            inside mice itself.
#
#  DATA      data/bullying_climate.csv           (run 01_generate_data.R)
#            data/bullying_climate_complete.csv
#
#  RUNTIME   About 10-15 minutes the first time (the number of
#            imputations is calculated, not fixed, so this varies).
#            Results are saved to cache/ and reload instantly after.
#
#  LIVE WALKTHROUGH -- 30 MINUTES
#     Sec 1   ~5 min   Look at the missingness, and at who gets deleted
#     Sec 2   ~7 min   Build an imputation model; how many imputations
#     Sec 3   ~5 min   THE DECISION POINT: does clustering change it?
#     Sec 4   ~7 min   The comparison table, and how to read it
#     Sec 5   ~4 min   MNAR, and a sensitivity analysis you can report
#     Sec 6   ~2 min   One honest caveat
#     Sec 7            Take-home
# ======================================================================


## ----------------------------------------------------------------------
## 0.  SETUP
## ----------------------------------------------------------------------

# library() loads an add-on package. suppressMessages() just hides the
```

```
# chatty start-up banners so the console stays readable.
# Three packages, and that is the whole list.
#   mice   does the imputing
#   lme4   fits the multilevel model
#   mitml  combines the results across imputed datasets
# (mice loads the `pan` engine by itself when it needs it -- you do not
#  have to attach it. Install it, though: install.packages("pan").)
suppressMessages({
  library(mice)      # the imputation engine
  library(lme4)      # fits multilevel (mixed-effects) models
  library(mitml)     # pools results across the imputed datasets
})

set.seed(4514)                            # makes random draws repeatable
dir.create("cache",  showWarnings = FALSE) # folder for saved results
dir.create("output", showWarnings = FALSE) # folder for figures and tables

# ---- SETTINGS YOU MIGHT CHANGE -----------------------------------
USE_CACHE <- TRUE   # TRUE = reuse saved results. Set FALSE to re-run
                    # everything from scratch (see VERBOSE below).
VERBOSE   <- TRUE   # TRUE = show each imputation as it runs, and a
                    # progress counter while models are being fitted.
CV_TARGET <- 0.05   # how much you will tolerate the standard error
                    # moving if you re-imputed. Section 2 turns this
                    # into the number of imputations.
MAXIT     <- 10     # iterations per imputation

# NOTE: there is no M_IMP setting. The number of imputations is CALCULATED
# in Section 2 from von Hippel's rule. Guessing it is the thing we are
# trying to stop people doing.

# TO LET THE ROOM WATCH IT RUN: set USE_CACHE <- FALSE and keep
# VERBOSE <- TRUE. mice then prints a live grid like
#
#      iter imp variable
#        1   1  frpl_ind  int_fall  vic_fall  sch_climate  int_spring
#        1   2  frpl_ind  int_fall  ...
#
# ...which is the algorithm cycling: for each ITERATION, for each
# IMPUTATION, it refills each incomplete VARIABLE in turn. That is
# literally the chained-equations loop, visible as it happens.
# Section 3 takes a few minutes at the calculated m. For a live run you
# can overwrite M_IMP with a small number right after Section 2 computes
# it -- but say out loud that you would not report SEs at that m.

## ---- the numeric convergence check -------------------------------
# Trace plots are an eyeball test. This is the number behind them.
#
# R-hat -- the "potential scale reduction factor" -- compares how much
# the chains differ FROM EACH OTHER against how much each one wobbles on
# its own. If they have mixed properly those two are the same thing and
# the ratio is about 1. Much above 1 means the chains are still telling
# different stories, and you need more iterations.
#
# Rule of thumb: below 1.05 is fine. Above it, raise MAXIT.
#
# Also reported: autocorrelation, i.e. how much each iteration still
# resembles the one before it. Near zero is good; high values mean the
# chain is crawling rather than exploring.
check_convergence <- function(imp, label) {

  # convergence() gives one row per variable per iteration.
  cv <- convergence(imp, diagnostic = "all")

  # Keep only the LAST iteration -- that is the state that actually
```

```
  # becomes your imputed dataset. Variables with no missing data return
  # NA, so drop those with !is.na().
  final <- cv[cv$.it == max(cv$.it) & !is.na(cv$psrf), c("vrb", "psrf", "ac")]
  final <- final[order(-final$psrf), ]        # worst offender first
  names(final) <- c("variable", "R_hat", "autocorr")

  cat("\nConvergence check --", label, "\n")
  print(final, row.names = FALSE, digits = 3)

  worst <- max(final$R_hat)                   # the number that decides it
  if (worst < 1.05) {
    say(sprintf("Largest R-hat = %.3f. Under 1.05, so the chains have mixed.",
                worst))
  } else {
    say(sprintf("Largest R-hat = %.3f, on '%s'.", worst, final$variable[1]),
        "That is ABOVE 1.05 -- those chains have not settled.",
        sprintf("Raise MAXIT (currently %d) and run again.", MAXIT))
  }
  invisible(final)
}


## ---- display the predictor matrix --------------------------------
# The predictor matrix is the grid of modelling decisions: for each
# variable with gaps, which other variables help fill it, and how.
#
# mice stores it with one ROW per variable being imputed. That is 13
# columns wide here, so it wraps in the console and the shape is lost.
# We flip it for display only -- edit pred_ml itself, never this.
show_predictors <- function(pred, meth) {

  rows <- names(meth)[meth != ""]        # only variables that have gaps
  m    <- t(pred[rows, , drop = FALSE])  # t() transposes: rows <-> columns

  cat("\nPREDICTOR MATRIX  (flipped to fit the screen)\n")
  cat("  columns = the variable being imputed\n")
  cat("  rows    = what gets used to predict it\n\n")
  print(m)
  say("",
      "  -2   the cluster identifier -- which school a student is in",
      "   0   not used",
      "   1   ordinary predictor",
      "   3   this predictor AND its school average",
      "",
      "Read one column at a time. Each column is the complete recipe for",
      "filling in that one variable.")
  invisible(m)
}


## ---- read the data -----------------------------------------------
# read.csv() reads a comma-separated file into a "data frame" -- R's
# name for a table with named columns.
obs   <- read.csv("data/bullying_climate.csv",          stringsAsFactors = FALSE)
truth <- read.csv("data/bullying_climate_complete.csv", stringsAsFactors = FALSE)

# obs   = what a researcher would actually have (has gaps, marked NA)
# truth = the complete data before we deleted anything (the answer key)


## ---- centring constants ------------------------------------------
# Subtracting the mean ("centring") makes the intercept interpretable
# and puts every method below on the same scale so the rows of the
# final table are comparable. We fix these ONCE from the complete data.
# In a real analysis you would centre on your observed sample means.
```

```
C_CLIM <- mean(truth$sch_climate)   # mean() averages a column
C_INTF <- mean(truth$int_fall)
C_FRPL <- mean(truth$sch_frpl)

# A function is a reusable recipe. This one takes a table `x`, adds
# three centred columns to it, and hands the table back.
prep <- function(x) {
  x$sch_climate_c <- x$sch_climate - C_CLIM   # "_c" = centred
  x$int_fall_c    <- x$int_fall    - C_INTF
  x$sch_frpl_c    <- x$sch_frpl    - C_FRPL
  x                                           # the last line is returned
}


## ---- the model we care about -------------------------------------
# Read the formula as: spring symptoms are EXPLAINED BY (~) the terms
# on the right. The last term, (1 | school_id), is what makes this
# multilevel: "let each school have its own intercept".
#
#   vic_fall       = the student's OWN victimisation       (within effect)
#   sch_climate_c  = administrator-reported school climate (CONTEXTUAL)
#
# IDENTIFICATION. Lagged, baseline-adjusted association -- not a causal
# effect. Identification is not the subject of this session.
FORM <- int_spring ~ vic_fall + sch_climate_c + int_fall_c + female +
        frpl_ind + factor(grade) + factor(race5) + sch_frpl_c +
        (1 | school_id)
# factor() tells R a number is a LABEL, not a quantity -- grade 8 is not
# "two more than" grade 6 in any meaningful sense here.

# Pull the numbers we care about out of a fitted model.
grab <- function(m) {
  s  <- summary(m)$coefficients        # the coefficient table
  vc <- as.data.frame(VarCorr(m))      # the variance components
  # c(...) builds a named vector; unname() strips labels R adds for us.
  c(est    = unname(s["sch_climate_c", "Estimate"]),
    se     = unname(s["sch_climate_c", "Std. Error"]),
    within = unname(s["vic_fall",      "Estimate"]),
    # ICC = between-school variance / total variance
    icc    = vc$vcov[1] / sum(vc$vcov))
}

# lmer() fits the multilevel model. We fit it to the COMPLETE data to
# get the truth every other method will be scored against.
TRUTH <- grab(lmer(FORM, data = prep(truth), REML = TRUE))

header("THE TRUTH (we only know this because we simulated it)")
say(sprintf("contextual effect of school climate = %.4f (SE %.4f)",
            TRUTH["est"], TRUTH["se"]),
    sprintf("ICC of the fitted model             = %.4f", TRUTH["icc"]),
    "",
    "Every method below is scored against BOTH -- and the second turns",
    "out to matter more than the first.")


## ------------------------------------------------------------------
## 1.  LOOK AT THE MISSINGNESS BEFORE YOU MODEL IT        [~5 min]
## ------------------------------------------------------------------
header("1. MISSINGNESS")

# A vector of the variable names in our analysis, so we can refer to
# the whole set at once instead of retyping it.
anal_vars <- c("int_spring","vic_fall","int_fall","frpl_ind","sch_climate")

# is.na(x) returns TRUE/FALSE for each value: is it missing?
```

```
# mean() of TRUE/FALSE gives the PROPORTION that are TRUE, so
# 100 * mean(is.na(x)) is the percentage missing.
# sapply() applies that little function to every column we name.
pct_missing <- sapply(obs[, c(anal_vars, "fight_fall")],
                      function(x) round(100 * mean(is.na(x)), 1))
print(pct_missing)

# complete.cases() is TRUE only for rows with NO missing values across
# the columns you give it. That is exactly what listwise deletion keeps.
cc <- complete.cases(obs[, anal_vars])

say("",
    sprintf("Listwise deletion keeps %d of %d students (%.1f%%)",
            sum(cc), nrow(obs), 100 * mean(cc)),
    sprintf("...and %d of %d SCHOOLS.",
            length(unique(obs$school_id[cc])),   # unique() removes repeats
            length(unique(obs$school_id))),
    "",
    "sch_climate is a LEVEL-2 variable. When a school did not return the",
    "administrator survey, listwise deletion removes every student in it.")

# md.pattern() tabulates WHICH combinations of variables go missing
# together. 1 = observed, 0 = missing. The left column counts students.
cat("Missing-data patterns (top 6):\n")
print(head(md.pattern(obs[, anal_vars], plot = FALSE), 6))

# --- WHO are the complete cases? The ethical argument, not just the
# --- statistical one.
# obs$attend[cc] means "the attendance column, for the rows where cc is
# TRUE". !cc flips TRUE/FALSE, so obs$attend[!cc] is the deleted rows.
comparison <- rbind(
  complete = c(attendance    = mean(obs$attend[cc]),
               teacher_peer  = mean(obs$teacher_peer[cc]),
               school_frpl   = mean(obs$sch_frpl[cc]),
               school_enroll = mean(obs$sch_enroll[cc])),
  deleted  = c(mean(obs$attend[!cc]),
               mean(obs$teacher_peer[!cc]),
               mean(obs$sch_frpl[!cc]),
               mean(obs$sch_enroll[!cc])))

cat("\n--- Complete cases vs. deleted cases ---\n")
print(round(comparison, 2))

say("",
    "Smaller schools were less likely to return the survey, so listwise",
    "deletion is not sampling students at random -- it is sampling SCHOOLS.")

# Do the auxiliary variables actually predict who goes missing? If they
# do, they belong in the imputation model. glm(..., family = binomial)
# is a logistic regression; the outcome here is "is int_spring missing?"
aux_test <- glm(is.na(int_spring) ~ attend + teacher_peer + sch_frpl,
                data = obs, family = binomial)
cat("--- Do the auxiliaries predict missingness on the outcome? ---\n")
print(round(summary(aux_test)$coefficients, 4))
say("",
    "All three are significant. That is the evidence that our MAR",
    "assumption is doing real work rather than being asserted.")


## ------------------------------------------------------------------
## 2.  BUILD THE IMPUTATION MODEL (SINGLE-LEVEL)          [~7 min]
## ------------------------------------------------------------------
header("2. IMPUTATION METHODS (single-level)")

mi_dat <- obs                               # work on a copy, not the original
```

```
mi_dat$race5       <- factor(mi_dat$race5)    # tell R these are categories
mi_dat$frpl_ind    <- factor(mi_dat$frpl_ind)
mi_dat$student_id  <- NULL                    # assigning NULL DELETES a column
mi_dat$vic_climate <- NULL                    # (an ID number predicts nothing)
mi_dat$fight_fall  <- NULL                    # MNAR variable, held for Sec 5

# Copy again, and drop the school identifier. Without it, the imputation
# model has no way to know two students share a school.
sl_dat <- mi_dat
sl_dat$school_id <- NULL                      # clustering IGNORED, on purpose

# make.method() asks mice to guess a method for each column. We then
# overwrite the ones we care about.
#   pmm    = predictive mean matching. Draws from values people ACTUALLY
#            reported, so an imputed count can never be -0.4.
#   logreg = logistic regression, for yes/no variables.
meth_sl <- make.method(sl_dat)
meth_sl[c("int_spring","vic_fall","int_fall","sch_climate")] <- "pmm"
meth_sl["frpl_ind"] <- "logreg"

print(meth_sl[meth_sl != ""])       # show only the variables being imputed

say("",
    "NOTE what is already wrong: sch_climate is a SCHOOL-level variable,",
    "but this model imputes it student by student. Two students in one",
    "school get different values for a quantity their school has exactly",
    "one of. Watch the ICC in Section 4.")

# --- HOW MANY IMPUTATIONS? ---------------------------------------
# You do not pick this number. You calculate it.
#
# "5 is fine" is adequate for a POINT ESTIMATE and not for a STANDARD
# ERROR. von Hippel (2020) shows the number needed for a REPLICABLE
# standard error grows QUADRATICALLY with the fraction of missing
# information -- not linearly, as the older rule M = 100 x FMI assumed.
#
#   von Hippel's two-stage rule
#   ---------------------------
#   STAGE 1   Run a small pilot. He recommends 20 imputations.
#   STAGE 2   Read the FMI off the pilot, then
#
#                     M  =  1 + (1/2) x (FMI / cv)^2
#
#             cv is how much you will accept the standard error moving
#             if you re-imputed the data. At the conventional cv = 0.05
#             this simplifies to the form you will see quoted:
#
#                     M  =  1 + 200 x FMI^2
#
# Use the LARGEST FMI across your coefficients, not the one for your
# focal term, so the standard errors are stable for every term you
# report.

# ---- STAGE 1: the pilot ----
if (USE_CACHE && file.exists("cache/pilot.rds")) {
  pilot <- readRDS("cache/pilot.rds")        # load a saved R object
} else {
  cat("STAGE 1 -- pilot at m = 20...\n")
  pilot <- mice(sl_dat, m = 20, maxit = 10, method = meth_sl,
                printFlag = VERBOSE,         # <-- shows the iteration grid
                seed = 2020)
  saveRDS(pilot, "cache/pilot.rds")          # save it for next time
}

# complete(pilot, "all") hands back a plain LIST of 20 completed tables.
```

```
# lapply() runs a function on each item of a list and gives back a list.
# So this fits the same model 20 times, once per completed dataset.
pilot_fits <- lapply(complete(pilot, "all"), function(d)
  lm(int_spring ~ vic_fall + sch_climate + int_fall + female + frpl_ind +
     factor(grade) + factor(race5) + sch_frpl, data = d))

# testEstimates() combines those 20 answers using Rubin's rules and
# reports the FMI for every coefficient. We fit the same terms as our
# analytic model because the FMI we want is the FMI for the coefficients
# we are actually going to report.
FMI <- max(testEstimates(pilot_fits)$estimates[, "FMI"], na.rm = TRUE)

# ---- STAGE 2: apply the rule ----
# ceiling() rounds UP -- you cannot run 97.4 imputations.
M_IMP <- ceiling(1 + 0.5 * (FMI / CV_TARGET)^2)

say(sprintf("STAGE 1  pilot m = 20, largest FMI = %.3f", FMI),
    sprintf("STAGE 2  M = 1 + 0.5 * (%.3f / %.2f)^2  =  %d imputations",
            FMI, CV_TARGET, M_IMP),
    "",
    sprintf("The old linear rule would have said %d. The old advice of 5",
            ceiling(100 * FMI)),
    "would leave your standard errors unreproducible.",
    "",
    "Note: howManyImputations::how_many_imputations() applies the same",
    "rule but plugs in the UPPER confidence limit of the FMI rather than",
    "the point estimate, so it returns a somewhat larger, more",
    "conservative M. Either is defensible; state which you used.")

sl_file <- sprintf("cache/imp_sl_m%d.rds", M_IMP)

if (USE_CACHE && file.exists(sl_file)) {
  imp_sl <- readRDS(sl_file)
} else {
  cat("Running single-level imputation...\n")
  imp_sl <- mice(sl_dat, m = M_IMP, maxit = MAXIT, method = meth_sl,
                 printFlag = VERBOSE, seed = 451)
  saveRDS(imp_sl, sl_file)
}

# --- DIAGNOSTICS: never skip these --------------------------------
# png() opens an image file, the plot is drawn into it, dev.off() closes
# it. Drop the png()/dev.off() lines to see plots in RStudio instead.
# IMPORTANT: the layout must be big enough for EVERY panel. There are two
# panels per incomplete variable (a mean and an SD). If the layout holds
# fewer, plot() silently spills onto a second page and png() saves only
# the LAST one -- so variables vanish from your diagnostic without warning.
n_imputed <- sum(imp_sl$method != "")        # how many variables have gaps
png("output/fig_convergence_singlelevel.png", 1100, 260 * n_imputed, res = 120)
print(plot(imp_sl, layout = c(2, n_imputed)))  # 2 columns, one row per variable
dev.off()

png("output/fig_density_singlelevel.png", 1100, 650, res = 120)
print(densityplot(imp_sl, ~ int_spring + vic_fall + sch_climate))
dev.off()

check_convergence(imp_sl, "single-level imputation")

say("Diagnostics written to output/.",
    "  Trace plots  -- lines should MIX rather than drift apart or run",
    "                  in parallel. Parallel lines mean it has not settled.",
    "  Density plots -- imputed values (red) should be plausible against",
    "                  observed (blue), but NOT identical to them.")
```

```
## ------------------------------------------------------------------
## 3.  DECISION POINT: DOES CLUSTERING CHANGE THE
##     IMPUTATION MODEL?                                    [~5 min]
## ------------------------------------------------------------------
header("3. MULTILEVEL IMPUTATION METHODS")

# The analytic model has a random intercept for school. The single-level
# imputation model does not. That is a CONGENIALITY failure: the imputation
# model is simpler than the analysis model, so it smooths away exactly the
# between-school variance we are trying to estimate.
#
#   2l.pan      level-1 continuous, clustered
#   2l.bin      level-1 binary, clustered (slow: ~7x logreg -- and
#               only worth it when the cluster variance is non-zero)
#   2lonly.pmm  LEVEL-2 variable -- one imputed value per school
#   -2          marks the cluster variable in the predictor matrix
#    3          adds the CLUSTER MEAN of a level-1 predictor

ml_dat <- mi_dat        # this copy KEEPS school_id

# The predictor matrix is a grid: one row per variable being imputed,
# one column per possible predictor. The number in each cell says HOW
# that predictor is used. make.predictorMatrix() gives a default to edit.
pred_ml <- make.predictorMatrix(ml_dat)

# Level-1 variables -- ones that vary from student to student.
L1 <- c("int_spring","vic_fall","int_fall","teacher_peer","attend",
        "female","frpl_ind","grade")

# WHY CODE 3 MATTERS: a random-intercept imputation model fits ONE
# pooled slope. The real world has a within-student effect AND a
# different contextual effect. Without cluster means the model cannot
# tell them apart and compresses the between-school structure toward the
# mean. See Grund, Ludtke & Robitzsch (2018).
#
# A for-loop repeats the same instruction for each item in a list.
# intersect() keeps only names present in both, which avoids errors if
# a column is missing.
for (r in c("int_spring","vic_fall","int_fall","frpl_ind")) {
  pred_ml[r, intersect(L1, colnames(pred_ml))] <- 3
}

diag(pred_ml) <- 0                # a variable must not predict itself
pred_ml[, "school_id"] <- -2      # -2 flags the cluster identifier

meth_ml <- make.method(ml_dat)
meth_ml[c("int_spring","vic_fall","int_fall")] <- "2l.pan"
meth_ml["sch_climate"] <- "2lonly.pmm"

# WHY frpl_ind IS *NOT* IMPUTED WITH THE CLUSTERED METHOD (2l.bin)
#   Individual lunch eligibility varies between schools only through the
#   school poverty rate -- and sch_frpl is already a predictor here. Once
#   you condition on it there is NO residual between-school variance left
#   for a random intercept to capture. Fit one anyway and glmer estimates
#   the variance at exactly zero and reports:
#       boundary (singular) fit: see help('isSingular')
#   ...on every iteration of every imputation. It is a warning, not an
#   error, and the imputations are usable -- but you are paying roughly a
#   sevenfold speed penalty to estimate a parameter that is zero by
#   construction.
#
#   So logreg here is not a shortcut. It is the correctly specified model
#   for THIS variable in THIS dataset. The general rule still holds: a
#   clustered binary with real between-cluster variation needs 2l.bin.
#   Check before you assume which case you are in:
```

```
#       glmer(frpl_ind ~ sch_frpl + (1|school_id), family = binomial)
#   A variance at or near zero means the random intercept is not earning
#   its keep.
meth_ml["frpl_ind"] <- "logreg"

print(meth_ml[meth_ml != ""])

# Show the grid of decisions we just built.
show_predictors(pred_ml, meth_ml)

ml_file <- sprintf("cache/imp_ml_m%d.rds", M_IMP)

if (USE_CACHE && file.exists(ml_file)) {
  imp_ml <- readRDS(ml_file)
} else {
  cat("Running multilevel imputation. Cached afterwards.\n\n")
  imp_ml <- mice(ml_dat, m = M_IMP, maxit = MAXIT, method = meth_ml,
                 predictorMatrix = pred_ml,
                 printFlag = VERBOSE, seed = 451)
  saveRDS(imp_ml, ml_file)
}

n_imputed <- sum(imp_ml$method != "")
png("output/fig_convergence_multilevel.png", 1100, 260 * n_imputed, res = 120)
print(plot(imp_ml, layout = c(2, n_imputed)))
dev.off()

check_convergence(imp_ml, "multilevel imputation")


# --- Did the level-2 imputer actually do its job? -----------------
# complete(imp, 1) hands back the FIRST completed dataset.
# tapply(X, GROUP, FUN) applies FUN to X separately within each GROUP.
# Here: within each school, how many DIFFERENT climate values are there?
# The right answer is 1. A school has one climate score.
chk  <- complete(imp_ml, 1)
chk2 <- complete(imp_sl, 1)
chk2$school_id <- obs$school_id       # put the school back for comparison

n_ml <- max(tapply(chk$sch_climate,  chk$school_id,
                   function(x) length(unique(x))))
n_sl <- max(tapply(chk2$sch_climate, chk2$school_id,
                   function(x) length(unique(x))))

say("",
    sprintf("2lonly.pmm  : most distinct climate values within one school = %d", n_ml),
    sprintf("single-level: most distinct climate values within one school = %d  <-- wrong", n_sl),
    "",
    "One school, one climate score. The single-level model invented",
    "within-school variation in a variable that has none.")


## ------------------------------------------------------------------
## 4.  THE COMPARISON TABLE                                 [~7 min]
## ------------------------------------------------------------------
header("4. THE COMPARISON")

# Fit the model to every imputed dataset and combine by Rubin's rules.
# We do the pooling by hand here (rather than mice's pool()) because we
# also want the ICC, which pool() does not carry.
# Fit the analytic model to every completed dataset, then combine.
#
# We used to do this by hand -- loop over the imputations, fit, then code
# Rubin's rules ourselves. Do not. mitml::testEstimates() does the pooling
# properly, AND with extra.pars = TRUE it pools the VARIANCE COMPONENTS
```

```
# too, which is where the ICC comes from. Averaging the ICCs from each
# dataset (the obvious hand-rolled move) is not a valid pooling rule.
pool_fits <- function(imp, school_vec = NULL) {

  datasets <- complete(imp, "all")   # a LIST of completed data frames

  # seq_along(x) gives 1, 2, 3, ... up to the length of x. We loop over
  # positions rather than the data itself only so we can show progress.
  fits <- lapply(seq_along(datasets), function(k) {
    progress(k, length(datasets))
    d <- datasets[[k]]                          # [[k]] pulls out item k
    if (!is.null(school_vec)) d$school_id <- school_vec
    d$frpl_ind <- as.numeric(as.character(d$frpl_ind))
    suppressMessages(suppressWarnings(
      lmer(FORM, data = prep(d), REML = TRUE)))
  })

  te <- testEstimates(fits, extra.pars = TRUE)  # <-- the pooling step

  # te$estimates  is the fixed-effects table (Estimate, Std.Error, FMI...)
  # te$extra.pars holds the variance components, including the ICC.
  c(est    = unname(te$estimates["sch_climate_c", "Estimate"]),
    se     = unname(te$estimates["sch_climate_c", "Std.Error"]),
    within = unname(te$estimates["vic_fall",      "Estimate"]),
    icc    = unname(te$extra.pars["ICC|school_id", 1]),
    fmi    = unname(te$estimates["sch_climate_c", "FMI"]))
}

# Listwise deletion: keep only complete rows, then fit once.
lw   <- obs[complete.cases(obs[, anal_vars]), ]
R_LW <- grab(lmer(FORM, data = prep(lw), REML = TRUE))

cat("Pooling the single-level imputations...\n")
R_SL <- pool_fits(imp_sl, school_vec = obs$school_id)

cat("Pooling the multilevel imputations...\n")
R_ML <- pool_fits(imp_ml)

# Build one row of the results table.
row <- function(lab, r, fmi = NA) {
  data.frame(method   = lab,
             estimate = round(unname(r["est"]), 4),
             SE       = round(unname(r["se"]),  4),
             SE_infl  = paste0(round(100*(unname(r["se"])/TRUTH["se"] - 1)), "%"),
             ICC      = round(unname(r["icc"]), 4),
             ICC_err  = paste0(round(100*(unname(r["icc"])/TRUTH["icc"] - 1)), "%"),
             FMI      = ifelse(is.na(fmi), NA, round(unname(fmi), 3)))
}

# rbind() stacks rows into one table.
TAB <- rbind(
  row("1. Complete data (truth)", TRUTH),
  row("2. Listwise deletion",     R_LW),
  row("3. Single-level MI",       R_SL, R_SL["fmi"]),
  row("4. Multilevel MI",         R_ML, R_ML["fmi"]))

header("CONTEXTUAL EFFECT OF SCHOOL VICTIMISATION CLIMATE")
print(TAB, row.names = FALSE)
write.csv(TAB, "output/comparison_table.csv", row.names = FALSE)

say("",
    "--- HOW TO READ THIS TABLE ---",
    "The point estimates sit within about half a standard error of each",
    "other. You could not choose between these methods on that column.",
    "We checked across random seeds and the direction of the error moves",
```

```
"around, so do not sell it as 'MI recovers the right answer'.",
"",
"Read the SE and ICC columns instead. Those differences are systematic:",
"  * Every method pays an SE penalty for the missing data. Multilevel",
"    MI pays the smallest -- it is the most efficient of the three.",
"  * Single-level MI UNDERSTATES the ICC. It imputed a school-level",
"    variable student by student and flattened the between-school",
"    variance. (Ludtke, Robitzsch & Grund, 2017.)",
"  * Listwise deletion OVERSTATES the ICC -- it dropped whole schools,",
"    distorting the variance decomposition the other way.",
"  * Only multilevel MI recovers the variance decomposition.",
"",
"If your research question is about a contextual effect, the variance",
"decomposition IS the finding. Getting it wrong by a third is not a",
"rounding error.")
```