

ABG-VBG Analysis

Brian Locke, Anila Mehta

Table of contents

1 Data Pre-Processing and Unweighted Analyses	1
1.0.1 Package Set Up	6
1.1 Helper functions for model diagnostics	13
1.1.1 Configuration for the IPW models	53
1.1.2 Outcome Variable Creation	79
1.2 Baseline tables	81
1.3 Three-level PCO ₂ categories (unweighted)	95
1.4 Observed outcome rates by CO ₂ bin (unweighted)	99
1.5 Restricted cubic spline regressions (unweighted)	102
1.5.1 Unweighted, Restricted Cubic Spline Regression - ABG by PaCO ₂	103
1.5.2 Unweighted, Restricted Cubic Spline - VBG	105
2 Inverse Propensity Weighting (GBM)	111
2.0.1 ABG IPW weighting and diagnostics	111
2.0.2 ABG IPW spline models	113
2.0.3 ABG IPW spline models (2–98th percentile)	115
2.0.4 VBG IPW weighting and spline models	119
2.0.5 Three-level PCO ₂ categories (weighted; ABG, VBG)	128
2.0.6 Observed outcome rates by CO ₂ bin (IPSW, non-MI)	131
2.1 Propensity score diagnostics	134
3 Multiple Imputation (and logistic IPSW)	147
3.0.1 Missingness structure and drivers	153

3.0.2	Monte Carlo error check after MI	158
3.1	Pre-imputation data prep (consistent types & predictors)	158
3.2	Imputation model specification (MICE)	158
3.2.1	Predictor matrix & methods. Run MICE (moderate settings for scale)	158
3.3	Refit propensity models within each imputation	239
3.3.1	FAIL-FAST CHECKS	239
3.3.2	ABG propensity (has_abg)	264
3.3.3	Balance diagnostics across imputations	265
3.3.4	VBG propensity (has_vbg)	266
3.3.5	VBG balance	266
3.4	Weighted outcome models within each imputation + pooling	268
3.4.1	Helper: fit + extract log-OR and SE from svyglm	268
3.4.2	VBG: MI pooled spline models (treated cohort only)	268
3.5	Explainability	269
3.6	MI three-level PCO2 helpers and checks	278
3.7	MI + IPW three-level PCO2 (ABG & VBG)	278
3.7.1	ABG: MI + IPW, three-level PCO2 outcomes	278
3.7.2	VBG: MI + IPW, three-level PCO2 outcomes	278
3.7.3	MI-pooled IPW associations (3-level CO2)	279
3.7.4	Summary: adjusted CO2-category associations across analysis tracks	280
3.8	Observed outcome rates by CO2 bin (MI + IPSW)	291
3.9	Manuscript outputs summary	297
3.9.1	Visualization: pooled three-level ORs	300
3.9.2	Visualization	310
4	Diagnostics and Exports	316
4.0.1	MI convergence and mixing	316
4.0.2	MI stability across m	319
4.0.3	MI maxit sensitivity (sampled)	324
4.0.4	Balance diagnostics	329
4.0.5	Outcome diagnostics	333
4.0.6	Diagnostics summary and audit	333
4.0.7	Performance / runtime log	340
4.0.8	Performance / runtime log	345
4.1	Save, export, and session info	349

1 Data Pre-Processing and Unweighted Analyses

This code pulls in the master database (a STATA file) and does some initial cleaning - this will only need to be run once, and then the data can be accessed in the usual way.

```
# put this in your first R chunk
stopifnot(requireNamespace("kableExtra", quietly = TRUE))
library(kableExtra)
library(gtsummary)
library(purrr)      # functional programming

# globally tighten gtsummary/gt tables (smaller font + tighter padding)
gtsummary::theme_gtsummary_compact()

# keep figures anchored in PDF to reduce heading-only/blank pages
if (knitr::is_latex_output()) {
  knitr::opts_chunk$set(fig.pos = "H")
}

# helper: turn any gtsummary table into a PDF-safe, auto-scaling LaTeX table
to_pdf_table <- function(tbl, font_size = 8, landscape = FALSE,
                           label_col_width = NULL, longtable = FALSE) {
  tbl_ncol <- tryCatch(ncol(gtsummary::as_tibble(tbl)), error = function(e) NA_integer_)
  if (is.finite(tbl_ncol) && tbl_ncol > 6 && font_size > 6) font_size <- 6
  kbl <- gtsummary::as_kable(
    tbl,
    format    = "latex",
    booktabs  = TRUE,
    longtable = longtable
  )

  # optional: set a fixed width for the first (label) column to encourage wrapping
  if (!is.null(label_col_width)) {
    kbl <- kableExtra::column_spec(kbl, 1, width = label_col_width)
  }

  latex_opts <- if (longtable) {
```

```

  c("repeat_header")
} else {
  c("hold_position", "scale_down")
}
kbl <- kableExtra::kable_styling(
  kbl,
  latex_options = latex_opts,
  font_size      = font_size,
  full_width     = !longtable,
  position       = "center"
)

if (landscape) kbl <- kableExtra::landscape(kbl) # needs pdflscape (enabled above)
mark_float_emitted()
kbl
}

# helper: scale generic data.frames for PDF output
kable_pdf <- function(df, caption = NULL, font_size = 7) {
  kbl <- knitr::kable(df, format = "latex", booktabs = TRUE, caption = caption)
  kbl <- kableExtra::kable_styling(
    kbl,
    latex_options = c("hold_position", "scale_down"),
    font_size      = font_size,
    full_width     = TRUE,
    position       = "center"
)
  mark_float_emitted()
  kbl
}

# helper: pretty, relative paths for PDF output
pretty_path <- function(p) {
  p_abs <- tryCatch(fs::path_abs(p, start = getwd()), error = function(e) NA_character_)
  res_abs <- tryCatch(fs::path_abs(results_dir, start = getwd()), error = function(e) NA_character_)
  if (!is.na(p_abs) && !is.na(res_abs)) {
    if (fs::path_has_parent(p_abs, res_abs) || identical(p_abs, res_abs)) {

```

```

    rel <- fs::path_rel(p_abs, res_abs)
    return(fs::path("Results", rel))
  }
}

out <- tryCatch(fs::path_rel(p, start = getwd()), error = function(e) NA_character_)
if (is.na(out) || out == "." || grepl("^\\\\.\\\"., out)) {
  return(basename(p))
}
out
}

# helper: strip manual "Table X." prefixes in PDF to avoid double numbering
strip_manual_table_number <- function(caption) {
  if (is.null(caption)) return(caption)
  if (!knitr:::is_latex_output()) return(caption)
  cap <- gsub("^\s*(\*\*\*)?Table\s+[0-9A-Za-z]+\.\.?*\s*", "", caption)
  cap
}

# helper: escape LaTeX special chars in plain captions
escape_latex_text <- function(x) {
  if (is.null(x)) return(x)
  if (!knitr:::is_latex_output()) return(x)
  gsub("[\\\$%{}^~]", "\\\\1", x, perl = TRUE)
}

# Float barrier helpers (use 4 modes to control float flushing)
mark_float_emitted <- function() {
  invisible(TRUE)
}

float_barrier <- function(mode = c("soft", "section", "hard", "flush")) {
  if (!knitr:::is_latex_output()) return(invisible(NULL))
  mode <- match.arg(mode)
  cmd <- switch(
    mode,
    soft = "\\FloatBarrier\\nopagebreak",
    section = "\\FloatBarrier\\nopagebreak\\par\\noindent",

```

```

hard = "\\FloatBarrier\\clearpage",
flush = "\\FloatBarrier\\pagebreak"
)
knitr::asis_output(cmd)
}

# helper: render full tables for PDF (no previews)
render_table_pdf <- function(df, caption, file_stub,
                             wide = FALSE,
                             digits = 2,
                             max_cols = 6,
                             landscape = NULL) {
stopifnot(is.data.frame(df))
out_csv <- results_path(paste0(file_stub, ".csv"))
write_csv_safely(df, out_csv, row_names = FALSE)

if (is.null(landscape)) landscape <- (wide || ncol(df) > max_cols)
use_longtable <- nrow(df) > 40
n_cols <- ncol(df)
parts <- if (n_cols <= max_cols) list(df) else {
  idx <- split(seq_len(n_cols), ceiling(seq_len(n_cols) / max_cols))
  lapply(idx, function(ii) df[, ii, drop = FALSE])
}
build_kbl <- function(df_part, cap) {
  cap <- escape_latex_text(strip_manual_table_number(cap))
  kbl <- knitr::kable(
    df_part,
    format      = "latex",
    booktabs   = TRUE,
    longtable  = use_longtable,
    caption    = cap,
    digits     = digits
  )
  kbl <- kableExtra::kable_styling(
    kbl,
    latex_options = if (use_longtable) c("repeat_header") else c("hold_position", "scale_down"),

```

```

font_size      = if (ncol(df_part) > max_cols || wide) 6 else 7,
full_width    = FALSE,
position       = "center"
)
if (landscape) kbl <- kableExtra::landscape(kbl)
kbl
}

if (length(parts) == 1L) {
  mark_float_emitted()
  return(build_kbl(parts[[1]], caption))
}

letters_part <- LETTERS[seq_along(parts)]
out <- character(0)
for (i in seq_along(parts)) {
  cap_i <- paste0(caption, " (Part ", letters_part[i], ")")
  out <- c(out, build_kbl(parts[[i]], cap_i))
}
mark_float_emitted()
knitr::asis_output(paste(out, collapse = "\n\n"))
}

render_table_pdf_maybe <- function(df, caption, file_stub,
                                    wide = FALSE,
                                    digits = 2,
                                    max_cols = 6,
                                    landscape = NULL,
                                    show = TRUE) {

  if (!isTRUE(show)) {
    write_csv_safely(df, results_path(paste0(file_stub, ".csv")), row_names = FALSE)
    return(invisible(NULL))
  }
  tbl <- render_table_pdf(
    df,
    caption = caption,
    file_stub = file_stub,

```

```

    wide = wide,
    digits = digits,
    max_cols = max_cols,
    landscape = landscape
)
if (isTRUE(show)) {
  return(tbl)
}
invisible(NULL)
}

```

1.0.1 Package Set Up

```

# Consolidated package management -----
required_pkgs <- c(
  "WeightIt", "broom", "cobalt", "codebookr", "dplyr", "flextable", "parallel",
  "gbm", "ggplot2", "gt", "gtsummary", "haven", "labelled", "Matrix", "scales",
  "modelsummary", "officer", "patchwork", "rms", "survey", "tibble", "lubridate",
  "sensitivitymw", "here", "fs", "dagitty", "ggdag", "naniar", "mice", "miceadds",
  "digest"
)

# Fail fast if packages are missing (use renv to install)
missing_pkgs <- setdiff(required_pkgs, rownames(installed.packages()))
if (length(missing_pkgs)) {
  stop(
    "Missing packages: ", paste(missing_pkgs, collapse = ", "),
    ". Install with renv::install(c(...)) and then run renv::snapshot()."
  )
}

# Load (or attach) all required packages
invisible(lapply(required_pkgs, require, character.only = TRUE))

# Centralize outputs under Results/, with paths relative to project root
qmd_dir <- here::here("Code Drafts")

```

```

data_dir_param <- Sys.getenv("ABGVBG_DATA_DIR", unset = params$data_dir)
results_dir_param <- Sys.getenv("ABGVBG_RESULTS_DIR", unset = params$results_dir)
data_dir_name <- fs::path_abs(data_dir_param, start = here::here())
results_dir <- fs::path_abs(results_dir_param, start = here::here())
if (!dir.exists(data_dir_name)) {
  stop("Data directory does not exist: ", data_dir_name)
}
fs::dir_create(results_dir, recurse = TRUE)
fig_dir <- fs::path(results_dir, "figs")
fig_path <- paste0(fs::path_rel(fig_dir, start = qmd_dir), "/")
results_path <- function(...) fs::path(results_dir, ...)
fs::dir_create(fig_dir, recurse = TRUE)

knitr::opts_chunk$set(
  fig.path    = fig_path,
  dev        = "ragg_png",
  dpi        = 200
)
# on macOS and some setups this prevents device headaches
options(bitmapType = "cairo")

# Purpose: cohort flow placeholder.
flow_png <- results_path("cohort_flow.png")
flow_pdf <- results_path("cohort_flow.pdf")
if (file.exists(flow_png) || file.exists(flow_pdf)) {
  knitr::include_graphics(if (file.exists(flow_png)) flow_png else flow_pdf)
} else {
  cat("Cohort flow diagram will be generated externally and inserted here in the final manuscript packet.\n")
}

```

Cohort flow diagram will be generated externally and inserted here in the final manuscript packet.

```

# Purpose: setup run config.
stopifnot(!is.null(params$run_mode))
stopifnot(!is.null(params$pilot_frac))
stopifnot(!is.null(params$mi_batch_threshold_pilot))
stopifnot(!is.null(params$mi_batch_threshold_full))

```

```

stopifnot(!is.null(params$sample_seed))

RUN_MODE <- match.arg(tolower(params$run_mode), c("pilot", "full"))
PILOT_FRAC <- as.numeric(params$pilot_frac)
stopifnot(is.finite(PILOT_FRAC), PILOT_FRAC > 0, PILOT_FRAC <= 1)
if (RUN_MODE == "full") PILOT_FRAC <- 1
SAMPLE_SEED <- as.integer(params$sample_seed)
stopifnot(is.finite(SAMPLE_SEED))

pilot_frac <- PILOT_FRAC
FULL_RUN <- identical(RUN_MODE, "full")
SHOW_LOW_VALUE_TABLES <- FALSE
MI_BATCH_THRESHOLD <- if (RUN_MODE == "pilot") {
  as.integer(params$mi_batch_threshold_pilot)
} else {
  as.integer(params$mi_batch_threshold_full)
}
stopifnot(is.finite(MI_BATCH_THRESHOLD), MI_BATCH_THRESHOLD > 0)
PLOT_DROP_POLICY <- "warn"
PROB_EPS <- 1e-6
MAX_SHAPE_LEVELS <- 6L
SPLINE_GRID_N <- 200L
OR_XLIM <- c(0.25, 16)
TAIL_NFINITE <- 5L
TAIL_WINDOW_ITERS <- 10L
BAL_XLIM_MAX <- 1.0
MC_ERR_RATIO_THRESH <- 0.10
SHOW_CHUNK_RUNTIME_TEXT <- FALSE
MAX_LEVELS_GBM <- 50L
GBM_MM_EST_BYTES_WARN <- 6e9
MAX_GBM_MM_COLS_WARN <- 200L
MAX_GBM_MM_COLS_STOP <- 400L
MAX_GBM_FACTOR_LEVELS_WARN <- 30L
MAX_GBM_FACTOR_LEVELS_STOP <- 60L

message(
  "CONFIG: RUN_MODE=", RUN_MODE,

```

```

" | PILOT_FRAC=", PILOT_FRAC,
" | SAMPLE_SEED=", SAMPLE_SEED
)

# Purpose: setup diagnostics output.
diag_run_id <- format(Sys.time(), "%Y%m%d_%H%M%S")
diag_run_ts <- as.character(Sys.time())
run_id <- diag_run_id
run_ts <- diag_run_ts
runtime_run_id <- diag_run_id
options(run_id = run_id, run_ts = run_ts)
runtime_log <- data.frame()
mi_warn_log <- data.frame(
  time = character(), stage = character(), component = character(),
  analysis_variant = character(), model_type = character(),
  group = character(), outcome = character(), imputation = integer(),
  batch = integer(), message = character(), stringsAsFactors = FALSE
)
mi_info_log <- mi_warn_log
mi_outcome_diag <- data.frame()
memory_snapshots <- data.frame()
gbm_preflight_design_dims <- data.frame()
gbm_preflight_warnings <- data.frame()
factor_diag_written <- character()
plot_drop_log_df <- data.frame(
  plot = character(), reason = character(), n_dropped = integer(),
  n_before = integer(), n_after = integer(),
  stage = character(), extra = character(),
  stringsAsFactors = FALSE
)
cleanup_diagnostics_outputs <- function() {
  diag_files <- c(
    "runtime_log.csv", "runtime_summary.csv", "runtime_summary_top15.csv",
    "warnings_log.csv", "mi_warnings_log.csv", "mi_info_log.csv",
    "diagnostics_summary.csv", "diagnostics_missingness.csv",
    "diagnostics_missingness-by-strata.csv",

```

```

"balance_target_imp_summary.csv", "balance_target_by_imp.csv",
"balance_target_worst_rows.csv", "balance_max_smd_by_imp.csv",
"balance_worst_terms.csv", "balance_worst10.csv", "balance_table.csv",
"weight_summary.csv", "ps_overlap_summary.csv",
"model_fit_diagnostics.csv", "mi_outcome_fit_diagnostics.csv",
"mi_fit_issue_summary.csv", "mi_m_stability.csv", "mi_maxit_sensitivity.csv",
"mi_obs_vs_imp_summary.csv", "mi_spline_curve_abg.csv",
"mi_spline_curve_vbg.csv", "mi_spline_coef_abg.csv",
"mi_spline_coef_vbg.csv", "diag-ps-shap-stability.csv",
"mice_smoketest.log", "mice_batches_log.csv", "mice_chain_diagnostics.csv",
"mice_pred_width_preflight.csv", "mice_logged_events_raw.csv",
"mice_logged_events_summary.csv", "mice_spec.rds",
"missingness-by-strata.csv", "missingness-drivers.csv",
"missingness-pattern.csv", "plot_drop_log.csv",
"plot_registry.csv",
"mi_logistic_ps_covariate_types.csv",
"mi_logistic_ps_abg_list.rds",
"mi_logistic_ps_vbg_list.rds"
)
diag_patterns <- c(
  "^mice_.*\\.(csv|log|rds|txt)$",
  "^diagnostics_.*\\(.csv$",
  "^runtime_.*\\(.csv$",
  "^warnings_.*\\(.csv$",
  "^balance_.*\\(.csv$",
  "^weight_summary\\(.csv$",
  "^ps_overlap_summary\\(.csv$",
  "^model_fit_diagnostics\\(.csv$",
  "^plot_drop_log\\(.csv$",
  "^diag-.*\\(.csv$",
  "^mi_logistic_ps_.*\\(.csv|rds)$"
)
files <- list.files(results_dir, full.names = TRUE)
base <- basename(files)
match_files <- base %in% diag_files
if (length(diag_patterns)) {
  match_files <- match_files | grepl(paste(diag_patterns, collapse = "|"), base)
}

```

```

}

to_delete <- files[match_files]
if (length(to_delete)) {
  safe_root <- normalizePath(results_dir, winslash = "/", mustWork = FALSE)
  del_paths <- normalizePath(to_delete, winslash = "/", mustWork = FALSE)
  stopifnot(all(startsWith(del_paths, safe_root)))
}
fs::file_delete(to_delete)

diag_figs <- list.files(fig_dir, full.names = TRUE)
diag_figs <- diag_figs[grep("(diag-|or-plot|ps-)", basename(diag_figs))]
if (length(diag_figs)) {
  safe_root <- normalizePath(results_dir, winslash = "/", mustWork = FALSE)
  del_paths <- normalizePath(diag_figs, winslash = "/", mustWork = FALSE)
  stopifnot(all(startsWith(del_paths, safe_root)))
}
fs::file_delete(diag_figs)
{
  d <- fs::path(results_dir, "mice_batches")
  if (dir.exists(d)) fs::dir_delete(d)
}
{
  d <- fs::path(results_dir, "diagnostics_audit_snippets")
  if (dir.exists(d)) fs::dir_delete(d)
}
{
  f <- fs::path(results_dir, "mi_partial_mids.rds")
  if (file.exists(f)) fs::file_delete(f)
}
}

cleanup_diagnostics_outputs()
# Ensure any stale Results artifacts from prior runs are removed
stale_patterns <- c(
  "^\$missingness_by_strata\\*.csv$",
  "^\$missingness-drivers\\*.csv$",
  "^\$missingness_by_strata_preview\\*.csv$",

```

```

"~missingness_drivers_top20\\.csv$",
"~missingness_top10\\.csv$",
"~outcome_counts_by_cohort\\.csv$",
"~weighting_diagnostics_non_mi\\.csv$",
"~mi_specification\\.csv$",
"~memory_snapshots\\.csv$",
"~gbm_preflight_design_dims\\.csv$",
"~gbm_preflight_factor_levels_.*\\.csv$",
"~diag_mi_shap_errors\\.csv$",
"~diag_mi_shap_method_used\\.csv$",
"~mi_shap_imp_diagnostics\\.csv$",
"~mi_logistic_ps_factor_levels_.*\\.csv$",
"~mi_logistic_ps_.*\\.(csv|rds)$",
"~get_imp_usage\\.csv$",
"~diagnostics_audit\\.md$",
"~pdf_page_text_stats\\.csv$",
"~pdf_blank_pages\\.json$",
"~pdf_hygiene_scan\\.csv$"
)
stale_files <- list.files(results_dir, full.names = TRUE)
stale_base <- basename(stale_files)
stale_files <- stale_files[grep(paste(stale_patterns, collapse = "|"), stale_base)]
if (length(stale_files)) fs::file_delete(stale_files)

```

1.1 Helper functions for model diagnostics

```

# Diagnostics helper functions for MI/IPW pipeline

assert_no_na_covars <- function(df, covars, context = "") {
  missing <- setdiff(covars, names(df))
  stopifnot(length(missing) == 0)
  na_counts <- vapply(df[, covars, drop = FALSE], function(x) sum(is.na(x)), numeric(1))
  if (any(na_counts > 0)) {
    msg <- paste0(
      "NA values found in covariates (", context, "): ",
      paste(names(na_counts)[na_counts > 0], na_counts[na_counts > 0], collapse = ", "))
  }
}
```

```

)
stop(msg)
}
invisible(TRUE)
}

summarize_logged_events <- function(mids_obj) {
  ev <- mids_obj$loggedEvents
  if (is.null(ev) || nrow(ev) == 0L) {
    return(data.frame(
      variable = character(),
      method = character(),
      event = character(),
      n = integer(),
      pct = numeric(),
      stringsAsFactors = FALSE
    ))
  }
  stopifnot(all(c("dep", "meth", "out") %in% names(ev)))
  tbl <- as.data.frame(table(ev$dep, ev$meth, ev$out), stringsAsFactors = FALSE)
  names(tbl) <- c("variable", "method", "event", "n")
  tbl <-tbl[tbl$n > 0, , drop = FALSE]
  tbl <-tbl[order(-tbl$n), , drop = FALSE]
  tbl$pct <-tbl$n / sum(tbl$n)
  tbl
}

extract_weightit_ps <- function(w) {
  stopifnot(!is.null(w$ps))
  as.numeric(w$ps)
}

ess <- function(w) {
  w <- w[is.finite(w)]
  stopifnot(length(w) > 0)
  sum(w)^2 / sum(w^2)
}

```

```

weight_concentration <- function(w, top_p = 0.01) {
  w <- w[is.finite(w)]
  stopifnot(length(w) > 0)
  ord <- sort(w, decreasing = TRUE)
  n_top <- max(1L, floor(length(ord) * top_p))
  sum(ord[seq_len(n_top)]) / sum(ord)
}

weight_summary <- function(w, ps, ps_floor, truncated) {
  w_ok <- w[is.finite(w)]
  stopifnot(length(w_ok) > 0)
  ps_ok <- ps[is.finite(ps)]
  stopifnot(length(ps_ok) > 0)
  out <- data.frame(
    n = length(w_ok),
    mean = mean(w_ok, na.rm = TRUE),
    sd = stats::sd(w_ok, na.rm = TRUE),
    min = min(w_ok, na.rm = TRUE),
    p01 = stats::quantile(w_ok, 0.01, na.rm = TRUE),
    p05 = stats::quantile(w_ok, 0.05, na.rm = TRUE),
    p95 = stats::quantile(w_ok, 0.95, na.rm = TRUE),
    p99 = stats::quantile(w_ok, 0.99, na.rm = TRUE),
    max = max(w_ok, na.rm = TRUE),
    sum_w = sum(w_ok, na.rm = TRUE),
    ess = ess(w_ok),
    top01_weight_share = weight_concentration(w_ok, top_p = 0.01),
    ps_floor = ps_floor,
    trunc_rate = mean(truncated, na.rm = TRUE),
    ps_min = min(ps_ok, na.rm = TRUE),
    ps_p01 = stats::quantile(ps_ok, 0.01, na.rm = TRUE),
    ps_p05 = stats::quantile(ps_ok, 0.05, na.rm = TRUE),
    ps_p95 = stats::quantile(ps_ok, 0.95, na.rm = TRUE),
    ps_max = max(ps_ok, na.rm = TRUE),
    stringsAsFactors = FALSE
)
out

```

```

}

compute_ipow_weights <- function(w, treat, ps_floor_quantile = 0.01,
                                  stabilize = TRUE) {
  ps <- extract_weightit_ps(w)
  if (length(ps) != length(treat)) stop("Propensity length mismatch.")
  treat <- as.integer(treat)
  if (!all(treat %in% c(0L, 1L))) stop("Treatment indicator must be 0/1.")

  ps_obs <- ps[treat == 1L & is.finite(ps)]
  if (!length(ps_obs)) stop("No treated observations with finite propensity.")
  ps_floor <- as.numeric(stats::quantile(ps_obs, probs = ps_floor_quantile, na.rm = TRUE))
  if (!is.finite(ps_floor) || ps_floor <= 0) stop("Invalid propensity floor.")

  w_raw <- rep(NA_real_, length(ps))
  w_raw[treat == 1L] <- 1 / ps[treat == 1L]

  cap <- 1 / ps_floor
  truncated <- treat == 1L & is.finite(ps) & ps < ps_floor
  w_trunc <- w_raw
  w_trunc[truncated] <- cap

  if (stabilize) {
    w_trunc <- w_trunc / mean(w_trunc[treat == 1L], na.rm = TRUE)
  }

  list(
    weights    = w_trunc,
    ps         = ps,
    ps_floor   = ps_floor,
    cap        = cap,
    truncated  = truncated
  )
}

assert_finite_weights <- function(w, name = "weights") {
  if (any(!is.finite(w))) {

```

```

    stop("Non-finite values detected in ", name, ".")
}
invisible(TRUE)
}

runtime_logger <- function(step_name, expr, notes = NA_character_) {
  start_time <- Sys.time()
  result <- eval(expr)
  end_time <- Sys.time()
  sec <- as.numeric(difftime(end_time, start_time, units = "secs"))
  row <- data.frame(
    step_name = step_name,
    seconds = sec,
    start_time = as.character(start_time),
    end_time = as.character(end_time),
    notes = notes,
    run_id = runtime_run_id,
    run_mode = RUN_MODE,
    n_subset = subset_n,
    stringsAsFactors = FALSE
  )
  runtime_log <- dplyr::bind_rows(runtime_log, row)
  result
}

make_context <- function(stage, component,
                        analysis_variant = NA_character_,
                        model_type = NA_character_,
                        group = NA_character_,
                        outcome = NA_character_,
                        imputation = NA_integer_,
                        batch = NA_integer_) {
  list(
    stage = stage,
    component = component,
    analysis_variant = analysis_variant,
    model_type = model_type,

```

```

group = group,
outcome = outcome,
imputation = imputation,
batch = batch
)
}

capture_warnings <- function(expr, context) {
  warn_rows <- list()
  stopifnot(all(c("stage", "component", "analysis_variant", "model_type",
    "group", "outcome", "imputation", "batch") %in% names(context)))
  ctx <- list(
    stage = as.character(context$stage),
    component = as.character(context$component),
    analysis_variant = as.character(context$analysis_variant),
    model_type = as.character(context$model_type),
    group = as.character(context$group),
    outcome = as.character(context$outcome),
    imputation = as.integer(context$imputation),
    batch = as.integer(context$batch)
  )
  val <- withCallingHandlers(
    expr,
    warning = function(w) {
      warn_rows <- append(warn_rows, list(data.frame(
        time = as.character(Sys.time()),
        stage = ctx$stage,
        component = ctx$component,
        analysis_variant = ctx$analysis_variant,
        model_type = ctx$model_type,
        group = ctx$group,
        outcome = ctx$outcome,
        imputation = ctx$imputation,
        batch = ctx$batch,
        message = conditionMessage(w),
        stringsAsFactors = FALSE
      )))
    }
  )
}

```

```

    invokeRestart("muffleWarning")
  }
)
warnings_df <- if (length(warn_rows)) {
  dplyr::bind_rows(warn_rows)
} else {
  data.frame(
    time = character(), stage = character(), component = character(),
    analysis_variant = character(), model_type = character(),
    group = character(), outcome = character(), imputation = integer(),
    batch = integer(), message = character(), stringsAsFactors = FALSE
  )
}
list(value = val, warnings = warnings_df)
}

append_warnings <- function(wlist) {
  stopifnot(is.data.frame(wlist) || is.list(wlist))
  df <- if (is.data.frame(wlist)) wlist else dplyr::bind_rows(wlist)
  stopifnot(is.data.frame(df))
  stopifnot("message" %in% names(df))
  info_rows <- df[grep("Number of logged events", df$message), , drop = FALSE]
  if (nrow(info_rows)) {
    mi_info_log <- dplyr::bind_rows(mi_info_log, info_rows)
  }
  df <- df[!grep("Number of logged events", df$message), , drop = FALSE]
  if (nrow(df) == 0L) return(invisible(FALSE))
  mi_warn_log <- dplyr::bind_rows(mi_warn_log, df)
  invisible(TRUE)
}

resolve_current_qmd <- function() {
  # Resolve qmd path robustly across render/knit working directories.
  candidates <- c(
    tryCatch(knitr::current_input(), error = function(e) NA_character_),
    tryCatch(knitr::current_input(dir = TRUE), error = function(e) NA_character_),
    here::here("Code Drafts", "ABG-VBG analysis 2025-12-11.qmd"),

```

```

"Code Drafts/ABG-VBG analysis 2025-12-11.qmd",
"ABG-VBG analysis 2025-12-11.qmd"
)
candidates <- candidates[!is.na(candidates) & nzchar(candidates)]
existing <- candidates[file.exists(candidates)]
if (!length(existing)) {
  stop("Could not resolve qmd source path for audit checks.")
}
normalizePath(existing[[1]], winslash = "/", mustWork = TRUE)
}

collect_warnings_from_list <- function(xlist) {
  warn_list <- lapply(xlist, function(x) {
    if (is.list(x) && !is.null(x$warnings)) x$warnings else NULL
  })
  warn_list <- warn_list[!vapply(warn_list, is.null, logical(1))]
  append_warnings(warn_list)
  invisible(TRUE)
}

append_outcome_diag <- function(df) {
  stopifnot(is.data.frame(df))
  if (nrow(df) == 0L) return(invisible(FALSE))
  mi_outcome_diag <- dplyr::bind_rows(mi_outcome_diag, df)
  invisible(TRUE)
}

summarize_warnings_log <- function(path = results_path("warnings_log.csv")) {
  warn_df <- read.csv(path, stringsAsFactors = FALSE)
  if (nrow(warn_df) == 0L) return(data.frame())
  warn_df |>
    dplyr::count(stage, component, analysis_variant, model_type, message, sort = TRUE)
}

write_csv_safely <- function(df, path, row_names = FALSE, required_cols = NULL) {
  run_id <- diag_run_id
  run_ts <- diag_run_ts

```

```

stopifnot(!is.null(df))
df <- as.data.frame(df)
# Normalize pillar/vctrs numeric classes to base numeric for CSV export
df <- dplyr::mutate(
  df,
  dplyr::across(
    where(~ inherits(.x, "pillar_num")),
    ~ suppressWarnings(as.numeric(.x))
  )
)
if (!is.null(required_cols)) {
  missing <- setdiff(required_cols, names(df))
  stopifnot(length(missing) == 0)
}
if (nrow(df) == 0L) {
  if (!("empty" %in% names(df))) df$empty <- logical(0)
}
df$run_id <- rep(run_id, nrow(df))
df$run_ts <- rep(run_ts, nrow(df))
utils::write.csv(df, path, row.names = row_names)
invisible(TRUE)
}

write_diag_lines <- function(lines, path) {
  run_id <- diag_run_id
  run_ts <- diag_run_ts
  header <- paste0("run_id: ", run_id, " run_ts: ", run_ts)
  writeLines(c(header, lines), con = path)
  invisible(TRUE)
}

droplevels_all <- function(df, vars = NULL) {
  if (is.null(vars)) {
    vars <- names(df)[vapply(df, is.factor, logical(1))]
  } else {
    vars <- intersect(vars, names(df))
    vars <- vars[vapply(df[,vars], is.factor, logical(1))]
  }
}

```

```

}

for (nm in vars) df[[nm]] <- droplevels(df[[nm]])
df
}

append_mem_snapshot <- function(stage_id, context_id = NA_character_, when = "pre") {
  g <- gc()
  cn <- colnames(g)
  used_col <- if ("used" %in% cn) "used" else cn[grepl("used", cn)][1]
  trig_col <- if ("gc trigger" %in% cn) "gc trigger" else cn[grepl("trigger", cn)][1]
  max_col <- if ("max used" %in% cn) "max used" else cn[grepl("max", cn)][1]
  n_used <- as.numeric(g["Ncells", used_col])
  v_used <- as.numeric(g["Vcells", used_col])
  v_trig <- as.numeric(g["Vcells", trig_col])
  v_max <- as.numeric(g["Vcells", max_col])
  mem_max <- tryCatch(mem.maxVSize(), error = function(e) NA_real_)
  row <- data.frame(
    run_id = diag_run_id,
    run_ts = diag_run_ts,
    stage_id = stage_id,
    context_id = context_id,
    when = when,
    time = as.character(Sys.time()),
    Ncells_used = n_used,
    Vcells_used = v_used,
    Vcells_gc_trigger = v_trig,
    Vcells_max_used = v_max,
    mem_max_vszie = mem_max,
    stringsAsFactors = FALSE
  )
  memory_snapshots <- dplyr::bind_rows(memory_snapshots, row)
  invisible(TRUE)
}

write_factor_levels_diag <- function(df, vars, context_id, file_prefix = "gbm_preflight_factor_levels") {
  key <- paste0(file_prefix, "::", context_id)
  if (key %in% factor_diag_written) {

```

```

    return(invisible(FALSE))
}
vars <- intersect(vars, names(df))
if (!length(vars)) {
  return(invisible(FALSE))
}
diag <- lapply(vars, function(v) {
  x <- df[[v]]
  if (!is.factor(x)) return(NULL)
  tab <- table(x, useNA = "ifany")
  n_unused <- sum(tab == 0)
  top <- utils::head(sort(tab, decreasing = TRUE), 5)
  data.frame(
    variable = v,
    class = class(x)[1],
    nlevels = nlevels(x),
    n_unused_levels = n_unused,
    top_levels = paste(names(top), as.integer(top), sep = "=", collapse = "; "),
    n_missing = sum(is.na(x)),
    context_id = context_id,
    stringsAsFactors = FALSE
  )
})
diag <- dplyr::bind_rows(diag)
if (nrow(diag) == 0L) return(invisible(FALSE))
out_path <- results_path(paste0(file_prefix, "_", context_id, ".csv"))
write_csv_safely(diag, out_path, row_names = FALSE)
factor_diag_written <- c(factor_diag_written, key)
invisible(TRUE)
}

log_design_dims <- function(df, covars, context_id, sample_n = 5000L) {
  covars <- intersect(covars, names(df))
  if (!length(covars)) return(invisible(FALSE))
  n_full <- nrow(df)
  n_samp <- min(sample_n, n_full)
  set.seed(20251206)
}

```

```

idx <- sample.int(n_full, n_samp)
df_s <- df[idx, covars, drop = FALSE]
fml <- stats::as.formula(paste("~", paste(covars, collapse = " + ")))
mm <- stats::model.matrix(fml, data = df_s)
n_factor <- sum(vapply(df_s, is.factor, logical(1)))
max_nlevels <- if (n_factor > 0) max(vapply(df_s[vapply(df_s, is.factor, logical(1))], nlevels, integer(1))) else 0L
row <- data.frame(
  run_id = diag_run_id,
  run_ts = diag_run_ts,
  context_id = context_id,
  nrow_full = n_full,
  nrow_sample = n_samp,
  ncol_model_matrix = ncol(mm),
  n_factor_vars = n_factor,
  max_nlevels_factor = max_nlevels,
  stringsAsFactors = FALSE
)
gbm_preflight_design_dims <- dplyr::bind_rows(gbm_preflight_design_dims, row)
invisible(TRUE)
}

gbm_preflight <- function(df, covars, context_id, sample_n = 50000L) {
  df <- droplevels_all(df)
  write_factor_levels_diag(df, covars, context_id, file_prefix = "gbm_preflight_factor_levels")
  covars <- intersect(covars, names(df))
  if (!length(covars)) return(invisible(FALSE))
  n_full <- nrow(df)
  n_samp <- min(sample_n, n_full)
  set.seed(20251206)
  idx <- sample.int(n_full, n_samp)
  df_s <- df[idx, covars, drop = FALSE]
  fml <- stats::as.formula(paste("~", paste(covars, collapse = " + ")))
  mm <- stats::model.matrix(fml, data = df_s)
  n_factor <- sum(vapply(df_s, is.factor, logical(1)))
  max_nlevels <- if (n_factor > 0) {
    max(vapply(df_s[vapply(df_s, is.factor, logical(1))], nlevels, integer(1)))
  } else 0L
}

```

```

est_bytes <- n_full * ncol(mm) * 8
row <- data.frame(
  run_id = diag_run_id,
  run_ts = diag_run_ts,
  context_id = context_id,
  nrow_full = n_full,
  nrow_sample = n_samp,
  ncol_model_matrix = ncol(mm),
  n_factor_vars = n_factor,
  max_nlevels_factor = max_nlevels,
  est_bytes = est_bytes,
  est_gb = est_bytes / 1024^3,
  stringsAsFactors = FALSE
)
gbm_preflight_design_dims <- dplyr::bind_rows(gbm_preflight_design_dims, row)
warn_msgs <- character()
if (is.finite(est_bytes) && est_bytes > GBM_MM_EST_BYTES_WARN) {
  warn_msgs <- c(warn_msgs, paste0("est_dense_mm_gb>", round(GBM_MM_EST_BYTES_WARN / 1024^3, 2)))
}
if (ncol(mm) > MAX_GBM_MM_COLS_WARN) {
  warn_msgs <- c(warn_msgs, paste0("mm_cols_warn>", MAX_GBM_MM_COLS_WARN))
}
if (max_nlevels > MAX_GBM_FACTOR_LEVELS_WARN) {
  warn_msgs <- c(warn_msgs, paste0("factor_levels_warn>", MAX_GBM_FACTOR_LEVELS_WARN))
}
if (length(warn_msgs)) {
  gbm_preflight_warnings <- dplyr::bind_rows(
    gbm_preflight_warnings,
    data.frame(
      run_id = diag_run_id,
      run_ts = diag_run_ts,
      context_id = context_id,
      nrow_full = n_full,
      ncol_model_matrix = ncol(mm),
      max_nlevels_factor = max_nlevels,
      est_gb = est_bytes / 1024^3,
      warning = paste(warn_msgs, collapse = ";"),
      stringsAsFactors = FALSE
    )
  )
}

```

```

        stringsAsFactors = FALSE
    )
}
}
if (ncol(mm) > MAX_GBM_MM_COLS_STOP || max_nlevels > MAX_GBM_FACTOR_LEVELS_STOP) {
  stop(
    "GBM preflight stop: context=", context_id,
    "; ncol_model_matrix=", ncol(mm),
    "; max_nlevels_factor=", max_nlevels,
    ". Consider collapsing factor levels or removing high-cardinality predictors."
  )
}
invisible(TRUE)
}

print_head <- function(df, n = 10, title = NULL) {
  if (!is.null(title)) message(title)
  if (is.null(df) || nrow(df) == 0L) {
    message("No rows to display.")
    return(invisible(FALSE))
  }
  print(utils::head(df, n))
  invisible(TRUE)
}

assert_is_df <- function(x, context = "") {
  if (is.null(x)) {
    stop(context, ": object is NULL")
  }
  if (!inherits(x, "data.frame")) {
    stop(context, ": expected data.frame/tibble; got class=",
        paste(class(x), collapse = ", "), " typeof=", typeof(x))
  }
  invisible(TRUE)
}

assert_has_cols <- function(df, cols, context = "") {

```

```

missing <- setdiff(cols, names(df))
if (length(missing)) {
  stop(context, ": missing required columns: ", paste(missing, collapse = ", "))
}
invisible(TRUE)
}

safe_nrow <- function(df, context = "") {
  assert_is_df(df, context)
  nr <- nrow(df)
  if (!is.numeric(nr) || length(nr) != 1L || is.na(nr)) {
    stop(context, ": nrow(df) returned invalid value: ",
         paste0(capture.output(str(nr)), collapse = " "))
  }
  as.integer(nr)
}

plot_drop_log <- function() {
  plot_drop_log_df
}

log_plot_drop <- function(plot_name, reason, n_dropped,
                           n_before = NA_integer_, n_after = NA_integer_,
                           stage = NA_character_, extra = NA_character_) {
  plot_drop_log_df <- dplyr::bind_rows(
    plot_drop_log(),
    data.frame(plot = plot_name, reason = reason, n_dropped = n_dropped,
               n_before = n_before, n_after = n_after,
               stage = stage, extra = extra,
               stringsAsFactors = FALSE)
  )
  if (identical(PLOT_DROP_POLICY, "stop")) {
    stop("Plot data dropped in ", plot_name, ":", reason)
  } else {
    warning("Plot data dropped in ", plot_name, ":", reason, call. = FALSE)
  }
}

```

```

fit_with_diagnostics <- function(fit_fun, context, prob_eps = PROB_EPS) {
  stopifnot(all(c("stage", "component", "analysis_variant", "model_type",
    "group", "outcome", "imputation", "batch") %in% names(context)))
  cap <- capture_warnings(
    tryCatch(fit_fun(), error = function(e) e),
    context = context
  )
  append_warnings(cap$warnings)
  fit <- cap$value
  warn_msgs <- if (is.data.frame(cap$warnings)) cap$warnings$message else character()
  warning_n <- if (is.data.frame(cap$warnings)) nrow(cap$warnings) else 0L
  top_warning <- if (warning_n) warn_msgs[1] else NA_character_

  if (inherits(fit, "error")) {
    diag <- data.frame(
      stage = as.character(context$stage),
      component = as.character(context$component),
      analysis_variant = as.character(context$analysis_variant),
      model_type = as.character(context$model_type),
      group = as.character(context$group),
      outcome = as.character(context$outcome),
      imputation = as.integer(context$imputation),
      n_used = NA_integer_, events = NA_integer_,
      converged = NA, iter = NA_integer_,
      sep_flag = NA, nonconv_flag = NA,
      min_phat = NA_real_, max_phat = NA_real_,
      warning_n = warning_n, top_warning = top_warning,
      error_message = conditionMessage(fit),
      stringsAsFactors = FALSE
    )
    return(list(fit = NULL, diag = diag, warnings = cap$warnings))
  }
  phat <- tryCatch(fitted(fit), error = function(e) NA_real_)
  min_phat <- if (all(is.na(phat))) NA_real_ else min(phat, na.rm = TRUE)
  max_phat <- if (all(is.na(phat))) NA_real_ else max(phat, na.rm = TRUE)
}

```

```

sep_flag <- FALSE
if (is.finite(min_phat) && min_phat < prob_eps) sep_flag <- TRUE
if (is.finite(max_phat) && max_phat > 1 - prob_eps) sep_flag <- TRUE
if (any(grepl("fitted probabilities numerically 0 or 1", warn_msgs, fixed = TRUE))) sep_flag <- TRUE

conv_val <- if (!is.null(fit$converged)) isTRUE(fit$converged) else NA
iter_val <- if (!is.null(fit$iter)) fit$iter else NA_integer_
nonconv_flag <- isFALSE(conv_val) || any(grepl("did not converge", warn_msgs, fixed = TRUE))

mf <- tryCatch(model.frame(fit), error = function(e) NULL)
y <- if (!is.null(mf)) tryCatch(model.response(mf), error = function(e) NULL) else NULL
n_used <- if (!is.null(y)) length(y) else NA_integer_
events <- if (!is.null(y) && is.numeric(y)) sum(y == 1, na.rm = TRUE) else NA_integer_

diag <- data.frame(
  stage = as.character(context$stage),
  component = as.character(context$component),
  analysis_variant = as.character(context$analysis_variant),
  model_type = as.character(context$model_type),
  group = as.character(context$group),
  outcome = as.character(context$outcome),
  imputation = as.integer(context$imputation),
  n_used = n_used, events = events,
  converged = conv_val, iter = iter_val,
  sep_flag = sep_flag, nonconv_flag = nonconv_flag,
  min_phat = min_phat, max_phat = max_phat,
  warning_n = warning_n, top_warning = top_warning,
  error_message = NA_character_,
  stringsAsFactors = FALSE
)
list(fit = fit, diag = diag, warnings = cap$warnings)
}

pooled_mi_vcov_check <- function(pooled) {
  if (is.null(pooled$variance)) stop("MIcombine object missing variance matrix.")
  V <- pooled$variance
  if (any(!is.finite(V))) stop("Non-finite pooled variance detected.")
}

```

```

within <- attr(pooled, "within")
between <- attr(pooled, "between")
if (!is.null(within) && any(!is.finite(within))) stop("Non-finite within variance.")
if (!is.null(between) && any(!is.finite(between))) stop("Non-finite between variance.")
invisible(TRUE)
}

save_diag_plot <- function(p, file, width = 8, height = 6, dpi = 200) {
  ggplot2::ggsave(filename = file, plot = p, width = width, height = height, dpi = dpi)
  invisible(file)
}

plot_registry_path <- results_path("plot_registry.csv")

read_plot_registry <- function() {
  if (file.exists(plot_registry_path)) {
    utils::read.csv(plot_registry_path, stringsAsFactors = FALSE)
  } else {
    data.frame(
      run_id = character(), run_ts = character(),
      plot_name = character(), fig_path = character(), md5 = character(),
      stringsAsFactors = FALSE
    )
  }
}

write_plot_registry <- function(df) {
  write_csv_safely(df, plot_registry_path, row_names = FALSE)
}

register_plot_file <- function(plot_name, file, run_id = diag_run_id, run_ts = diag_run_ts) {
  md5 <- as.character(tools::md5sum(file))
  reg <- read_plot_registry()
  dup_name <- nrow(reg) > 0 &&
    any(reg$run_id == run_id & reg$plot_name == plot_name, na.rm = TRUE)
  if (dup_name) {
    log_plot_drop(

```

```

plot_name,
"duplicate_figure",
n_dropped = 1,
n_before = NA_integer_,
n_after = NA_integer_,
stage = "plot_registry",
extra = file
)
return(FALSE)
}
reg <- dplyr::bind_rows(
  reg,
  data.frame(run_id = run_id, run_ts = run_ts,
             plot_name = plot_name, fig_path = file, md5 = md5,
             stringsAsFactors = FALSE)
)
write_plot_registry(reg)
TRUE
}

print_plot_once <- function(p, plot_name, width = 8, height = 6, dpi = 200) {
  file <- results_path("figs", paste0(plot_name, ".png"))
  save_diag_plot(p, file, width = width, height = height, dpi = dpi)
  if (register_plot_file(plot_name, file)) {
    mark_float_emitted()
    knitr::include_graphics(file)
  } else {
    invisible(NULL)
  }
}

print_plot_force <- function(p, plot_name, width = 8, height = 6, dpi = 200) {
  # Use this for key publication figures that must always appear in the PDF.
  # We still register file metadata, but we never suppress the on-page display.
  file <- results_path("figs", paste0(plot_name, ".png"))
  save_diag_plot(p, file, width = width, height = height, dpi = dpi)
  register_plot_file(plot_name, file)
}

```

```

mark_float_emitted()
knitr::include_graphics(file)
}

term_to_parent_feature <- function(term, feature_names) {
  if (is.na(term) || identical(term, "(Intercept)")) return(NA_character_)
  clean_term <- gsub(``, "", term, fixed = TRUE)
  feature_names <- sort(unique(feature_names), decreasing = TRUE)
  hit <- feature_names[vapply(feature_names, function(v) grepl(v, clean_term, fixed = TRUE), logical(1))]
  if (length(hit)) return(hit[[1]])
  NA_character_
}

sample_rows_for_shap <- function(df, n_max = SHAP_SAMPLE_N, seed = 20251206L) {
  stopifnot(is.data.frame(df) || is.matrix(df))
  if (nrow(df) <= n_max) return(df)
  set.seed(seed)
  df[sample.int(nrow(df), size = n_max, replace = FALSE), , drop = FALSE]
}

fit_weightit_gbm_for_shap <- function(formula_obj, data_obj) {
  args <- c(
    list(
      formula = formula_obj,
      data = data_obj,
      method = "gbm",
      estimand = "ATE",
      missing = "ind",
      include.obj = TRUE
    ),
    gbm_params
  )
  do.call(WeightIt::weightit, args)
}

extract_nonmi_gbm_shap_top <- function(weight_obj, x_df, feature_names, group_label,
                                         nsim = SHAP_NSIM, top_n = SHAP_TOP_N) {

```

```

if (!HAS_FASTSHAP) {
  return(
    data.frame(
      group = character(),
      feature = character(),
      mean_abs_shap = numeric(),
      stringsAsFactors = FALSE
    )
  )
}

stopifnot(inherits(weight_obj, "weightit"))
stopifnot(!is.null(weight_obj$obj))
gbm_obj <- weight_obj$obj
best_tree <- weight_obj$info$best.tree
if (is.null(best_tree) || !is.finite(best_tree) || best_tree <= 0) {
  best_tree <- gbm_obj$n.trees
}
x_use <- x_df[, feature_names, drop = FALSE]
char_cols <- names(x_use)[vapply(x_use, is.character, logical(1))]
if (length(char_cols)) {
  for (nm in char_cols) x_use[[nm]] <- factor(x_use[[nm]])
}
x_use <- droplevels_all(x_use)
x_background <- as.data.frame(
  sample_rows_for_shap(x_use, n_max = max(SHAP_SAMPLE_N, 5000L), seed = 20251206L)
)
x_shap <- as.data.frame(
  sample_rows_for_shap(x_use, n_max = SHAP_SAMPLE_N, seed = 20251206L)
)

pred_fun <- function(object, newdata) {
  new_use <- newdata[, object$var.names, drop = FALSE]
  as.numeric(stats::predict(object, newdata = new_use, n.trees = best_tree, type = "link"))
}

shap_mat <- tryCatch(
  fastshap::explain(

```

```

object = gbm_obj,
feature_names = feature_names,
X = x_background,
newdata = x_shap,
pred_wrapper = pred_fun,
nsim = nsim,
adjust = FALSE
),
error = function(e) {
warning(
  "fastshap failed for ", group_label,
  "; falling back to GBM relative influence. ",
  conditionMessage(e),
  call. = FALSE
)
NULL
}
)

if (is.null(shap_mat)) {
  gbm_imp <- tryCatch(
    gbm::summary.gbm(gbm_obj, plotit = FALSE),
    error = function(e) data.frame(var = character(), rel.inf = numeric(), stringsAsFactors = FALSE)
  )
  if (!nrow(gbm_imp)) {
    return(data.frame(group = character(), feature = character(), mean_abs_shap = numeric(), stringsAsFactors =
      FALSE))
  }
  return(
    gbm_imp |>
      dplyr::transmute(
        group = group_label,
        feature = as.character(var),
        mean_abs_shap = as.numeric(rel.inf) / 100
      ) |>
      dplyr::arrange(dplyr::desc(mean_abs_shap)) |>
      dplyr::slice_head(n = top_n)
  )
}

```

```

)
}

out <- data.frame(
  group = group_label,
  feature = colnames(shap_mat),
  mean_abs_shap = colMeans(abs(shap_mat), na.rm = TRUE),
  stringsAsFactors = FALSE
) |>
  dplyr::arrange(dplyr::desc(mean_abs_shap)) |>
  dplyr::slice_head(n = top_n)
out
}

plot_shap_top10_two_panel <- function(df_top, title_text, x_label) {
  # Keep only top-N per cohort and force descending order inside each panel.
  df_top_plot <- df_top |>
    dplyr::group_by(group) |>
    dplyr::arrange(dplyr::desc(mean_abs_shap), .by_group = TRUE) |>
    dplyr::slice_head(n = SHAP_TOP_N) |>
    dplyr::mutate(
      feature_rank = dplyr::row_number(),
      feature_plot = paste0(sprintf("%02d", feature_rank), ". ", feature),
      group = factor(group, levels = c("ABG", "VBG"))
    ) |>
    dplyr::ungroup()

  # Prefixing with rank makes ordering deterministic in each facet.
  ggplot2::ggplot(
    df_top_plot,
    ggplot2::aes(
      x = mean_abs_shap,
      y = stats::reorder(feature_plot, mean_abs_shap),
      fill = group
    )
  ) +
  ggplot2::geom_col(show.legend = FALSE) +

```

```

ggplot2::facet_wrap(~group, scales = "free_y", ncol = 2) +
  ggplot2::labs(
    title = title_text,
    x = x_label,
    y = NULL
  ) +
  ggplot2::scale_y_discrete(labels = function(x) sub("^[0-9]+\\.\\s*", "", x)) +
  ggplot2::theme_minimal(base_size = 10)
}

add_overall_love_rows <- function(df_in, overall_label = "Overall (mean |SMD|)") {
  # Summarize whole-plot balance for each cohort/sample pair and display it
  # as the top row so readers can quickly judge global balance shift.
  overall <- df_in |>
    dplyr::group_by(group, sample) |>
    dplyr::summarise(
      abs_smd = mean(abs_smd, na.rm = TRUE),
      abs_smd_raw = mean(abs_smd_raw, na.rm = TRUE),
      .groups = "drop"
    ) |>
    dplyr::mutate(term = overall_label)
  dplyr::bind_rows(overall, df_in)
}

loveplot_style <- function(df_in, title_text) {
  # Build a classic love-plot view with:
  # 1) a segment from Raw -> IPW for each term, and
  # 2) explicit Raw/IPW points so both samples are always visible.
  # This avoids overplotting where one sample can hide the other.
  df_plot <- df_in |>
    dplyr::mutate(
      sample = factor(sample, levels = c("Raw", "IPW")),
      order_val = dplyr::if_else(
        term == "Overall (mean |SMD|)",
        max(abs_smd_raw, na.rm = TRUE) + 1,
        abs_smd_raw
      )
    )
}

```

```

)

df_wide <- df_plot |>
  dplyr::select(group, term, order_val, sample, abs_smd) |>
  tidyr::pivot_wider(names_from = sample, values_from = abs_smd)

ggplot2::ggplot(df_plot, ggplot2::aes(y = stats::reorder(term, order_val))) +
  ggplot2::geom_segment(
    data = df_wide,
    ggplot2::aes(
      x = Raw,
      xend = IPW,
      y = stats::reorder(term, order_val),
      yend = stats::reorder(term, order_val)
    ),
    inherit.aes = FALSE,
    color = "grey65",
    linewidth = 0.3,
    alpha = 0.8
  ) +
  ggplot2::geom_point(
    data = df_plot |> dplyr::filter(sample == "Raw"),
    ggplot2::aes(x = abs_smd, color = sample, shape = sample, size = term == "Overall (mean |SMD|)", alpha = 0.95
  ) +
  ggplot2::geom_point(
    data = df_plot |> dplyr::filter(sample == "IPW"),
    ggplot2::aes(x = abs_smd, color = sample, shape = sample, size = term == "Overall (mean |SMD|)", alpha = 0.95
  ) +
  ggplot2::geom_vline(xintercept = 0.10, linetype = "dashed", color = "grey30") +
  ggplot2::facet_wrap(~group, ncol = 2) +
  ggplot2::scale_color_manual(values = c("Raw" = "#F8766D", "IPW" = "#00BFC4")) +
  ggplot2::scale_shape_manual(values = c("Raw" = 16, "IPW" = 17)) +
  ggplot2::scale_size_manual(values = c(`TRUE` = 3.0, `FALSE` = 1.9), guide = "none") +
  ggplot2::labs(
    title = title_text,

```

```

    x = "Absolute Standardized Mean Differences",
    y = NULL,
    color = "Sample"
) +
ggplot2::theme_minimal(base_size = 10) +
ggplot2::theme(legend.position = "top")
}

CO2_RATE_OUTCOMES <- c("imv_proc", "niv_proc", "death_60d", "hypercap_resp_failure")
CO2_RATE_OUTCOME_LABELS <- c(
  imv_proc = "IMV",
  niv_proc = "NIV",
  death_60d = "Death (60d)",
  hypercap_resp_failure = "Hypercapnic RF"
)

compute_co2_bin_rates <- function(df, co2_var, outcomes, outcome_labels,
                                    weight_var = NULL, q = c(0.02, 0.98),
                                    bin_width = 1, cohort_label = NA_character_) {
  assert_is_df(df, context = paste0("compute_co2_bin_rates(", co2_var, ")"))
  stopifnot(co2_var %in% names(df))
  stopifnot(all(outcomes %in% names(df)))
  stopifnot(length(q) == 2L, q[1] < q[2], is.finite(bin_width), bin_width > 0)
  if (!is.null(weight_var)) stopifnot(weight_var %in% names(df))

  co2_vals <- suppressWarnings(as.numeric(df[[co2_var]]))
  keep <- is.finite(co2_vals)
  if (!any(keep)) {
    return(tibble::tibble(
      cohort = character(), outcome = character(), outcome_label = character(),
      bin_lo = numeric(), bin_mid = numeric(), n_bin = integer(), rate = numeric()
    ))
  }

  q_lohi <- stats::quantile(co2_vals[keep], probs = q, na.rm = TRUE, names = FALSE)
  keep <- keep & co2_vals >= q_lohi[1] & co2_vals <= q_lohi[2]
  if (!is.null(weight_var)) {

```

```

w <- suppressWarnings(as.numeric(df[[weight_var]]))
keep <- keep & is.finite(w) & !is.na(w) & (w > 0)
}
if (!any(keep)) {
  return(tibble::tibble(
    cohort = character(), outcome = character(), outcome_label = character(),
    bin_lo = numeric(), bin_mid = numeric(), n_bin = integer(), rate = numeric()
  ))
}

d <- df[keep, , drop = FALSE]
d$co2_work <- co2_vals[keep]
d$bin_lo <- floor(d$co2_work / bin_width) * bin_width
d$bin_mid <- d$bin_lo + (bin_width / 2)
if (!is.null(weight_var)) {
  d$w_work <- suppressWarnings(as.numeric(d[[weight_var]]))
}

rate_rows <- lapply(outcomes, function(outcome_nm) {
  y <- suppressWarnings(as.numeric(d[[outcome_nm]] == 1))
  y[is.na(d[[outcome_nm]])] <- NA_real_
  tmp <- d[, c("bin_lo", "bin_mid"), drop = FALSE]
  tmp$y <- y
  if (!is.null(weight_var)) tmp$w <- d$w_work
  if (!is.null(weight_var)) {
    tmp <- tmp[is.finite(tmp$w) & !is.na(tmp$w) & !is.na(tmp$y), , drop = FALSE]
  } else {
    tmp <- tmp[!is.na(tmp$y), , drop = FALSE]
  }
  if (!nrow(tmp)) return(NULL)

  if (!is.null(weight_var)) {
    out <- tmp |>
      dplyr::group_by(bin_lo, bin_mid) |>
      dplyr::summarise(
        n_bin = dplyr::n(),
        rate = ifelse(sum(w, na.rm = TRUE) > 0, sum(y * w, na.rm = TRUE) / sum(w, na.rm = TRUE), NA_real_),

```

```

        .groups = "drop"
    )
} else {
  out <- tmp |>
    dplyr::group_by(bin_lo, bin_mid) |>
    dplyr::summarise(
      n_bin = dplyr::n(),
      rate = mean(y, na.rm = TRUE),
      .groups = "drop"
    )
}

out |>
  dplyr::mutate(
    cohort = cohort_label,
    outcome = outcome_nm,
    outcome_label = dplyr::if_else(
      outcome_nm %in% names(outcome_labels),
      as.character(outcome_labels[[outcome_nm]]),
      outcome_nm
    )
  ) |>
  dplyr::select(cohort, outcome, outcome_label, bin_lo, bin_mid, n_bin, rate)
})

dplyr::bind_rows(rate_rows)
}

pool_mi_bin_rates <- function(rate_df_all_imps) {
  assert_is_df(rate_df_all_imps, context = "pool_mi_bin_rates")
  stopifnot("imp" %in% names(rate_df_all_imps))
  rate_df_all_imps |>
    dplyr::group_by(cohort, outcome, outcome_label, bin_lo, bin_mid) |>
    dplyr::summarise(
      n_bin = mean(n_bin, na.rm = TRUE),
      rate = mean(rate, na.rm = TRUE),
      n_imp_contrib = sum(is.finite(rate)),

```

```

    .groups = "drop"
)
}

plot_co2_bin_rate_panel <- function(rate_df, title, x_label, vline_at) {
  assert_is_df(rate_df, context = "plot_co2_bin_rate_panel")
  stopifnot(all(c("bin_mid", "n_bin", "rate", "outcome_label") %in% names(rate_df)))
  lbl_levels <- unname(CO2_RATE_OUTCOME_LABELS[CO2_RATE_OUTCOMES])
  rate_df$outcome_label <- factor(rate_df$outcome_label, levels = lbl_levels)

  ggplot2::ggplot(rate_df, ggplot2::aes(x = bin_mid, y = rate)) +
    ggplot2::geom_point(color = "#2C7FB8", alpha = 0.8, size = 1.2) +
    ggplot2::geom_smooth(
      ggplot2::aes(weight = n_bin),
      method = "loess", span = 0.5, se = FALSE,
      color = "#D95FOE", linewidth = 0.8
    ) +
    ggplot2::geom_vline(xintercept = vline_at, linetype = "dashed", color = "grey40") +
    ggplot2::facet_wrap(~ outcome_label, ncol = 2, scales = "free_y") +
    ggplot2::scale_y_continuous(labels = scales::percent_format(accuracy = 1)) +
    ggplot2::labs(
      title = title,
      x = x_label,
      y = "Observed outcome rate",
      caption = paste0("Points are 1-mmHg bins; LOESS span = 0.5; dashed line at ", vline_at, " mmHg.")
    ) +
    ggplot2::theme_minimal(base_size = 10)
}

compute_or_axis_spec <- function(df_list, lo_col = "conf.low", hi_col = "conf.high",
                                  min_pow = -6, max_pow = 6,
                                  default_limits = c(0.25, 16)) {
  if (inherits(df_list, "data.frame")) df_list <- list(df_list)
  if (!is.list(df_list)) df_list <- list(df_list)
  vals <- unlist(lapply(df_list, function(df) {
    assert_is_df(df, context = "compute_or_axis_spec")
    stopifnot(all(c(lo_col, hi_col) %in% names(df)))
  }))
}

```

```

lo <- df[[lo_col]]
hi <- df[[hi_col]]
c(lo, hi)
}), use.names = FALSE)
vals <- vals[is.finite(vals) & vals > 0]
if (!length(vals)) {
  lo_pow <- floor(log2(default_limits[1]))
  hi_pow <- ceiling(log2(default_limits[2]))
} else {
  lo_pow <- floor(log2(min(vals)))
  hi_pow <- ceiling(log2(max(vals)))
}
lo_pow <- max(lo_pow, min_pow)
hi_pow <- min(hi_pow, max_pow)
if (lo_pow > hi_pow) hi_pow <- lo_pow
limits <- 2^c(lo_pow, hi_pow)
breaks <- 2^(lo_pow:hi_pow)
list(limits = limits, breaks = breaks)
}

or_axis_scale <- function(spec) {
  list(
    ggplot2::scale_y_log10(
      breaks = spec$breaks,
      labels = scales::number_format(accuracy = 0.01)
    ),
    ggplot2::coord_cartesian(ylim = spec$limits)
  )
}

map_or_exposure <- function(df, plot_name) {
  assert_is_df(df, context = paste0("map_or_exposure(", plot_name, ")"))
  stopifnot(nrow(df) > 0L)
  stopifnot("term" %in% names(df))
  df$term <- gsub("^", "", df$term)
  df$exposure <- dplyr::case_when(
    grepl("^pc02_cat_abg", df$term) ~ gsub("^pc02_cat_abg", "", df$term),

```

```

grepl("^\$co2_cat_vbg", df$term) ~ gsub("^\$co2_cat_vbg", "", df$term),
grepl("^\$co2_cat", df$term) ~ gsub("^\$co2_cat", "", df$term),
TRUE ~ NA_character_
)
bad <- is.na(df$exposure) | !df$exposure %in% CO2_CAT_CONTRAST_LEVELS
if (any(bad)) {
  out_path <- results_path("or_term_mapping_failures.csv")
  bad_terms <- data.frame(term = df$term[bad], group = df$group[bad],
                           plot_name = plot_name, stringsAsFactors = FALSE)
  write_csv_safely(bad_terms, out_path, row_names = FALSE)
  stop(paste0("Unmapped OR terms in ", plot_name, ". See ", out_path))
}
df$exposure <- factor(df$exposure, levels = CO2_CAT_CONTRAST_LEVELS)
df
}

build_or_plot_df <- function(df, plot_name, expected_exposure_levels) {
  df <- tibble::as_tibble(df)
  req_cols <- c("outcome", "group", "estimate", "conf.low", "conf.high")
  req_cols <- c(req_cols, "exposure")
  assert_has_cols(df, unique(req_cols), context = paste0("build_or_plot_df(", plot_name, ")"))
  df <- dplyr::mutate(
    df,
    estimate = as.numeric(estimate),
    conf.low = as.numeric(conf.low),
    conf.high = as.numeric(conf.high)
  )
  if (safe_nrow(df, paste0("build_or_plot_df(", plot_name, ")")) == 0L) {
    stop("Empty OR plot data in ", plot_name)
  }
  drop <- !is.finite(df$estimate) | !is.finite(df$conf.low) | !is.finite(df$conf.high)
  if (any(drop)) {
    log_plot_drop(plot_name, "non-finite estimate/conf", sum(drop),
                  n_before = nrow(df), n_after = nrow(df) - sum(drop),
                  stage = "build_or_plot_df")
    df <- df[!drop, , drop = FALSE]
  }
}

```

```

drop_pos <- (df$estimate <= 0) | (df$conf.low <= 0) | (df$conf.high <= 0)
if (any(drop_pos, na.rm = TRUE)) {
  log_plot_drop(plot_name, "non-positive estimate/conf", sum(drop_pos, na.rm = TRUE),
                n_before = nrow(df), n_after = nrow(df) - sum(drop_pos, na.rm = TRUE),
                stage = "build_or_plot_df")
  df <- df[!drop_pos, , drop = FALSE]
}
exp_drop <- is.na(df$exposure) | trimws(as.character(df$exposure)) == ""
if (any(exp_drop)) {
  log_plot_drop(plot_name, "missing exposure level", sum(exp_drop),
                n_before = nrow(df), n_after = nrow(df) - sum(exp_drop),
                stage = "build_or_plot_df")
  df <- df[!exp_drop, , drop = FALSE]
}
df$exposure <- factor(df$exposure, levels = expected_exposure_levels)
bad_exp <- is.na(df$exposure)
if (any(bad_exp)) {
  bad_df <- df[bad_exp, , drop = FALSE]
  bad_path <- results_path(paste0("or_unmapped_terms_", plot_name, ".csv"))
  write_csv_safely(bad_df, bad_path, row_names = FALSE)
  stop(paste0("Unmapped exposure terms in ", plot_name, ". See ", bad_path))
}
missing_levels <- setdiff(expected_exposure_levels, unique(as.character(df$exposure)))
if (length(missing_levels)) {
  stop(paste0("Missing exposure levels in ", plot_name, ":", paste(missing_levels, collapse = ", ")))
}
if (nrow(df) == 0L) {
  msg <- paste0("OR plot data empty after filtering in ", plot_name)
  stop(msg)
}
df
}

plot_or_safe <- function(df, plot_name, axis_spec,
                         color_var = "group", shape_var = "exposure",
                         facet_var = NULL, title = NULL, caption = NULL) {
  df <- tibble::as_tibble(df)

```

```

assert_is_df(df, context = paste0("plot_or_safe()", plot_name, ""))
nr <- safe_nrow(df, paste0("plot_or_safe()", plot_name, ""))
if (nr == 0L) {
  stop("plot_or_safe()", plot_name, "): no rows to plot.")
}
assert_has_cols(df, c("outcome", "group", "exposure", "estimate", "conf.low", "conf.high"),
                context = paste0("plot_or_safe()", plot_name, ""))
if (isTRUEgetOption("ABG_VBG_DEBUG", FALSE)) {
  message("plot_or_safe()", plot_name, "): class=", paste(class(df), collapse = ", "),
          " nrow=", nr, " ncol=", ncol(df))
}
stopifnot(color_var %in% names(df))
stopifnot(shape_var %in% names(df))
stopifnot(!is.null(axis_spec))
df[[shape_var]] <- as.factor(df[[shape_var]])
n_shape <- dplyr::n_distinct(df[[shape_var]])
if (n_shape > MAX_SHAPE_LEVELS) {
  message("OR plot: too many shape levels (", n_shape, ") in ", plot_name,
         ". Dropping shape and faceting by ", shape_var, ".")
  facet_var <- shape_var
  shape_var <- NULL
}

df$.grp <- 1L
if (!is.null(color_var) && !is.null(shape_var)) {
  df$.grp <- interaction(df[[color_var]], df[[shape_var]], drop = TRUE)
} else if (!is.null(color_var)) {
  df$.grp <- df[[color_var]]
} else if (!is.null(shape_var)) {
  df$.grp <- df[[shape_var]]
}

p <- ggplot2::ggplot(df, ggplot2::aes(x = outcome, y = estimate, ymin = conf.low, ymax = conf.high, group = .grp))
if (!is.null(color_var)) {
  p <- p + ggplot2::aes(color = .data[[color_var]])
}
if (!is.null(shape_var)) {

```

```

    p <- p + ggplot2::aes(shape = .data[[shape_var]])
}

p <- p +
  ggplot2::geom_pointrange(position = ggplot2::position_dodge(width = 0.7), size = 0.6) +
  ggplot2::geom_hline(yintercept = 1, linetype = "dashed", colour = "grey40") +
  or_axis_scale(axis_spec) +
  ggplot2::labs(
    title = title,
    x = "Outcome",
    y = "Odds Ratio (log scale, 95% CI)",
    color = if (!is.null(color_var)) "Blood-gas type" else NULL,
    shape = if (!is.null(shape_var)) "PCO2 category" else NULL,
    caption = caption
  ) +
  ggplot2::theme_minimal(base_size = 9) +
  ggplot2::theme(
    plot.caption = ggplot2::element_text(hjust = 0),
    axis.text.x = ggplot2::element_text(angle = 15, hjust = 1),
    legend.position = "bottom",
    plot.margin = ggplot2::margin(6, 6, 6, 6)
  )
if (!is.null(color_var) || !is.null(shape_var)) {
  p <- p + ggplot2::guides(
    color = ggplot2::guide_legend(nrow = 2, byrow = TRUE),
    shape = ggplot2::guide_legend(nrow = 1, byrow = TRUE)
  )
}

if (!is.null(facet_var)) {
  p <- p + ggplot2::facet_wrap(stats::as.formula(paste("~", facet_var)))
}
p
}

# Purpose: preflight scan for very long source lines that can stress LaTeX code-block rendering.
qmd_src_scan <- resolve_current_qmd()

```

```

qmd_lines <- readLines(qmd_src_scan, warn = FALSE)
risk_threshold <- 220L
line_nchars <- nchar(qmd_lines, type = "chars", allowNA = FALSE, keepNA = FALSE)
risk_idx <- which(line_nchars > risk_threshold)
risk_df <- if (length(risk_idx)) {
  data.frame(
    line_no = risk_idx,
    n_chars = line_nchars[risk_idx],
    line_preview = substr(qmd_lines[risk_idx], 1, 200),
    stringsAsFactors = FALSE
  )
} else {
  data.frame(
    line_no = integer(),
    n_chars = integer(),
    line_preview = character(),
    stringsAsFactors = FALSE
  )
}
write_csv_safely(risk_df, results_path("latex_codeblock_risk_scan.csv"), row_names = FALSE)

# Purpose: setup model diagrams.
library(dagitty)
library(ggdag)

# Toggle to show diagrams inline in the PDF
show_model_diagrams <- FALSE

model_diagram_dir <- fs::path(results_dir, "figs", "model-diagrams")
fs::dir_create(model_diagram_dir)

.make_safe_name <- function(x) {
  x <- gsub("[^A-Za-z0-9]+", "-", x)
  x <- gsub("(^-|-$)", "", x)
  tolower(x)
}

```

```

.extract_model_vars <- function(fml) {
  tt <- stats::terms(fml)
  response <- all.vars(stats::update(fml, . ~ 1))
  response <- if (length(response)) response[1] else NA_character_
  term_labels <- attr(tt, "term.labels")
  preds <- unique(unlist(lapply(term_labels, function(t) {
    all.vars(stats::as.formula(paste("~", t)))
  })))
  preds <- setdiff(preds, response)
  list(response = response, preds = preds)
}

.build_model_dag <- function(fml) {
  vars <- .extract_model_vars(fml)
  if (is.na(vars$response) || length(vars$preds) == 0L) {
    stop("Model diagram: formula has no response or predictors: ", deparse(fml))
  }
  edges <- paste(sprintf("%s -> %s", vars$preds, vars$response), collapse = "\n ")
  dagitty::dagitty(sprintf("dag { %s }", edges))
}

save_model_diagram <- function(name, fml, width = NULL, height = NULL) {
  dag <- .build_model_dag(fml)

  vars <- .extract_model_vars(fml)
  n_pred <- length(vars$preds)
  n_nodes <- n_pred + 1L

  label_size <- if (n_pred >= 35) 1.4 else if (n_pred >= 25) 1.8 else if (n_pred >= 15) 2.2 else 2.8
  point_size <- if (n_pred >= 25) 2.0 else if (n_pred >= 15) 2.5 else 3.0

  base_width <- if (is.null(width)) 8 else width
  base_height <- if (is.null(height)) 6 else height
  width <- max(base_width, min(24, 0.4 * n_nodes + 4))
  height <- max(base_height, min(16, 0.25 * n_nodes + 4))

  layout <- if (n_pred >= 8) "sugiyama" else "nicely"
}

```

```

dag_data <- ggdag::tidy_dagitty(dag, layout = layout)$data

p <- ggplot2::ggplot(
  dag_data,
  ggplot2::aes(x = x, y = y, xend = xend, yend = yend)
) +
  ggdag::geom_dag_edges() +
  ggdag::geom_dag_point(size = point_size) +
  ggdag::geom_dag_text_repel(
    ggplot2::aes(label = name),
    size = label_size,
    max.overlaps = Inf
  ) +
  ggplot2::scale_x_continuous(expand = ggplot2::expansion(mult = 0.15)) +
  ggplot2::scale_y_continuous(expand = ggplot2::expansion(mult = 0.15)) +
  ggplot2::theme_void() +
  ggplot2::labs(title = name) +
  ggplot2::theme(
    plot.title = ggplot2::element_text(size = 10, hjust = 0.5),
    plot.margin = ggplot2::margin(10, 10, 10, 10)
  )
)

file <- fs::path(model_diagram_dir, paste0(.make_safe_name(name), ".png"))
ggplot2::ggsave(file, p, width = width, height = height, dpi = 200)
if (isTRUE(show_model_diagrams)) print(p)
file
}

model_diagrams <- list()
register_model_diagram <- function(name, fml, width = 8, height = 6) {
  file <- save_model_diagram(name, fml, width = width, height = height)
  model_diagrams[[name]] <- file
  invisible(file)
}

register_model_diagrams <- function(forms, width = 8, height = 6) {
  purrr::iwalk(forms, function(fml, nm) {

```

```

    register_model_diagram(nm, fml, width = width, height = height)
  })
}

# Purpose: setup shapviz.
# Keep SHAP enabled in both pilot and full so the report has figure parity
# across IPSW sections. Sampling controls below limit runtime/memory.

RUN_SHAP <- TRUE
SHAP_TOP_N <- 10L
SHAP_SAMPLE_N <- 2000L
SHAP_NSIM <- 20L
MI_SHAP_SAMPLE_N_FULL <- 400L
MI_SHAP_SAMPLE_N_PILOT <- 1000L
MI_SHAP_SAMPLE_N <- if (RUN_MODE == "full") MI_SHAP_SAMPLE_N_FULL else MI_SHAP_SAMPLE_N_PILOT
MI_SHAP_MAX_TERMS <- 50000L
MI_SHAP_FAIL_OPEN <- TRUE

HAS_FASTSHAP <- requireNamespace("fastshap", quietly = TRUE)
if (RUN_SHAP && !HAS_FASTSHAP) {
  warning(
    "fastshap is not installed; SHAP figures will be skipped. ",
    "Install with renv::install('fastshap').",
    call. = FALSE
  )
}

# Chunk runtime annotation (printed into PDF) - disabled by default
if (isTRUE(SHOW_CHUNK_RUNTIME_TEXT)) {
  knitr::knit_hooks$set(runtimelog = local{
    starts <- list()
    escape_latex <- function(x) {
      gsub("(\\\\\\\$\\{\\}_\\^~)", "\\\\\\1", x, perl = TRUE)
    }
    function(before, options) {
      if (before) {
        starts[[options$label]] <- proc.time()
      } else {
        st <- starts[[options$label]]

```

```

    if (is.null(st)) return(NULL)
    elapsed <- (proc.time() - st)[["elapsed"]]
    lbl <- escape_latex(options$label)
    paste0(
      "\n\n",
      "\\textit{Chunk ", lbl, " runtime: ", sprintf("%.2f", elapsed), " s}",
      "\n\n"
    )
  }
}
})))
knitr::opts_chunk$set(runtimelog = TRUE)
} else {
  knitr::opts_chunk$set(runtimelog = FALSE)
}

# Purpose: seed escrow.
seed_escrow <- data.frame(
  component = c(
    "Multiple imputation (mice)",
    "ABG propensity GBM (non-MI)",
    "VBG propensity GBM (non-MI)",
    "MI PS (glm RCS) seeds (ABG per imputation)",
    "MI PS (glm RCS) seeds (VBG per imputation)"
  ),
  seed = c(
    "20251206",
    "42",
    "42",
    "20251206 + imputation index",
    "30251206 + imputation index"
  ),
  stringsAsFactors = FALSE
)
render_table_pdf(
  seed_escrow,
  caption = "Seed escrow for MI and GBM runs",

```

```

file_stub = "seed_escrow",
digits = 0
)

```

Table 1: Seed escrow for MI and GBM runs

component	seed
Multiple imputation (mice)	20251206
ABG propensity GBM (non-MI)	42
VBG propensity GBM (non-MI)	42
MI PS (glm RCS) seeds (ABG per imputation)	20251206 + imputation index
MI PS (glm RCS) seeds (VBG per imputation)	30251206 + imputation index

```

DEBUG_SPLINE <- FALSE

# Make gt tables robust in PDF: full width, caption, small font
gt_pdf <- function(x, title = NULL, subtitle = NULL) {
  out <- x |>
    gt::tab_options(
      table.width          = gt::pct(100),
      table.align           = "left",
      table.font.size       = gt::px(9),
      data_row.padding      = gt::px(1),
      column_labels.font.size = gt::px(9),
      heading.title.font.size = gt::px(10),
      heading.subtitle.font.size = gt::px(9)
    ) |>
    gt::opt_align_table_header(align = "left")
  if (!is.null(title))   out <- out |> gt::tab_caption(title)
  if (!is.null(subtitle)) out <- out |> gt::tab_source_note(subtitle)
  out
}

```

Converts the data from a STATA format to rdata if the rdata file does not exist. If it does already exist, it just loads that.

```

# Purpose: stage1 start.
append_mem_snapshot("stage1", "start", "pre")

# data_dir_name resolved in setup-packages from params/env

```

```

stata_file <- file.path(data_dir_name, "full_db.dta")

stata_data <- read_dta(stata_file)

# Sanitize variable labels for PDF/TOC safety (avoid Unicode subscripts)
sanitize_label_text <- function(x) {
  if (is.null(x)) return(x)
  x <- gsub("\u2080", "0", x, fixed = TRUE)
  x <- gsub("\u2081", "1", x, fixed = TRUE)
  x <- gsub("\u2082", "2", x, fixed = TRUE)
  x <- gsub("\u2083", "3", x, fixed = TRUE)
  x <- gsub("\u2084", "4", x, fixed = TRUE)
  x <- gsub("\u2085", "5", x, fixed = TRUE)
  x <- gsub("\u2086", "6", x, fixed = TRUE)
  x <- gsub("\u2087", "7", x, fixed = TRUE)
  x <- gsub("\u2088", "8", x, fixed = TRUE)
  x <- gsub("\u2089", "9", x, fixed = TRUE)
  x
}
sanitize_var_labels <- function(df) {
  labs <- labelled::var_label(df)
  labs <- lapply(labs, sanitize_label_text)
  labelled::var_label(df) <- labs
  df
}
stata_data <- sanitize_var_labels(stata_data)

var_labels <- labelled::var_label(stata_data)
value_labels <- lapply(stata_data, function(x) if (is.labelled(x)) val_labels(x))
saveRDS(
  list(var_labels = var_labels, value_labels = value_labels),
  results_path("stata_labels.rds")
)

```

1.1.1 Configuration for the IPW models

```
# Purpose: propensity config.
drop_vars_ultra_missing <- c("bpn", "spo2")
cat_vars <- c("sex", "race_ethnicity", "location", "encounter_type")
numeric_vars <- c(
  "age_at_encounter", "curr_bmi", "temp_new", "sbp", "dbp", "hr",
  "sodium", "serum_cr", "serum_hco3", "serum_cl", "serum_lac", "serum_k",
  "wbc", "plt", "serum_phos", "serum_ca"
)
co2_vars <- c("paco2", "vbg_co2", "vbg_o2sat")

covars_gbm <- c(
  "age_at_encounter", "sex", "race_ethnicity", "curr_bmi",
  "copd", "asthma", "osa", "chf", "acute_nmd", "phtn", "ckd", "dm",
  "location", "encounter_type", "temp_new", "sbp", "dbp", "hr",
  "sodium", "serum_cr", "serum_hco3", "serum_cl", "serum_lac", "serum_k",
  "wbc", "plt", "serum_phos", "serum_ca"
)
covars_gbm <- setdiff(covars_gbm, drop_vars_ultra_missing)
covars_ps <- covars_gbm

# Core adjustment set for conditional prognostic models
adj_core <- c("age_at_encounter", "sex", "race_ethnicity", "location", "encounter_type")

gbm_params <- list(
  n.trees = 800,
  interaction.depth = 3,
  shrinkage = 0.01,
  bag.fraction = 0.8,
  n.minobsinnode = 10,
  cv.folds = 0,
  stop.method = "smd.max",
  n.cores = 1L
)
stopifnot(gbm_params$stop.method == "smd.max")
SPLINE_BASIS <- "ns"
```

```

SPLINE_DF <- 4L
stopifnot(SPLINE_BASIS %in% c("ns", "rcs"))
get_gbm_cores <- function() {
  n_rows <- nrow(subset_data)
  if (is.finite(n_rows) && n_rows > 200000L) return(1L)
  gbm_params$n.cores
}
ps_trunc_quantile <- 0.01
stopifnot(ps_trunc_quantile > 0, ps_trunc_quantile < 0.5)

formula_abg      <- reformulate(covars_gbm, response = "has_abg")
formula_vbg      <- reformulate(covars_gbm, response = "has_vbg")

# Model diagrams: propensity models (GBM PS)
register_model_diagram("PS model: ABG test (GBM)", formula_abg, width = 10, height = 7)
register_model_diagram("PS model: VBG test (GBM)", formula_vbg, width = 10, height = 7)

# TODO: consider stop.method = "es.mean" (ATS version) if smd.max remains unstable on full N.

# Purpose: build subset data.
# Start from the complete extracted dataset.
subset_data <- stata_data

# In pilot mode, sample a reproducible fraction for faster iteration.
# In full mode, PILOT_FRAC should be 1 and no sampling is applied.
if (PILOT_FRAC < 1) {
  set.seed(SAMPLE_SEED)
  subset_data <- dplyr::sample_frac(stata_data, size = PILOT_FRAC)
}
sampled_n <- nrow(subset_data)

# Emit and persist row counts so each render is auditable.
message("DATA: full_n=", nrow(stata_data), " | sampled_n=", sampled_n)
if (RUN_MODE == "full" && PILOT_FRAC < 1) {
  stop("RUN_MODE='full' but PILOT_FRAC < 1. Set pilot_frac=1 or run_mode='pilot'.")
}

```

```

# Remove encounters with legacy code 1 before encounter-type normalization.
subset_data <- subset_data %>%
  filter(episode_type != 1)
subset_n <- nrow(subset_data)

message("DATA: analysis_subset_n=", subset_n)

run_rowcounts <- data.frame(
  run_id = diag_run_id,
  run_ts = diag_run_ts,
  run_mode = RUN_MODE,
  pilot_frac = PILOT_FRAC,
  full_n = nrow(stata_data),
  sampled_n = sampled_n,
  subset_n = subset_n,
  stringsAsFactors = FALSE
)
write_csv_safely(run_rowcounts, results_path("run_rowcounts.csv"), row_names = FALSE)

# Normalize once early and reuse everywhere

# Keep raw copy for missingness reporting
subset_data_raw <- subset_data
episode_type_raw <- subset_data_raw$episode_type

to01 <- function(x) {
  # Convert mixed encodings (logical/factor/character/numeric labels)
  # to a strict integer indicator in {0, 1, NA}.
  if (inherits(x, "haven_labelled")) x <- unclass(x)
  if (is.logical(x)) return(as.integer(x))
  if (is.factor(x)) x <- as.character(x)
  out <- rep(NA_integer_, length(x))
  xs <- suppressWarnings(as.numeric(x))
  is_num <- !is.na(xs)
  out[is_num & xs %in% c(0, 1)] <- as.integer(xs[is_num & xs %in% c(0, 1)])
  if (any(!is_num)) {
    idx <- which(!is_num)

```

```

s <- trimws(tolower(as.character(x[idx])))
# Explicit text mappings used in this project.
# We support both binary labels and testing-status labels.
out[idx[s %in% c("0","no","false","female","f")]] <- 0L
out[idx[s %in% c("1","yes","true","male","m")]]   <- 1L
out[idx[s %in% c("no test", "not tested", "untested", "control", "absent")]] <- 0L
out[idx[s %in% c("test", "tested", "treated", "present")]] <- 1L
}
out
}

normalize_encounter_type <- function(x) {
  # Harmonize encounter type across numeric codes and free-text variants.
  # Output is the canonical factor used in all models/tables:
  # Emergency / Inpatient.
  s_chr <- trimws(tolower(as.character(x)))
  num_from_text <- suppressWarnings(as.numeric(gsub("[^0-9]+", "", s_chr)))
  lab <- rep(NA_character_, length(s_chr))
  lab[!is.na(num_from_text) & num_from_text == 2] <- "Emergency"
  lab[!is.na(num_from_text) & num_from_text == 3] <- "Inpatient"
  is_na <- is.na(lab)
  is_em <- grepl("\\bemerg(?:ency)?\\b", s_chr) |
    grepl("(^|[^a-z])ed([a-z]|$)", s_chr) |
    grepl("\\ba&e\\b", s_chr) |
    grepl("\\bemergency\\s+dept\\b", s_chr)
  is_ip <- grepl("\\binpatient\\b", s_chr) |
    grepl("\\binpt\\b", s_chr) |
    grepl("\\binpat\\b", s_chr) |
    grepl("(^|[^a-z])ip([a-z]|$)", s_chr)
  lab[is_na & is_em] <- "Emergency"
  lab[is_na & is_ip] <- "Inpatient"
  factor(lab, levels = c("Emergency", "Inpatient"))
}

coerce_num <- function(x) {
  # Robust numeric coercion for labelled/factor/character inputs.
  # Non-numeric symbols are stripped to avoid parse failures.
}

```

```

if (inherits(x, "haven_labelled")) x <- unclass(x)
if (is.factor(x)) x <- as.character(x)
if (is.character(x)) x <- gsub("[^0-9.+-]", "", x)
suppressWarnings(as.numeric(x))
}

# Reference factor levels are captured once from source data.
# We then reuse them to keep category ordering stable across stages.
levels_ref <- list(
  sex = c("Female", "Male"),
  race_ethnicity = levels(factor(stata_data$race_ethnicity)),
  location = levels(factor(stata_data$location)),
  encounter_type = levels(normalize_encounter_type(stata_data$encounter_type))
)

normalize_types <- function(df, levels_ref = NULL) {
  # 1) Drop variables intentionally excluded due to extreme missingness.
  df <- df[, setdiff(names(df), drop_vars_ultra_missing), drop = FALSE]

  # 2) Coerce core numeric/lab/gas variables to numeric once.
  num_vars <- intersect(c(numeric_vars, co2_vars), names(df))
  for (nm in num_vars) df[[nm]] <- coerce_num(df[[nm]])

  # 3) Force sex to the canonical binary factor.
  stopifnot("sex" %in% names(df))
  df$sex <- factor(to01(df$sex), levels = c(0L, 1L), labels = c("Female", "Male"))

  # 4) Normalize encounter type into Emergency/Inpatient.
  stopifnot("encounter_type" %in% names(df))
  df$encounter_type <- normalize_encounter_type(df$encounter_type)

  # 5) Convert remaining categorical predictors to factors.
  # If reference levels are supplied, enforce them for consistency.
  for (nm in setdiff(cat_vars, c("sex", "encounter_type"))) {
    stopifnot(nm %in% names(df))
    if (!is.null(levels_ref) && !is.null(levels_ref[[nm]])) {
      df[[nm]] <- factor(df[[nm]], levels = levels_ref[[nm]])
    }
  }
}

```

```

} else {
  df[[nm]] <- factor(df[[nm]])
}
}

# 6) Ensure treatment indicators are explicit 0/1 integers.
stopifnot(all(c("has_abg", "has_vbg") %in% names(df)))
df$has_abg <- to01(df$has_abg)
df$has_vbg <- to01(df$has_vbg)

# Keep row count unchanged: this function is for type normalization only.
df
}

subset_data <- normalize_types(subset_data_raw, levels_ref)
subset_data <- droplevels_all(subset_data)
subset_data$encounter_type <- droplevels(subset_data$encounter_type)
stopifnot(nlevels(subset_data$encounter_type) == 2L)
subset_n <- nrow(subset_data)

# Factor-level diagnostic (CSV + short summary)
factor_diag <- lapply(names(subset_data), function(v) {
  x <- subset_data[[v]]
  data.frame(
    variable = v,
    class = class(x)[1],
    nlevels = if (is.factor(x)) nlevels(x) else NA_integer_,
    n_missing = sum(is.na(x)),
    pct_missing = if (nrow(subset_data) > 0) 100 * sum(is.na(x)) / nrow(subset_data) else NA_real_,
    stringsAsFactors = FALSE
  )
})
factor_diag <- dplyr::bind_rows(factor_diag)
write_csv_safely(factor_diag, results_path("factor_levels_diagnostic.csv"), row_names = FALSE)
render_table_pdf(
  factor_diag,
  caption = "Factor level diagnostics (all variables)",

```

```

file_stub = "factor_levels_diagnostic",
digits = 1
)

```

Table 2: Factor level diagnostics (all variables)

variable	class	nlevels	n_missing	pct_missing
encounter_id	character	NA	0	0.0
rfs	character	NA	0	0.0
sex	factor	2	0	0.0
race	haven_labelled	NA	0	0.0
ethnicity	haven_labelled	NA	0	0.0
location	factor	4	0	0.0
age_at_encounter	numeric	NA	0	0.0
los	numeric	NA	0	0.0
curr_bmi	numeric	NA	2934	56.7
hr	numeric	NA	1890	36.5
curr_height	numeric	NA	1462	28.3
vbg_temp	numeric	NA	5175	100.0
abg_temp	numeric	NA	5175	100.0
vbg_o2sat	numeric	NA	4474	86.5
abg_o2sat	numeric	NA	4097	79.2
sao2_blood	numeric	NA	4932	95.3
value_prev_weight	numeric	NA	4720	91.2
value_prev_height	numeric	NA	4488	86.7
value_prev_bmi	numeric	NA	4859	93.9
value_highest_20198	numeric	NA	3929	75.9
value_highest_115576	numeric	NA	4652	89.9
value_highest_327718	numeric	NA	4939	95.4
vbg_co2	numeric	NA	3730	72.1
highest_vbg_co2	numeric	NA	3726	72.0
pco2_nos	numeric	NA	4568	88.3
highest_pco2_nos	numeric	NA	4568	88.3
abg_ph	numeric	NA	3240	62.6
vbg_ph	numeric	NA	3638	70.3
abg_hco3	numeric	NA	4021	77.7
vbg_hco3	numeric	NA	3859	74.6
sodium	numeric	NA	261	5.0
serum_cr	numeric	NA	481	9.3
hgb	numeric	NA	566	10.9
serum_hco3	numeric	NA	282	5.4
serum_cl	numeric	NA	298	5.8
serum_lac	numeric	NA	3125	60.4
serum_k	numeric	NA	402	7.8
temp_cor_oxygen	numeric	NA	5090	98.4
vbg_ph_temp_cor	numeric	NA	5091	98.4
vbg_po2	numeric	NA	3878	74.9

Table 2: Factor level diagnostics (all variables) (*continued*)

variable	class	nlevels	n_missing	pct_missing
vbg_lactate	numeric	NA	4993	96.5
vbg_hco3_calc	numeric	NA	5050	97.6
abg_po2	numeric	NA	3983	77.0
abg_po2_temp_cor	numeric	NA	4925	95.2
abg_ph_temp_cor	numeric	NA	4933	95.3
abg_lactate	numeric	NA	4975	96.1
ph_blood	numeric	NA	4621	89.3
po2_blood	numeric	NA	4670	90.2
wbc	numeric	NA	894	17.3
plt	numeric	NA	401	7.7
bnp_date	character	NA	0	0.0
serum_phos_date	character	NA	0	0.0
serum_phos	numeric	NA	2732	52.8
serum_ca_date	character	NA	0	0.0
serum_ca	numeric	NA	515	10.0
serum_albumin_date	character	NA	0	0.0
serum_albumin	numeric	NA	1859	35.9
serum_tprot_date	character	NA	0	0.0
serum_tprot	numeric	NA	1961	37.9
has_j9612	numeric	NA	0	0.0
has_j9622	numeric	NA	0	0.0
has_j9602	numeric	NA	0	0.0
has_j9692	numeric	NA	0	0.0
ohs_code	numeric	NA	0	0.0
has_j9600	numeric	NA	0	0.0
principal_diagnosis_indicator	character	NA	0	0.0
admitting_diagnosis	character	NA	0	0.0
reason_for_visit	character	NA	0	0.0
has_j9601	numeric	NA	0	0.0
has_j961	numeric	NA	0	0.0
has_j9610	numeric	NA	0	0.0
has_j9611	numeric	NA	0	0.0
has_j962	numeric	NA	0	0.0
has_j9620	numeric	NA	0	0.0
has_j9621	numeric	NA	0	0.0
has_j9690	numeric	NA	0	0.0
has_j9691	numeric	NA	0	0.0
other_abn_of_br	haven_labelled	NA	0	0.0
cfdo	haven_labelled	NA	0	0.0
has_i50_acute	numeric	NA	0	0.0
acute_nmd	haven_labelled	NA	0	0.0
sepsis_dx	haven_labelled	NA	0	0.0
stupor_dx	haven_labelled	NA	0	0.0
cog_signs_dx	haven_labelled	NA	0	0.0
mal_fat_dx	haven_labelled	NA	0	0.0

Table 2: Factor level diagnostics (all variables) (*continued*)

variable	class	nlevels	n_missing	pct_missing
resp_acid_dx	haven_labelled	NA	0	0.0
sleep_hypovent_dx	haven_labelled	NA	0	0.0
cchs_dx	haven_labelled	NA	0	0.0
other_sleep_hypovent_dx	haven_labelled	NA	0	0.0
acidosis_unspec	haven_labelled	NA	0	0.0
headache_dx	haven_labelled	NA	0	0.0
cpap	haven_labelled	NA	0	0.0
tte_proc	haven_labelled	NA	0	0.0
aero	haven_labelled	NA	0	0.0
inh_teaching	haven_labelled	NA	0	0.0
cxr1v	haven_labelled	NA	0	0.0
cxr2v	haven_labelled	NA	0	0.0
ctcnnoncon	haven_labelled	NA	0	0.0
ctcccon	haven_labelled	NA	0	0.0
cc_time	haven_labelled	NA	0	0.0
meas_venous_o2_proc	haven_labelled	NA	0	0.0
meas_arterial_gas_proc	haven_labelled	NA	0	0.0
blood_cx_proc	haven_labelled	NA	0	0.0
art_punct_proc	haven_labelled	NA	0	0.0
ctabdpelv	haven_labelled	NA	0	0.0
osa	haven_labelled	NA	0	0.0
asthma	haven_labelled	NA	0	0.0
copd	haven_labelled	NA	0	0.0
chf	haven_labelled	NA	0	0.0
stroke	haven_labelled	NA	0	0.0
ckd	haven_labelled	NA	0	0.0
pvd	haven_labelled	NA	0	0.0
oud	haven_labelled	NA	0	0.0
sedatives	haven_labelled	NA	0	0.0
phtn	haven_labelled	NA	0	0.0
polycy	haven_labelled	NA	0	0.0
po_steroid	haven_labelled	NA	0	0.0
narcan	haven_labelled	NA	0	0.0
inpt_inh	haven_labelled	NA	0	0.0
vasodilators	haven_labelled	NA	0	0.0
ip_diuretics	haven_labelled	NA	0	0.0
ip_abx	haven_labelled	NA	0	0.0
paralytic	haven_labelled	NA	0	0.0
op_diuretics	haven_labelled	NA	0	0.0
op_opiate	haven_labelled	NA	0	0.0
op_mat	haven_labelled	NA	0	0.0
op_nrt	haven_labelled	NA	0	0.0
copd_med	haven_labelled	NA	0	0.0
muscle_relax	haven_labelled	NA	0	0.0
pat_enc_hash	character	NA	0	0.0

Table 2: Factor level diagnostics (all variables) (*continued*)

variable	class	nlevels	n_missing	pct_missing
ABG_rfs	numeric	NA	0	0.0
is_amb	numeric	NA	0	0.0
encounter_date	Date	NA	0	0.0
age_by_ten	numeric	NA	0	0.0
age_decade	haven_labelled	NA	0	0.0
death_date	numeric	NA	4467	86.3
died	numeric	NA	0	0.0
months_death_or_cens	numeric	NA	0	0.0
curr_weight	numeric	NA	1624	31.4
curr_weight_date	numeric	NA	1624	31.4
prev_weight_date	numeric	NA	4720	91.2
curr_height_date	numeric	NA	1462	28.3
prev_height_date	numeric	NA	4488	86.7
curr_bmi_date	numeric	NA	2934	56.7
prev_bmi_date	numeric	NA	4858	93.9
height	numeric	NA	1459	28.2
height_date	numeric	NA	1459	28.2
weight	numeric	NA	1620	31.3
weight_date	numeric	NA	1620	31.3
calc_bmi	numeric	NA	2191	42.3
calc_bmi_date	numeric	NA	2191	42.3
working_bmi	numeric	NA	2932	56.7
working_bmi_date	numeric	NA	2932	56.7
bmi	numeric	NA	1777	34.3
bmi_date	numeric	NA	1777	34.3
bmi_int	numeric	NA	1777	34.3
bmi_by_five	numeric	NA	1777	34.3
rr	numeric	NA	2845	55.0
rr_date	numeric	NA	2838	54.8
temp_new	numeric	NA	2501	48.3
new_temp_date	numeric	NA	2501	48.3
sbp	numeric	NA	1544	29.8
sbp_date	numeric	NA	1544	29.8
dbp	numeric	NA	1556	30.1
dbp_date	numeric	NA	1550	30.0
spo2_date	numeric	NA	3665	70.8
hr_date	numeric	NA	1889	36.5
abg_ph_date	numeric	NA	4054	78.3
vbg_ph_date	numeric	NA	3754	72.5
abg_hco3_date	numeric	NA	4004	77.4
abg_hco3_int	numeric	NA	4021	77.7
vbg_hco3_date	numeric	NA	3841	74.2
sodium_date	numeric	NA	260	5.0
serum_k_date	numeric	NA	402	7.8
hgb_date	numeric	NA	426	8.2

Table 2: Factor level diagnostics (all variables) (*continued*)

variable	class	nlevels	n_missing	pct_missing
wbc_date	numeric	NA	759	14.7
plt_date	numeric	NA	344	6.6
serum_hco3_date	numeric	NA	276	5.3
any_bicarb	numeric	NA	230	4.4
int_bicarb	numeric	NA	230	4.4
hco3_cat	haven_labelled	NA	282	5.4
serum_cl_date	numeric	NA	292	5.6
serum_cr_date	numeric	NA	444	8.6
serum_lac_date	numeric	NA	3085	59.6
vbg_co2_date	numeric	NA	3723	71.9
pco2_nos_date	numeric	NA	4568	88.3
highest_vbg_co2_date	numeric	NA	3723	71.9
highest_pco2_nos_date	numeric	NA	4568	88.3
paco2	numeric	NA	3264	63.1
paco2_date_1	numeric	NA	3928	75.9
paco2_date_2	numeric	NA	4652	89.9
paco2_date_3	numeric	NA	4939	95.4
paco2_date	numeric	NA	3259	63.0
paco2_int	numeric	NA	3264	63.1
highest_paco2	numeric	NA	3260	63.0
paco2_date_highest_1	numeric	NA	3928	75.9
paco2_date_highest_2	numeric	NA	4652	89.9
paco2_date_highest_3	numeric	NA	4939	95.4
paco2_date_highest	numeric	NA	3259	63.0
temp_cor_oxygen_date	numeric	NA	5060	97.8
temp_cor_vbg_ph_date	numeric	NA	5091	98.4
vbg_po2_date	numeric	NA	3847	74.3
vbg_lactate_date	numeric	NA	4992	96.5
vbg_hco3_calc_date	numeric	NA	5049	97.6
abg_po2_date	numeric	NA	3982	76.9
abg_po2_temp_cor_date	numeric	NA	4925	95.2
abg_ph_temp_cor_date	numeric	NA	4933	95.3
abg_lactate_date	numeric	NA	4974	96.1
ph_blood_date	numeric	NA	4621	89.3
po2_blood_date	numeric	NA	4662	90.1
vbg_temp_date	numeric	NA	5175	100.0
abg_temp_date	numeric	NA	5166	99.8
vbg_o2sat_date	numeric	NA	4468	86.3
abg_o2sat_date	numeric	NA	3959	76.5
sao2_blood_date	numeric	NA	4890	94.5
paco2_flag	haven_labelled	NA	3264	63.1
highest_paco2_flag	haven_labelled	NA	3260	63.0
paco2_52_flag	haven_labelled	NA	3264	63.1
vbg_co2_flag	haven_labelled	NA	3730	72.1
highest_vbg_co2_flag	haven_labelled	NA	3726	72.0

Table 2: Factor level diagnostics (all variables) (*continued*)

variable	class	nlevels	n_missing	pct_missing
miss_paco2_flag	numeric	NA	0	0.0
miss_vbg_co2_flag	numeric	NA	0	0.0
miss_vbg_or_abg_co2_flag	numeric	NA	0	0.0
hco3_flag	haven_labelled	NA	282	5.4
not_paco2_flag	numeric	NA	3264	63.1
not_hco3_flag	numeric	NA	282	5.4
k_cat	haven_labelled	NA	402	7.8
acidemia	haven_labelled	NA	2284	44.1
abg_sbe	numeric	NA	4062	78.5
vbg_sbe	numeric	NA	3867	74.7
cw_simple_acute_resp_acid	haven_labelled	NA	4331	83.7
paco2_52_comp_flag	haven_labelled	NA	3266	63.1
po_steroid_date	numeric	NA	2857	55.2
narcan_date	numeric	NA	4148	80.2
inpt_inh_date	numeric	NA	2994	57.9
vasodilators_date	numeric	NA	5152	99.6
ip_diuretics_date	numeric	NA	5112	98.8
ip_abx_date	numeric	NA	5065	97.9
paralytic_date	numeric	NA	5134	99.2
inpt_inh_0	numeric	NA	0	0.0
ip_abx_0	numeric	NA	0	0.0
ip_diuretics_0	numeric	NA	0	0.0
narcan_0	numeric	NA	0	0.0
paralytic_0	numeric	NA	0	0.0
po_steroid_0	numeric	NA	0	0.0
vasodilators_0	numeric	NA	0	0.0
op_diuretics_first_date	numeric	NA	3245	62.7
op_diuretics_last_date	numeric	NA	3256	62.9
op_diuretics_365d	haven_labelled	NA	0	0.0
op_opiate_first_date	numeric	NA	1907	36.9
op_opiate_last_date	numeric	NA	1917	37.0
op_opiate_365d	haven_labelled	NA	0	0.0
op_mat_first_date	numeric	NA	4967	96.0
op_mat_last_date	numeric	NA	4970	96.0
op_mat_365d	haven_labelled	NA	0	0.0
op_nrt_first_date	numeric	NA	4588	88.7
op_nrt_last_date	numeric	NA	4590	88.7
op_nrt_365d	haven_labelled	NA	0	0.0
copd_med_first_date	numeric	NA	2608	50.4
copd_med_last_date	numeric	NA	2615	50.5
copd_med_365d	haven_labelled	NA	0	0.0
muscle_relax_first_date	numeric	NA	3983	77.0
muscle_relax_last_date	numeric	NA	3989	77.1
muscle_relax_365d	haven_labelled	NA	0	0.0
tte_proc_first_date	numeric	NA	4527	87.5

Table 2: Factor level diagnostics (all variables) (*continued*)

variable	class	nlevels	n_missing	pct_missing
tte_proc_last_date	numeric	NA	4527	87.5
vent_proc	haven_labelled	NA	0	0.0
niv_proc	haven_labelled	NA	0	0.0
imv_proc	haven_labelled	NA	0	0.0
cpap_first_date	numeric	NA	4868	94.1
cpap_last_date	numeric	NA	4868	94.1
niv_proc_first_date	numeric	NA	4841	93.5
niv_proc_last_date	numeric	NA	4841	93.5
imv_proc_first_date	numeric	NA	4621	89.3
imv_proc_last_date	numeric	NA	4621	89.3
vent_proc_first_date	numeric	NA	4344	83.9
vent_proc_last_date	numeric	NA	4344	83.9
aero_first_date	numeric	NA	4506	87.1
aero_last_date	numeric	NA	4506	87.1
inh_teaching_first_date	numeric	NA	5020	97.0
inh_teaching_last_date	numeric	NA	5020	97.0
cxr1v_first_date	numeric	NA	3799	73.4
cxr1v_last_date	numeric	NA	3799	73.4
cxr2v_first_date	numeric	NA	5025	97.1
cxr2v_last_date	numeric	NA	5025	97.1
ctcnoncon_first_date	numeric	NA	5025	97.1
ctcnoncon_last_date	numeric	NA	5025	97.1
ctccon_first_date	numeric	NA	4944	95.5
ctccon_last_date	numeric	NA	4944	95.5
ctabdpelv_first_date	numeric	NA	4870	94.1
ctabdpelv_last_date	numeric	NA	4870	94.1
meas_venous_o2_proc_first_date	numeric	NA	5109	98.7
meas_venous_o2_proc_last_date	numeric	NA	5109	98.7
meas_art_gas_proc_first_date	numeric	NA	5130	99.1
meas_art_gas_proc_last_date	numeric	NA	5130	99.1
blood_cx_proc_first_date	numeric	NA	4493	86.8
blood_cx_proc_last_date	numeric	NA	4493	86.8
art_punct_proc_first_date	numeric	NA	4892	94.5
art_punct_proc_last_date	numeric	NA	4892	94.5
cc_time_first_date	numeric	NA	4002	77.3
cc_time_last_date	numeric	NA	4002	77.3
aero_0	numeric	NA	0	0.0
blood_cx_proc_0	numeric	NA	0	0.0
cc_time_0	numeric	NA	0	0.0
cpap_0	numeric	NA	0	0.0
ctabdpelv_0	numeric	NA	0	0.0
ctccon_0	numeric	NA	0	0.0
ctcnoncon_0	numeric	NA	0	0.0
cxr1v_0	numeric	NA	0	0.0
cxr2v_0	numeric	NA	0	0.0

Table 2: Factor level diagnostics (all variables) (*continued*)

variable	class	nlevels	n_missing	pct_missing
imv_proc_0	numeric	NA	0	0.0
inh_teaching_0	numeric	NA	0	0.0
niv_proc_0	numeric	NA	0	0.0
tte_proc_0	numeric	NA	0	0.0
vent_proc_0	numeric	NA	0	0.0
aero_dur	numeric	NA	4506	87.1
blood_cx_proc_dur	numeric	NA	4493	86.8
cpap_dur	numeric	NA	4868	94.1
ctabdpelv_dur	numeric	NA	4870	94.1
ctccon_dur	numeric	NA	4944	95.5
ctcnnoncon_dur	numeric	NA	5025	97.1
cxr1v_dur	numeric	NA	3799	73.4
cxr2v_dur	numeric	NA	5025	97.1
imv_proc_dur	numeric	NA	4621	89.3
inh_teaching_dur	numeric	NA	5020	97.0
niv_proc_dur	numeric	NA	4841	93.5
tte_proc_dur	numeric	NA	4527	87.5
hypercap_resp_failure	haven_labelled	NA	0	0.0
j9612_date	numeric	NA	5139	99.3
j9612_pcpl	haven_labelled	NA	5139	99.3
j9612_adm	haven_labelled	NA	5139	99.3
j9612_vr	haven_labelled	NA	5139	99.3
j9622_date	numeric	NA	5064	97.9
j9622_pcpl	haven_labelled	NA	5067	97.9
j9622_adm	haven_labelled	NA	5064	97.9
j9622_vr	haven_labelled	NA	5064	97.9
j9602_date	numeric	NA	5007	96.8
j9602_pcpl	haven_labelled	NA	5010	96.8
j9602_adm	haven_labelled	NA	5007	96.8
j9602_vr	haven_labelled	NA	5007	96.8
j9692_date	numeric	NA	5153	99.6
j9692_pcpl	haven_labelled	NA	5153	99.6
j9692_adm	haven_labelled	NA	5153	99.6
j9692_vr	haven_labelled	NA	5153	99.6
ohs_code_date	numeric	NA	5124	99.0
e662_pcpl	haven_labelled	NA	5124	99.0
e662_adm	haven_labelled	NA	5124	99.0
e662_vr	haven_labelled	NA	5124	99.0
hypercap_resp_failure_date	numeric	NA	4873	94.2
other_resp_failure	haven_labelled	NA	0	0.0
j9600_date	numeric	NA	4995	96.5
j9601_date	numeric	NA	4265	82.4
j961_date	numeric	NA	5175	100.0
j9610_date	numeric	NA	5140	99.3
j9611_date	numeric	NA	5047	97.5

Table 2: Factor level diagnostics (all variables) (*continued*)

variable	class	nlevels	n_missing	pct_missing
j962_date	numeric	NA	5175	100.0
j9620_date	numeric	NA	5153	99.6
j9621_date	numeric	NA	4916	95.0
j9690_date	numeric	NA	5050	97.6
j9691_date	numeric	NA	5105	98.6
other_resp_failure_date	numeric	NA	3864	74.7
sepsis_dx_date	numeric	NA	4571	88.3
stupor_dx_date	numeric	NA	5057	97.7
cog_signs_dx_date	numeric	NA	4689	90.6
mal_fat_dx_date	numeric	NA	4598	88.9
resp_acid_dx_date	numeric	NA	5127	99.1
sleep_hypovent_dx_date	numeric	NA	5170	99.9
cchs_dx_date	numeric	NA	5175	100.0
other_sleep_hypovent_dx_date	numeric	NA	5169	99.9
acidosis_unspec_date	numeric	NA	5016	96.9
headache_dx_date	numeric	NA	5023	97.1
dysp_dx	haven_labelled	NA	0	0.0
dysp_dx_date	numeric	NA	4423	85.5
symp_obs	haven_labelled	NA	0	0.0
symp_obs_date	numeric	NA	5124	99.0
abn_br_dx	haven_labelled	NA	0	0.0
abn_br_dx_date	numeric	NA	5167	99.8
resp_abnormality	haven_labelled	NA	0	0.0
r0689_date	numeric	NA	5089	98.3
resp_abnormality_date	numeric	NA	5076	98.1
other_abn_of_br_date	numeric	NA	5089	98.3
fast_br	haven_labelled	NA	0	0.0
fast_br_date	numeric	NA	5147	99.5
pulm_edema_dx	haven_labelled	NA	0	0.0
pulm_edema_dx_date	numeric	NA	5018	97.0
pna_dx	haven_labelled	NA	0	0.0
pna_dx_date	numeric	NA	4440	85.8
acute_chf	numeric	NA	0	0.0
acute_old	haven_labelled	NA	0	0.0
acute_old_date	numeric	NA	0	0.0
resp_dep_compl	haven_labelled	NA	0	0.0
resp_dep_compl_date	numeric	NA	5004	96.7
acute_nmd_date	numeric	NA	5164	99.8
osa_first_date	numeric	NA	4294	83.0
osa_last_date	numeric	NA	4296	83.0
asthma_first_date	numeric	NA	4491	86.8
asthma_last_date	numeric	NA	4491	86.8
copd_first_date	numeric	NA	4159	80.4
copd_last_date	numeric	NA	4160	80.4
chf_first_date	numeric	NA	4122	79.7

Table 2: Factor level diagnostics (all variables) (*continued*)

variable	class	nlevels	n_missing	pct_missing
chf_last_date	numeric	NA	4126	79.7
stroke_first_date	numeric	NA	4805	92.9
stroke_last_date	numeric	NA	4807	92.9
ckd_first_date	numeric	NA	4267	82.5
ckd_last_date	numeric	NA	4268	82.5
ctd	haven_labelled	NA	0	0.0
ctd_first_date	numeric	NA	4897	94.6
ctd_last_date	numeric	NA	4897	94.6
dem	haven_labelled	NA	0	0.0
dem_first_date	numeric	NA	4853	93.8
dem_last_date	numeric	NA	4853	93.8
dm	haven_labelled	NA	0	0.0
dm_first_date	numeric	NA	3654	70.6
dm_last_date	numeric	NA	3658	70.7
pvd_first_date	numeric	NA	4669	90.2
pvd_last_date	numeric	NA	4669	90.2
oud_first_date	numeric	NA	4892	94.5
oud_last_date	numeric	NA	4893	94.6
sedatives_first_date	numeric	NA	5124	99.0
sedatives_last_date	numeric	NA	5124	99.0
cfdo_first_date	numeric	NA	5166	99.8
cfdo_last_date	numeric	NA	5166	99.8
phtn_first_date	numeric	NA	4739	91.6
phtn_last_date	numeric	NA	4739	91.6
polocy_first_date	numeric	NA	5112	98.8
polocy_last_date	numeric	NA	5112	98.8
nmd	haven_labelled	NA	0	0.0
nmd_first_date	numeric	NA	4942	95.5
nmd_last_date	numeric	NA	4943	95.5
nic	haven_labelled	NA	0	0.0
nic_first_date	numeric	NA	3951	76.3
nic_last_date	numeric	NA	3952	76.4
ovs	haven_labelled	NA	0	0.0
ats_ohs_flag	numeric	NA	4531	87.6
pos_ohs_flag	numeric	NA	0	0.0
ats_copd_flag	numeric	NA	4560	88.1
guidelines	haven_labelled	NA	4560	88.1
OBESITY_rfs	numeric	NA	0	0.0
_merge_amb_obes	haven_labelled	NA	5175	100.0
PREDISPOSITION_rfs	numeric	NA	0	0.0
_merge_amb_predis	haven_labelled	NA	5175	100.0
RESPFAIL_rfs	numeric	NA	0	0.0
_merge_amb_respfail	haven_labelled	NA	5175	100.0
VBG_rfs	numeric	NA	0	0.0
_merge_amb_vbg	haven_labelled	NA	5175	100.0

Table 2: Factor level diagnostics (all variables) (*continued*)

variable	class	nlevels	n_missing	pct_missing
VENTSUPPORT_rfs	numeric	NA	0	0.0
_merge_amb_ventsupp	haven_labelled	NA	5175	100.0
is_emer	numeric	NA	0	0.0
_merge_emer_obes	haven_labelled	NA	4674	90.3
_merge_emer_predisp	haven_labelled	NA	3875	74.9
_merge_emer_respfail	haven_labelled	NA	3818	73.8
_merge_emer_vbg	haven_labelled	NA	3480	67.2
_merge_emer_ventsupp	haven_labelled	NA	3465	67.0
_merge_emer	haven_labelled	NA	3465	67.0
is_inp	haven_labelled	NA	1710	33.0
_merge_inpat_obes	haven_labelled	NA	3752	72.5
_merge_inpat_predisp	haven_labelled	NA	2575	49.8
_merge_inpat_respfail	haven_labelled	NA	2264	43.7
_merge_inpat_vbg	haven_labelled	NA	1841	35.6
_merge_inpat_ventsupp	haven_labelled	NA	1710	33.0
_merge_inpat	haven_labelled	NA	0	0.0
patient_id	numeric	NA	0	0.0
rfsgroup	haven_labelled	NA	0	0.0
encounter_type	factor	2	0	0.0
first_encounter	haven_labelled	NA	0	0.0
has_abg	integer	NA	0	0.0
has_vbg	integer	NA	0	0.0
max_hco3_or_its_met_alk	numeric	NA	3264	63.1
prim_met_alk	numeric	NA	4560	88.1
max_hco3_or_its_comb_met_alk	numeric	NA	3264	63.1
combo_met_alk	numeric	NA	4560	88.1
alkalemia	numeric	NA	2284	44.1
paco2_flag_and_alk	numeric	NA	4560	88.1
paco2_50_flag	haven_labelled	NA	3264	63.1
hypercap_dx_adm_flag	numeric	NA	4873	94.2
hypercap_dx_pcpl_flag	numeric	NA	4877	94.2
hypercap_dx_vr_flag	numeric	NA	4873	94.2
hypercap_on_abg	numeric	NA	0	0.0
hypercap_on_vbg	numeric	NA	0	0.0
has_both_abg_vbg	haven_labelled	NA	0	0.0
has_neither_abg_vbg	haven_labelled	NA	0	0.0
vbg_or_abg_co2_flag	haven_labelled	NA	0	0.0
dx_hypercap_on_abg	numeric	NA	4873	94.2
sugg_hypercap_dx_on_vbg	numeric	NA	4873	94.2
dx_hypercap_on_vbg	numeric	NA	4873	94.2
vbg_o2sat_calc	numeric	NA	0	0.0
abg_o2sat_calc	numeric	NA	0	0.0
corr_vbg_co2	numeric	NA	3730	72.1
corr_vbg_co2_flag	haven_labelled	NA	3730	72.1
corr_hypercap_on_vbg	numeric	NA	0	0.0

Table 2: Factor level diagnostics (all variables) (*continued*)

variable	class	nlevels	n_missing	pct_missing
race_ethnicity	factor	7	0	0.0
has_vbg_and_cat	haven_labelled	NA	0	0.0
has_bmi_and_cat	haven_labelled	NA	0	0.0
has_weight_and_cat	haven_labelled	NA	0	0.0
has_height_and_cat	haven_labelled	NA	0	0.0
has_hr_and_cat	haven_labelled	NA	0	0.0
has_sbp_and_cat	haven_labelled	NA	0	0.0
has_rr_and_cat	haven_labelled	NA	0	0.0
has_temp_and_cat	haven_labelled	NA	0	0.0
has_spo2_and_cat	haven_labelled	NA	0	0.0
has_cl_and_cat	haven_labelled	NA	0	0.0
has_k_and_cat	haven_labelled	NA	0	0.0
has_hco3_and_cat	haven_labelled	NA	0	0.0
has_lactate_and_cat	haven_labelled	NA	0	0.0
has_na_and_cat	haven_labelled	NA	0	0.0
has_cr_and_cat	haven_labelled	NA	0	0.0
has_hgb_and_cat	haven_labelled	NA	0	0.0
has_wbc_and_cat	haven_labelled	NA	0	0.0
has_plt_and_cat	haven_labelled	NA	0	0.0
has_bnp_and_cat	haven_labelled	NA	0	0.0
has_phos_and_cat	haven_labelled	NA	0	0.0
has_ca_and_cat	haven_labelled	NA	0	0.0
has_alb_and_cat	haven_labelled	NA	0	0.0
has_tprot_and_cat	haven_labelled	NA	0	0.0
encounter_type_dummy1	numeric	NA	0	0.0
encounter_type_dummy2	numeric	NA	0	0.0
encounter_type_dummy3	numeric	NA	0	0.0
female	haven_labelled	NA	0	0.0
male	numeric	NA	0	0.0
white_race	numeric	NA	0	0.0
black_race	numeric	NA	0	0.0
unknown_race	numeric	NA	0	0.0
asian_race	numeric	NA	0	0.0
nat_am_race	numeric	NA	0	0.0
nhpi_race	numeric	NA	0	0.0
not_hisp_eth	numeric	NA	0	0.0
hisp_eth	numeric	NA	0	0.0
unknown_eth	numeric	NA	0	0.0
location_dummy1	numeric	NA	0	0.0
location_dummy2	numeric	NA	0	0.0
location_dummy3	numeric	NA	0	0.0
location_dummy4	numeric	NA	0	0.0
ps	numeric	NA	0	0.0
tx_pr_decile	numeric	NA	0	0.0
ps_trunc	numeric	NA	0	0.0

Table 2: Factor level diagnostics (all variables) (*continued*)

variable	class	nlevels	n_missing	pct_missing
sumofweights	numeric	NA	0	0.0
ipw	numeric	NA	0	0.0
stabilized_weight	numeric	NA	0	0.0
approx_ipw_fweight	numeric	NA	0	0.0
vbg_ps	numeric	NA	0	0.0
vbg_tx_pr_decile	numeric	NA	0	0.0
vbg_ps_trunc	numeric	NA	0	0.0
vbg_sumofweights	numeric	NA	0	0.0
vbg_ipw	numeric	NA	0	0.0
vbg_stabilized_weight	numeric	NA	0	0.0
vbg_approx_ipw_fweight	numeric	NA	0	0.0

```

# Factor expansion guard (logs only)
factor_guard <- factor_diag |>
  dplyr::filter(!is.na(nlevels) & nlevels > MAX_LEVELS_GBM)
write_csv_safely(factor_guard, results_path("factor_expansion_guard.csv"), row_names = FALSE)
if (nrow(factor_guard)) {
  warning("GBM factor expansion guard: factors exceeding MAX_LEVELS_GBM detected. See
    ↵ Results/factor_expansion_guard.csv.",
    call. = FALSE)
}

tab_enc <- table(subset_data$encounter_type, useNA = "ifany")
if (sum(!is.na(subset_data$encounter_type)) == 0) {
  message("All encounter_type values are NA after normalization. Showing top raw values:")
  s_raw <- trimws(tolower(as.character(encounter_type_raw)))
  print(utils::head(sort(table(s_raw), decreasing = TRUE), 20))
  stop("normalize_encounter_type produced all NA; extend the synonym map to your raw values.")
}

# Reference profile for conditional curves
stopifnot(all(adj_core %in% names(subset_data)))

mode_level <- function(x) {
  if (is.factor(x)) {
    tab <- table(x, useNA = "no")

```

```

if (!length(tab)) return(factor(NA, levels = levels(x)))
lev <- names(tab)[which.max(tab)]
return(factor(lev, levels = levels(x)))
}
if (is.character(x)) {
  tab <- table(x, useNA = "no")
  if (!length(tab)) return(NA_character_)
  return(names(tab)[which.max(tab)])
}
stats::median(as.numeric(x), na.rm = TRUE)
}

make_ref_profile <- function(data, adj_vars) {
  out <- lapply(adj_vars, function(v) mode_level(data[[v]]))
  names(out) <- adj_vars
  as.data.frame(out, stringsAsFactors = FALSE)
}

make_co2_grid <- function(co2_var, co2_vals, ref_df) {
  grid <- data.frame(co2_vals)
  names(grid) <- co2_var
  stopifnot(!is.null(ref_df))
  stopifnot(nrow(ref_df) > 0L)
  ref_rep <- ref_df[rep(1L, nrow(grid)), , drop = FALSE]
  grid <- cbind(grid, ref_rep)
  grid
}

make_co2_grid_ref <- function(co2_var, co2_vals, ref_df, co2_ref) {
  grid_vals <- c(co2_vals, co2_ref)
  grid_vals <- grid_vals[is.finite(grid_vals)]
  grid_vals <- sort(unique(as.numeric(grid_vals)))
  if (length(grid_vals) < 2) {
    fallback_center <- NA_real_
    if (is.finite(co2_ref)) {
      fallback_center <- co2_ref
    } else if (length(co2_vals) && is.finite(co2_vals[1])) {

```

```

    fallback_center <- co2_vals[1]
}
if (is.finite(fallback_center)) {
  grid_vals <- sort(unique(c(fallback_center - 1, fallback_center, fallback_center + 1)))
  grid_vals <- grid_vals[is.finite(grid_vals)]
}
}
if (length(grid_vals) < 2) stop("CO2 grid is too small for ", co2_var, ".")
grid <- make_co2_grid(co2_var, grid_vals, ref_df)
ref_val <- if (is.finite(co2_ref)) co2_ref else stats::median(grid[[co2_var]], na.rm = TRUE)
ref_idx <- match(ref_val, grid[[co2_var]])
if (is.na(ref_idx)) {
  ref_idx <- which.min(abs(grid[[co2_var]] - ref_val))
}
if (any(diff(grid[[co2_var]]) <= 0)) {
  stop("CO2 grid is not strictly increasing for ", co2_var, ".")
}
list(grid = grid, ref_idx = ref_idx, co2_ref = grid[[co2_var]][ref_idx])
}

predict_or_curve_from_fit <- function(fit, grid_df, ref_idx, co2_var) {
  # Build a model matrix consistent with the fitted model (handles factor levels)
  tt <- stats::delete.response(stats::terms(fit))
  xlev <- stats:::getXlevels(tt, stats::model.frame(fit))
  mf_new <- stats::model.frame(tt, grid_df, na.action = stats::na.pass, xlev = xlev)
  mm <- stats::model.matrix(tt, mf_new)
  beta <- stats::coef(fit)
  V <- stats::vcov(fit)

  # Align model matrix columns to coefficient names (fill missing with 0, drop extras)
  bn <- names(beta)
  if (is.null(bn) || any(!nzchar(bn))) {
    if (length(beta) == ncol(mm)) {
      bn <- colnames(mm)
      names(beta) <- bn
    } else {
      stop("predict_or_curve_from_fit: coefficient names missing and dimensions do not match.")
    }
  }
}

```

```

    }
}

if (!all(bn %in% colnames(mm))) {
  missing_cols <- setdiff(bn, colnames(mm))
  if (length(missing_cols)) {
    mm <- cbind(mm, matrix(0, nrow = nrow(mm), ncol = length(missing_cols),
                           dimnames = list(NULL, missing_cols)))
  }
}

mm <- mm[, bn, drop = FALSE]
stopifnot(ncol(mm) == length(beta))

eta <- as.numeric(mm %*% beta)
mmV <- mm %*% V
var_eta <- rowSums(mmV * mm)

mm_ref <- mm[ref_idx, , drop = FALSE]
eta_ref <- eta[ref_idx]
var_ref <- as.numeric(mm_ref %*% V %*% t(mm_ref))
cov_ref <- as.numeric(mmV %*% t(mm_ref))

logOR <- eta - eta_ref
var_logOR <- var_eta + var_ref - 2 * cov_ref
var_logOR <- pmax(var_logOR, 0)
logOR[ref_idx] <- 0
var_logOR[ref_idx] <- 0
se <- sqrt(var_logOR)

data.frame(
  grid_df,
  logOR = logOR,
  SE_logOR = se,
  var_logOR = var_logOR,
  OR = exp(logOR),
  LCL = exp(logOR - 1.96 * se),
  UCL = exp(logOR + 1.96 * se),
  co2_ref = grid_df[[co2_var]][ref_idx],

```

```

    ref_idx = ref_idx,
    row.names = NULL
)
}

x_ref_abg <- make_ref_profile(
  subset_data |> dplyr::filter(has_abg == 1, !is.na(paco2)),
  adj_core
)
x_ref_vbg <- make_ref_profile(
  subset_data |> dplyr::filter(has_vbg == 1, !is.na(vbg_co2)),
  adj_core
)

# Purpose: run metadata.
run_meta <- tibble::tibble(
  run_id      = diag_run_id,
  run_mode    = RUN_MODE,
  pilot_frac = PILOT_FRAC,
  mi_batch_threshold = MI_BATCH_THRESHOLD,
  full_n      = nrow(stata_data),
  sampled_n   = sampled_n,
  subset_n    = nrow(subset_data)
)
render_table_pdf(run_meta,
                 "Run metadata (pilot vs full)",
                 "run_metadata",
                 digits = 2)

```

Table 3: Run metadata (pilot vs full) (Part A)

run_id	run_mode	pilot_frac	mi_batch_threshold	full_n	sampled_n
20260208_095917	pilot	0.01		5000	833476

Table 4: Run metadata (pilot vs full) (Part B)

subset_n
5175

```

# Write run config JSON for portability
json_escape <- function(x) gsub("\"", "\\\"", x)
run_cfg <- list(
  run_id = diag_run_id,
  run_mode = RUN_MODE,
  pilot_frac = PILOT_FRAC,
  mi_batch_threshold = MI_BATCH_THRESHOLD,
  data_dir = data_dir_name,
  results_dir = results_dir,
  full_n = nrow(stata_data),
  sampled_n = sampled_n,
  subset_n = nrow(subset_data)
)
json_lines <- c(
  "{",
  paste0("  \"run_id\": \"", json_escape(run_cfg$run_id), "\","),
  paste0("  \"run_mode\": \"", json_escape(run_cfg$run_mode), "\","),
  paste0("  \"pilot_frac\": ", run_cfg$pilot_frac, ","),
  paste0("  \"mi_batch_threshold\": ", run_cfg$mi_batch_threshold, ","),
  paste0("  \"data_dir\": \"", json_escape(run_cfg$data_dir), "\","),
  paste0("  \"results_dir\": \"", json_escape(run_cfg$results_dir), "\","),
  paste0("  \"full_n\": ", run_cfg$full_n, ","),
  paste0("  \"sampled_n\": ", run_cfg$sampled_n, ","),
  paste0("  \"subset_n\": ", run_cfg$subset_n),
  "}"
)
writeLines(json_lines, results_path("run_config.json"))

```

Codebook exported to Results/codebookr.docx.

```

# Purpose: codebook export full.
study_codebook <- codebookr::codebook(
  stata_data,
  title = "Full TrinetX",
  subtitle = "Dataset Documentation",
  description = "This dataset contains patient-level records from the TrinetX database.
                It has been processed and converted from the original Stata file."
)

```

```
codebook_file <- results_path("codebookr.docx")
print(study_codebook, codebook_file)
```

1.1.2 Outcome Variable Creation

```
# Purpose: derive death 60d.
subset_data <- subset_data %>%
  mutate(
    ## 1. Did the patient die?
    died = if_else(!is.na(death_date), 1L, 0L),

    ## 2. Absolute death date (if death_date is an offset)
    death_abs = if_else(!is.na(death_date),
                        encounter_date + death_date,
                        as.Date(NA)),

    ## 3. Year month (YM) for encounter and death
    enc_ym = floor_date(encounter_date, unit = "month"),
    death_ym = floor_date(death_abs, unit = "month"),

    ## 4. Reference censoring date: 1 Jun 2024
    ref_ym = ymd("2024-06-01"),

    ## 5. Months from encounter to death or censoring
    months_death_or_cens = case_when(
      !is.na(death_ym) ~ interval(enc_ym, death_ym) %/% months(1),
      TRUE           ~ interval(enc_ym, ref_ym)   %/% months(1)
    ),

    ## 6. Remove impossible values
    months_death_or_cens = if_else(
      months_death_or_cens < 0 | months_death_or_cens > 16,
      NA_integer_, months_death_or_cens
    ),

    ## 7. Death within one or two months
```

```

died_1mo = if_else(died == 1 & months_death_or_cens < 1, 1L, 0L),
died_2mo = if_else(died == 1 & months_death_or_cens <= 1, 1L, 0L),

## 8. Month of death (missing if censored)
death_time = if_else(died == 1, months_death_or_cens, NA_integer_),

## 9. Death within 60 days (new variable)
death_60d = if_else(died == 1 & death_abs <= (encounter_date + days(60)), 1L, 0L)
) %>%
select(-enc_ym, -death_ym)

subset_data <- subset_data %>%
mutate(
  death_60d = if_else(died == 1 & death_abs <= (encounter_date + days(60)), 1L, 0L)
)

# Purpose: death 60d summary.
table(subset_data$death_60d, useNA = "ifany")

```

0	1
4641	534

```
prop.table(table(subset_data$death_60d, useNA = "ifany"))
```

0	1
0.8968116	0.1031884

```
summary(subset_data$death_60d)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.0000	0.0000	0.0000	0.1032	0.0000	1.0000

1.2 Baseline tables

```
# Robust derivation of analysis variables + helper for Table 1 production
# -----
#
# helper: label binary 0/1 → "No"/"Yes"
bin_lab <- function(x) {
  y <- to01(x)
  if (all(is.na(y))) {
    return(factor(y, levels = c(0, 1), labels = c("No", "Yes")))
  }
  factor(y, levels = c(0, 1), labels = c("No", "Yes"))
}

# helper: preserve labeled factors if already present; otherwise map numeric codes
label_from_codes <- function(x, codes, labels) {
  if (is.factor(x)) {
    lev <- levels(x)
    if (all(lev %in% labels)) {
      return(factor(x, levels = labels))
    }
    lev_num <- suppressWarnings(as.numeric(lev))
    if (all(!is.na(lev_num)) && all(lev_num %in% codes)) {
      return(factor(as.numeric(as.character(x)), levels = codes, labels = labels))
    }
    return(x)
  }
  x_num <- suppressWarnings(as.numeric(as.character(x)))
  if (all(is.na(x_num))) return(factor(x, levels = labels))
  if (all(x_num %in% codes, na.rm = TRUE)) {
    return(factor(x_num, levels = codes, labels = labels))
  }
  factor(x)
}

subset_data <- subset_data %>%
  mutate(
```

```

## ensure 0/1 numerics (avoids factor-level coercion)
across(c(has_abg, has_vbg),
       ~ to01(.)),

## derive ABG / VBG status groups (binary test status only)
abg_group = case_when(
  has_abg == 0 ~ "No ABG",
  has_abg == 1 ~ "ABG",
  TRUE          ~ "Missing"
),
vbg_group = case_when(
  has_vbg == 0 ~ "No VBG",
  has_vbg == 1 ~ "VBG",
  TRUE          ~ "Missing"
),

## factorise groups with explicit NA/Missing level
abg_group = factor(
  abg_group,
  levels = c("No ABG", "ABG", "Missing")
),
vbg_group = factor(
  vbg_group,
  levels = c("No VBG", "VBG", "Missing")
),

## labelled covariates (robust to factor or numeric codes)
sex_label = label_from_codes(sex, c(0, 1), c("Female", "Male")),
race_ethnicity_label = label_from_codes(
  race_ethnicity,
  0:6,
  c("White", "Black or African American", "Hispanic",
    "Asian", "American Indian", "Pacific Islander", "Unknown")
),
location_label = label_from_codes(
  location,
  0:3,

```

```

    c("South", "Northeast", "Midwest", "West")
),
encounter_type_label = label_from_codes(
  encounter_type,
  c(2, 3),
  c("Emergency", "Inpatient")
),
osa_label      = bin_lab(osa),
asthma_label   = bin_lab(asthma),
copd_label     = bin_lab(copd),
chf_label      = bin_lab(chf),
nmd_label      = bin_lab(nmd),
phtn_label     = bin_lab(phtn),
ckd_label      = bin_lab(ckd),
diabetes_label = bin_lab(dm)
)

# variables to summarise
vars <- c(
  "age_at_encounter", "curr_bmi", "sex_label", "race_ethnicity_label", "location_label",
  "osa_label", "asthma_label", "copd_label", "chf_label", "nmd_label",
  "phtn_label", "ckd_label", "diabetes_label", "encounter_type_label", "vbg_co2", "paco2"
)
vars_baseline <- setdiff(vars, c("vbg_co2", "paco2"))
vars_abg <- c(vars_baseline, "paco2")
vars_vbg <- c(vars_baseline, "vbg_co2")

# Table 1 constructor
make_table1 <- function(data, group_var, caption = "") {
  group_sym <- rlang::sym(group_var)

  df <- data %>%
    filter(!is.na (!!group_sym),                                # drop explicit NA
           !!group_sym != "Missing") %>%                      # drop "Missing" cohort
    mutate (!!group_sym := droplevels (!!group_sym)) %>% # only drop group levels
    select(all_of(c(group_var, vars_baseline)))
}

```

```

empty_fac <- names(which(vapply(df, function(z) is.factor(z) && length(levels(z)) == 0L, logical(1))))
if (length(empty_fac) > 0) {
  warning("0-level factor columns detected: ", paste(empty_fac, collapse = ", "),
  ". Converting to character to prevent gtsummary failure.", call. = FALSE)
  df[empty_fac] <- lapply(df[empty_fac], as.character)
}

df %>%
  gtsummary::tbl_summary(
    by   = !!group_sym,
    type = list(sex_label ~ "categorical"),
    statistic = list(
      gtsummary::all_continuous() ~ "{mean} ± {sd}; {N_miss}/{N_obs} missing ({p_miss}%)",
      gtsummary::all_categorical() ~ "{n} ({p}%)"
    ),
    digits  = list(gtsummary::all_continuous() ~ 1),
    missing  = "no"                                # no gtsummary missing column/row
  ) %>%
  gtsummary::modify_header(label = "***Variable***") %>%
  gtsummary::modify_caption(strip_manual_table_number(caption))
}

if (sum(!is.na(subset_data$sex_label)) == 0L || length(levels(subset_data$sex_label)) == 0L) {
  warning("sex_label is all NA or has zero levels; check sex normalization/mapping.", call. = FALSE)
  stopifnot("sex" %in% names(subset_data))
}

# build tables
table1A <- make_table1(subset_data, "abg_group", caption = "Table 1A: ABG cohorts")
table1B <- make_table1(subset_data, "vbg_group", caption = "Table 1B: VBG cohorts")

tbl1a_pdf <- to_pdf_table(table1A, font_size = 7, landscape = FALSE, label_col_width = "2.0in",
                           longtable = TRUE)
tbl1b_pdf <- to_pdf_table(table1B, font_size = 7, landscape = FALSE, label_col_width = "2.0in",
                           longtable = TRUE)
tbl1a_pdf

```

Table 5: ABG cohorts

Variable	**No ABG** N = 3,264	**ABG** N = 1,911
age_at_encounter	58.4 ± 17.7; 0.0/3,264.0 missing (0.0%)	61.3 ± 17.2; 0.0/1,911.0 missing (0.0%)
curr_bmi	32.6 ± 8.9; 1,821.0/3,264.0 missing (55.8%)	28.8 ± 7.0; 1,113.0/1,911.0 missing (58.2%)
sex_label		
Female	1,737 (53%)	854 (45%)
Male	1,527 (47%)	1,057 (55%)
race_ethnicity_label		
White	2,009 (62%)	1,271 (67%)
Black or African American	601 (18%)	283 (15%)
Hispanic	250 (7.7%)	84 (4.4%)
Asian	46 (1.4%)	37 (1.9%)
American Indian	11 (0.3%)	25 (1.3%)
Pacific Islander	8 (0.2%)	2 (0.1%)
Unknown	339 (10%)	209 (11%)
location_label		
South	1,374 (42%)	1,057 (55%)
Northeast	921 (28%)	379 (20%)
Midwest	246 (7.5%)	160 (8.4%)
West	723 (22%)	315 (16%)
osa_label	576 (18%)	303 (16%)
asthma_label	464 (14%)	220 (12%)
copd_label	579 (18%)	434 (23%)
chf_label	587 (18%)	457 (24%)
nmd_label	130 (4.0%)	103 (5.4%)
phtn_label	239 (7.3%)	195 (10%)
ckd_label	519 (16%)	385 (20%)
diabetes_label	954 (29%)	559 (29%)
encounter_type_label		
Emergency	1,427 (44%)	283 (15%)
Inpatient	1,837 (56%)	1,628 (85%)

tbl1b_pdf

Table 6: VBG cohorts

Variable	**No VBG** N = 3,730	**VBG** N = 1,445
age_at_encounter	59.6 ± 17.6; 0.0/3,730.0 missing (0.0%)	59.0 ± 17.7; 0.0/1,445.0 missing (0.0%)
curr_bmi	31.9 ± 8.6; 1,939.0/3,730.0 missing (52.0%)	28.9 ± 7.5; 995.0/1,445.0 missing (68.9%)
sex_label		
Female	1,921 (52%)	670 (46%)
Male	1,809 (48%)	775 (54%)
race_ethnicity_label		
White	2,513 (67%)	767 (53%)
Black or African American	616 (17%)	268 (19%)

Table 6: VBG cohorts (*continued*)

Variable	**No VBG** N = 3,730	**VBG** N = 1,445
Hispanic	221 (5.9%)	113 (7.8%)
Asian	54 (1.4%)	29 (2.0%)
American Indian	13 (0.3%)	23 (1.6%)
Pacific Islander	8 (0.2%)	2 (0.1%)
Unknown	305 (8.2%)	243 (17%)
location_label		
South	1,987 (53%)	444 (31%)
Northeast	703 (19%)	597 (41%)
Midwest	264 (7.1%)	142 (9.8%)
West	776 (21%)	262 (18%)
osa_label	648 (17%)	231 (16%)
asthma_label	513 (14%)	171 (12%)
copd_label	731 (20%)	282 (20%)
chf_label	741 (20%)	303 (21%)
nmd_label	175 (4.7%)	58 (4.0%)
phtn_label	295 (7.9%)	139 (9.6%)
ckd_label	628 (17%)	276 (19%)
diabetes_label	1,034 (28%)	479 (33%)
encounter_type_label		
Emergency	1,247 (33%)	463 (32%)
Inpatient	2,483 (67%)	982 (68%)

```
# Purpose: export table1a table1b word.
ft_table1A <- as_flex_table(table1A)
ft_table1B <- as_flex_table(table1B)

doc <- read_docx() %>%
  body_add_par("Table 1A. Baseline Characteristics by ABG Group", style = "heading 1") %>%
  body_add_flextable(ft_table1A) %>%
  body_add_par("Table 1B. Baseline Characteristics by VBG Group", style = "heading 1") %>%
  body_add_flextable(ft_table1B)

print(doc, target = results_path("Table1_ABG_VBG.docx"))

# Status factors (column labels are taken from factor levels)
subset_data <- subset_data %>%
  mutate(
    abg_status = factor(has_abg, levels = c(0, 1),
    
```

```

    labels = c("Did not get ABG", "Did get ABG")),
vbg_status = factor(has_vbg, levels = c(0, 1),
                    labels = c("Did not get VBG", "Did get VBG"))
)

# ABG table with "Everyone" column first
tbl1_abg <- subset_data %>%
  select(all_of(vars_baseline), abg_status) %>%
  gtsummary::tbl_summary(
    by = abg_status,
    type = list(sex_label ~ "categorical"),
    statistic = list(
      gtsummary::all_continuous() ~ "{mean} ± {sd}; {N_miss}/{N_obs} missing ({p_miss}%)",
      gtsummary::all_categorical() ~ "{n} ({p}%)"
    ),
    digits = list(gtsummary::all_continuous() ~ 1),
    missing = "no"
  ) %>%
  gtsummary::add_overall(last = FALSE, col_label = "Everyone") %>%
  gtsummary::modify_header(label = "***Variable***")

# VBG table with "Everyone" column first
tbl1_vbg <- subset_data %>%
  select(all_of(vars_baseline), vbg_status) %>%
  gtsummary::tbl_summary(
    by = vbg_status,
    type = list(sex_label ~ "categorical"),
    statistic = list(
      gtsummary::all_continuous() ~ "{mean} ± {sd}; {N_miss}/{N_obs} missing ({p_miss}%)",
      gtsummary::all_categorical() ~ "{n} ({p}%)"
    ),
    digits = list(gtsummary::all_continuous() ~ 1),
    missing = "no"
  ) %>%
  gtsummary::add_overall(last = FALSE, col_label = "Everyone") %>%
  gtsummary::modify_header(label = "***Variable***")

```

```

tbl1_abg <- tbl1_abg %>%
  modify_caption(strip_manual_table_number("Table 1A. Baseline summary: Everyone and ABG status"))
tbl1_vbg <- tbl1_vbg %>%
  modify_caption(strip_manual_table_number("Table 1B. Baseline summary: Everyone and VBG status"))

# merged table for Word export (Table 1: Everyone + ABG/VBG status)
tbl1 <- gtsummary::tbl_merge(
  list(tbl1_abg, tbl1_vbg),
  tab_spanner = c("**ABG status**", "**VBG status**")
)

tbl1_abg_pdf <- to_pdf_table(tbl1_abg, font_size = 7, landscape = FALSE, label_col_width = "2.0in",
                               longtable = TRUE)
tbl1_vbg_pdf <- to_pdf_table(tbl1_vbg, font_size = 7, landscape = FALSE, label_col_width = "2.0in",
                               longtable = TRUE)
tbl1_abg_pdf

```

Table 7: Baseline summary: Everyone and ABG status**

Variable	Everyone	**Did not get ABG** N = 3,264	**Did get ABG** N = 1,911
age_at_encounter	59.5 ± 17.6; 0.0/5,175.0 missing (0.0%)	58.4 ± 17.7; 0.0/3,264.0 missing (0.0%)	61.3 ± 17.2; 0.0/1,911.0 missing (0.0%)
curr_bmi	31.3 ± 8.5; 2,934.0/5,175.0 missing (56.7%)	32.6 ± 8.9; 1,821.0/3,264.0 missing (55.8%)	28.8 ± 7.0; 1,113.0/1,911.0 missing (58.2%)
sex_label			
Female	2,591 (50%)	1,737 (53%)	854 (45%)
Male	2,584 (50%)	1,527 (47%)	1,057 (55%)
race_ethnicity_label			
White	3,280 (63%)	2,009 (62%)	1,271 (67%)
Black or African American	884 (17%)	601 (18%)	283 (15%)
Hispanic	334 (6.5%)	250 (7.7%)	84 (4.4%)
Asian	83 (1.6%)	46 (1.4%)	37 (1.9%)
American Indian	36 (0.7%)	11 (0.3%)	25 (1.3%)
Pacific Islander	10 (0.2%)	8 (0.2%)	2 (0.1%)
Unknown	548 (11%)	339 (10%)	209 (11%)
location_label			
South	2,431 (47%)	1,374 (42%)	1,057 (55%)
Northeast	1,300 (25%)	921 (28%)	379 (20%)
Midwest	406 (7.8%)	246 (7.5%)	160 (8.4%)
West	1,038 (20%)	723 (22%)	315 (16%)
osa_label	879 (17%)	576 (18%)	303 (16%)
asthma_label	684 (13%)	464 (14%)	220 (12%)
copd_label	1,013 (20%)	579 (18%)	434 (23%)
chf_label	1,044 (20%)	587 (18%)	457 (24%)
nmd_label	233 (4.5%)	130 (4.0%)	103 (5.4%)

Table 7: Baseline summary: Everyone and ABG status** (*continued*)

Variable	Everyone	**Did not get ABG** N = 3,264	**Did get ABG** N = 1,911
phtn_label	434 (8.4%)	239 (7.3%)	195 (10%)
ckd_label	904 (17%)	519 (16%)	385 (20%)
diabetes_label	1,513 (29%)	954 (29%)	559 (29%)
encounter_type_label			
Emergency	1,710 (33%)	1,427 (44%)	283 (15%)
Inpatient	3,465 (67%)	1,837 (56%)	1,628 (85%)

tbl1_vbg_pdf

Table 8: Baseline summary: Everyone and VBG status**

Variable	Everyone	**Did not get VBG** N = 3,730	**Did get VBG** N = 1,445
age_at_encounter	59.5 ± 17.6; 0.0/5,175.0 missing (0.0%)	59.6 ± 17.6; 0.0/3,730.0 missing (0.0%)	59.0 ± 17.7; 0.0/1,445.0 missing (0.0%)
curr_bmi	31.3 ± 8.5; 2,934.0/5,175.0 missing (56.7%)	31.9 ± 8.6; 1,939.0/3,730.0 missing (52.0%)	28.9 ± 7.5; 995.0/1,445.0 missing (68.9%)
sex_label			
Female	2,591 (50%)	1,921 (52%)	670 (46%)
Male	2,584 (50%)	1,809 (48%)	775 (54%)
race_ethnicity_label			
White	3,280 (63%)	2,513 (67%)	767 (53%)
Black or African American	884 (17%)	616 (17%)	268 (19%)
Hispanic	334 (6.5%)	221 (5.9%)	113 (7.8%)
Asian	83 (1.6%)	54 (1.4%)	29 (2.0%)
American Indian	36 (0.7%)	13 (0.3%)	23 (1.6%)
Pacific Islander	10 (0.2%)	8 (0.2%)	2 (0.1%)
Unknown	548 (11%)	305 (8.2%)	243 (17%)
location_label			
South	2,431 (47%)	1,987 (53%)	444 (31%)
Northeast	1,300 (25%)	703 (19%)	597 (41%)
Midwest	406 (7.8%)	264 (7.1%)	142 (9.8%)
West	1,038 (20%)	776 (21%)	262 (18%)
osa_label	879 (17%)	648 (17%)	231 (16%)
asthma_label	684 (13%)	513 (14%)	171 (12%)
copd_label	1,013 (20%)	731 (20%)	282 (20%)
chf_label	1,044 (20%)	741 (20%)	303 (21%)
nmd_label	233 (4.5%)	175 (4.7%)	58 (4.0%)
phtn_label	434 (8.4%)	295 (7.9%)	139 (9.6%)
ckd_label	904 (17%)	628 (17%)	276 (19%)
diabetes_label	1,513 (29%)	1,034 (28%)	479 (33%)
encounter_type_label			
Emergency	1,710 (33%)	1,247 (33%)	463 (32%)
Inpatient	3,465 (67%)	2,483 (67%)	982 (68%)

```

# Purpose: co2 category definitions.
ABG_CO2_VAR <- "paco2"
V рG_CO2_VAR <- "vbg_co2"
ABG_CO2_LOW <- 35
ABG_CO2_HIGH <- 45
V рG_CO2_LOW <- 40
V рG_CO2_HIGH <- 50
CO2_SPEC <- list(
  ABG = list(var = ABG_CO2_VAR, normal_lo = ABG_CO2_LOW, normal_hi = ABG_CO2_HIGH),
  V рG = list(var = V рG_CO2_VAR, normal_lo = V рG_CO2_LOW, normal_hi = V рG_CO2_HIGH)
)
CO2_SPEC$ABG$ref <- (CO2_SPEC$ABG$normal_lo + CO2_SPEC$ABG$normal_hi) / 2
CO2_SPEC$V рG$ref <- (CO2_SPEC$V рG$normal_lo + CO2_SPEC$V рG$normal_hi) / 2
# Reference values for spline OR curves (midpoint of normal range)
ABG_CO2_REF <- CO2_SPEC$ABG$ref
V рG_CO2_REF <- CO2_SPEC$V рG$ref
CO2_CAT_LEVELS <- c("Normal", "Low", "High")
CO2_CAT_CONTRAST_LEVELS <- setdiff(CO2_CAT_LEVELS, "Normal")

make_co2_cat3 <- function(x, low_cut, high_cut) {
  x <- suppressWarnings(as.numeric(x))
  out <- dplyr::case_when(
    is.na(x) ~ NA_character_,
    x < low_cut ~ "Low",
    x > high_cut ~ "High",
    TRUE ~ "Normal"
  )
  factor(out, levels = CO2_CAT_LEVELS)
}

subset_data <- subset_data %>%
  mutate(
    pco2_cat_abg = make_co2_cat3(.data[[ABG_CO2_VAR]], ABG_CO2_LOW, ABG_CO2_HIGH),
    pco2_cat_vbg = make_co2_cat3(.data[[V рG_CO2_VAR]], V рG_CO2_LOW, V рG_CO2_HIGH)
  )

stopifnot("Normal" %in% levels(subset_data$pco2_cat_abg),

```

```

"Normal" %in% levels(subset_data$pco2_cat_vbg))

warn_low_counts <- function(cat, label) {
  tab <- table(cat, useNA = "no")
  if (length(tab) && any(tab < 10)) {
    warning(label, ": low counts in CO2 categories: ",
            paste(names(tab), tab, collapse = ", "), call. = FALSE)
  }
}
warn_low_counts(subset_data$pco2_cat_abg[subset_data$has_abg == 1 & !is.na(subset_data$paco2)],
                "ABG")
warn_low_counts(subset_data$pco2_cat_vbg[subset_data$has_vbg == 1 & !is.na(subset_data$vbg_co2)],
                "VBG")

# Fail-fast if any binary CO2 indicator references remain in this QMD
qmd_path <- resolve_current_qmd()
txt <- readLines(qmd_path, warn = FALSE)
pat_var <- paste0("hypercap", "_on_")
if (any(grepl(pat_var, txt))) {
  stop("Binary CO2 indicator variable references remain in the QMD.")
}

# ABG cohort (has_abg == 1)
tbl2_abg <- subset_data %>%
  filter(has_abg == 1) %>%
  select(all_of(vars_abg), pco2_cat_abg) %>%
  gtsummary::tbl_summary(
    by = pco2_cat_abg,
    type = list(sex_label ~ "categorical"),
    statistic = list(
      gtsummary::all_continuous() ~ "{mean} ± {sd}; {N_miss}/{N_obs} missing ({p_miss}%)",
      gtsummary::all_categorical() ~ "{n} ({p}%)"
    ),
    digits = list(gtsummary::all_continuous() ~ 1),
    missing = "no"
  ) %>%
  gtsummary::modify_header(
    label = "**Variable**",

```

```

stat_1 = "##Normal##",
stat_2 = "##Low##",
stat_3 = "##High##"
) %>%
gtsummary::modify_caption("Baseline summary by CO2 category within ABG cohort")

# VBG cohort (has_vbg == 1)
tbl2_vbg <- subset_data %>%
filter(has_vbg == 1) %>%
select(all_of(vars_vbg), pco2_cat_vbg) %>%
gtsummary::tbl_summary(
  by = pco2_cat_vbg,
  type = list(sex_label ~ "categorical"),
  statistic = list(
    gtsummary::all_continuous() ~ "{mean} ± {sd}; {N_miss}/{N_obs} missing ({p_miss}%)",
    gtsummary::all_categorical() ~ "{n} ({p}%)"
  ),
  digits = list(gtsummary::all_continuous() ~ 1),
  missing = "no"
) %>%
gtsummary::modify_header(
  label = "##Variable##",
  stat_1 = "##Normal##",
  stat_2 = "##Low##",
  stat_3 = "##High##"
) %>%
gtsummary::modify_caption("Baseline summary by CO2 category within VBG cohort")

# Render ABG and VBG as separate PDF tables to avoid clipped multi-spanner output.
tbl2_abg_pdf <- to_pdf_table(tbl2_abg, font_size = 7, landscape = FALSE, label_col_width = "2.0in",
                               longtable = TRUE)
tbl2_abg_pdf

```

Table 9: Baseline summary by CO2 category within ABG cohort

##Variable##	##Normal##	##Low##	##High##
age_at_encounter	61.9 ± 16.8; 0.0/825.0 missing (0.0%)	59.6 ± 18.4; 0.0/521.0 missing (0.0%)	61.9 ± 16.5; 0.0/565.0 missing (0.0%)
curr_bmi	28.5 ± 6.8; 457.0/825.0 missing (55.4%)	28.1 ± 6.5; 318.0/521.0 missing (61.0%)	29.9 ± 7.7; 338.0/565.0 missing (59.8%)

Table 9: Baseline summary by CO2 category within ABG cohort (*continued*)

Variable	**Normal**	**Low**	**High**
sex_label			
Female	372 (45%)	225 (43%)	257 (45%)
Male	453 (55%)	296 (57%)	308 (55%)
race_ethnicity_label			
White	556 (67%)	330 (63%)	385 (68%)
Black or African American	115 (14%)	82 (16%)	86 (15%)
Hispanic	36 (4.4%)	29 (5.6%)	19 (3.4%)
Asian	12 (1.5%)	14 (2.7%)	11 (1.9%)
American Indian	15 (1.8%)	8 (1.5%)	2 (0.4%)
Pacific Islander	2 (0.2%)	0 (0%)	0 (0%)
Unknown	89 (11%)	58 (11%)	62 (11%)
location_label			
South	467 (57%)	284 (55%)	306 (54%)
Northeast	148 (18%)	84 (16%)	147 (26%)
Midwest	69 (8.4%)	47 (9.0%)	44 (7.8%)
West	141 (17%)	106 (20%)	68 (12%)
osa_label	116 (14%)	53 (10%)	134 (24%)
asthma_label	84 (10%)	58 (11%)	78 (14%)
copd_label	154 (19%)	83 (16%)	197 (35%)
chf_label	175 (21%)	115 (22%)	167 (30%)
nmd_label	48 (5.8%)	25 (4.8%)	30 (5.3%)
phtn_label	82 (9.9%)	45 (8.6%)	68 (12%)
ckd_label	157 (19%)	112 (21%)	116 (21%)
diabetes_label	243 (29%)	145 (28%)	171 (30%)
encounter_type_label			
Emergency	127 (15%)	65 (12%)	91 (16%)
Inpatient	698 (85%)	456 (88%)	474 (84%)
paco2	39.8 ± 3.0; 0.0/825.0 missing (0.0%)	29.6 ± 4.5; 0.0/521.0 missing (0.0%)	59.1 ± 19.3; 0.0/565.0 missing (0.0%)

```
tbl2_vbg_pdf <- to_pdf_table(tbl2_vbg, font_size = 7, landscape = FALSE, label_col_width = "2.0in",
                                longtable = TRUE)
tbl2_vbg_pdf
```

Table 10: Baseline summary by CO2 category within VBG cohort

Variable	**Normal**	**Low**	**High**
age_at_encounter	57.2 ± 18.3; 0.0/647.0 missing (0.0%)	59.4 ± 17.7; 0.0/402.0 missing (0.0%)	61.6 ± 16.3; 0.0/396.0 missing (0.0%)
curr_bmi	29.7 ± 7.6; 465.0/647.0 missing (71.9%)	27.7 ± 6.3; 249.0/402.0 missing (61.9%)	29.4 ± 8.6; 281.0/396.0 missing (71.0%)
sex_label			
Female	323 (50%)	177 (44%)	170 (43%)
Male	324 (50%)	225 (56%)	226 (57%)
race_ethnicity_label			
White	325 (50%)	209 (52%)	233 (59%)

Table 10: Baseline summary by CO2 category within VBG cohort (*continued*)

Variable	**Normal**	**Low**	**High**
Black or African American	115 (18%)	71 (18%)	82 (21%)
Hispanic	59 (9.1%)	33 (8.2%)	21 (5.3%)
Asian	16 (2.5%)	5 (1.2%)	8 (2.0%)
American Indian	9 (1.4%)	11 (2.7%)	3 (0.8%)
Pacific Islander	1 (0.2%)	1 (0.2%)	0 (0%)
Unknown	122 (19%)	72 (18%)	49 (12%)
location_label			
South	205 (32%)	101 (25%)	138 (35%)
Northeast	273 (42%)	154 (38%)	170 (43%)
Midwest	61 (9.4%)	33 (8.2%)	48 (12%)
West	108 (17%)	114 (28%)	40 (10%)
osa_label	95 (15%)	53 (13%)	83 (21%)
asthma_label	66 (10%)	50 (12%)	55 (14%)
copd_label	89 (14%)	74 (18%)	119 (30%)
chf_label	110 (17%)	88 (22%)	105 (27%)
nmd_label	19 (2.9%)	13 (3.2%)	26 (6.6%)
phtn_label	47 (7.3%)	37 (9.2%)	55 (14%)
ckd_label	103 (16%)	103 (26%)	70 (18%)
diabetes_label	194 (30%)	157 (39%)	128 (32%)
encounter_type_label			
Emergency	240 (37%)	112 (28%)	111 (28%)
Inpatient	407 (63%)	290 (72%)	285 (72%)
vbg_co2	44.5 ± 3.0; 0.0/647.0 missing (0.0%)	33.1 ± 5.5; 0.0/402.0 missing (0.0%)	61.6 ± 13.6; 0.0/396.0 missing (0.0%)

```
# Keep merged object for Word export only.
tbl2 <- gtsummary::tbl_merge(
  tbls = list(tbl2_abg, tbl2_vbg),
  tab_spanner = c("**ABG (PaCO2)**", "**VBG (PvCO2)**")
) %>%
  gtsummary::modify_caption(strip_manual_table_number("**Table 2. Baseline summary by CO2 category within ABG and VBG cohorts**"))

# Purpose: export table1 tbl2 word.
library(gtsummary)
library(flextable)
library(officer)

# gtsummary objects (example: tbl1, tbl2)
ft1 <- as_flex_table(tbl1)
```

Table 11: Table 2a. Crude outcomes by CO₂ category

Cohort	Outcome	Normal	Low	High
ABG	IMV	174/825 (21.1%)	145/521 (27.8%)	144/565 (25.5%)
ABG	NIV	60/825 (7.3%)	38/521 (7.3%)	85/565 (15.0%)
ABG	Death (60d)	110/825 (13.3%)	103/521 (19.8%)	109/565 (19.3%)
ABG	Hypercapnic RF	33/825 (4.0%)	28/521 (5.4%)	145/565 (25.7%)
VBG	IMV	72/647 (11.1%)	63/402 (15.7%)	73/396 (18.4%)
VBG	NIV	31/647 (4.8%)	23/402 (5.7%)	46/396 (11.6%)
VBG	Death (60d)	69/647 (10.7%)	68/402 (16.9%)	63/396 (15.9%)
VBG	Hypercapnic RF	26/647 (4.0%)	16/402 (4.0%)	91/396 (23.0%)

```

ft2 <- as_flex_table(tbl2)

doc1 <- read_docx() %>%
  body_add_par("Table 1. Baseline summary: Everyone, ABG status, and VBG status",
               style = "heading 1") %>%
  body_add_flextable(ft1)
print(doc1, target = results_path("Table1.docx"))

doc2 <- read_docx() %>%
  body_add_par("Table 2. Baseline summary by CO2 category within ABG and VBG cohorts",
               style = "heading 1") %>%
  body_add_flextable(ft2)
print(doc2, target = results_path("Table2.docx"))

```

1.3 Three-level PCO₂ categories (unweighted)

Three groups using low/normal/high CO₂ categories

```

# Purpose: or data three level unweighted.
stopifnot(all(c("pco2_cat_abg", "pco2_cat_vbg") %in% names(subset_data)))

library(broom)
library(tidyr)
library(dplyr)

```

```

run_logit <- function(data, outcome, exposure, group_name, adj_vars = NULL, model_type = "Crude") {
  f <- if (length(adj_vars)) {
    reformulate(c(exposure, adj_vars), response = outcome)
  } else {
    as.formula(paste(outcome, "~", exposure))
  }
  fit_res <- fit_with_diagnostics(
    function() glm(f, data = data, family = binomial, control = glm.control(maxit = 50)),
    context = make_context(
      stage = "outcome",
      component = "cat3",
      analysis_variant = "unweighted",
      model_type = "cat3",
      group = group_name,
      outcome = outcome,
      imputation = NA_integer_,
      batch = NA_integer_
    )
  )
  append_outcome_diag(fit_res$diag)
  if (is.null(fit_res$fit)) {
    stop("run_logit: model fit failed for outcome=", outcome,
         " exposure=", exposure, " group=", group_name)
  }
  tidy(fit_res$fit, exponentiate = TRUE, conf.int = TRUE) %>%
    filter(term != "(Intercept)", startsWith(term, exposure)) %>%
    mutate(
      outcome = outcome,
      group = group_name,
      model = model_type
    )
}
}

outcomes_unw <- c("imv_proc", "niv_proc", "death_60d", "hypercap_resp_failure")

unw_three_level_forms <- list(
  "ABG 3-level: IMV ~ CO2 category + X"      = reformulate(c("pco2_cat_abg", adj_core), response = "imv_proc"),

```

```

"ABG 3-level: NIV ~ CO2 category + X"      = reformulate(c("pc02_cat_abg", adj_core), response = "niv_proc"),
"ABG 3-level: Death60d ~ CO2 category + X" = reformulate(c("pc02_cat_abg", adj_core), response = "death_60d"),
"ABG 3-level: HCRF ~ CO2 category + X"      = reformulate(c("pc02_cat_abg", adj_core), response =
  ↵  "hypercap_resp_failure"),
"VBG 3-level: IMV ~ CO2 category + X"       = reformulate(c("pc02_cat_vbg", adj_core), response = "imv_proc"),
"VBG 3-level: NIV ~ CO2 category + X"       = reformulate(c("pc02_cat_vbg", adj_core), response = "niv_proc"),
"VBG 3-level: Death60d ~ CO2 category + X" = reformulate(c("pc02_cat_vbg", adj_core), response = "death_60d"),
"VBG 3-level: HCRF ~ CO2 category + X"       = reformulate(c("pc02_cat_vbg", adj_core), response =
  ↵  "hypercap_resp_failure")
)
register_model_diagrams(unw_three_level_forms)

unw_results_crude <- bind_rows(
  lapply(outcomes_unw, function(o) run_logit(subset_data, o, "pc02_cat_abg", "ABG")),
  lapply(outcomes_unw, function(o) run_logit(subset_data, o, "pc02_cat_vbg", "VBG"))
)
unw_results_adj <- bind_rows(
  lapply(outcomes_unw, function(o) run_logit(subset_data, o, "pc02_cat_abg", "ABG", adj_core, "Adjusted")),
  lapply(outcomes_unw, function(o) run_logit(subset_data, o, "pc02_cat_vbg", "VBG", adj_core, "Adjusted"))
)

unw_threelvel_results <- unw_results_adj %>%
  mutate(method = "Unweighted adjusted")

unw_combined_or_df <- unw_results_adj %>%
  mutate(
    outcome = recode(outcome,
      imv_proc = "Intubation",
      niv_proc = "NIV",
      death_60d = "Death (60d)",
      hypercap_resp_failure = "Hypercapnic RF")
  )
unw_combined_or_df <- map_or_exposure(unw_combined_or_df, "or-plot-three-level-unweighted") |>
  select(outcome, group, exposure, estimate, conf.low, conf.high)

# Purpose: or plot three level unweighted.
library(scales)

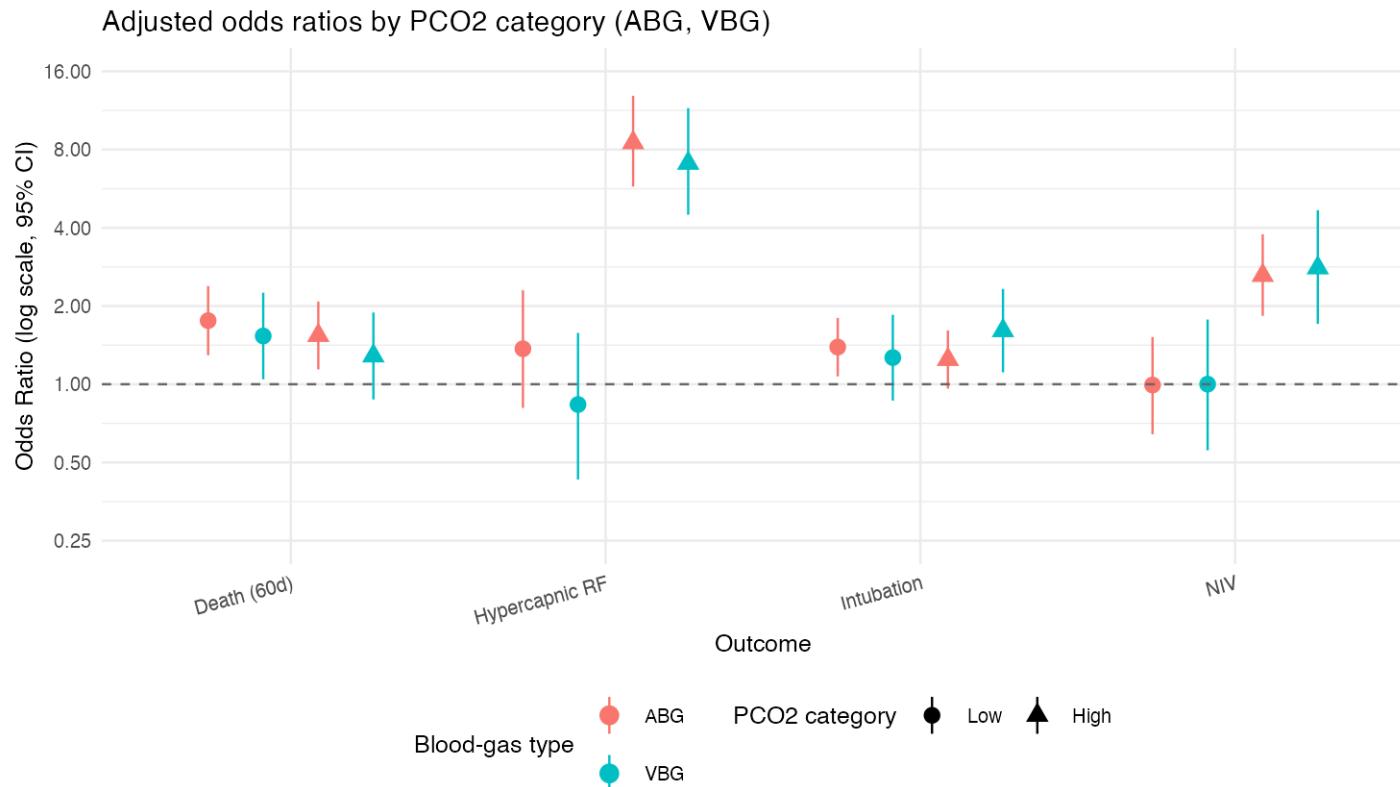
```

```

unw_combined_or_df$group <- factor(
  unw_combined_or_df$group,
  levels = c("ABG", "VBG")
)

unw_plot_df <- build_or_plot_df(unw_combined_or_df, "or-plot-three-level-unweighted",
                                 expected_exposure_levels = CO2_CAT_CONTRAST_LEVELS)
stopifnot(is.data.frame(unw_plot_df))
unw_axis_spec <- compute_or_axis_spec(unw_plot_df, lo_col = "conf.low", hi_col = "conf.high",
                                       default_limits = OR_XLIM)
unw_p_or <- plot_or_safe(
  unw_plot_df,
  plot_name = "or-plot-three-level-unweighted",
  axis_spec = unw_axis_spec,
  title = "Adjusted odds ratios by PCO2 category (ABG, VBG)",
  caption = paste(
    "Adjusted for age, sex, race/ethnicity, location, and encounter type.",
    "Reference = patients in the normal PCO2 range.",
    "Low: <35 mmHg (ABG) or <40 mmHg (VBG); High: >45 mmHg (ABG) or >50 mmHg (VBG).",
    "Because the underlying cohorts differ (ABG, VBG), denominators are not identical across groups.",
    sep = "\n"
  )
)
print_plot_once(unw_p_or, "or-plot-three-level-unweighted", width = 7.5, height = 4.8)

```



Adjusted for age, sex, race/ethnicity, location, and encounter type.

Reference = patients in the normal PCO₂ range.

Low: <35 mmHg (ABG) or <40 mmHg (VBG); High: >45 mmHg (ABG) or >50 mmHg (VBG).

Because the underlying cohorts differ (ABG, VBG), denominators are not identical across groups.

1.4 Observed outcome rates by CO₂ bin (unweighted)

```
# Purpose: summarize raw event rates in 1-mmHg CO2 bins to inspect local jumps.
```

```
co2_rate_unw_abg <- compute_co2_bin_rates(
  df = subset_data |> dplyr::filter(has_abg == 1),
  co2_var = "paco2",
  outcomes = CO2_RATE_OUTCOMES,
  outcome_labels = CO2_RATE_OUTCOME_LABELS,
  weight_var = NULL,
```

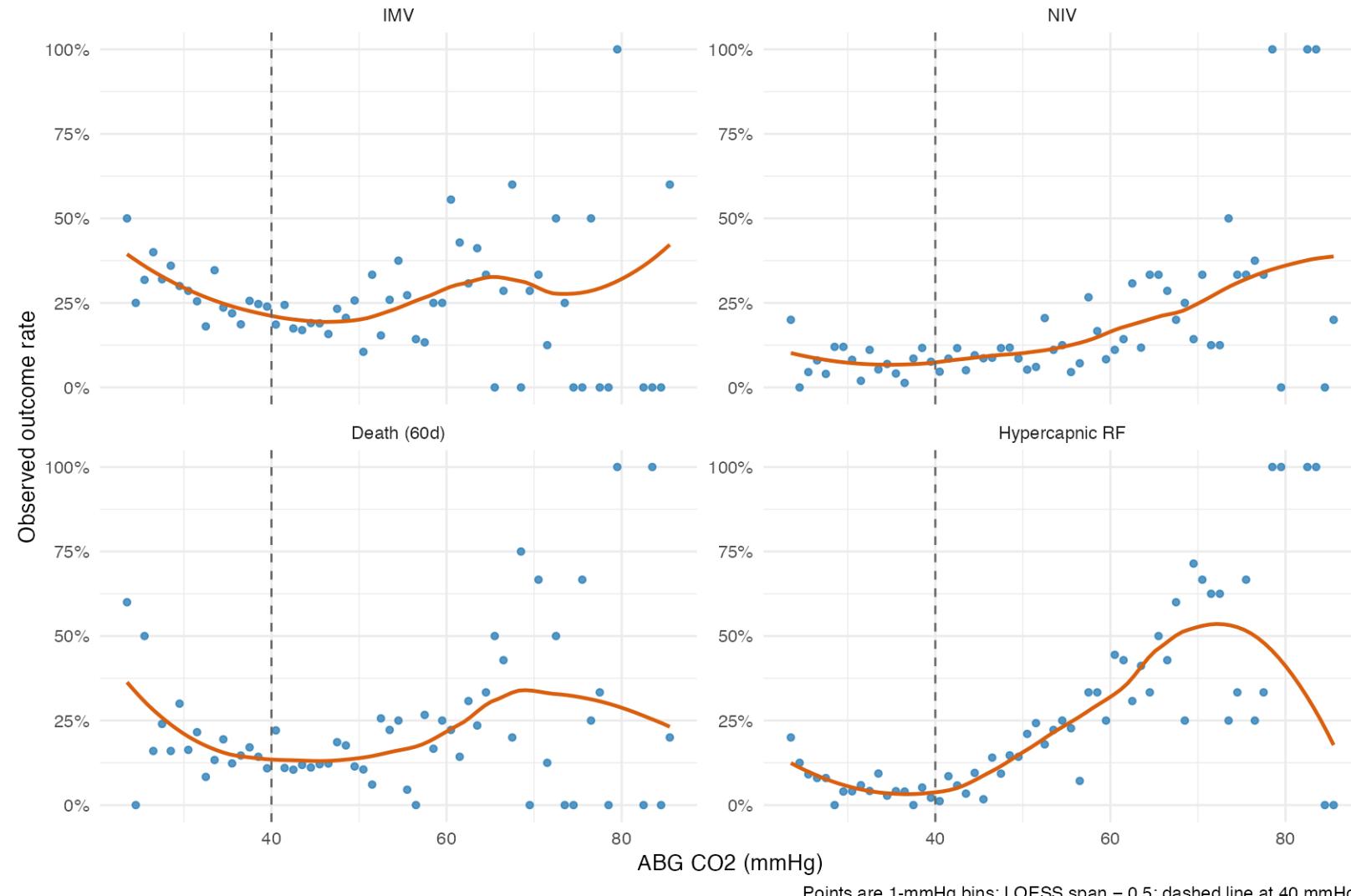
```

q = c(0.02, 0.98),
bin_width = 1,
cohort_label = "ABG"
)
co2_rate_unw_vbg <- compute_co2_bin_rates(
  df = subset_data |> dplyr::filter(has_vbg == 1),
  co2_var = "vbg_co2",
  outcomes = CO2_RATE_OUTCOMES,
  outcome_labels = CO2_RATE_OUTCOME_LABELS,
  weight_var = NULL,
  q = c(0.02, 0.98),
  bin_width = 1,
  cohort_label = "VBG"
)
write_csv_safely(co2_rate_unw_abg, results_path("co2_binned_rates_unweighted_abg.csv"), row_names = FALSE)
write_csv_safely(co2_rate_unw_vbg, results_path("co2_binned_rates_unweighted_vbg.csv"), row_names = FALSE)

p_co2_rate_unw_abg <- plot_co2_bin_rate_panel(
  co2_rate_unw_abg,
  title = "Observed outcome rates by ABG CO2 bin (unweighted)",
  x_label = "ABG CO2 (mmHg)",
  vline_at = ABG_CO2_REF
)
p_co2_rate_unw_vbg <- plot_co2_bin_rate_panel(
  co2_rate_unw_vbg,
  title = "Observed outcome rates by VBG CO2 bin (unweighted)",
  x_label = "VBG CO2 (mmHg)",
  vline_at = VBG_CO2_REF
)
print_plot_once(p_co2_rate_unw_abg, "co2-binned-rates-unweighted-abg", width = 8.5, height = 6.0)

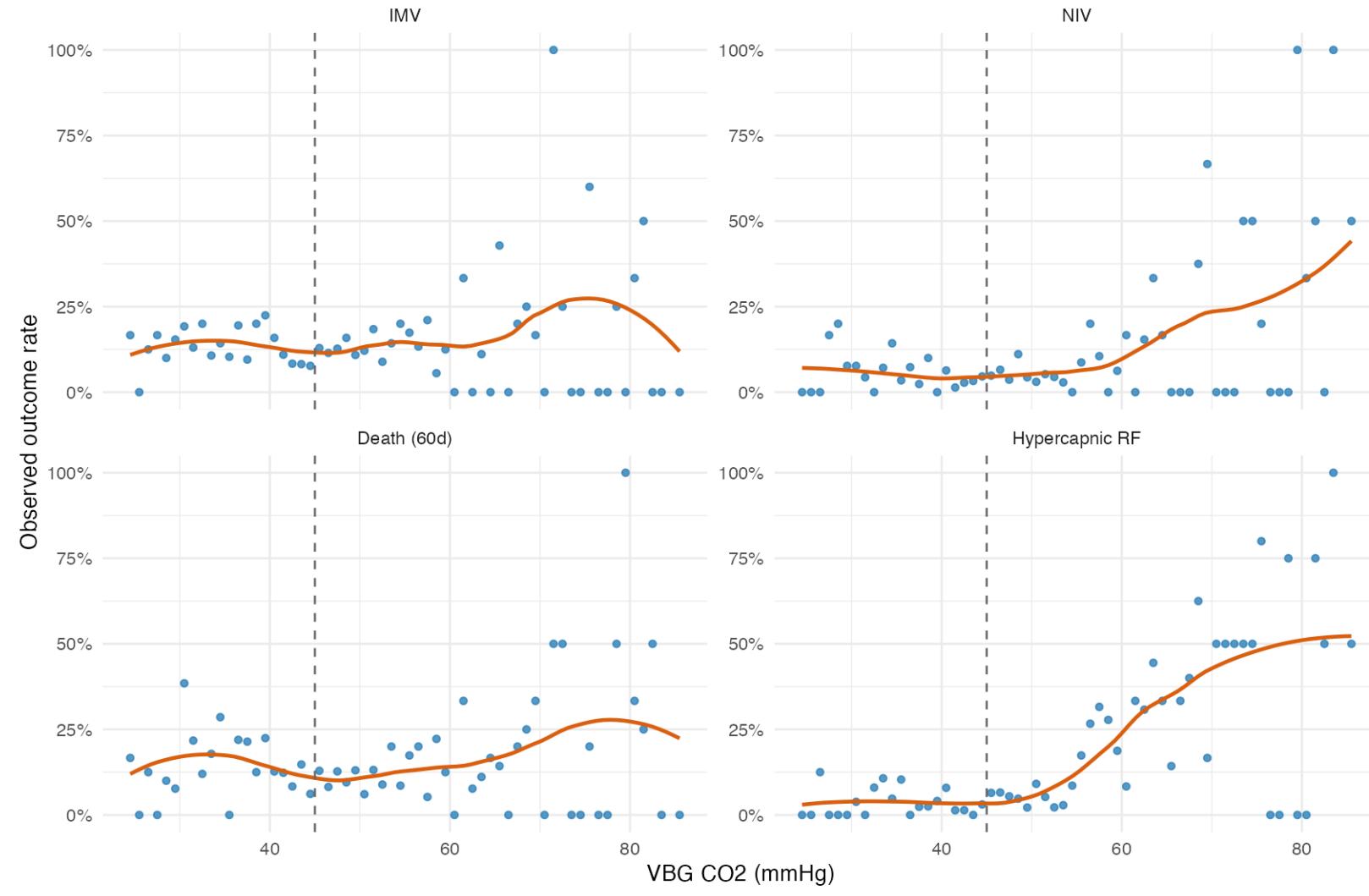
```

Observed outcome rates by ABG CO₂ bin (unweighted)



```
print_plot_once(p_co2_rate_unw_vbg, "co2-binned-rates-unweighted-vbg", width = 8.5, height = 6.0)
```

Observed outcome rates by VBG CO₂ bin (unweighted)



1.5 Restricted cubic spline regressions (unweighted)

Spline curves are shown as odds ratios relative to CO₂_ref (midpoint of the normal range), holding covariates at the reference profile.

```
# ABG spline dataset
subset_data_abg <- subset_data %>%
  filter(has_abg == 1, !is.na(paco2)) %>%
  select(paco2, imv_proc, niv_proc, death_60d, hypercap_resp_failure, all_of(adj_core)) %>%
  filter(complete.cases(.))
```

1.5.1 Unweighted, Restricted Cubic Spline Regression - ABG by PaCO2

```
# Purpose: rcs abg unweighted models.
make_spline_fml <- function(outcome, co2_var, adj_vars) {
  spline_term <- if (SPLINE_BASIS == "rcs") {
    sprintf("rms::rcs(%s, %d)", co2_var, SPLINE_DF)
  } else {
    sprintf("splines::ns(%s, %d)", co2_var, SPLINE_DF)
  }
  stats::as.formula(
    paste0(outcome, " ~ ", spline_term,
           if (length(adj_vars)) paste0(" + ", paste(adj_vars, collapse = " + "))) else ""))
}
}

#| code-block-title: "Unweighted ABG spline models (adjusted)"
abg_spline_forms <- list(
  "ABG spline (adjusted): IMV ~ CO2 spline + X"      = make_spline_fml("imv_proc", "paco2", adj_core),
  "ABG spline (adjusted): NIV ~ CO2 spline + X"      = make_spline_fml("niv_proc", "paco2", adj_core),
  "ABG spline (adjusted): Death60d ~ CO2 spline + X" = make_spline_fml("death_60d", "paco2", adj_core),
  "ABG spline (adjusted): HCRF ~ CO2 spline + X"     = make_spline_fml("hypercap_resp_failure", "paco2", adj_core)
)
register_model_diagrams(abg_spline_forms)

co2_seq_abg <- stats::quantile(subset_data_abg$paco2, probs = c(0.02, 0.98), na.rm = TRUE)
grid_abg_info_unw <- make_co2_grid_ref(
  "paco2",
  seq(co2_seq_abg[1], co2_seq_abg[2], length.out = SPLINE_GRID_N),
  x_ref_abg,
  ABG_CO2_REF
```

```

)
grid_abg_unw <- grid_abg_info_unw$grid
ref_idx_abg_unw <- grid_abg_info_unw$ref_idx
co2_ref_abg_unw <- grid_abg_info_unw$co2_ref

fit_spline_glm <- function(outcome, co2_var, data, group_label) {
  fit_res <- fit_with_diagnostics(
    function() glm(make_spline_fml(outcome, co2_var, adj_core),
                  data = data, family = binomial,
                  control = glm.control(maxit = 50)),
    context = make_context(
      stage = "outcome",
      component = "spline",
      analysis_variant = "unweighted",
      model_type = "spline",
      group = group_label,
      outcome = outcome,
      imputation = NA_integer_,
      batch = NA_integer_
    )
  )
  append_outcome_diag(fit_res$diag)
  fit_res$fit
}

fit_imv <- fit_spline_glm("imv_proc", "paco2", subset_data_abg, "ABG")
fit_niv <- fit_spline_glm("niv_proc", "paco2", subset_data_abg, "ABG")
fit_death <- fit_spline_glm("death_60d", "paco2", subset_data_abg, "ABG")
fit_hcrcf <- fit_spline_glm("hypercap_resp_failure", "paco2", subset_data_abg, "ABG")
if (any(vapply(list(fit_imv, fit_niv, fit_death, fit_hcrcf), is.null, logical(1)))) {
  stop("Unweighted ABG spline fits failed; see model_fit_diagnostics.csv.")
}

pred_imv <- predict_or_curve_from_fit(fit_imv, grid_abg_unw, ref_idx_abg_unw, "paco2")
pred_niv <- predict_or_curve_from_fit(fit_niv, grid_abg_unw, ref_idx_abg_unw, "paco2")
pred_death <- predict_or_curve_from_fit(fit_death, grid_abg_unw, ref_idx_abg_unw, "paco2")
pred_hcrcf <- predict_or_curve_from_fit(fit_hcrcf, grid_abg_unw, ref_idx_abg_unw, "paco2")

```

```
## Plotting deferred until VBG curves are computed so axes can be shared.
```

1.5.2 Unweighted, Restricted Cubic Spline - VBG

```
# --- VBG dataset ---
subset_data_vbg <- subset_data %>%
  dplyr::filter(has_vbg == 1, !is.na(vbg_co2)) %>%
  dplyr::select(vbg_co2, imv_proc, niv_proc, death_60d, hypercap_resp_failure, all_of(adj_core)) %>%
  dplyr::filter(complete.cases(.))

# Purpose: rcs vbg unweighted models.
vbg_spline_forms <- list(
  "VBG spline (adjusted): IMV ~ CO2 spline + X"      = make_spline_fml("imv_proc", "vbg_co2", adj_core),
  "VBG spline (adjusted): NIV ~ CO2 spline + X"      = make_spline_fml("niv_proc", "vbg_co2", adj_core),
  "VBG spline (adjusted): Death60d ~ CO2 spline + X" = make_spline_fml("death_60d", "vbg_co2", adj_core),
  "VBG spline (adjusted): HCRF ~ CO2 spline + X"     = make_spline_fml("hypercap_resp_failure", "vbg_co2", adj_core)
)
register_model_diagrams(vbg_spline_forms)

co2_seq_vbg <- stats::quantile(subset_data_vbg$vbg_co2, probs = c(0.02, 0.98), na.rm = TRUE)
grid_vbg_info_unw <- make_co2_grid_ref(
  "vbg_co2",
  seq(co2_seq_vbg[1], co2_seq_vbg[2], length.out = SPLINE_GRID_N),
  x_ref_vbg,
  VBG_CO2_REF
)
grid_vbg_unw <- grid_vbg_info_unw$grid
ref_idx_vbg_unw <- grid_vbg_info_unw$ref_idx
co2_ref_vbg_unw <- grid_vbg_info_unw$co2_ref

fit_imv_vbg <- fit_spline_glm("imv_proc", "vbg_co2", subset_data_vbg, "VBG")
fit_niv_vbg <- fit_spline_glm("niv_proc", "vbg_co2", subset_data_vbg, "VBG")
fit_death_vbg <- fit_spline_glm("death_60d", "vbg_co2", subset_data_vbg, "VBG")
fit_hcrf_vbg <- fit_spline_glm("hypercap_resp_failure", "vbg_co2", subset_data_vbg, "VBG")
if (any(vapply(list(fit_imv_vbg, fit_niv_vbg, fit_death_vbg, fit_hcrf_vbg), is.null, logical(1)))) {
  stop("Unweighted VBG spline fits failed; see model_fit_diagnostics.csv.")
}
```

```

pred_imv_vbg <- predict_or_curve_from_fit(fit_imv_vbg, grid_vbg_unw, ref_idx_vbg_unw, "vbg_co2")
pred_niv_vbg <- predict_or_curve_from_fit(fit_niv_vbg, grid_vbg_unw, ref_idx_vbg_unw, "vbg_co2")
pred_death_vbg <- predict_or_curve_from_fit(fit_death_vbg, grid_vbg_unw, ref_idx_vbg_unw, "vbg_co2")
pred_hcrf_vbg <- predict_or_curve_from_fit(fit_hcrf_vbg, grid_vbg_unw, ref_idx_vbg_unw, "vbg_co2")
axis_unw_common <- compute_or_axis_spec(
  list(pred_imv, pred_niv, pred_death, pred_hcrf,
    pred_imv_vbg, pred_niv_vbg, pred_death_vbg, pred_hcrf_vbg),
  lo_col = "LCL", hi_col = "UCL"
)

plot_imv <- ggplot(pred_imv, aes(x = paco2, y = OR)) +
  geom_line(color = "blue", linewidth = 1.2) +
  geom_ribbon(aes(ymin = LCL, ymax = UCL), fill = "blue", alpha = 0.2) +
  geom_hline(yintercept = 1, linetype = "dashed", color = "grey40") +
  or_axis_scale(axis_unw_common) +
  labs(title = "Intubation (adjusted)", x = "PaCO2 (mmHg)",
    y = paste0("Odds ratio (ref PaCO2 = ", co2_ref_abg_unw, "; log scale)")) +
  theme_minimal()

plot_niv <- ggplot(pred_niv, aes(x = paco2, y = OR)) +
  geom_line(color = "green", linewidth = 1.2) +
  geom_ribbon(aes(ymin = LCL, ymax = UCL), fill = "green", alpha = 0.2) +
  geom_hline(yintercept = 1, linetype = "dashed", color = "grey40") +
  or_axis_scale(axis_unw_common) +
  labs(title = "NIV (adjusted)", x = "PaCO2 (mmHg)",
    y = paste0("Odds ratio (ref PaCO2 = ", co2_ref_abg_unw, "; log scale)")) +
  theme_minimal()

plot_death <- ggplot(pred_death, aes(x = paco2, y = OR)) +
  geom_line(color = "red", linewidth = 1.2) +
  geom_ribbon(aes(ymin = LCL, ymax = UCL), fill = "red", alpha = 0.2) +
  geom_hline(yintercept = 1, linetype = "dashed", color = "grey40") +
  or_axis_scale(axis_unw_common) +
  labs(title = "Death (60d, adjusted)", x = "PaCO2 (mmHg)",
    y = paste0("Odds ratio (ref PaCO2 = ", co2_ref_abg_unw, "; log scale)")) +
  theme_minimal()

```

```

plot_hcrcf <- ggplot(pred_hcrcf, aes(x = paco2, y = OR)) +
  geom_line(color = "purple", linewidth = 1.2) +
  geom_ribbon(aes(ymin = LCL, ymax = UCL), fill = "purple", alpha = 0.2) +
  geom_hline(yintercept = 1, linetype = "dashed", color = "grey40") +
  or_axis_scale(axis_unw_common) +
  labs(title = "Hypercapnic RF (adjusted)", x = "PaCO2 (mmHg)",
       y = paste0("Odds ratio (ref PaCO2 = ", co2_ref_abg_unw, "; log scale)")) +
  theme_minimal()

plot_imv_vbg <- ggplot(pred_imv_vbg, aes(x = vbg_co2, y = OR)) +
  geom_line(color = "blue") +
  geom_ribbon(aes(ymin = LCL, ymax = UCL), fill = "blue", alpha = 0.2) +
  geom_hline(yintercept = 1, linetype = "dashed", color = "grey40") +
  or_axis_scale(axis_unw_common) +
  labs(title = "IMV (adjusted)", x = "VBG CO2 (mmHg)",
       y = paste0("Odds ratio (ref VBG CO2 = ", co2_ref_vbg_unw, "; log scale)")) +
  theme_minimal()

plot_niv_vbg <- ggplot(pred_niv_vbg, aes(x = vbg_co2, y = OR)) +
  geom_line(color = "green") +
  geom_ribbon(aes(ymin = LCL, ymax = UCL), fill = "green", alpha = 0.2) +
  geom_hline(yintercept = 1, linetype = "dashed", color = "grey40") +
  or_axis_scale(axis_unw_common) +
  labs(title = "NIV (adjusted)", x = "VBG CO2 (mmHg)",
       y = paste0("Odds ratio (ref VBG CO2 = ", co2_ref_vbg_unw, "; log scale)")) +
  theme_minimal()

plot_death_vbg <- ggplot(pred_death_vbg, aes(x = vbg_co2, y = OR)) +
  geom_line(color = "red") +
  geom_ribbon(aes(ymin = LCL, ymax = UCL), fill = "red", alpha = 0.2) +
  geom_hline(yintercept = 1, linetype = "dashed", color = "grey40") +
  or_axis_scale(axis_unw_common) +
  labs(title = "Death (60d, adjusted)", x = "VBG CO2 (mmHg)",
       y = paste0("Odds ratio (ref VBG CO2 = ", co2_ref_vbg_unw, "; log scale)")) +
  theme_minimal()

```

```

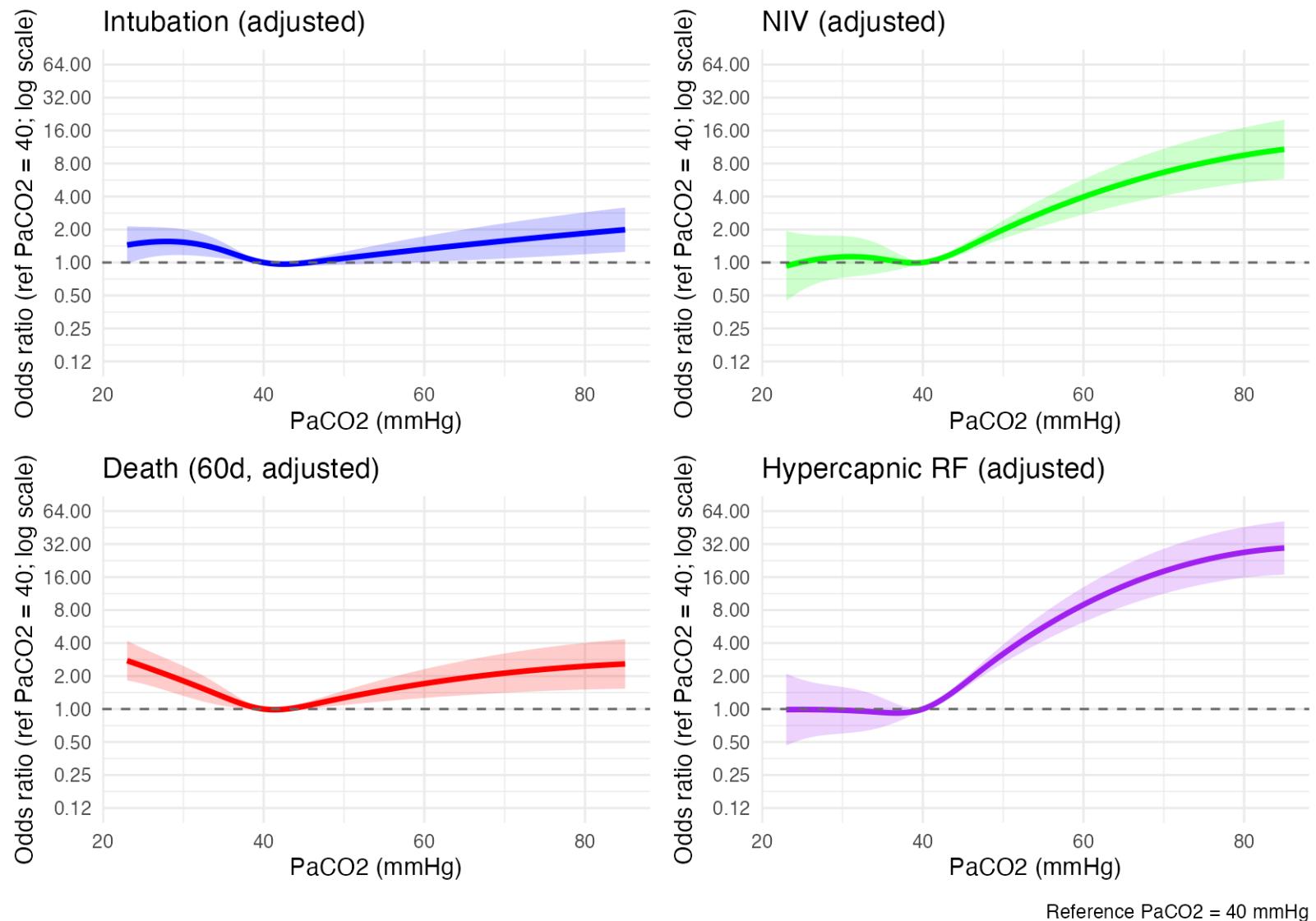
plot_hcrcf_vbg <- ggplot(pred_hcrcf_vbg, aes(x = vbg_co2, y = OR)) +
  geom_line(color = "purple") +
  geom_ribbon(aes(ymin = LCL, ymax = UCL), fill = "purple", alpha = 0.2) +
  geom_hline(yintercept = 1, linetype = "dashed", color = "grey40") +
  or_axis_scale(axis_unw_common) +
  labs(title = "Hypercapnic RF (adjusted)", x = "VBG CO2 (mmHg)",
       y = paste0("Odds ratio (ref VBG CO2 = ", co2_ref_vbg_unw, "; log scale)")) +
  theme_minimal()

unw_abg_panel <- (plot_imv | plot_niv) / (plot_death | plot_hcrcf) +
  plot_annotation(caption = paste0("Reference PaCO2 = ", co2_ref_abg_unw, " mmHg"))

unw_vbg_panel <- ((plot_imv_vbg | plot_niv_vbg) /
  (plot_death_vbg | plot_hcrcf_vbg)) +
  plot_annotation(
    title = paste0("Adjusted odds ratios by VBG CO2 (ref = ", co2_ref_vbg_unw, ")"),
    caption = paste0("Reference VBG CO2 = ", co2_ref_vbg_unw, " mmHg")
  )

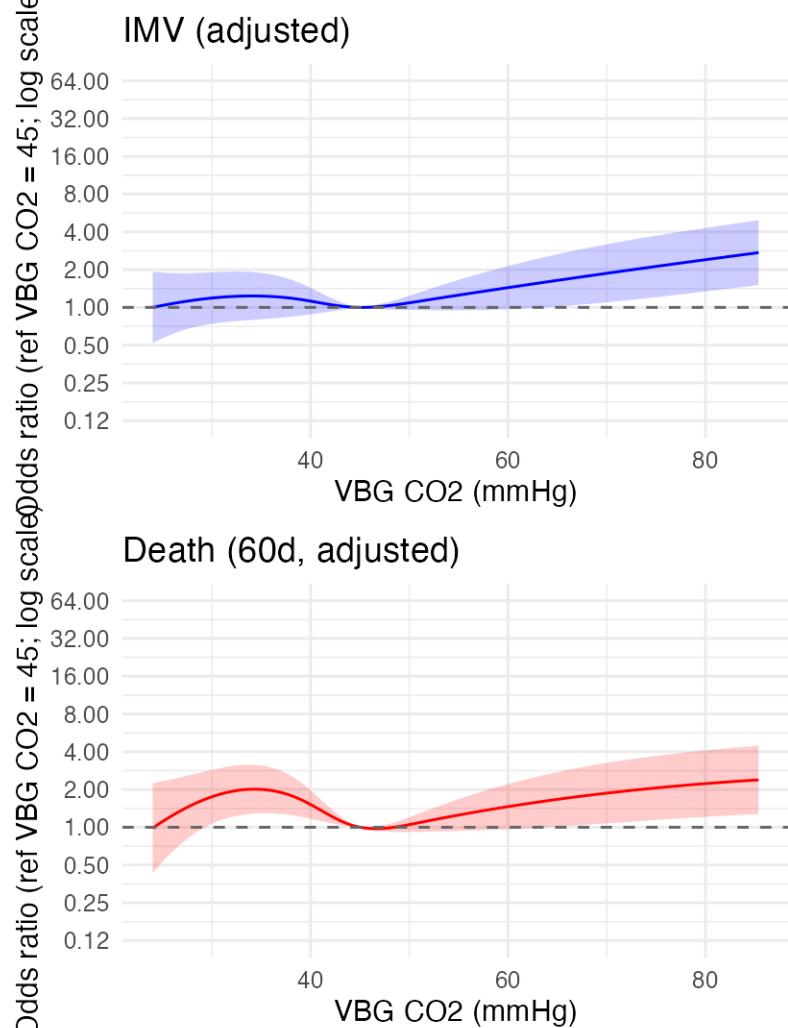
print_plot_once(unw_abg_panel, "spline-unweighted-abg", width = 8.5, height = 6)

```

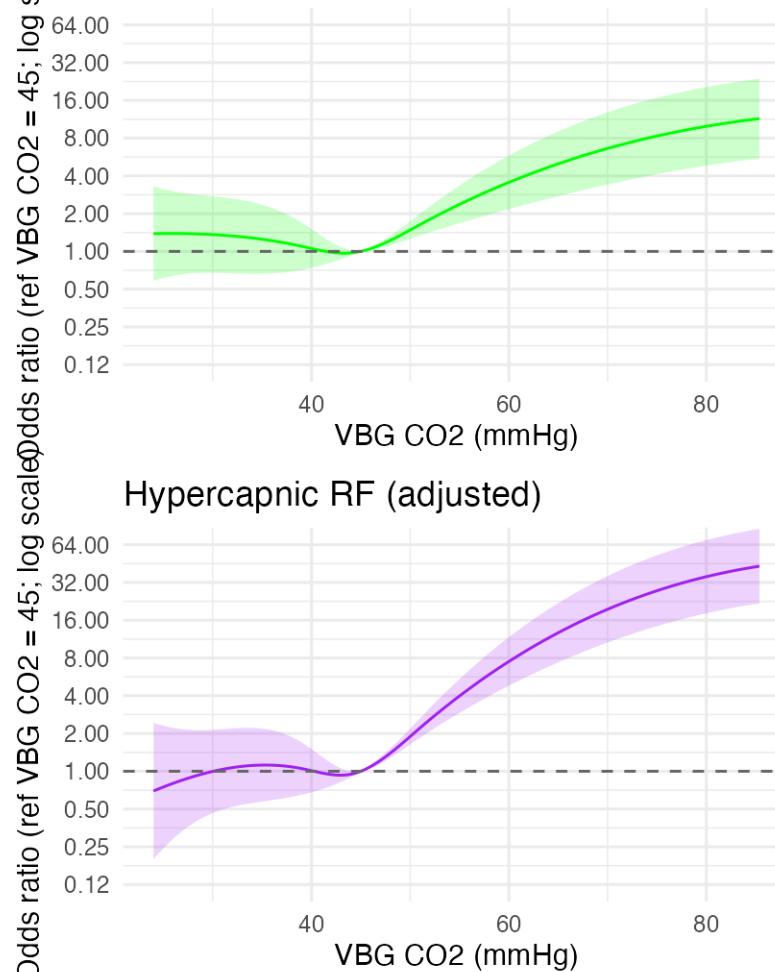


```
print_plot_once(unw_vbg_panel, "spline-unweighted-vbg", width = 8.5, height = 6)
```

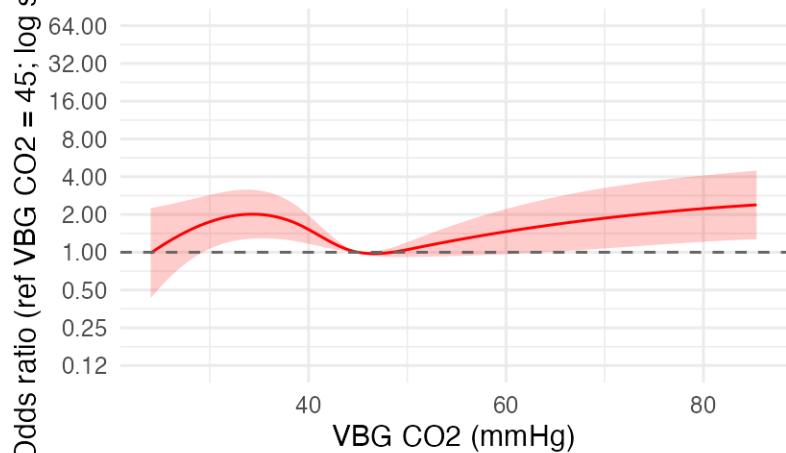
Adjusted odds ratios by VBG CO₂ (ref = 45)



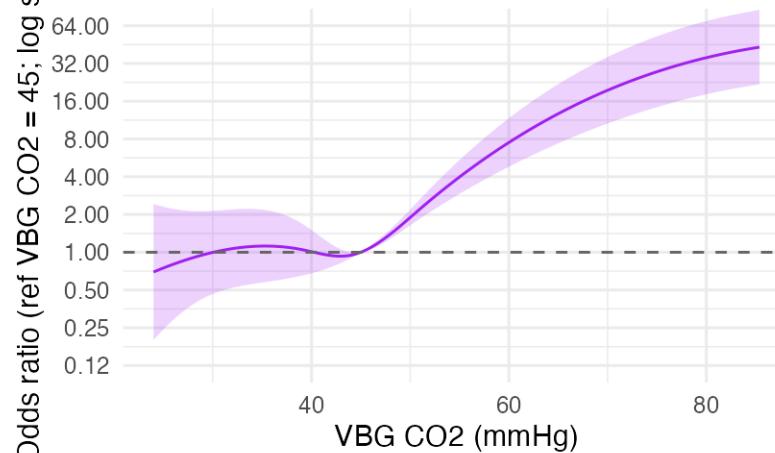
NIV (adjusted)



Death (60d, adjusted)



Hypercapnic RF (adjusted)



Reference VBG CO₂ = 45 mmHg

2 Inverse Propensity Weighting (GBM)

IPW done using Gradient Boosting Methods (GBM) - a type of decision-tree based machine learning. “***Random forests and GBM are designed to automatically include relevant interactions for variables included in the model.*** As such, using a GBM to estimate the PS model, can reduce model misspecification, since ***the analyst is not required to identify relevant interactions or nonlinearities.***” from this citation: PMID: [39947224](#). PMCID: [PMC11825193](#).

Current propensity score uses `covars_gbm` (demographics, comorbidities, encounter type, vitals, labs) as defined above; in this block only `encounter_type` is explicitly factored before weighting.

Note: for all these, I suggested new GBM adjustments that accomplish the following:

1. Smaller GBM & balance-based stopping (`stop.method = “smd.max”`) → faster fit, avoids over-fitting, lighter tails (which lead to extreme weights that are problematic).
2. Target balance compares weighted treated cohort to the full sample; aim for $|SMD| < 0.1$.
3. Weight stabilization (divide by mean) mitigates a few huge weights. We use one-sided truncation at very small propensities (caps large weights only).
4. Uses robust variance estimation (e.g. allows the variances to change by PaCO2) for IP-weighted GLM; works with splines via `rcs()`. This is a bit nuanced but I think good to change even though it adds complexity
5. Deterministic seed ensures result replication.

2.0.1 ABG IPW weighting and diagnostics

```
# Already normalized globally; just drop unused levels
subset_data$encounter_type <- droplevels(subset_data$encounter_type)
```

GBM tuning is shared across ABG and VBG via `gbm_params` to keep symmetry; update there if needed.

```
# 1. fit GBM propensity model, ABG
# Build a minimal modeling frame (treatment + covariates only).
# Including has_vbg preserves shared normalization checks used elsewhere.
set.seed(42)
gbm_df_abg <- subset_data[, c("has_abg", "has_vbg", covars_gbm), drop = FALSE]

# Normalize data types before fitting GBM so predictors are consistent.
```

```

gbm_df_abg <- normalize_types(gbm_df_abg, levels_ref)
gbm_df_abg <- droplevels_all(gbm_df_abg)

# Log potential design-size problems and memory usage around model fit.
gbm_preflight(gbm_df_abg, covars_gbm, "unimp_abg")
append_mem_snapshot("gbm_unimp", "unimp_abg", "pre")

weight_model <- do.call(
  weightit,
  c(
    list(
      formula_abg,
      data      = gbm_df_abg,
      method    = "gbm",
      estimand  = "ATE",
      missing   = "ind",
      include.obj = FALSE
    ),
    gbm_params
  )
)
append_mem_snapshot("gbm_unimp", "unimp_abg", "post")

# 2. One-sided IPSW (ABG observed only) + truncation of small propensities
# `compute_ipow_weights()` applies the project's one-sided weighting rule:
# treated rows receive inverse propensity weights, non-treated rows are anchors.
ipow_abg <- compute_ipow_weights(
  weight_model,
  treat = gbm_df_abg$has_abg,
  ps_floor_quantile = ps_trunc_quantile,
  stabilize = TRUE
)
w_abg <- ipow_abg$weights
ps_floor_abg <- ipow_abg$ps_floor
subset_data$trunc_abg <- ipow_abg$truncated
subset_data$ps_abg <- ipow_abg$ps
subset_data$w_abg <- w_abg

```

```

# Sanity-check: treated weights must be finite before outcome modeling.
assert_finite_weights(w_abg[subset_data$has_abg == 1], "w_abg")
rm(weight_model, gbm_df_abg)
invisible(gc())

# Balance diagnostics and treated-only outcome models are handled later.

```

Inverse Propensity-Weighted Logistic Regressions with CO2 predictor represented as a restricted cubic spline.

These are covariate-adjusted outcome models ($\text{outcome} \sim \text{spline}(\text{CO2}) + \text{X}$), fit separately for ABG and VBG cohorts using `survey::svyglm` with robust (design-based) SEs. Spline curves are shown as odds ratios relative to CO2_{ref} (midpoint of the normal range).

2.0.2 ABG IPW spline models

```

# set.seed(42) # reproducible GBM fit
#
# # 1. inverse-probability weights for receiving an ABG
#
# # done in the last block, so not needed
#
# Model diagrams: IPW ABG spline models
ipw_abg_rcs_forms <- list(
  "ABG IPW spline (adjusted): IMV ~ CO2 spline + X"      = make_spline_fml("imv_proc", "paco2", adj_core),
  "ABG IPW spline (adjusted): NIV ~ CO2 spline + X"      = make_spline_fml("niv_proc", "paco2", adj_core),
  "ABG IPW spline (adjusted): Death60d ~ CO2 spline + X" = make_spline_fml("death_60d", "paco2", adj_core),
  "ABG IPW spline (adjusted): HCRF ~ CO2 spline + X"     = make_spline_fml("hypercap_resp_failure", "paco2", adj_core)
)
register_model_diagrams(ipw_abg_rcs_forms)

# 2. analysis sample: rows with a measured PaCO2
subset_data_abg <- subset_data %>%
  filter(!is.na(paco2)) %>%                                # implies has_abg == 1
  select(paco2, imv_proc, niv_proc, death_60d,
         hypercap_resp_failure, w_abg, all_of(adj_core)) %>%
  filter(complete.cases(.))

```

```

# 3. weighted logistic spline models with robust SEs
fitfun <- function(formula, outcome) {
  fit_res <- fit_with_diagnostics(
    function() svyglm(
      formula,
      design = svydesign(ids = ~1, weights = ~w_abg, data = subset_data_abg),
      family = quasibinomial(),
      control = glm.control(maxit = 50)
    ),
    context = make_context(
      stage = "outcome",
      component = "spline",
      analysis_variant = "ipw",
      model_type = "spline",
      group = "ABG",
      outcome = outcome,
      imputation = NA_integer_,
      batch = NA_integer_
    )
  )
  append_outcome_diag(fit_res$diag)
  fit_res$fit
}

fit_imv_abg  <- fitfun(make_spline_fml("imv_proc", "paco2", adj_core), "imv_proc")
fit_niv_abg  <- fitfun(make_spline_fml("niv_proc", "paco2", adj_core), "niv_proc")
fit_death_abg <- fitfun(make_spline_fml("death_60d", "paco2", adj_core), "death_60d")
fit_hcrcf_abg <- fitfun(make_spline_fml("hypercap_resp_failure", "paco2", adj_core),
                           "hypercap_resp_failure")
if (any(vapply(list(fit_imv_abg, fit_niv_abg, fit_death_abg, fit_hcrcf_abg), is.null, logical(1)))) {
  stop("IPW ABG spline fits failed; see model_fit_diagnostics.csv.")
}

# 4. prediction helper
mkpred <- function(fit, data_ref, co2_var, ref_df, co2_ref) {

```

```

co2_seq <- stats::quantile(data_ref[[co2_var]], probs = c(0.02, 0.98), na.rm = TRUE)
grid_info <- make_co2_grid_ref(
  co2_var,
  seq(co2_seq[1], co2_seq[2], length.out = SPLINE_GRID_N),
  ref_df,
  co2_ref
)
predict_or_curve_from_fit(fit, grid_info$grid, grid_info$ref_idx, co2_var)
}

pred_imv_abg <- mkpred(fit_imv_abg, subset_data_abg, "paco2", x_ref_abg, ABG_CO2_REF)
pred_niv_abg <- mkpred(fit_niv_abg, subset_data_abg, "paco2", x_ref_abg, ABG_CO2_REF)
pred_death_abg <- mkpred(fit_death_abg, subset_data_abg, "paco2", x_ref_abg, ABG_CO2_REF)
pred_hcrcf_abg <- mkpred(fit_hcrcf_abg, subset_data_abg, "paco2", x_ref_abg, ABG_CO2_REF)
axis_abg_ipw_trim <- compute_or_axis_spec(
  list(pred_imv_abg, pred_niv_abg, pred_death_abg, pred_hcrcf_abg),
  lo_col = "LCL", hi_col = "UCL"
)
# 5. plotting
# Plotting deferred until VBG curves are computed so axes can be shared.

```

Restricting plots bewtween 0.02 and 0.98

2.0.3 ABG IPW spline models (2–98th percentile)

```

# Purpose: ipw abg rcs trimmed.
subset_data_abg <- subset_data %>%
  filter(!is.na(paco2)) %>% # implies has_abg == 1
  select(paco2, imv_proc, niv_proc, death_60d,
         hypercap_resp_failure, w_abg, all_of(adj_core)) %>%
  filter(complete.cases(.))

fitfun <- function(formula, outcome) {
  fit_res <- fit_with_diagnostics(
    function() svyglm(
      formula,
      design = svydesign(ids = ~1, weights = ~w_abg, data = subset_data_abg),

```

```

family = quasibinomial(),
control = glm.control(maxit = 50)
),
context = make_context(
  stage = "outcome",
  component = "spline",
  analysis_variant = "ipw",
  model_type = "spline",
  group = "ABG",
  outcome = outcome,
  imputation = NA_integer_,
  batch = NA_integer_
)
)
append_outcome_diag(fit_res$diag)
fit_res$fit
}

fit_imv_abg  <- fitfun(make_spline_fml("imv_proc", "paco2", adj_core), "imv_proc")
fit_niv_abg  <- fitfun(make_spline_fml("niv_proc", "paco2", adj_core), "niv_proc")
fit_death_abg <- fitfun(make_spline_fml("death_60d", "paco2", adj_core), "death_60d")
fit_hcrcf_abg <- fitfun(make_spline_fml("hypercap_resp_failure", "paco2", adj_core),
                           "hypercap_resp_failure")
if (any(vapply(list(fit_imv_abg, fit_niv_abg, fit_death_abg, fit_hcrcf_abg), is.null, logical(1)))) {
  stop("IPW ABG spline fits (trimmed) failed; see model_fit_diagnostics.csv.")
}

# 4. prediction helper
mkpred <- function(fit, data_ref, co2_var, ref_df, co2_ref) {
  q <- stats::quantile(data_ref[[co2_var]], probs = c(0.02, 0.98), na.rm = TRUE)
  grid_info <- make_co2_grid_ref(
    co2_var,
    seq(q[1], q[2], length.out = SPLINE_GRID_N),
    ref_df,
    co2_ref
  )
  predict_or_curve_from_fit(fit, grid_info$grid, grid_info$ref_idx, co2_var)
}

```

```

}

pred_imv_abg    <- mkpred(fit_imv_abg,    subset_data_abg, "paco2", x_ref_abg, ABG_CO2_REF)
pred_niv_abg    <- mkpred(fit_niv_abg,    subset_data_abg, "paco2", x_ref_abg, ABG_CO2_REF)
pred_death_abg <- mkpred(fit_death_abg,  subset_data_abg, "paco2", x_ref_abg, ABG_CO2_REF)
pred_hcrcf_abg <- mkpred(fit_hcrcf_abg, subset_data_abg, "paco2", x_ref_abg, ABG_CO2_REF)

# 5. plotting
xlab <- expression(paste("ABG CO" [2], " (mmHg)"))

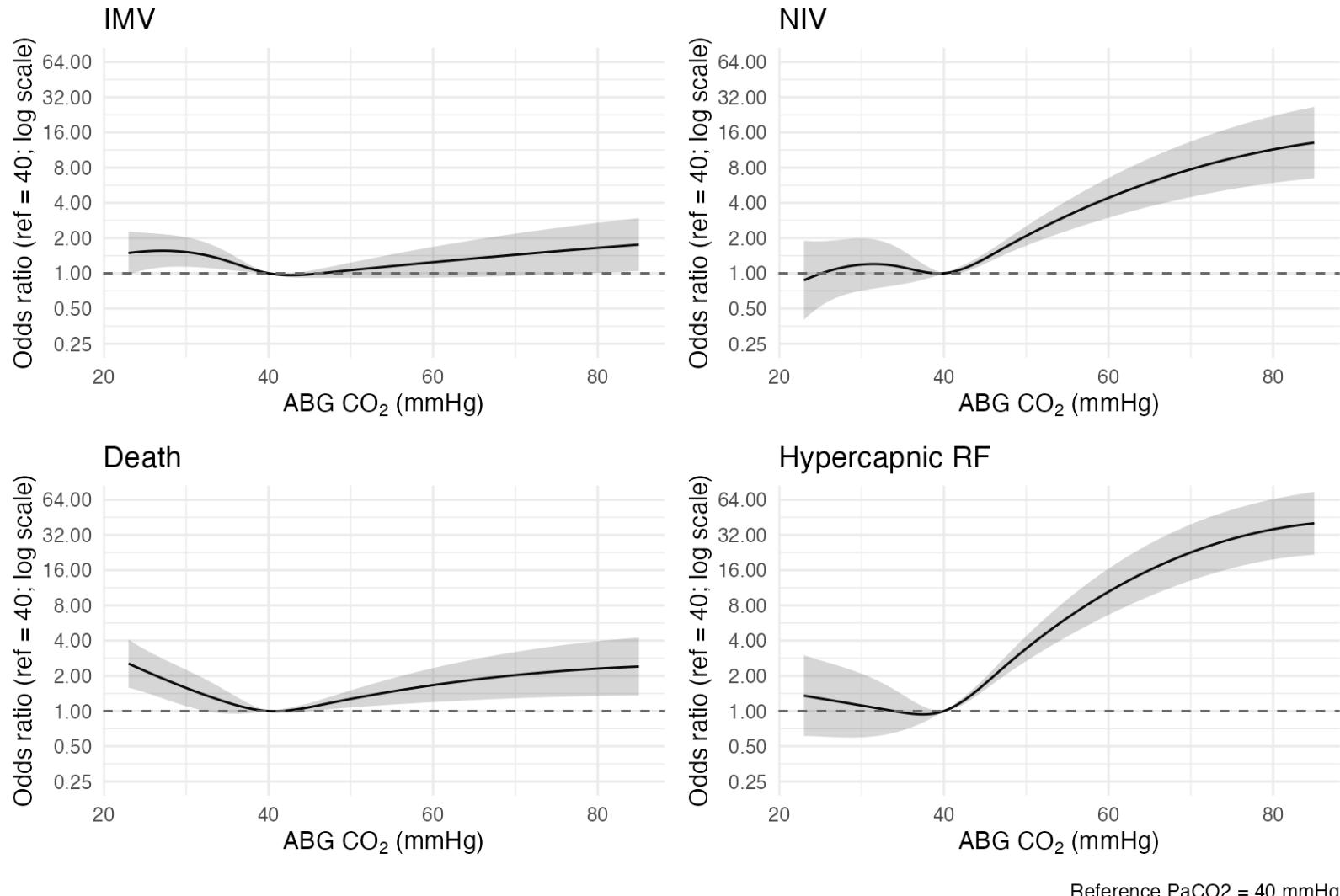
plt <- function(dat, title)
  ggplot(dat, aes(paco2, OR)) +
    geom_line() +
    geom_ribbon(aes(ymin = LCL, ymax = UCL), alpha = 0.2) +
    geom_hline(yintercept = 1, linetype = "dashed", color = "grey40") +
    or_axis_scale(axis_abg_ipw_trim) +
    labs(title = title, x = xlab,
         y = paste0("Odds ratio (ref = ", ABG_CO2_REF, "; log scale)")) +
    theme_minimal()

ipw_abg_panel <- (patchwork::wrap_plots(
  plt(pred_imv_abg,    "IMV"),
  plt(pred_niv_abg,    "NIV"),
  plt(pred_death_abg,  "Death"),
  plt(pred_hcrcf_abg,  "Hypercapnic RF"),
  ncol = 2
)) +
  plot_annotation(
    title = paste0("Propensity-weighted adjusted odds ratios by ABG CO2 (ref = ",
                  ABG_CO2_REF, "; conditional on X; 2-98% range)"),
    caption = paste0("Reference PaCO2 = ", ABG_CO2_REF, " mmHg")
  )

print_plot_once(ipw_abg_panel, "spline-ipw-abg-trimmed", width = 8.5, height = 6)

```

Propensity-weighted adjusted odds ratios by ABG CO₂ (ref = 40; conditional on X; 2–98% range)



VBG uses the same GBM tuning as ABG (shared `gbm_params`).

2.0.4 VBG IPW weighting and spline models

```
# Inverse-propensity weighting & outcome modelling for **VBG** cohort
#   - mirrored 1-to-1 to the validated ABG workflow

set.seed(42)

# 1. IPW for VBG -----
set.seed(42)
# Mirror ABG logic: create a minimal design frame and normalize types first.
gbm_df_vbg <- subset_data[, c("has_abg", "has_vbg", covars_gbm), drop = FALSE]
gbm_df_vbg <- normalize_types(gbm_df_vbg, levels_ref)
gbm_df_vbg <- droplevels_all(gbm_df_vbg)

# Capture model-size and memory telemetry before/after fit.
gbm_preflight(gbm_df_vbg, covars_gbm, "unimp_vbg")
append_mem_snapshot("gbm_unimp", "unimp_vbg", "pre")
w_vbg <- do.call(
  weightit,
  c(
    list(
      formula_vbg,
      data      = gbm_df_vbg,
      method    = "gbm",
      estimand  = "ATE",
      missing   = "ind",
      include.obj = FALSE
    ),
    gbm_params
  )
)
append_mem_snapshot("gbm_unimp", "unimp_vbg", "post")

# One-sided IPSW (VBG observed only) + truncation of small propensities
# Same weighting convention as ABG, but for VBG testing indicator.
ipow_vbg <- compute_ipow_weights(
  w_vbg,
```

```

treat = gbm_df_vbg$has_vbg,
ps_floor_quantile = ps_trunc_quantile,
stabilize = TRUE
)
w_vbg_ipow <- ipow_vbg$weights
ps_floor_vbg <- ipow_vbg$ps_floor
subset_data$trunc_vbg <- ipow_vbg$truncated
subset_data$ps_vbg <- ipow_vbg$ps
subset_data$w_vbg <- w_vbg_ipow

# Guard against non-finite treated weights before survey models.
assert_finite_weights(w_vbg_ipow[subset_data$has_vbg == 1], "w_vbg")
rm(w_vbg, gbm_df_vbg)
invisible(gc())

# Balance diagnostics are handled later.

# 2. Analysis set (VBG only) -----
subset_data_vbg <- subset_data %>%
  filter(!is.na(vbg_co2)) %>%
  select(vbg_co2, imv_proc, niv_proc, death_60d,
         hypercap_resp_failure, w_vbg, all_of(adj_core)) %>%
  filter(complete.cases(.))

fitfun <- function(formula, outcome) {
  fit_res <- fit_with_diagnostics(
    function() svyglm(
      formula,
      design = svydesign(ids = ~1, weights = ~w_vbg, data = subset_data_vbg),
      family = quasibinomial(),
      control = glm.control(maxit = 50)
    ),
    context = make_context(
      stage = "outcome",
      component = "spline",
      analysis_variant = "ipw",
      model_type = "spline",

```

```

group = "VBG",
outcome = outcome,
imputation = NA_integer_,
batch = NA_integer_
)
)
append_outcome_diag(fit_res$diag)
fit_res$fit
}

# Model diagrams: IPW VBG spline models
ipw_vbg_rcs_forms <- list(
  "VBG IPW spline (adjusted): IMV ~ CO2 spline + X"      = make_spline_fml("imv_proc", "vbg_co2", adj_core),
  "VBG IPW spline (adjusted): NIV ~ CO2 spline + X"      = make_spline_fml("niv_proc", "vbg_co2", adj_core),
  "VBG IPW spline (adjusted): Death60d ~ CO2 spline + X" = make_spline_fml("death_60d", "vbg_co2", adj_core),
  "VBG IPW spline (adjusted): HCRF ~ CO2 spline + X"     = make_spline_fml("hypercap_resp_failure", "vbg_co2",
    adj_core)
)
register_model_diagrams(ipw_vbg_rcs_forms)

fit_imv_vbg   <- fitfun(make_spline_fml("imv_proc", "vbg_co2", adj_core), "imv_proc")
fit_niv_vbg   <- fitfun(make_spline_fml("niv_proc", "vbg_co2", adj_core), "niv_proc")
fit_death_vbg <- fitfun(make_spline_fml("death_60d", "vbg_co2", adj_core), "death_60d")
fit_hcrf_vbg  <- fitfun(make_spline_fml("hypercap_resp_failure", "vbg_co2", adj_core),
                           "hypercap_resp_failure")
if (any(vapply(list(fit_imv_vbg, fit_niv_vbg, fit_death_vbg, fit_hcrf_vbg), is.null, logical(1)))) {
  stop("IPW VBG spline fits failed; see model_fit_diagnostics.csv.")
}

# 4. Prediction helper -----
mkpred <- function(fit, data_ref, co2_var, ref_df, co2_ref) {
  co2_seq <- stats::quantile(data_ref[[co2_var]], probs = c(0.02, 0.98), na.rm = TRUE)
  grid_info <- make_co2_grid_ref(
    co2_var,
    seq(co2_seq[1], co2_seq[2], length.out = SPLINE_GRID_N),
    ref_df,
    co2_ref
}

```

```

)
predict_or_curve_from_fit(fit, grid_info$grid, grid_info$ref_idx, co2_var)
}

pred_imv_vbg   <- mkpred(fit_imv_vbg, subset_data_vbg, "vbg_co2", x_ref_vbg, VBG_CO2_REF)
pred_niv_vbg   <- mkpred(fit_niv_vbg, subset_data_vbg, "vbg_co2", x_ref_vbg, VBG_CO2_REF)
pred_death_vbg <- mkpred(fit_death_vbg, subset_data_vbg, "vbg_co2", x_ref_vbg, VBG_CO2_REF)
pred_hcrf_vbg  <- mkpred(fit_hcrf_vbg, subset_data_vbg, "vbg_co2", x_ref_vbg, VBG_CO2_REF)
axis_ipw_common <- compute_or_axis_spec(
  list(pred_imv_abg, pred_niv_abg, pred_death_abg, pred_hcrf_abg,
       pred_imv_vbg, pred_niv_vbg, pred_death_vbg, pred_hcrf_vbg),
  lo_col = "LCL", hi_col = "UCL"
)

# 5. Plotting (gray scheme) -----
xlab <- expression(paste("VBG CO" [2], " (mmHg)"))

plt <- function(dat, title)
  ggplot(dat, aes(vbg_co2, OR)) +
    geom_line() +
    geom_ribbon(aes(ymin = LCL, ymax = UCL), alpha = 0.2) +
    geom_hline(yintercept = 1, linetype = "dashed", color = "grey40") +
    or_axis_scale(axis_ipw_common) +
    labs(title = title, x = xlab,
         y = paste0("Odds ratio (ref = ", VBG_CO2_REF, "; log scale)")) +
    theme_minimal()

ipw_vbg_panel <- (patchwork::wrap_plots(
  plt(pred_imv_vbg, "IMV"),
  plt(pred_niv_vbg, "NIV"),
  plt(pred_death_vbg, "Death"),
  plt(pred_hcrf_vbg, "Hypercapnic RF"),
  ncol = 2
)) +
  plot_annotation(
    title = paste0("Propensity-weighted adjusted odds ratios by VBG CO2 (ref = ",
                  VBG_CO2_REF, "; conditional on X)"),

```

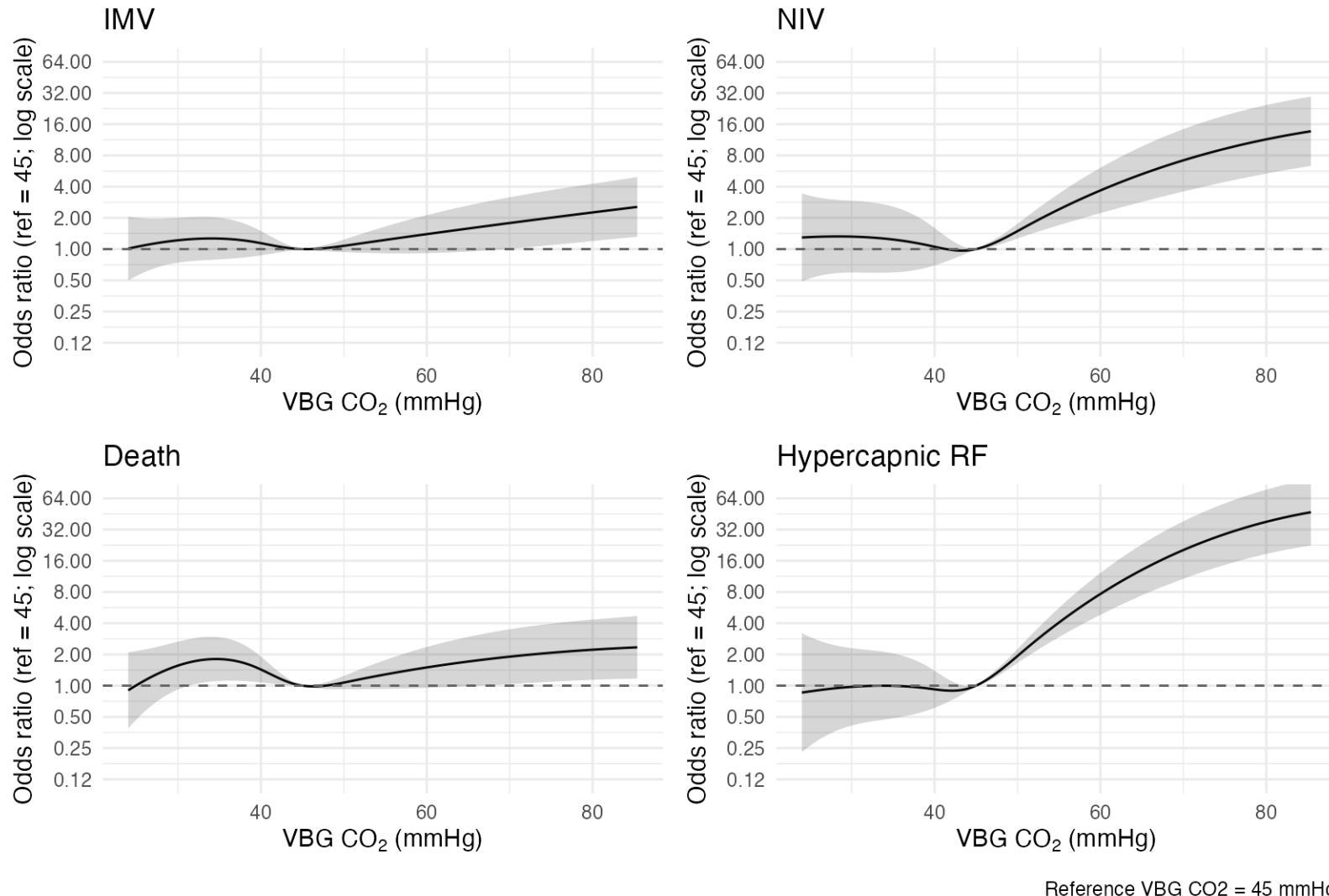
```

    caption = paste0("Reference VBG CO2 = ", VBG_CO2_REF, " mmHg")
)

print_plot_once(ipw_vbg_panel, "spline-ipw-vbg", width = 8.5, height = 6)

```

Propensity-weighted adjusted odds ratios by VBG CO₂ (ref = 45; conditional on X)



```

# ABG plots with the same axis (shared with VBG)
xlab_abg <- expression(paste("ABG CO"[2], " (mmHg)"))

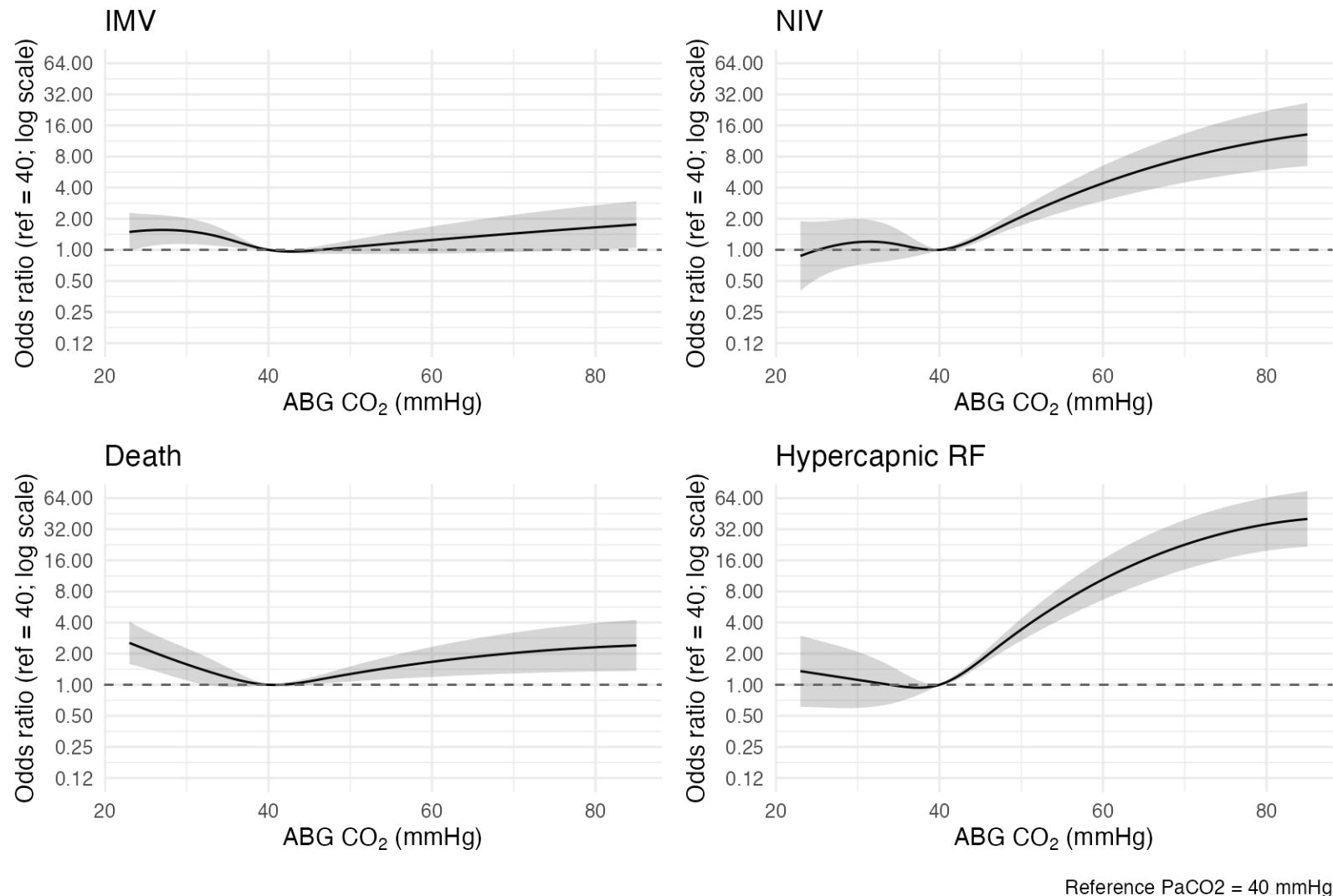
plt_abg <- function(dat, title)
  ggplot(dat, aes(paco2, OR)) +
    geom_line() +
    geom_ribbon(aes(ymin = LCL, ymax = UCL), alpha = 0.2) +
    geom_hline(yintercept = 1, linetype = "dashed", color = "grey40") +
    or_axis_scale(axis_ipw_common) +
    labs(title = title, x = xlab_abg,
         y = paste0("Odds ratio (ref = ", ABG_CO2_REF, "; log scale)")) +
    theme_minimal()

ipw_abg_shared_panel <- (patchwork::wrap_plots(
  plt_abg(pred_imv_abg,      "IMV"),
  plt_abg(pred_niv_abg,      "NIV"),
  plt_abg(pred_death_abg,    "Death"),
  plt_abg(pred_hcrcf_abg,    "Hypercapnic RF"),
  ncol = 2
)) +
  plot_annotation(
    title = paste0("Propensity-weighted adjusted odds ratios by ABG CO2 (ref = ",
                  ABG_CO2_REF, "; conditional on X)"),
    caption = paste0("Reference PaCO2 = ", ABG_CO2_REF, " mmHg")
  )

print_plot_once(ipw_abg_shared_panel, "spline-ipw-abg-shared", width = 8.5, height = 6)

```

Propensity-weighted adjusted odds ratios by ABG CO₂ (ref = 40; conditional on X)



```
# ABG + VBG overlaid spline panel (non-MI IPSW), matching MI-style comparison
ipw_curve_abg <- dplyr::bind_rows(
  pred_imv_abg |> dplyr::mutate(outcome = "IMV"),
  pred_niv_abg |> dplyr::mutate(outcome = "NIV"),
```

```

pred_death_abg |> dplyr::mutate(outcome = "Death (60d)",
pred_hcrf_abg |> dplyr::mutate(outcome = "Hypercapnic RF")
) |>
dplyr::mutate(group = "ABG", co2 = paco2) |>
dplyr::select(group, outcome, co2, OR, LCL, UCL)

ipw_curve_vbg <- dplyr::bind_rows(
  pred_imv_vbg |> dplyr::mutate(outcome = "IMV"),
  pred_niv_vbg |> dplyr::mutate(outcome = "NIV"),
  pred_death_vbg |> dplyr::mutate(outcome = "Death (60d"),
  pred_hcrf_vbg |> dplyr::mutate(outcome = "Hypercapnic RF")
) |>
dplyr::mutate(group = "VBG", co2 = vbg_co2) |>
dplyr::select(group, outcome, co2, OR, LCL, UCL)

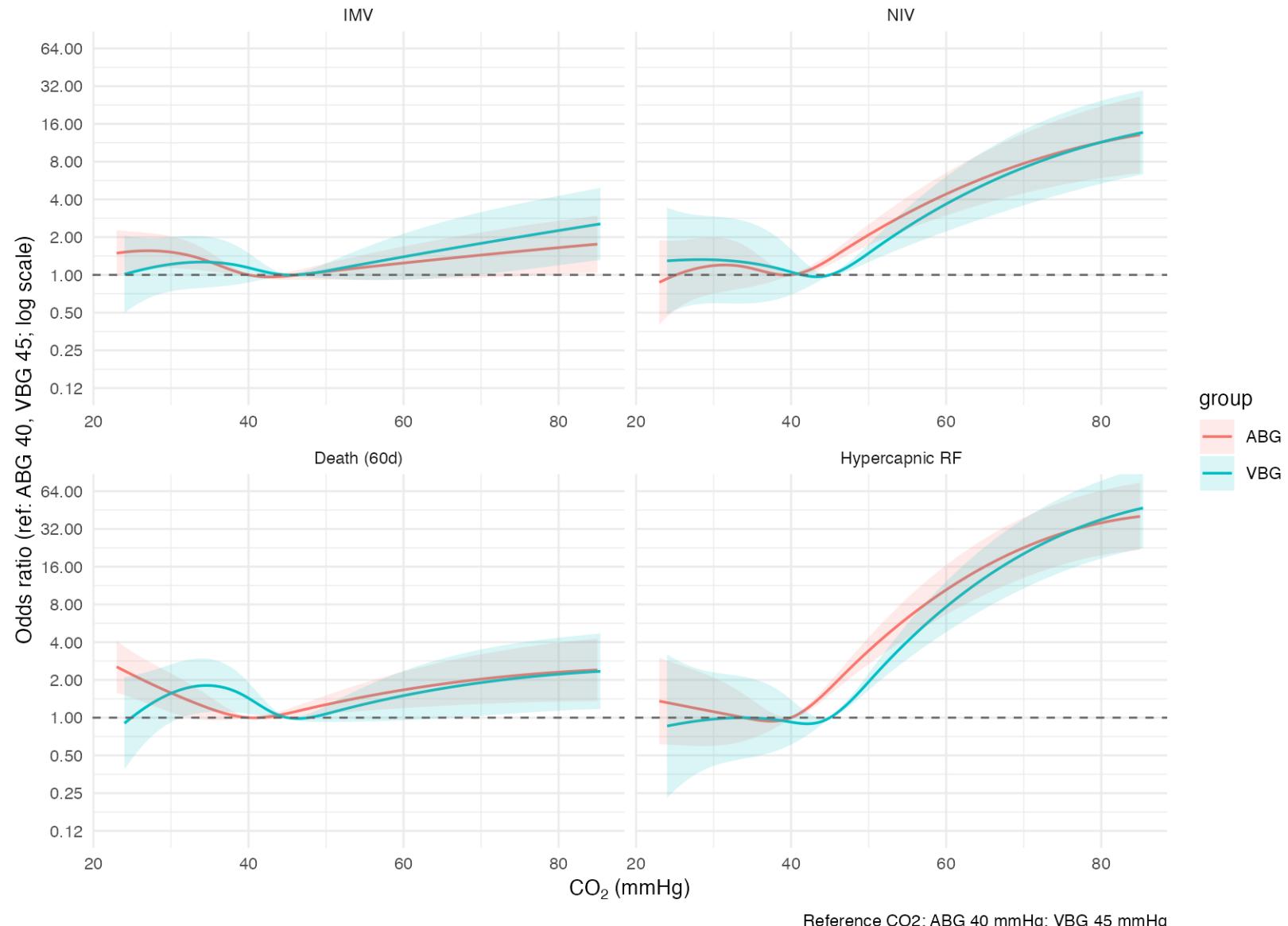
ipw_curve_overlay <- dplyr::bind_rows(ipw_curve_abg, ipw_curve_vbg) |>
dplyr::mutate(
  group = factor(group, levels = c("ABG", "VBG")),
  outcome = factor(outcome, levels = c("IMV", "NIV", "Death (60d)", "Hypercapnic RF"))
)

ipw_overlay_plot <- ggplot(ipw_curve_overlay, aes(x = co2, y = OR, color = group, fill = group)) +
  geom_line(linewidth = 0.6) +
  geom_ribbon(aes(ymin = LCL, ymax = UCL), alpha = 0.15, color = NA) +
  geom_hline(yintercept = 1, linetype = "dashed", color = "grey40") +
  or_axis_scale(axis_ipw_common) +
  facet_wrap(~ outcome, scales = "free_x") +
  labs(
    title = expression(
      paste("IPSW-adjusted spline odds ratios: ABG vs VBG CO[2], " (non-MI))
    ),
    x = expression(CO[2]~"(mmHg)"),
    y = paste0("Odds ratio (ref: ABG ", ABG_CO2_REF, ", VBG ", VBG_CO2_REF, "; log scale"),
    caption = paste0("Reference CO2: ABG ", ABG_CO2_REF, " mmHg; VBG ", VBG_CO2_REF, " mmHg")
  ) +
  theme_minimal(base_size = 10)

```

```
print_plot_once(ipw_overlay_plot, "spline-ipw-overlay-abg-vbg", width = 8.5, height = 6.5)
```

IPSW-adjusted spline odds ratios: ABG vs VBG CO₂ (non-MI)



2.0.5 Three-level PCO2 categories (weighted; ABG, VBG)

Three groups with weights and covariate adjustment

```
# Purpose: ipw three level pco2 all.
library(dplyr)
library(survey)
library(broom)
library(ggplot2)
library(scales)

# 1. Ensure PCO2 categories are present
stopifnot(all(c("pco2_cat_abg", "pco2_cat_vbg") %in% names(subset_data)))

# 2. Function: weighted logistic regression & OR extraction
run_weighted_or <- function(data, outcome, cat_var, weight_var, group_name,
                           treat_var, adj_vars) {
  stopifnot(!is.null(treat_var))
  stopifnot(!is.null(adj_vars))
  dat <- data %>%
    filter(
      .data[[treat_var]] == 1,
      !is.na(.data[[cat_var]]),
      !is.na(.data[[outcome]]),
      !is.na(.data[[weight_var]]),
      .data[[weight_var]] > 0
    ) %>%
    mutate(
      !!cat_var := factor(.data[[cat_var]],
                           levels = CO2_CAT_LEVELS)
    ) %>%
    droplevels()

  design <- svydesign(
    ids = ~1,
    weights = as.formula(paste0("~", weight_var)),
    data = dat
```

```

)
rhs_terms <- c(cat_var, adj_vars)
fml <- stats::reformulate(rhs_terms, response = outcome)
fit_res <- fit_with_diagnostics(
  function() svyglm(fml, design = design, family = quasibinomial(),
    control = glm.control(maxit = 50)),
  context = make_context(
    stage = "outcome",
    component = "cat3",
    analysis_variant = "ipw",
    model_type = "cat3",
    group = group_name,
    outcome = outcome,
    imputation = NA_integer_,
    batch = NA_integer_
  )
)
append_outcome_diag(fit_res$diag)
if (is.null(fit_res$fit)) {
  stop("run_weighted_or: model fit failed for outcome=", outcome,
    " cat_var=", cat_var, " group=", group_name)
}

tidy(fit_res$fit, exponentiate = TRUE, conf.int = TRUE) %>%
  filter(term != "(Intercept)", startsWith(term, cat_var)) %>%
  mutate(
    group      = group_name,
    outcome    = outcome
  )
}

# 3. Run across outcomes & cohorts
outcomes_ipw <- c("imv_proc", "niv_proc", "death_60d", "hypercap_resp_failure")

ipw_three_level_forms <- list(
  "ABG IPW 3-level: IMV ~ CO2 category + X"      = reformulate(c("pco2_cat_abg", adj_core), response = "imv_proc"),

```

```

"ABG IPW 3-level: NIV ~ CO2 category + X"      = reformulate(c("pc02_cat_abg", adj_core), response = "niv_proc"),
"ABG IPW 3-level: Death60d ~ CO2 category + X" = reformulate(c("pc02_cat_abg", adj_core), response = "death_60d"),
"ABG IPW 3-level: HCRF ~ CO2 category + X"      = reformulate(c("pc02_cat_abg", adj_core), response =
  ↵ "hypercap_resp_failure"),
"VBG IPW 3-level: IMV ~ CO2 category + X"       = reformulate(c("pc02_cat_vbg", adj_core), response = "imv_proc"),
"VBG IPW 3-level: NIV ~ CO2 category + X"       = reformulate(c("pc02_cat_vbg", adj_core), response = "niv_proc"),
"VBG IPW 3-level: Death60d ~ CO2 category + X" = reformulate(c("pc02_cat_vbg", adj_core), response = "death_60d"),
"VBG IPW 3-level: HCRF ~ CO2 category + X"       = reformulate(c("pc02_cat_vbg", adj_core), response =
  ↵ "hypercap_resp_failure")
)
register_model_diagrams(ipw_three_level_forms)

ipw_combined_or_df <- bind_rows(
  lapply(outcomes_ipw, function(out)
    run_weighted_or(subset_data, out, "pc02_cat_abg", "w_abg", "ABG",
      treat_var = "has_abg", adj_vars = adj_core)),
  lapply(outcomes_ipw, function(out)
    run_weighted_or(subset_data, out, "pc02_cat_vbg", "w_vbg", "VBG",
      treat_var = "has_vbg", adj_vars = adj_core))
)

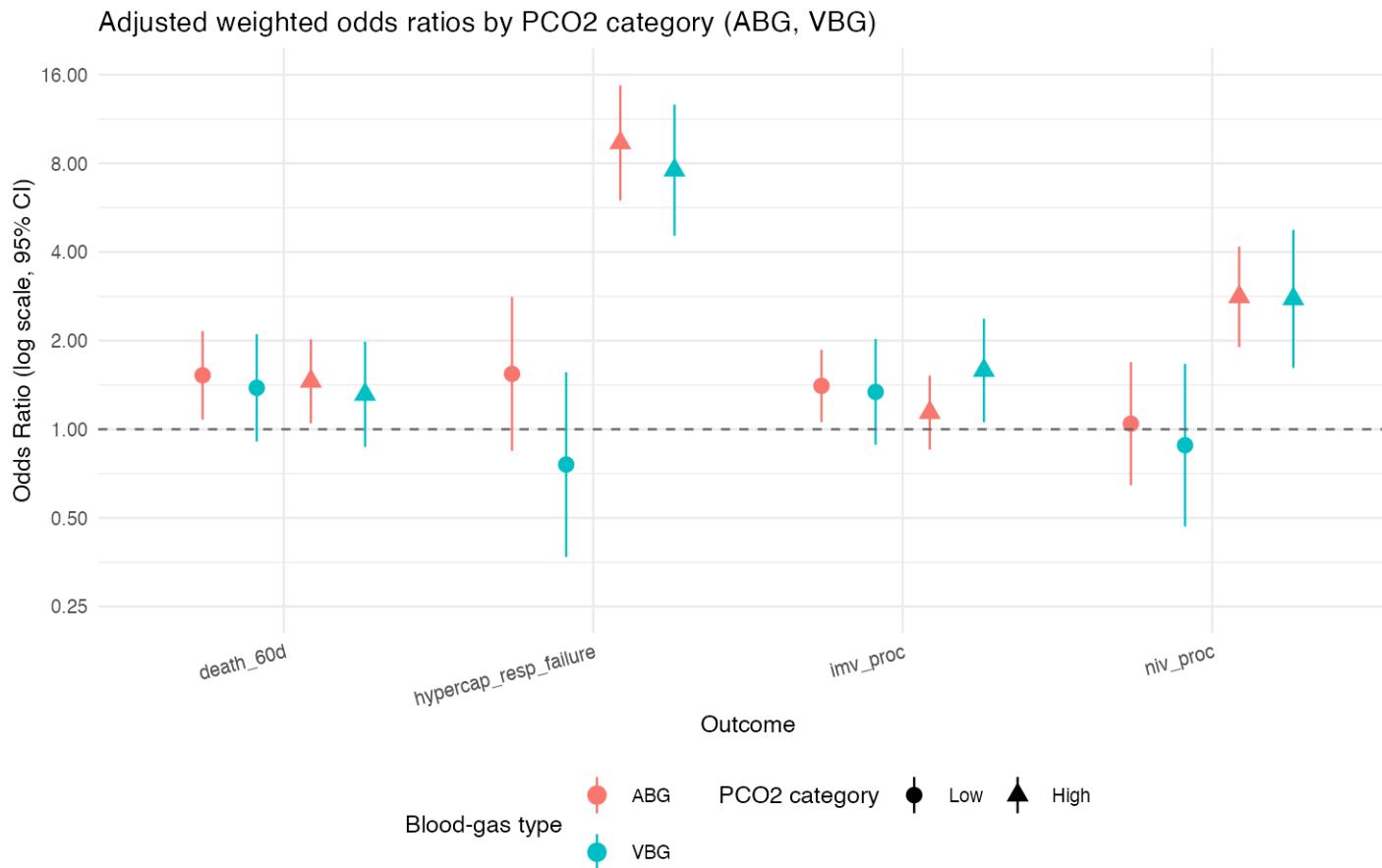
ipw_threelvel_results <- ipw_combined_or_df %>%
  mutate(method = "IPW adjusted")

ipw_combined_or_df <- map_or_exposure(ipw_combined_or_df, "or-plot-three-level-weighted")
ipw_combined_or_df$group <- factor(ipw_combined_or_df$group, levels = c("ABG", "VBG"))

# 4. Plot weighted odds ratios
ipw_plot_df <- build_or_plot_df(ipw_combined_or_df, "or-plot-three-level-weighted",
  expected_exposure_levels = CO2_CAT_CONTRAST_LEVELS)
ipw_axis_spec <- compute_or_axis_spec(ipw_plot_df, lo_col = "conf.low", hi_col = "conf.high",
  default_limits = OR_XLIM)
ipw_p_or <- plot_or_safe(
  ipw_plot_df,
  plot_name = "or-plot-three-level-weighted",
  axis_spec = ipw_axis_spec,
  title = "Adjusted weighted odds ratios by PCO2 category (ABG, VBG)"
)

```

```
)
print_plot_once(ipw_p_or, "or-plot-three-level-weighted", width = 7.5, height = 4.8)
```



2.0.6 Observed outcome rates by CO₂ bin (IPSW, non-MI)

```
# Purpose: summarize weighted event rates in 1-mmHg bins for the non-MI IPSW analyses.
stopifnot(all(c("w_abg", "w_vbg") %in% names(subset_data)))
```

```
co2_rate_ipw_abg <- compute_co2_bin_rates(
```

```

df = subset_data |> dplyr::filter(has_abg == 1),
co2_var = "paco2",
outcomes = CO2_RATE_OUTCOMES,
outcome_labels = CO2_RATE_OUTCOME_LABELS,
weight_var = "w_abg",
q = c(0.02, 0.98),
bin_width = 1,
cohort_label = "ABG"
)
co2_rate_ipw_vbg <- compute_co2_bin_rates(
  df = subset_data |> dplyr::filter(has_vbg == 1),
  co2_var = "vbg_co2",
  outcomes = CO2_RATE_OUTCOMES,
  outcome_labels = CO2_RATE_OUTCOME_LABELS,
  weight_var = "w_vbg",
  q = c(0.02, 0.98),
  bin_width = 1,
  cohort_label = "VBG"
)

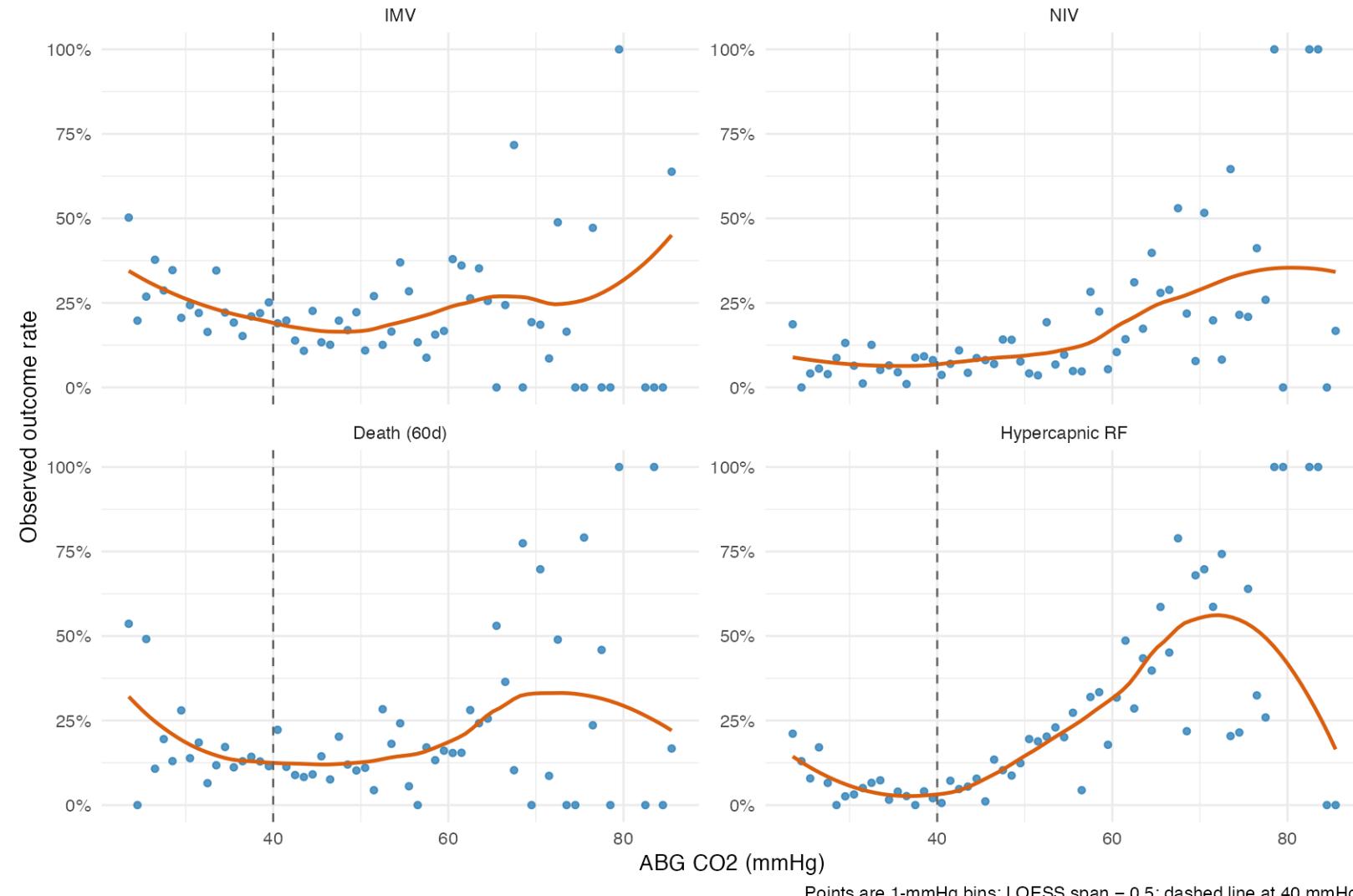
write_csv_safely(co2_rate_ipw_abg, results_path("co2_binned_rates_ipw_abg.csv"), row_names = FALSE)
write_csv_safely(co2_rate_ipw_vbg, results_path("co2_binned_rates_ipw_vbg.csv"), row_names = FALSE)

p_co2_rate_ipw_abg <- plot_co2_bin_rate_panel(
  co2_rate_ipw_abg,
  title = "Observed outcome rates by ABG CO2 bin (IPSW, non-MI)",
  x_label = "ABG CO2 (mmHg)",
  vline_at = ABG_CO2_REF
)
p_co2_rate_ipw_vbg <- plot_co2_bin_rate_panel(
  co2_rate_ipw_vbg,
  title = "Observed outcome rates by VBG CO2 bin (IPSW, non-MI)",
  x_label = "VBG CO2 (mmHg)",
  vline_at = VBG_CO2_REF
)

print_plot_once(p_co2_rate_ipw_abg, "co2-binned-rates-ipw-abg", width = 8.5, height = 6.0)

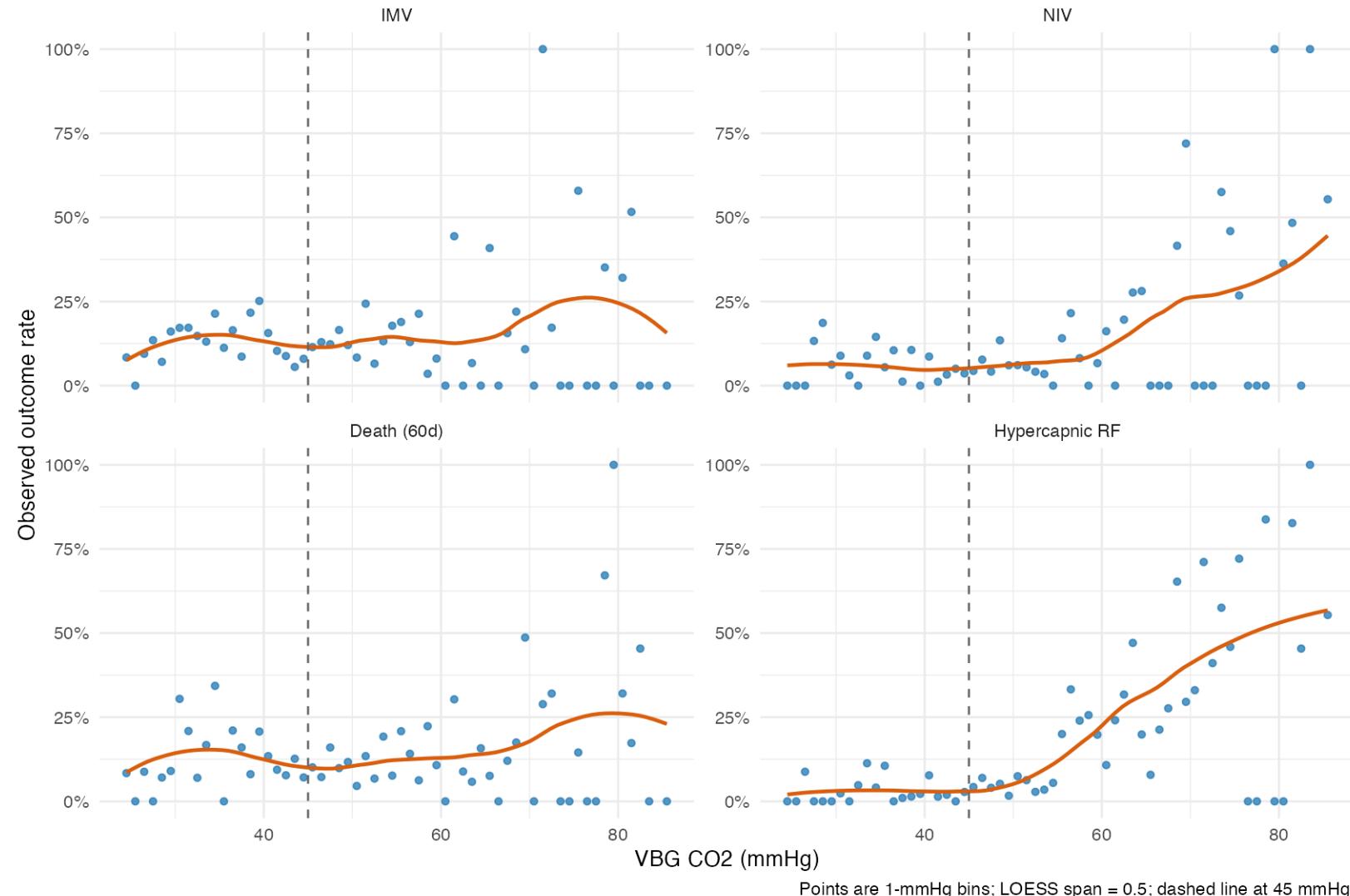
```

Observed outcome rates by ABG CO₂ bin (IPSW, non-MI)



```
print_plot_once(p_co2_rate_ipw_vbg, "co2-binned-rates-ipw-vbg", width = 8.5, height = 6.0)
```

Observed outcome rates by VBG CO₂ bin (IPSW, non-MI)



2.1 Propensity score diagnostics

Plotting propensity scores

```

# --- Propensity score histograms (ABG / VBG) -----
# ABG = arterial blood gas; VBG = venous blood gas

library(dplyr)
library(ggplot2)
library(scales)

stopifnot("has_abg" %in% names(subset_data))
stopifnot("has_vbg" %in% names(subset_data))
stopifnot(all(c("ps_abg", "ps_vbg") %in% names(subset_data)))

# Build list of per-cohort PS data frames conditionally (so missing cohorts don't error)
ps_dfs_cond <- list(
  ABG = data.frame(
    ps      = subset_data$ps_abg,
    treat   = subset_data$has_abg,
    ScoreType = "ABG"
  ),
  VBG = data.frame(
    ps      = subset_data$ps_vbg,
    treat   = subset_data$has_vbg,
    ScoreType = "VBG"
  )
)

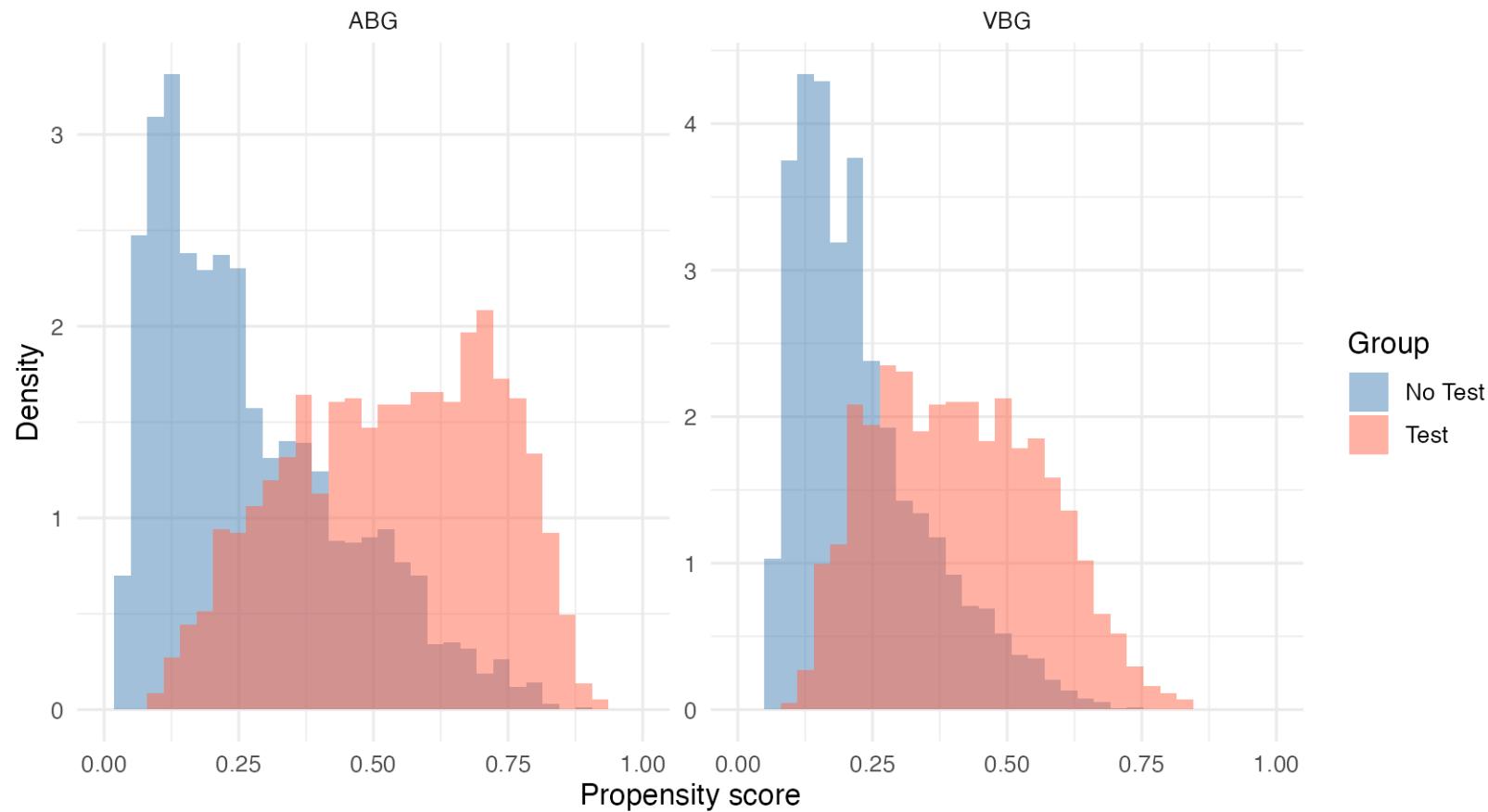
# Bind, clean, and factorize for plotting
df_ps_cond <- bind_rows(ps_dfs_cond) %>%
  filter(!is.na(ps), !is.na(treat)) %>%
  mutate(
    treat    = factor(treat, levels = c(0, 1), labels = c("No Test", "Test")),
    ScoreType = factor(ScoreType, levels = c("ABG", "VBG"))
  )

# Plot
p_ps_cond <- ggplot(df_ps_cond, aes(x = ps, fill = treat)) +
  geom_histogram(aes(y = after_stat(density)), alpha = 0.5,
                 position = "identity", bins = 30) +

```

```
scale_fill_manual(values = c("No Test" = "steelblue", "Test" = "tomato")) +
facet_wrap(~ScoreType, scales = "free_y") +
coord_cartesian(xlim = c(0, 1)) +
labs(
  title = "Propensity Score Distributions",
  x      = "Propensity score",
  y      = "Density",
  fill   = "Group"
) +
theme_minimal(base_size = 12)
print_plot_once(p_ps_cond, "propensity-histograms-conditional", width = 8.5, height = 5)
```

Propensity Score Distributions



```
# Purpose: propensity histograms all.  
stopifnot(all(c("ps_abg", "ps_vbg") %in% names(subset_data)))
```

```
ps_dfs_all <- list(  
  ABG = data.frame(  
    ps      = subset_data$ps_abg,  
    treat   = subset_data$has_abg,  
    ScoreType = "ABG"  
  ),
```

```

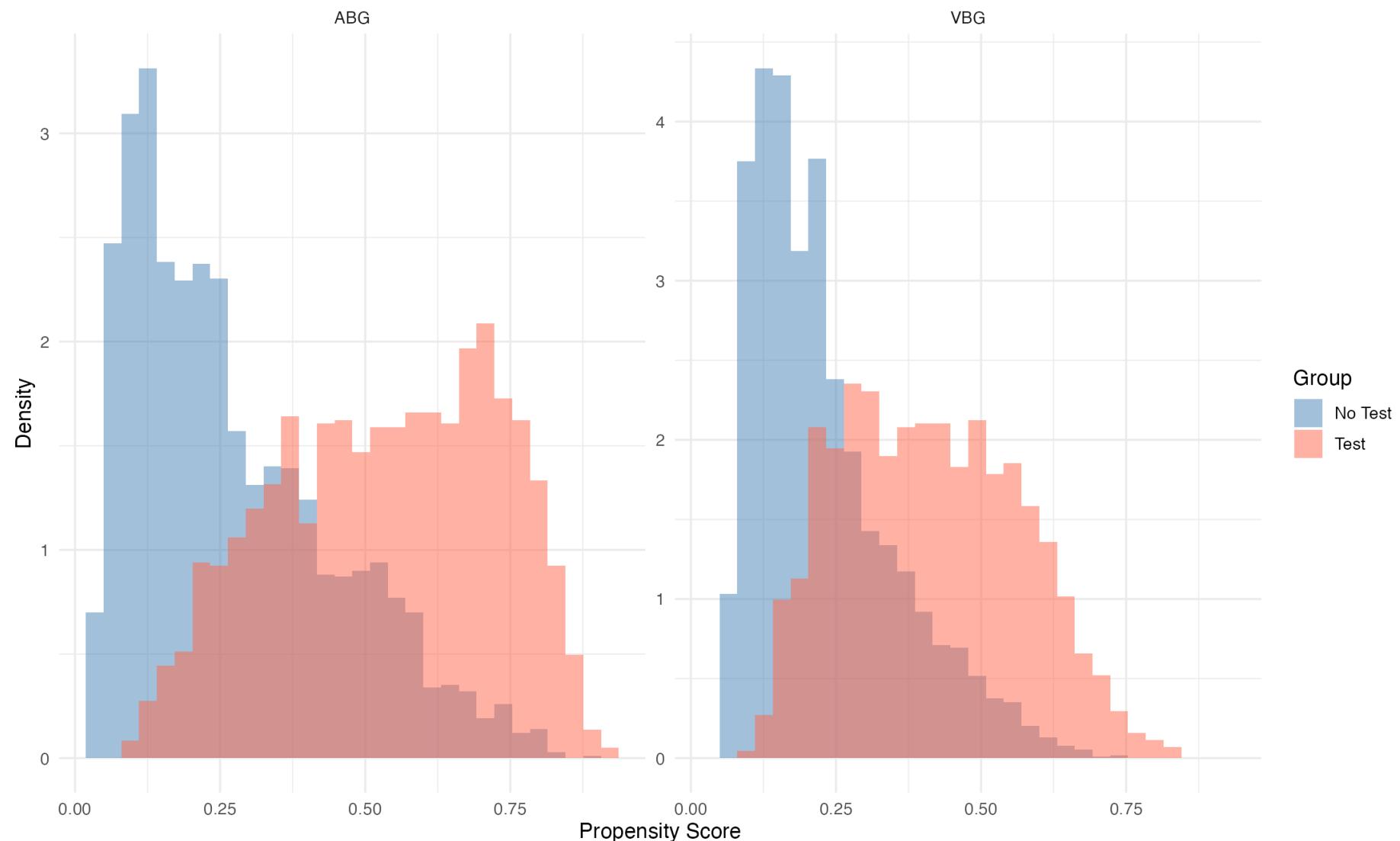
VBG = data.frame(
  ps      = subset_data$ps_vbg,
  treat   = subset_data$has_vbg,
  ScoreType = "VBG"
)
)

df_ps_all <- bind_rows(ps_dfs_all) %>%
  mutate(
    treat   = factor(treat, levels = c(0,1), labels = c("No Test", "Test")),
    ScoreType = factor(ScoreType, levels = c("ABG","VBG"))
  )

ggplot(df_ps_all, aes(x = ps, fill = treat)) +
  geom_histogram(aes(y = after_stat(density)), alpha = 0.5,
                 position = "identity", bins = 30) +
  scale_fill_manual(values = c("No Test" = "steelblue", "Test" = "tomato")) +
  facet_wrap(~ScoreType, scales = "free_y") +
  labs(
    title = "Propensity Score Distributions",
    x = "Propensity Score",
    y = "Density",
    fill = "Group"
  ) +
  theme_minimal(base_size = 12)

```

Propensity Score Distributions



```
# Purpose: shap top10 ipw gbm.  
# This chunk fits lightweight explainability GBM models for ABG/VBG  
# and reports top predictors by mean absolute SHAP value.  
nonmi_shap_abg_file <- results_path("shap_top10_ipw_gbm_abg.csv")
```

```

nonmi_shap_vbg_file <- results_path("shap_top10_ipw_gbm_vbg.csv")

if (RUN_SHAP && HAS_FASTSHAP) {
  # Build ABG/VBG design frames using the same covariates as propensity models.
  gbm_shap_df_abg <- subset_data[, c("has_abg", "has_vbg", covars_gbm), drop = FALSE]
  gbm_shap_df_abg <- normalize_types(gbm_shap_df_abg, levels_ref)
  gbm_shap_df_abg <- droplevels_all(gbm_shap_df_abg)

  gbm_shap_df_vbg <- subset_data[, c("has_abg", "has_vbg", covars_gbm), drop = FALSE]
  gbm_shap_df_vbg <- normalize_types(gbm_shap_df_vbg, levels_ref)
  gbm_shap_df_vbg <- droplevels_all(gbm_shap_df_vbg)

  # Fit separate GBM explainers for ABG and VBG testing models.
  set.seed(20251206)
  shap_fit_abg <- fit_weightit_gbm_for_shap(formula_abg, gbm_shap_df_abg)
  set.seed(30251206)
  shap_fit_vbg <- fit_weightit_gbm_for_shap(formula_vbg, gbm_shap_df_vbg)

  # Compute and rank mean |SHAP| values (top N per cohort).
  shap_top_abg <- extract_nonmi_gbm_shap_top(
    shap_fit_abg, gbm_shap_df_abg, covars_gbm, "ABG",
    nsim = SHAP_NSIM, top_n = SHAP_TOP_N
  )
  shap_top_vbg <- extract_nonmi_gbm_shap_top(
    shap_fit_vbg, gbm_shap_df_vbg, covars_gbm, "VBG",
    nsim = SHAP_NSIM, top_n = SHAP_TOP_N
  )

  write_csv_safely(shap_top_abg, nonmi_shap_abg_file, row_names = FALSE)
  write_csv_safely(shap_top_vbg, nonmi_shap_vbg_file, row_names = FALSE)

  # Draw publication figure: ABG panel on left, VBG panel on right.
  shap_top_nonmi <- dplyr::bind_rows(shap_top_abg, shap_top_vbg)
  if (nrow(shap_top_nonmi) > 0) {
    p_shap_nonmi <- plot_shap_top10_two_panel(
      shap_top_nonmi,
      "Non-MI IPSW-GBM: top 10 features by mean absolute SHAP",

```

```

    "Mean |SHAP value|"
)
# Save + register; display occurs after love plot for narrative flow.
shap_file <- results_path("figs", "shap-top10-ipw-gbm-abg-vbg.png")
save_diag_plot(p_shap_nonmi, shap_file, width = 9, height = 5, dpi = 200)
register_plot_file("shap-top10-ipw-gbm-abg-vbg", shap_file)
}

rm(
  gbm_shap_df_abg, gbm_shap_df_vbg, shap_fit_abg, shap_fit_vbg,
  shap_top_abg, shap_top_vbg, shap_top_nonmi
)
invisible(gc())
} else {
  # Keep output schema stable even when SHAP is disabled/unavailable.
  write_csv_safely(data.frame(), nonmi_shap_abg_file, row.names = FALSE)
  write_csv_safely(data.frame(), nonmi_shap_vbg_file, row.names = FALSE)
}

# Purpose: loveplot ipw gbm.
# Produce a standard love-plot style balance display:
# raw |SMD| vs weighted |SMD| for ABG and VBG cohorts.
covars_balance <- intersect(covars_ps, names(subset_data))
love_data_from_bal <- function(data, treat_var, weights_vec, covars, group_label) {
  id_vars <- intersect(c("has_abg", "has_vbg"), names(data))
  bal_cols <- unique(c(id_vars, treat_var, covars))
  bal_df <- data[, bal_cols, drop = FALSE]
  weights_vec <- ifelse(is.finite(weights_vec) & !is.na(weights_vec), weights_vec, 1)
  if (nrow(bal_df) == 0L) {
    return(data.frame(group = character(), term = character(),
                      abs_smd_raw = numeric(), abs_smd_ipw = numeric(),
                      stringsAsFactors = FALSE))
  }
  treat_vals <- unique(stats::na.omit(bal_df[[treat_var]]))
  if (length(treat_vals) < 2L) {
    return(data.frame(group = character(), term = character(),
                      abs_smd_raw = numeric(), abs_smd_ipw = numeric(),

```

```

            stringsAsFactors = FALSE))
}
bal_df <- normalize_types(bal_df, levels_ref)
bal_df <- droplevels_all(bal_df)
bal_formula <- stats::as.formula(paste(treat_var, "~", paste(covars, collapse = " + ")))
bal_obj <- cobalt::bal.tab(
  bal_formula,
  data = bal_df,
  weights = weights_vec,
  un = TRUE,
  method = "weighting",
  estimand = "ATE",
  quick = FALSE
)
bal_tbl <- as.data.frame(bal_obj$Balance, stringsAsFactors = FALSE)
bal_tbl$term <- rownames(bal_tbl)
if (!("Diff.Un" %in% names(bal_tbl))) bal_tbl$Diff.Un <- NA_real_
if (!("Diff.Adj" %in% names(bal_tbl))) bal_tbl$Diff.Adj <- NA_real_
bal_tbl |>
  dplyr::transmute(
    group = group_label,
    term = term,
    abs_smd_raw = abs(.data$Diff.Un),
    abs_smd_ipw = abs(.data$Diff.Adj)
  )
}

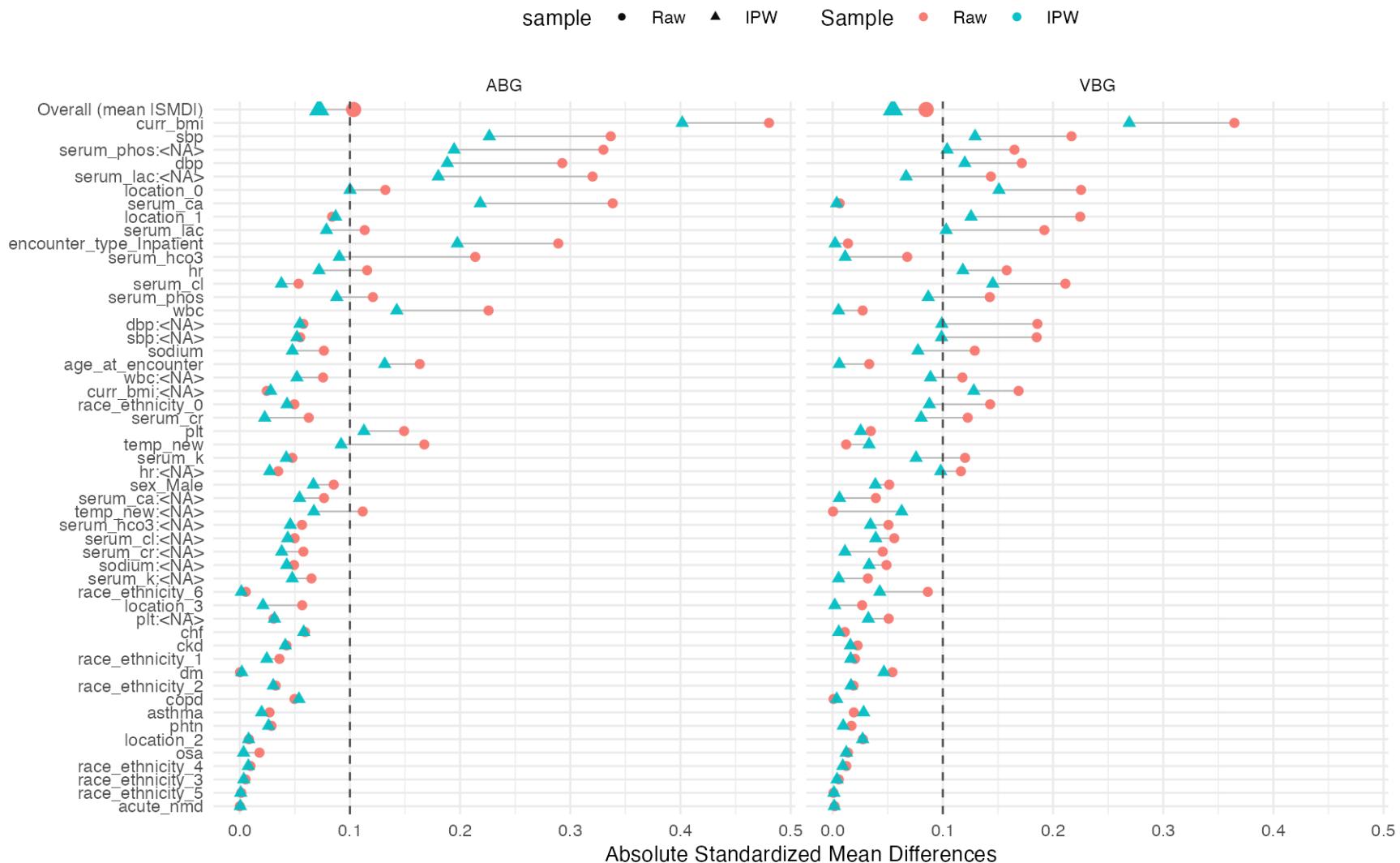
lov_nonmi <- dplyr::bind_rows(
  love_data_from_bal(subset_data, "has_abg", subset_data$w_abg, covars_balance, "ABG"),
  love_data_from_bal(subset_data, "has_vbg", subset_data$w_vbg, covars_balance, "VBG")
)

lov_nonmi_long <- dplyr::bind_rows(
  lov_nonmi |>
    dplyr::transmute(group, term, sample = "Raw", abs_smd = abs_smd_raw, abs_smd_raw),
  lov_nonmi |>
    dplyr::transmute(group, term, sample = "IPW", abs_smd = abs_smd_ipw, abs_smd_raw)
)

```

```
)  
lov_nonmi_long <- add_overall_love_rows(lov_nonmi_long)  
  
write_csv_safely(lov_nonmi_long, results_path("loveplot_ipw_gbm_data.csv"), row_names = FALSE)  
p_lov_nonmi <- loveplot_style(lov_nonmi_long, "Non-MI IPSW-GBM covariate balance")  
print_plot_once(p_lov_nonmi, "loveplot-ipw-gbm-abg-vbg", width = 9, height = 6)
```

Non-MI IPSW-GBM covariate balance



```
# Purpose: render SHAP top-10 directly from CSV to ensure PDF display.
# Read precomputed top-10 SHAP values and re-plot deterministically.
shap_abg_file <- results_path("shap_top10_ipw_gbm_abg.csv")
shap_vbg_file <- results_path("shap_top10_ipw_gbm_vbg.csv")
```

```

shap_abg <- utils::read.csv(shap_abg_file, stringsAsFactors = FALSE)
shap_vbg <- utils::read.csv(shap_vbg_file, stringsAsFactors = FALSE)
shap_abg$group <- "ABG"
shap_vbg$group <- "VBG"

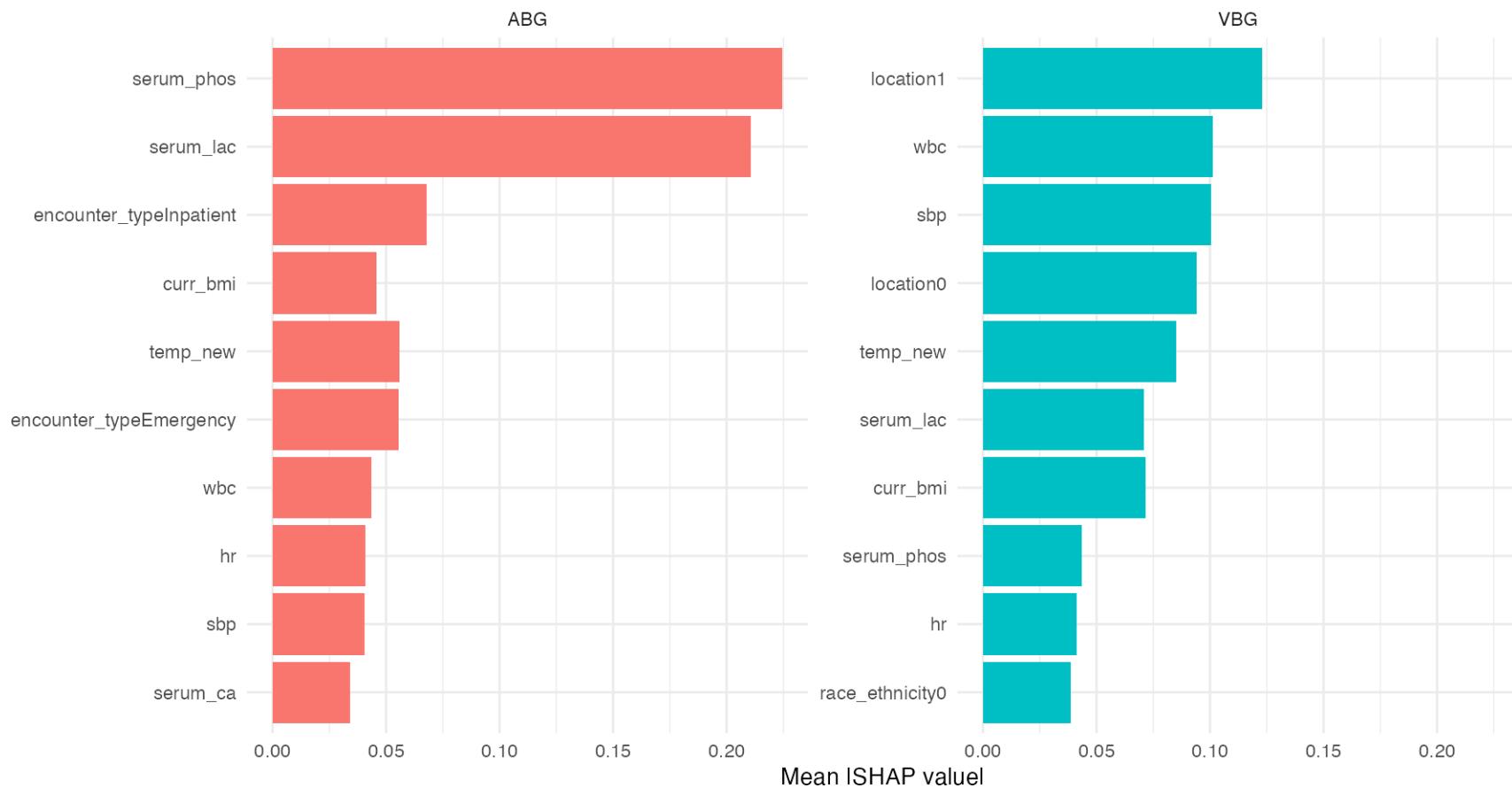
shap_plot_df <- dplyr::bind_rows(shap_abg, shap_vbg) |>
  dplyr::filter(is.finite(mean_abs_shap)) |>
  dplyr::group_by(group) |>
  dplyr::arrange(dplyr::desc(mean_abs_shap), .by_group = TRUE) |>
  dplyr::slice_head(n = 10L) |>
  dplyr::ungroup()

stopifnot(nrow(shap_plot_df) > 0)

p_shap <- plot_shap_top10_two_panel(
  shap_plot_df,
  "IPSW-GBM SHAP top contributors (ABG vs VBG)",
  "Mean |SHAP value|"
)
print(p_shap)

```

IPSW-GBM SHAP top contributors (ABG vs VBG)



```
# Purpose: stage2 cleanup.
append_mem_snapshot("stage2", "end", "post")
stage2_rm_required <- c(
  "ipow_abg", "ipow_vbg", "w_abg", "w_vbg_ipow",
  "subset_data_abg", "subset_data_vbg",
  "fit_imv_abg", "fit_niv_abg", "fit_death_abg", "fit_hcrf_abg",
  "pred_imv_abg", "pred_niv_abg", "pred_death_abg", "pred_hcrf_abg",
  "fit_imv_vbg", "fit_niv_vbg", "fit_death_vbg", "fit_hcrf_vbg",
  "pred_imv_vbg", "pred_niv_vbg", "pred_death_vbg", "pred_hcrf_vbg",
  "plt", "mkpred",
```

```

"ipw_combined_or_df", "ipw_plot_df", "ipw_p_or", "ipw_axis_spec", "outcomes_ipw",
"df_ps_cond", "ps_dfs_cond", "p_ps_cond",
"df_ps_all", "ps_dfs_all"
)
stage2_rm_optional <- c(
  "covars_balance", "lov_abg", "lov_vbg", "lov_nonmi", "lov_nonmi_long", "p_lov_nonmi",
  "nonmi_shap_abg_file", "nonmi_shap_vbg_file", "p_shap_nonmi"
)
missing_stage2 <- setdiff(stage2_rm_required, ls())
stopifnot(length(missing_stage2) == 0)
rm(list = c(stage2_rm_required, intersect(stage2_rm_optional, ls())))
invisible(gc())
append_mem_snapshot("stage2", "cleanup", "post")

```

3 Multiple Imputation (and logistic IPSW)

added 12/6/2025

```

# Core MI + diagnostics
library(mice)          # chained equations (MICE)
library(miceadds)        # pooling helpers & utilities
library(naniar)          # missingness summaries/plots
library(visdat)          # quick type/missingness viz
library(skimr)           # data skim for large frames

# Modeling
library(WeightIt)        # GBM propensity with weights
library(gbm)              # underlying GBM engine
library(survey)           # svyglm outcome models
library(cobalt)           # balance diagnostics
library(broom)             # tidy model outputs
library(dplyr)             # data manipulation
library(ggplot2)

# Pooling and MI bookkeeping
library(mitoools)         # MIcombine for pooling (generic)

```

```

library(parallel)      # basic parallel where helpful

# Parallel + progress setup
library(future)

# setup
library(future.apply)
library(progressr)

mi_mids_file    <- results_path("mi_abg_vbg_mids.rds")
mi_logistic_ps_abg_file    <- results_path("mi_logistic_ps_abg_list.rds")
mi_logistic_ps_vbg_file    <- results_path("mi_logistic_ps_vbg_list.rds")
mi_pooled_file   <- results_path("mi_pooled_results.rds")

# Use sequential futures to avoid PSOCK cluster startup failures during render
future::plan(sequential)

# choose a handler, but DO NOT make it global inside a knitted document
progressr::handlers(progressr::handler_rstudio)    # or handler_txtprogressbar
options(future.rng.onMisuse = "error")               # safer RNG with futures

set.seed(20251206)

# ensure a writable figure dir + stable device on macOS
fs::dir_create(fig_dir, recurse = TRUE)
knitr::opts_chunk$set(fig.path = fig_path, dev = "png", dpi = 144)
options(bitmapType = "cairo")  # prevents device issues on macOS

# Purpose: mi settings.
M_IMP      <- 80
MAXIT_MI <- 20
MI_SEED    <- 20251206
DEBUG_SPLINE <- FALSE
M_IMP_TARGET <- 80
M_IMP_MIN <- 20
M_IMP_STEP <- 10
MCERR_RATIO_TARGET <- 0.10

```

```

ALLOW_M_IMP_EARLY_STOP <- FALSE

# MI propensity model (MI-only)
MI_PS_METHOD      <- "glm_rcs4"
MI_PS_SPLINE_K    <- 4L
MI_GLM_MAXIT     <- 25L

MINCOR_QUICKPRED <- 0.05
MINPUC_QUICKPRED <- 0.25
MAX_PRED_PER_VAR <- 40L
MAX_MM_COLS       <- 300L
MAX_LEVELS_PRED  <- 100L
COR_SAMPLE_N      <- 50000L
MI_MAX_BYTES      <- 8e9

MI_RAM_GB <- 16L
MI_BATCH_START <- ifelse(MI_RAM_GB <= 16, 2L, 5L)
MI_BATCH_MIN   <- 1L
MI_BATCH_SEED_STRIDE <- 100000L
MI_GC_EVERY_BATCH <- TRUE
MI_PREEMPTIVE_BATCH_REDUCE <- TRUE
MI_VCELLS_FRAC_THRESHOLD <- 0.80
MI_SMOKE_TEST <- TRUE
MI_DEBUG_PRINTFLAG <- FALSE
MI_MEMORY_HYGIENE <- TRUE
FORCE_MI_BATCHED <- FALSE

stopifnot(exists("RUN_MODE"), exists("FULL_RUN"))

M_IMP <- M_IMP_TARGET
M_IMP_MIN <- max(M_IMP_MIN, 50L)
if (isTRUE(DEBUG_SPLINE)) {
  stop("DEBUG_SPLINE must be FALSE; pilot and full runs must use the same model workflow.")
}
stopifnot(M_IMP >= M_IMP_MIN, M_IMP <= 100)
stopifnot(MI_PS_METHOD %in% c("glm_rcs4"))

```

```

# Key covariates + outcomes used in MI (plus report-only BNP/Spo2)
extra_miss_vars <- intersect(c("bnp", "spo2"), names(subset_data_raw))
miss_vars <- unique(c(
  covars_ps,
  extra_miss_vars,
  "paco2", "vbg_co2", "vbg_o2sat",
  "has_abg", "has_vbg",
  "imv_proc", "niv_proc", "death_60d", "hypercap_resp_failure"
))
miss_vars <- intersect(miss_vars, names(subset_data_raw))

miss_tbl <- subset_data_raw |>
  dplyr::select(dplyr::all_of(miss_vars)) |>
  dplyr::summarise(dplyr::across(dplyr::everything(), ~ sum(is.na(.)))) |>
  tidyverse::pivot_longer(dplyr::everything(), names_to = "variable", values_to = "n_missing") |>
  dplyr::mutate(
    total_n      = nrow(subset_data_raw),
    pct_missing = 100 * n_missing / total_n
  ) |>
  dplyr::arrange(dplyr::desc(pct_missing))

miss_tbl_disp <- miss_tbl |>
  dplyr::mutate(
    n_missing    = scales::comma(n_missing),
    pct_missing = sprintf("%.1f%%", pct_missing)
  ) |>
  dplyr::select(variable, n_missing, pct_missing)

# Keep the full table in artifacts; suppress this low-value, very long display in PDF.
write_csv_safely(miss_tbl, results_path("missingness_preimputation_full.csv"), row_names = FALSE)

# MI propensity model helpers (glm with RCS on continuous covariates)
ps_is_continuous <- function(v, df) {
  x <- df[[v]]
  if (!is.numeric(x)) return(FALSE)
  x <- x[is.finite(x)]
  if (length(x) < 10) return(FALSE)
}

```

```

if (length(unique(x)) < 10) return(FALSE)
TRUE
}

build_mi_ps_formula <- function(treat, covars, df, k = 4L, basis = "rcs") {
  cont <- covars[vapply(covars, ps_is_continuous, logical(1), df = df)]
  catv <- setdiff(covars, cont)
  term_cont <- character()
  if (length(cont)) {
    term_cont <- if (basis == "rcs") {
      paste0("rms::rcs(`", cont, "`, ", k, ")")
    } else {
      paste0("splines::ns(`", cont, "`, ", k, ")")
    }
  }
  term_cat <- if (length(catv)) paste0("`", catv, "`") else character()
  rhs <- c(term_cont, term_cat)
  if (!length(rhs)) stop("No covariates available for MI PS model.")
  stats::as.formula(paste0("`", treat, " ` ~ ", paste(rhs, collapse = " + ")))
}

fit_mi_ps_glm <- function(df, treat, covars, k = 4L, maxit = 25L, context = NULL) {
  stopifnot(all(c(treat, covars) %in% names(df)))
  df <- droplevels_all(df)
  basis <- if (requireNamespace("rms", quietly = TRUE)) "rcs" else "ns"
  if (is.null(context)) {
    context <- make_context(
      stage = "MI", component = "mi_ps_glm",
      analysis_variant = "weighted_imputed",
      model_type = "ps",
      group = NA_character_,
      outcome = NA_character_,
      imputation = NA_integer_,
      batch = NA_integer_
    )
  }
}

```

```

fit_once <- function(basis_type) {
  if (basis_type == "rcs") {
    dd <- rms:::datadist(df)
    old_opt <- options(datadist = ".__dd_ps__")
    assign(".__dd_ps__", dd, envir = .GlobalEnv)
    on.exit({
      options(old_opt)
      rm(list = ".__dd_ps__", envir = .GlobalEnv)
    }, add = TRUE)
  }
  form <- build_mi_ps_formula(treat, covars, df, k = k, basis = basis_type)
  cap <- capture_warnings(
    tryCatch(
      stats::glm(form, data = df, family = stats::binomial(),
                 control = stats::glm.control(maxit = maxit),
                 model = FALSE, x = FALSE, y = FALSE),
      error = function(e) e
    ),
    context = context
  )
  append_warnings(cap$warnings)
  list(fit = cap$value, formula = form, basis = basis_type)
}

res <- fit_once(basis)
if (inherits(res$fit, "error") && basis == "rcs") {
  res <- fit_once("ns")
}
if (inherits(res$fit, "error")) {
  return(list(error = conditionMessage(res$fit), method = paste0("glm_", res$basis)))
}

ps <- as.numeric(res$fit$fitted.values)
ps <- pmin(pmax(ps, 1e-8), 1 - 1e-8)
vc <- tryCatch(stats::vcov(res$fit), error = function(e) NULL)
list(
  ps = ps,

```

```

fit_ok = TRUE,
converged = if (!is.null(res$fit$converged)) isTRUE(res$fit$converged) else NA,
formula = paste(deparse(res$formula), collapse = " "),
n = nrow(df),
p = length(res$fit$coefficients),
method = paste0("glm_", res$basis),
basis = res$basis,
coef = stats::coef(res$fit),
vcov_diag = if (is.null(vc)) NULL else diag(vc)
)
}

```

3.0.1 Missingness structure and drivers

```

# Purpose: mi missing structure.
library(dplyr)
library(tidyr)
library(rlang)

# Use raw data for missingness rates; normalized data for model-based drivers
miss_data <- subset_data_raw
model_data <- subset_data

# Focus on key variables (same as MI set) for consistency
extra_miss_vars <- intersect(c("bpn", "spo2"), names(miss_data))
miss_vars <- unique(c(
  covars_ps,
  extra_miss_vars,
  "paco2", "vbg_co2", "vbg_o2sat",
  "has_abg", "has_vbg",
  "imv_proc", "niv_proc", "death_60d", "hypercap_resp_failure"
))
miss_vars <- intersect(miss_vars, names(miss_data))
stopifnot(length(miss_vars) > 0)

# Strata to compare missingness across

```

```

strata_vars <- c("has_abg", "has_vbg",
                 "imv_proc", "death_60d", "encounter_type", "location")
strata_vars <- intersect(strata_vars, names(miss_data))

# Overall missingness %
miss_overall <- miss_data |>
  summarise(across(all_of(miss_vars), ~ mean(is.na(.)) * 100)) |>
  pivot_longer(everything(), names_to = "variable", values_to = "pct_missing_overall")

# Missingness % on model_data (used for driver models)
miss_overall_model <- model_data |>
  summarise(across(all_of(intersect(miss_vars, names(model_data))), ~ mean(is.na(.)) * 100)) |>
  pivot_longer(everything(), names_to = "variable", values_to = "pct_missing_overall_model")

# Missingness by strata (percent within each level)
miss_by_strata <- purrr::map_dfr(strata_vars, function(sv) {
  # drop the current grouping variable from targets
  vars_here <- setdiff(intersect(miss_vars, names(miss_data)), sv)
  stopifnot(length(vars_here) > 0L)
  stopifnot(sv %in% names(miss_data))

  miss_data |>
    group_by(.data[[sv]]) |>
    summarise(across(all_of(vars_here), ~ mean(is.na(.)) * 100), .groups = "drop") |>
    pivot_longer(-all_of(sv), names_to = "variable", values_to = "pct_missing") |>
    rename(level = !!sym(sv)) |>
    mutate(
      stratum = sv,
      level   = as.character(level)
    )
  }
)

# Combine overall + strata (save full table; display full table in PDF)
miss_panel_full <- miss_by_strata |>
  left_join(miss_overall, by = "variable") |>
  arrange(desc(pct_missing_overall), stratum, level)

```

```

miss_panel_file <- results_path("missingness-by-strata.csv")
write_csv_safely(miss_panel_full, miss_panel_file, row_names = FALSE)

render_table_pdf_maybe(
  miss_panel_full,
  caption = "Missingness by key strata (pre-imputation).",
  file_stub = "missingness_by_strata",
  digits = 1,
  show = SHOW_LOW_VALUE_TABLES
)

# --- Drivers of missingness (logit I(NA) on observed covariates) -----
# Candidate predictors (observed covariates only)
driver_covars <- intersect(
  c("age_at_encounter", "sex", "encounter_type", "location", "curr_bmi",
    "has_abg", "has_vbg", "imv_proc", "death_60d"),
  names(model_data)
)

# Fit models for variables with any missingness
vars_to_model <- miss_overall_model |>
  filter(pct_missing_overall_model > 0) |>
  pull(variable)
vars_to_model <- intersect(vars_to_model, names(model_data))

skip_log <- list()

model_results <- purrr::map_dfr(vars_to_model, function(v) {
  df <- model_data |>
    select(all_of(c(v, driver_covars))) |>
    mutate(miss = as.integer(is.na(.data[[v]])))

  # use only rows with observed predictors and varying miss indicator
  df <- df[stats::complete.cases(df[driver_covars]), , drop = FALSE]
  if (nrow(df) == 0L) {
    skip_log[[length(skip_log) + 1L]] <- data.frame(
      variable = v,

```

```

    reason = "no_complete_cases",
    stringsAsFactors = FALSE
)
return(tibble::tibble())
}
if (dplyr::n_distinct(df$miss) < 2L) {
  skip_log[[length(skip_log) + 1L]] <- data.frame(
    variable = v,
    reason = "miss_indicator_constant",
    stringsAsFactors = FALSE
)
return(tibble::tibble())
}
fml <- as.formula(paste0("miss ~ ", paste(driver_covars, collapse = " + ")))
if (length(driver_covars) > 0L) {
  diag_outcome <- paste0("missing_", make.names(v))
  diag_terms <- paste0(~, driver_covars, ~, collapse = " + ")
  diag_fml <- stats::as.formula(paste0(diag_outcome, " ~ ", diag_terms))
  register_model_diagram(paste("Missingness model:", v), diag_fml, width = 10, height = 6)
}
fit <- tryCatch(
  suppressWarnings(glm(fml, data = df, family = binomial())),
  error = function(e) e
)
if (inherits(fit, "error")) {
  stop("Missingness driver: glm failed for variable ", v)
}

broom::tidy(fit, conf.int = FALSE, exponentiate = FALSE) |>
  filter(term != "(Intercept)") |>
  mutate(
    OR = exp(estimate),
    LCL = exp(estimate - 1.96 * std.error),
    UCL = exp(estimate + 1.96 * std.error)
  ) |>
  transmute(

```

```

variable = v,
term,
OR,
LCL,
UCL,
p.value
)
})

skip_tbl <- dplyr::bind_rows(skip_log)
write_csv_safely(skip_tbl, results_path("missingness_driver_skips.csv"), row_names = FALSE)

if (nrow(model_results) > 0) {
  model_results_file <- results_path("missingness-drivers.csv")
  model_results_out <- model_results |>
    dplyr::mutate(run_id = diag_run_id)
  write_csv_safely(model_results_out, model_results_file, row_names = FALSE)

  model_results_disp <- model_results_out |>
    dplyr::select(-run_id) |>
    arrange(p.value) |>
    mutate(
      OR = round(OR, 2),
      LCL = round(LCL, 2),
      UCL = round(UCL, 2),
      p.value = signif(p.value, 3)
    )
}

render_table_pdf_maybe(
  model_results_disp,
  caption = "Predictors of missingness (logit OR).",
  file_stub = "missingness-drivers",
  digits = 2,
  show = SHOW_LOW_VALUE_TABLES
)
} else {
  model_results_stub <- data.frame(

```

```

variable = character(),
term = character(),
OR = numeric(),
LCL = numeric(),
UCL = numeric(),
p.value = numeric(),
stringsAsFactors = FALSE
)
write_csv_safely(model_results_stub, results_path("missingness-drivers.csv"), row_names = FALSE)
message("No modelable missingness signals (all complete or no variation).")
}

```

3.0.2 Monte Carlo error check after MI

3.1 Pre-imputation data prep (consistent types & predictors)

Why: MI models need coherent types; using exactly the same covariates as the propensity score models avoids model drift.

```
# Types are normalized in the schema-normalize block.
```

3.2 Imputation model specification (MICE)

3.2.1 Predictor matrix & methods. Run MICE (moderate settings for scale)

```

# --- variables for propensity score model (kept identical to main analysis) ---
# ----- MICE setup: include PaCO2/VBG CO2 as predictors but do not impute -----
library(mice)
library(dplyr)

# --- add analysis targets and CO2 measures explicitly -----
mi_vars <- setdiff(unique(c(
  covars_ps,
  "has_abg", "has_vbg",                                # treatments (NOT imputed)
  "imv_proc", "niv_proc", "death_60d", "hypercap_resp_failure", # outcomes (NOT imputed)
  co2_vars
)), drop_vars_ultra_missing)

```

```

mi_df <- subset_data[, mi_vars, drop = FALSE]
mi_df <- normalize_types(mi_df, levels_ref)

mi_df_size <- utils::object.size(mi_df)
message("MI data size (bytes): ", format(mi_df_size, units = "auto"))

# Rough memory preflight based on total missing cells (imputed values only).
miss_counts <- vapply(mi_df, function(x) sum(is.na(x)), numeric(1))
miss_total <- sum(miss_counts)
if (is.finite(miss_total) && miss_total > 0) {
  est_bytes <- miss_total * M_IMP * 8
  message("MI imputation storage estimate: ", format(structure(est_bytes, class = "object_size"), units = "auto"))
  if (est_bytes > MI_MAX_BYTES) {
    m_max <- floor(MI_MAX_BYTES / (miss_total * 8))
    m_target <- max(50L, m_max)
    if (m_target < 50L) {
      stop("Estimated MI storage exceeds memory (m=", M_IMP,
           ", estimated=", format(structure(est_bytes, class = "object_size"), units = "auto"),
           "). Reduce missingness or MI scope, or increase available memory.")
    }
    if (m_target < M_IMP) {
      message("Reducing M_IMP from ", M_IMP, " to ", m_target,
              " to stay within MI_MAX_BYTES.")
      M_IMP <- m_target
    }
  }
}

# Make binary comorbid factors so "logreg" is used (and stays binary)
bin_covars <- c("copd", "asthma", "osa", "chf", "acute_nmd", "phtn", "ckd", "dm")
missing_bin <- setdiff(bin_covars, names(mi_df))
stopifnot(length(missing_bin) == 0)
mi_df[bin_covars] <- lapply(mi_df[bin_covars], function(z) {
  if (is.factor(z)) return(droplevels(z))
  zz <- suppressWarnings(as.integer(z))
  factor(zz, levels = c(0L, 1L), labels = c("0", "1"))
})

```

```

})

# For MICE: convert any remaining characters → factors
mi_df <- dplyr::mutate(mi_df, across(where(is.character), ~ factor(.x)))

# Guardrail: high-cardinality factors can blow up MICE model matrices.
# Exclude them from predictorMatrix and (if missing) make "Missing" explicit.
high_card <- names(which(vapply(mi_df, function(x) is.factor(x) && nlevels(x) > MAX_LEVELS_PRED, logical(1))))
if (length(high_card)) {
  message("MICE: high-cardinality factors detected (nlevels > ", MAX_LEVELS_PRED, "): ",
         paste(high_card, collapse = ", "))
  for (v in high_card) {
    if (any(is.na(mi_df[[v]]))) {
      lv <- levels(mi_df[[v]])
      tmp <- as.character(mi_df[[v]])
      tmp[is.na(tmp)] <- "Missing"
      mi_df[[v]] <- factor(tmp, levels = unique(c(lv, "Missing")))
      if (!is.null(levels_ref) && !is.null(levels_ref[[v]])) &&
        !"Missing" %in% levels_ref[[v]]) {
        levels_ref[[v]] <- c(levels_ref[[v]], "Missing")
      }
    }
  }
}

# --- methods & predictor matrix aligned to *mi_df* -----
meth <- mice::make.method(mi_df)

is_fac     <- vapply(mi_df, is.factor, logical(1))
is_num     <- vapply(mi_df, is.numeric, logical(1))
is_bin_fac <- vapply(mi_df, function(x) is.factor(x) && nlevels(x) == 2, logical(1))
is_multicat <- vapply(mi_df, function(x) is.factor(x) && nlevels(x) > 2, logical(1))

# robust defaults
meth[is_num]     <- "pmm"      # numerics: predictive mean matching
meth[is_multicat] <- "polyreg"  # unordered multicategory
meth[is_bin_fac] <- "logreg"   # binary factors: logistic regression

```

```

# never impute treatments, outcomes, or CO2 exposures
no_imp <- c("has_abg", "has_vbg", "imv_proc", "niv_proc", "death_60d", "hypercap_resp_failure",
           "paco2", "vbg_co2")
if (length(high_card)) no_imp <- unique(c(no_imp, high_card))
meth[intersect(names(meth), no_imp)] <- ""

# predictor matrix; force has_abg/has_vbg as predictors, but do not impute no_imp
pred <- mice:::quickpred(mi_df, mincor = MINCOR_QUICKPRED, minpuc = MINPUC_QUICKPRED)
for (nm in intersect(c("has_abg", "has_vbg"), colnames(pred))) {
  pred[, nm] <- 1
}
pred[intersect(rownames(pred), no_imp), ] <- 0

if (length(high_card)) {
  pred[, intersect(high_card, colnames(pred))] <- 0
  pred[intersect(high_card, rownames(pred)), ] <- 0
}

# Ensure dropped covariates do not appear as predictors
drop_covars <- intersect(drop_vars_ultra_missing, colnames(pred))
if (length(drop_covars)) {
  pred[, drop_covars] <- 0
}

# Non-imputed variables with missingness should NOT be predictors
no_imp_with_missing <- intersect(no_imp, names(mi_df))
no_imp_with_missing <- no_imp_with_missing[
  vapply(mi_df[no_imp_with_missing], function(x) any(is.na(x)), logical(1))]
]
no_imp_with_missing <- setdiff(no_imp_with_missing, c("has_abg", "has_vbg"))

pred_cols_check <- intersect(c("paco2", "vbg_co2"), colnames(pred))
if (length(pred_cols_check)) {
  message(
    "MICE: predictor column sums (pre-exclude) for ",
    paste(pred_cols_check, collapse = ", "))
}

```

```

": ",
paste(colSums(pred[, pred_cols_check, drop = FALSE]), collapse = ", ")
)
}

if (length(no_imp_with_missing)) {
  pred[, intersect(no_imp_with_missing, colnames(pred))] <- 0
  message("MICE: excluded non-imputed missing predictors: ",
         paste(no_imp_with_missing, collapse = ", "))
}

if (length(pred_cols_check)) {
  message(
    "MICE: predictor column sums (post-exclude) for ",
    paste(pred_cols_check, collapse = ", "),
    ": ",
    paste(colSums(pred[, pred_cols_check, drop = FALSE]), collapse = ", "))
}

# Ensure key covariates with missingness have predictors (avoid zero-row pred)
core_preds <- intersect(
  c("age_at_encounter", "sex", "race_ethnicity", "location", "encounter_type",
    "has_abg", "has_vbg", "imv_proc", "niv_proc", "death_60d",
    "hypercap_resp_failure", "paco2", "vbg_co2"),
  colnames(pred)
)

core_preds <- setdiff(core_preds, no_imp_with_missing)
vars_need_pred <- intersect(covars_ps, rownames(pred))
vars_need_pred <- setdiff(vars_need_pred, no_imp)
vars_need_pred <- vars_need_pred[vapply(mi_df[vars_need_pred], function(x) any(is.na(x)), logical(1))]
zero_pred <- vars_need_pred[rowSums(pred[vars_need_pred, , drop = FALSE] != 0, na.rm = TRUE) == 0]
if (length(zero_pred)) {
  message("MICE predictor rows empty for: ", paste(zero_pred, collapse = ", "),
         ". Adding core predictors.")
  pred[zero_pred, core_preds] <- 1
}
pred[intersect(rownames(pred), no_imp), ] <- 0
pred[c(zero_pred), c(zero_pred)] <- 0

```

```

# --- Preflight and cap predictor rows to avoid huge model matrices ----

n_mm_cols <- function(pred_row, df) {
  preds <- names(which(pred_row != 0))
  cols <- 1L
  for (p in preds) {
    x <- df[[p]]
    if (is.factor(x)) {
      lv <- nlevels(x)
      cols <- cols + max(1L, lv - 1L)
    } else {
      cols <- cols + 1L
    }
  }
  cols
}

pred_width_preflight <- function(pred, df, meth) {
  vars <- names(meth)[meth != ""]
  vars <- vars[vapply(df[vars], function(x) any(is.na(x)), logical(1))]
  rows <- lapply(vars, function(v) {
    pred_row <- pred[v, ]
    n_pred <- sum(pred_row != 0, na.rm = TRUE)
    mm_cols <- n_mm_cols(pred_row, df)
    x <- df[[v]]
    nlevels_v <- if (is.factor(x)) nlevels(x) else NA_integer_
    miss_n <- sum(is.na(x))
    data.frame(
      variable = v,
      method = meth[[v]],
      n_pred = n_pred,
      mm_cols = mm_cols,
      nlevels_v = nlevels_v,
      miss_n = miss_n,
      stringsAsFactors = FALSE
    )
  })
}

```

```

    })
  if (!length(rows)) {
    return(data.frame(variable = character(), method = character(), n_pred = integer(),
                      mm_cols = integer(), nlevels_v = integer(), miss_n = integer()))
  }
  dplyr::bind_rows(rows)
}

preflight_pre <- pred_width_preflight(pred, mi_df, meth) |>
  dplyr::mutate(stage = "pre")

core_preds <- intersect(
  c("age_at_encounter", "sex", "race_ethnicity", "location", "encounter_type",
    "has_abg", "has_vbg", "imv_proc", "niv_proc", "death_60d",
    "hypercap_resp_failure", "paco2", "vbg_co2"),
  colnames(pred))
)
core_preds <- setdiff(core_preds, no_imp_with_missing)

set.seed(MI_SEED + 100)
idx <- sample.int(nrow(mi_df), min(COR_SAMPLE_N, nrow(mi_df)))

vars_cap <- names(meth)[meth != ""]
vars_cap <- vars_cap[vapply(mi_df[vars_cap], function(x) any(is.na(x))), logical(1))]

pred_nlevels <- function(x) {
  if (is.factor(x)) nlevels(x) else 1L
}

for (v in vars_cap) {
  cand <- names(which(pred[v, ] != 0))
  if (!length(cand)) next
  keep <- intersect(core_preds, cand)
  rem <- setdiff(cand, keep)

  if (length(rem)) {
    y_raw <- mi_df[[v]][idx]
  }
}

```

```

y <- if (is.factor(y_raw)) as.integer(y_raw) else suppressWarnings(as.numeric(y_raw))
scores <- vapply(rem, function(r) {
  x_raw <- mi_df[[r]][idx]
  x <- if (is.factor(x_raw)) as.integer(x_raw) else suppressWarnings(as.numeric(x_raw))
  suppressWarnings(abs(stats::cor(y, x, use = "pairwise.complete.obs")))
}, numeric(1))
ord <- order(is.na(scores), -scores)
rem_keep <- rem[ord]
rem_keep <- rem_keep[seq_len(min(length(rem_keep), max(0L, MAX_PRED_PER_VAR - length(keep))))]
} else {
  rem_keep <- character()
}

keep_all <- unique(c(keep, rem_keep))
pred[v, ] <- 0
pred[v, keep_all] <- 1
pred[v, v] <- 0

mm_cols <- n_mm_cols(pred[v, ], mi_df)
if (mm_cols > MAX_MM_COLS) {
  drop_pool <- setdiff(keep_all, keep)
  if (!length(drop_pool)) drop_pool <- keep
  drop_order <- drop_pool[order(vapply(drop_pool, function(p) pred_nlevels(mi_df[[p]]), integer(1)),
                                 decreasing = TRUE)]
  for (p in drop_order) {
    if (p %in% keep && length(keep) == 1) break
    pred[v, p] <- 0
    mm_cols <- n_mm_cols(pred[v, ], mi_df)
    if (mm_cols <= MAX_MM_COLS) break
  }
}
}

preflight_post <- pred_width_preflight(pred, mi_df, meth) |>
  dplyr::mutate(stage = "post")
preflight_all <- dplyr::bind_rows(preflight_pre, preflight_post)
write_csv_safely(preflight_all, results_path("mice_pred_width_preflight.csv"), row_names = FALSE)

```

```

# MI integrity: treatments/outcomes excluded; binaries use logreg
stopifnot(all(meth[no_imp] == ""))
stopifnot(all(rowSums(pred[intersect(rownames(pred), no_imp), , drop = FALSE]) == 0))
bin_fac <- names(which(vapply(mi_df, function(x) is.factor(x) && nlevels(x) == 2, logical(1))))
bin_fac <- setdiff(bin_fac, no_imp)
stopifnot(all(meth[bin_fac] == "logreg"))

# integrity checks
stopifnot(
  ncol(pred) == ncol(mi_df),
  nrow(pred) == ncol(mi_df),
  length(meth) == ncol(mi_df),
  identical(names(meth), colnames(mi_df)))
)

# --- run MICE -----
mi_df_run <- mi_df
M_IMP_RUN <- M_IMP
MAXIT_RUN <- MAXIT_MI
FORCE_MI_BATCHED <- nrow(mi_df_run) > MI_BATCH_THRESHOLD

if (!requireNamespace("digest", quietly = TRUE)) {
  stop("Package 'digest' is required for MI checkpoint signatures.")
}

make_mi_signature <- function(df) {
  classes <- vapply(df, function(x) class(x)[1], character(1))
  na_counts <- vapply(df, function(x) sum(is.na(x)), integer(1))
  nlevels <- vapply(df, function(x) if (is.factor(x)) nlevels(x) else NA_integer_, integer(1))
  lvl_hash <- vapply(df, function(x) {
    if (!is.factor(x)) return(NA_character_)
    digest::digest(levels(x), algo = "xxhash64")
  }, character(1))
  col_sig <- data.frame(
    name = names(df),
    class = classes,

```

```

na = na_counts,
nlevels = nlevels,
lvl_hash = lvl_hash,
stringsAsFactors = FALSE
)
hash <- digest::digest(
  list(dim = dim(df), col_sig = col_sig),
  algo = "xxhash64"
)
list(
  hash = hash,
  nrow = nrow(df),
  ncol = ncol(df),
  run_mode = RUN_MODE,
  pilot_frac = PILOT_FRAC,
  mi_pilot_mode = RUN_MODE,
  sample_seed = SAMPLE_SEED,
  mi_seed = MI_SEED,
  col_sig = col_sig
)
}

mi_smoke_log_path <- results_path("mice_smoketest.log")
write_smoke_log <- function(lines) {
  write_diag_lines(lines, mi_smoke_log_path)
}

mem_max <- tryCatch(mem.maxVSize(), error = function(e) NA_real_)
mem_env <- Sys.getenv("R_MAX_VSIZE", unset = NA_character_)
gc_pre <- utils::capture.output(gc())
write_smoke_log(c(
  paste0("MI smoke test log: ", Sys.time()),
  paste0("mem.maxVSize: ", ifelse(is.na(mem_max), "NA", format(mem_max, scientific = FALSE))),
  paste0("R_MAX_VSIZE env: ", ifelse(nchar(mem_env), mem_env, "NA")),
  paste0("mi_df size: ", format(utils::object.size(mi_df), units = "auto")),
  paste0("mi_df_run size: ", format(utils::object.size(mi_df_run), units = "auto")),
  "gc() pre:",
```

```

    gc_pre
))
if (RUN_MODE == "full" && is.finite(mem_max) && mem_max < 2.2e10) {
  message("Full run on ~16GB memory: consider running MI/weights on a >32GB machine for speed.")
}

subset_data_saved <- FALSE
subset_data_path <- results_path("subset_data_pre_mi.rds")
if (MI_MEMORY_HYGIENE) {
  saveRDS(subset_data, subset_data_path)
  rm(subset_data)
  subset_data_saved <- TRUE
  invisible(gc())
}

run_mice_call <- function(m_val, maxit, seed_val, print_flag = FALSE) {
  mice::mice(
    data      = mi_df_run,
    m         = m_val,
    maxit     = maxit,
    predictorMatrix = pred,
    method    = meth,
    printFlag = print_flag,
    seed      = seed_val
  )
}

if (MI_SMOKE_TEST) {
  smoke_con <- file(mi_smoke_log_path, open = "at")
  sink(smoke_con, type = "output")

  cap_smoke <- capture_warnings(
    tryCatch(
      run_mice_call(m_val = 1L, maxit = 1L, seed_val = MI_SEED, print_flag = TRUE),
      error = function(e) e
    ),
    context = make_context(

```

```

stage = "MI",
component = "mice_smoketest",
analysis_variant = "mi",
model_type = "mice",
group = NA_character_,
outcome = NA_character_,
imputation = NA_integer_,
batch = NA_integer_
)
)
append_warnings(cap_smoke$warnings)
sink(type = "output")
close(smoke_con)

if (inherits(cap_smoke$value, "error")) {
  smoke_msg <- conditionMessage(cap_smoke$value)
  message("MICE smoke test failed: ", smoke_msg)
  write_smoke_log(c(
    "Smoke test failed.",
    paste0("Error: ", smoke_msg)
  ))
  if (grepl("vector memory limit", smoke_msg, fixed = TRUE)) {
    MI_DEBUG_PRINTFLAG <- TRUE
  }
  if (subset_data_saved) {
    subset_data <- readRDS(subset_data_path)
  }
  if (MI_DEBUG_PRINTFLAG) {
    debug_path <- results_path("mice_debug_print.txt")
    dbg_con <- file(debug_path, open = "wt")
    sink(dbg_con, type = "output")
    message("Debug MI: running m=1, maxit=1 with printFlag=TRUE.")
    tryCatch(
      run_mice_call(m_val = 1L, maxit = 1L, seed_val = MI_SEED, print_flag = TRUE),
      error = function(e) e
    )
    sink(type = "output")
  }
}

```

```

        close(dbg_con)
    }
    stop("MICE smoke test failed; see ", mi_smoke_log_path, " and Results/mice_debug_print.txt.")
} else {
    message("MICE smoke test succeeded (m=1, maxit=1).")
}
}

get_vcells_stats <- function() {
    g <- gc()
    cn <- colnames(g)
    mb_cols <- cn[grep("\\\\(Mb\\\\)", cn)]
    used_mb_col <- if (length(mb_cols)) mb_cols[1] else NA_character_
    limit_col <- cn[grep("limit", cn, ignore.case = TRUE)]
    limit_col <- if (length(limit_col)) limit_col[1] else NA_character_
    trig_col <- cn[grep("trigger", cn, ignore.case = TRUE)]
    trig_col <- if (length(trig_col)) trig_col[1] else NA_character_
    max_col <- cn[grep("max", cn, ignore.case = TRUE)]
    max_col <- if (length(max_col)) max_col[1] else NA_character_

    used_mb <- if (!is.na(used_mb_col)) as.numeric(g["Vcells", used_mb_col]) else NA_real_
    limit_mb <- if (!is.na(limit_col)) as.numeric(g["Vcells", limit_col]) else NA_real_
    trig_mb <- if (!is.na(trig_col)) as.numeric(g["Vcells", trig_col]) else NA_real_
    max_mb <- if (!is.na(max_col)) as.numeric(g["Vcells", max_col]) else NA_real_

    if (!is.finite(limit_mb) || limit_mb <= 0) {
        mem_lim <- tryCatch(mem.maxVSize(), error = function(e) NA_real_)
        if (is.finite(mem_lim) && mem_lim > 0) {
            limit_mb <- if (mem_lim < 1e8) mem_lim else mem_lim / 1024^2
        }
    }

    data.frame(
        gc_vcells_used_mb = used_mb,
        gc_vcells_limit_mb = limit_mb,
        gc_vcells_frac = ifelse(is.finite(limit_mb) && limit_mb > 0, used_mb / limit_mb, NA_real_),
        gc_vcells_trigger_mb = trig_mb,

```

```

    gc_vcells_max_used_mb = max_mb,
    stringsAsFactors = FALSE
  )
}

set.seed(MI_SEED)
run_mice_single <- function(m_val) {
  runtime_logger(
    "mice_imputation",
    run_mice_call(m_val = m_val, maxit = MAXIT_RUN, seed_val = MI_SEED, print_flag = FALSE),
    notes = paste0("m=", m_val, "; maxit=", MAXIT_RUN)
  )
}

mc_progress <- list()
sentinel_specs <- list(
  list(name = "abg_imv", outcome = "imv_proc", treat = "has_abg", co2_var = "paco2",
       low = ABG_CO2_LOW, high = ABG_CO2_HIGH),
  list(name = "vbg_imv", outcome = "imv_proc", treat = "has_vbg", co2_var = "vbg_co2",
       low = VBG_CO2_LOW, high = VBG_CO2_HIGH)
)

mcerr_ratio_for_spec <- function(imp_obj, spec) {
  fits <- lapply(seq_len(imp_obj$m), function(i) {
    d <- mice::complete(imp_obj, action = i)
    d <- d[, c(spec$outcome, spec$treat, spec$co2_var), drop = FALSE]
    d <- d[d[[spec$treat]] == 1 & is.finite(d[[spec$co2_var]]), , drop = FALSE]
    if (nrow(d) == 0L) return(NULL)
    d$co2_cat <- make_co2_cat3(d[[spec$co2_var]], spec$low, spec$high)
    d$co2_cat <- stats::relevel(base::droplevels(d$co2_cat), ref = "Normal")
    if (dplyr::n_distinct(d[[spec$outcome]]) < 2L) return(NULL)
    fit <- tryCatch(
      stats::glm(stats::reformulate("co2_cat", response = spec$outcome),
                 data = d, family = binomial(), x = FALSE, y = FALSE, model = FALSE),
      error = function(e) NULL
    )
    if (is.null(fit)) return(NULL)
  })
}

```

```

    list(coef = stats::coef(fit), vcov = stats::vcov(fit))
  })
fits <- fits[!vapply(fits, is.null, logical(1))]
if (length(fits) < 2L) return(NA_real_)
results <- lapply(fits, `[[`, "coef")
variances <- lapply(fits, `[[`, "vcov")
pooled <- mitools::MIcombine(results = results, variances = variances)
est <- as.numeric(stats::coef(pooled))
names(est) <- names(stats::coef(pooled))
se <- sqrt(diag(pooled$variance))
names(se) <- names(stats::coef(pooled))
coef_mat <- do.call(cbind, lapply(results, function(v) v[names(est)]))
B <- apply(coef_mat, 1, stats::var, na.rm = TRUE)
mcerr <- sqrt(B / length(results))
ratio <- mcerr / se
idx <- grep("co2_cat", names(ratio))
if (!any(idx)) return(max(ratio, na.rm = TRUE))
max(ratio[idx], na.rm = TRUE)
}

run_mice_batched <- function(m_total, m_batch_start, maxit, base_seed) {
  imp_acc <- NULL
  m_done <- 0L
  batch_attempt_idx <- 0L
  m_batch <- m_batch_start
  batch_log <- list()
  logged_events_acc <- list()

  while (m_done < m_total) {
    if (MI_GC_EVERY_BATCH) invisible(gc())
    mem_stats <- get_vcells_stats()
    if (MI_PREEMPTIVE_BATCH_REDUCE &&
        is.finite(mem_stats$gc_vcells_frac) &&
        mem_stats$gc_vcells_frac > MI_VCELLS_FRAC_THRESHOLD &&
        m_batch > MI_BATCH_MIN) {
      m_batch <- max(MI_BATCH_MIN, floor(m_batch / 2))
      message("Preemptively reducing MI batch size to ", m_batch,

```

```

    " (Vcells pressure: ", round(mem_stats$gc_vcells_frac, 2), ".")
}

m_b <- min(m_batch, m_total - m_done)
batch_attempt_idx <- batch_attempt_idx + 1L
seed_b <- base_seed + batch_attempt_idx * MI_BATCH_SEED_STRIDE
message("MICE batch ", batch_attempt_idx, " (m=", m_b, ", seed=", seed_b, ")")

t0 <- Sys.time()
mem_pre <- get_vcells_stats()
cap <- capture_warnings(
  tryCatch(
    runtime_logger(
      paste0("mice_batch_", batch_attempt_idx),
      mice::mice(
        data = mi_df_run,
        m = m_b,
        maxit = maxit,
        predictorMatrix = pred,
        method = meth,
        printFlag = FALSE,
        seed = seed_b
      ),
      notes = paste0("batch=", batch_attempt_idx, "; m=", m_b, "; maxit=", maxit)
    ),
    error = function(e) e
  ),
  context = make_context(
    stage = "MI",
    component = "mice_batch",
    analysis_variant = "mi",
    model_type = "mice",
    group = NA_character_,
    outcome = NA_character_,
    imputation = NA_integer_,
    batch = batch_attempt_idx
  )
)

```

```

)
append_warnings(cap$warnings)

imp_b <- cap$value
if (inherits(imp_b, "error")) {
  err_msg <- conditionMessage(imp_b)
  message("MICE batch ", batch_attempt_idx, " failed: ", err_msg)
  mem_stats <- get_vcells_stats()
  batch_log[[batch_attempt_idx]] <- data.frame(
    batch = batch_attempt_idx,
    m_batch = m_b,
    seed = seed_b,
    ok = FALSE,
    error_message = err_msg,
    seconds = as.numeric(difftime(Sys.time(), t0, units = "secs")),
    gc_vcells_used_mb_pre = mem_pre$gc_vcells_used_mb,
    gc_vcells_limit_mb_pre = mem_pre$gc_vcells_limit_mb,
    gc_vcells_frac_pre = mem_pre$gc_vcells_frac,
    gc_vcells_used_mb_post = mem_stats$gc_vcells_used_mb,
    gc_vcells_limit_mb_post = mem_stats$gc_vcells_limit_mb,
    gc_vcells_frac_post = mem_stats$gc_vcells_frac,
    stringsAsFactors = FALSE
  )
  if (grepl("vector memory limit", err_msg, fixed = TRUE) && m_batch > MI_BATCH_MIN) {
    m_batch <- max(MI_BATCH_MIN, floor(m_batch / 2))
    message("Reducing MI batch size to ", m_batch, " and retrying.")
    invisible(gc())
    next
  }
  write_csv_safely(dplyr::bind_rows(batch_log), results_path("mice_batches_log.csv"), row_names = FALSE)
  stop("MICE batch ", batch_attempt_idx, " failed; see log: ", results_path("mice_batches_log.csv"))
}

if (is.null(imp_acc)) {
  imp_acc <- imp_b
} else {
  imp_acc <- mice::ibind(imp_acc, imp_b)
}

```

```

}

le_b <- imp_b$loggedEvents
le_b <- if (is.null(le_b)) data.frame() else as.data.frame(le_b)
if (NROW(le_b) > 0) {
  le_b <- le_b |>
    dplyr::mutate(
      batch = batch_attempt_idx,
      seed = seed_b,
      m_global_start = m_done + 1L
    )
  logged_events_acc[[length(logged_events_acc) + 1L]] <- le_b
}

m_done <- imp_acc$m
mem_stats <- get_vcells_stats()
batch_log[[batch_attempt_idx]] <- data.frame(
  batch = batch_attempt_idx,
  m_batch = m_b,
  seed = seed_b,
  ok = TRUE,
  error_message = NA_character_,
  seconds = as.numeric(difftime(Sys.time(), t0, units = "secs")),
  gc_vcells_used_mb_pre = mem_pre$gc_vcells_used_mb,
  gc_vcells_limit_mb_pre = mem_pre$gc_vcells_limit_mb,
  gc_vcells_frac_pre = mem_pre$gc_vcells_frac,
  gc_vcells_used_mb_post = mem_stats$gc_vcells_used_mb,
  gc_vcells_limit_mb_post = mem_stats$gc_vcells_limit_mb,
  gc_vcells_frac_post = mem_stats$gc_vcells_frac,
  stringsAsFactors = FALSE
)
rm(imp_b)
if (MI_GC_EVERY_BATCH) invisible(gc())

if (ALLOW_M_IMP_EARLY_STOP &&
  m_done >= M_IMP_MIN &&
  (m_done %% M_IMP_STEP == 0 || m_done == m_total)) {

```

```

ratios <- vapply(sentinel_specs, function(s) mcerr_ratio_for_spec(imp_acc, s), numeric(1))
mc_progress[[length(mc_progress) + 1L]] <- data.frame(
  m = m_done,
  abg_ratio = ratios[["abg_imv"]],
  vbg_ratio = ratios[["vbg_imv"]],
  max_ratio = max(ratios, na.rm = TRUE),
  stringsAsFactors = FALSE
)
write_csv_safely(dplyr::bind_rows(mc_progress),
  results_path("mi_mcerr_progress.csv"),
  row_names = FALSE)
if (all(is.finite(ratios)) && max(ratios, na.rm = TRUE) <= MCERR_RATIO_TARGET) {
  message("MC error criterion met at m=", m_done,
    " (max MCerr/SE=", round(max(ratios, na.rm = TRUE), 3), "). Stopping early.")
  break
}
}
}

write_csv_safely(dplyr::bind_rows(batch_log), results_path("mice_batches_log.csv"), row_names = FALSE)
log_events_raw_batched <- dplyr::bind_rows(logged_events_acc)
attr(imp_acc, "logged_events_batched") <- log_events_raw_batched
imp_acc
}

imp <- NULL
use_batched <- isTRUE(FORCE_MI_BATCHED)
if (!use_batched) {
  cap_mice <- capture_warnings(
    tryCatch(run_mice_single(M_IMP_RUN), error = function(e) e),
    context = make_context(
      stage = "MI",
      component = "mice",
      analysis_variant = "mi",
      model_type = "mice",
      group = NA_character_
    )
  )
}

```

```

    outcome = NA_character_,
    imputation = NA_integer_,
    batch = NA_integer_
  )
)
append_warnings(cap_mice$warnings)
imp <- cap_mice$value
if (inherits(imp, "error") && grepl("vector memory limit", conditionMessage(imp), fixed = TRUE)) {
  message("MICE memory limit hit; switching to batched mode.")
  use_batched <- TRUE
}
}
if (use_batched) {
  message("MICE: running in batches (start=", MI_BATCH_START, ").")
  imp <- run_mice_batched(M_IMP_RUN, MI_BATCH_START, MAXIT_RUN, MI_SEED)
} else {
  write_csv_safely(data.frame(), results_path("mice_batches_log.csv"), row_names = FALSE)
}
if (inherits(imp, "error")) stop(imp)
stopifnot(inherits(imp, "mids"))
if (ALLOW_M_IMP_EARLY_STOP && imp$m < M_IMP_RUN) {
  message("Early stop: stopping at m=", imp$m, " (target ", M_IMP_RUN, ").")
  M_IMP_RUN <- imp$m
} else {
  stopifnot(imp$m == M_IMP_RUN)
}
if (M_IMP != M_IMP_RUN) M_IMP <- M_IMP_RUN
if (MAXIT_MI != MAXIT_RUN) MAXIT_MI <- MAXIT_RUN
write_csv_safely(dplyr::bind_rows(mc_progress), results_path("mi_mcerr_progress.csv"), row_names = FALSE)
saveRDS(imp, file = mi_mids_file)

# Save MICE spec for reproducibility
saveRDS(
  list(method = imp$method, predictorMatrix = imp$predictorMatrix),
  results_path("mice_spec.rds")
)
if (use_batched) {

```

```

  message("Multiple imputation was run in batches and combined via mice::ibind() .")
}

if (subset_data_saved) {
  subset_data <- readRDS(subset_data_path)
}

# Logged events: raw + summary (by dep/out/meth)
log_events_raw <- as.data.frame(imp$loggedEvents)
log_events_batched <- as.data.frame(attr(imp, "logged_events_batched"))
log_events_raw <- dplyr::bind_rows(log_events_raw, log_events_batched)

if (nrow(log_events_raw)) {
  write_csv_safely(log_events_raw, results_path("mice_logged_events_raw.csv"), row_names = FALSE,
                  required_cols = c("dep", "out", "meth"))
  log_events_summary <- log_events_raw |>
    dplyr::count(dep, out, meth, name = "n") |>
    dplyr::mutate(variable = dep) |>
    dplyr::arrange(dplyr::desc(n)) |>
    dplyr::mutate(pct = n / sum(n))
  write_csv_safely(log_events_summary, results_path("mice_logged_events_summary.csv"), row_names = FALSE,
                  required_cols = c("variable", "n", "pct"))
} else {
  log_events_raw_empty <- data.frame(
    dep = character(), out = character(), meth = character(),
    stringsAsFactors = FALSE
)
  log_events_summary_empty <- data.frame(
    variable = character(), n = integer(), pct = numeric(),
    stringsAsFactors = FALSE
)
  write_csv_safely(log_events_raw_empty, results_path("mice_logged_events_raw.csv"), row_names = FALSE,
                  required_cols = c("dep", "out", "meth"))
  write_csv_safely(log_events_summary_empty, results_path("mice_logged_events_summary.csv"), row_names = FALSE,
                  required_cols = c("variable", "n", "pct"))
  log_events_summary <- log_events_summary_empty
}
if (nrow(mi_info_log)) {

```

```

warns_events <- mi_info_log |>
  dplyr::filter(stage == "MI", component %in% c("mice", "mice_batch"))
if (nrow(warns_events) && nrow(log_events_raw) == 0L) {
  warning("Mismatch: main MI run reported logged events but loggedEvents table is empty; ",
         "check batch capture and loggedEvents exports.", call. = FALSE)
}
}

# Chain diagnostics (lightweight; no complete("long"))
chain_diag <- data.frame()
chain_diag_stats <- list(
  n_imputed_vars = 0L,
  n_with_chainMean = 0L,
  n_with_drift_tail = 0L,
  drift_tail_na_frac = NA_real_,
  tail_window_na_mean = NA_real_
)
stopifnot(!is.null(imp$method))
impute_vars <- names(imp$method)[imp$method != ""]
impute_vars <- intersect(impute_vars, names(imp$data))
if (length(impute_vars)) {
  impute_vars <- impute_vars[vapply(imp$data[impute_vars], function(x) any(is.na(x)), logical(1))]
}
stopifnot(!is.null(imp$chainMean))
{
  cm <- imp$chainMean
  dims <- dim(cm)
  dn <- dimnames(cm)
  iter_candidates <- unique(c(imp$iteration, MAXIT_MI, MAXIT_MI + 1L))
  iter_candidates <- iter_candidates[is.finite(iter_candidates) & iter_candidates > 0]
  imp_m <- imp$m

  extract_chain_mean <- function(cm_obj, dims_obj, dn_obj, imp_m_val, iter_cand) {
    mean_chain <- NULL
    var_names <- NULL
    iter_dim <- NA_integer_
    var_dim <- NA_integer_

```

```

m_dim <- NA_integer_
if (!is.null(dims_obj) && length(dims_obj) == 2) {
  iter_dim <- which(dims_obj %in% iter_cand)[1]
  if (length(iter_dim) == 0) iter_dim <- 1L
  var_dim <- setdiff(seq_along(dims_obj), iter_dim)[1]
  mean_chain <- cm_obj
  if (iter_dim == 2L) mean_chain <- t(mean_chain)
} else if (!is.null(dims_obj) && length(dims_obj) == 3) {
  m_dim <- which(dims_obj == imp_m_val)
  if (length(m_dim)) {
    m_dim <- m_dim[1]
  } else {
    m_dim <- 3L
  }
  iter_dim <- setdiff(which(dims_obj %in% iter_cand), m_dim)[1]
  if (length(iter_dim) == 0) iter_dim <- setdiff(1:3, c(m_dim, NA_integer_))[1]
  var_dim <- setdiff(1:3, c(iter_dim, m_dim))[1]
  if (all(is.finite(c(iter_dim, var_dim, m_dim)))) {
    cm_std <- aperm(cm_obj, c(iter_dim, var_dim, m_dim))
    mean_chain <- apply(cm_std, c(1, 2), mean, na.rm = TRUE)
    if (!is.null(dimnames(cm_std))) {
      if (!is.null(dimnames(cm_std)[[2]])) {
        var_names <- dimnames(cm_std)[[2]]
      }
      if (!is.null(dimnames(cm_std)[[1]])) {
        rownames(mean_chain) <- dimnames(cm_std)[[1]]
      }
    }
  }
}
if (is.null(mean_chain)) {
  return(list(mean_chain = NULL, var_names = NULL, iter_dim = iter_dim, var_dim = var_dim, m_dim = m_dim))
}
if (is.null(var_names) && !is.null(dn_obj) && length(dn_obj) >= var_dim) {
  var_names <- dn_obj[[var_dim]]
}
if (is.null(var_names)) {

```

```

    var_names <- colnames(mean_chain)
}
list(mean_chain = mean_chain, var_names = var_names, iter_dim = iter_dim, var_dim = var_dim, m_dim = m_dim)
}

res_chain <- extract_chain_mean(cm, dims, dn, imp_m, iter_candidates)
mean_chain <- res_chain$mean_chain
var_names <- res_chain$var_names
if (is.null(mean_chain)) {
  stop("Chain diagnostics: unable to construct mean_chain from imp$chainMean.")
}

{
  iter_idx <- seq_len(nrow(mean_chain))
  numeric_names <- !is.null(var_names) && all(grepl("^\\d+$", var_names))
  if (is.null(var_names) || numeric_names) {
    if (length(impute_vars) && length(impute_vars) == ncol(mean_chain)) {
      var_names <- impute_vars
      numeric_names <- FALSE
    }
  }
  if (is.null(var_names)) {
    stop("Chain diagnostics: variable names missing and could not be matched to imp$method.")
  }
  colnames(mean_chain) <- var_names

  if (numeric_names) {
    warn_df <- data.frame(
      time = as.character(Sys.time()),
      stage = "MI",
      component = "chain_diagnostics",
      analysis_variant = NA_character_,
      model_type = NA_character_,
      group = NA_character_,
      outcome = NA_character_,
      imputation = NA_integer_,
      batch = NA_integer_,

```

```

    message = "Chain diagnostics variable name mapping failed; using numeric/fallback names.",
    stringsAsFactors = FALSE
)
append_warnings(warn_df)
}

vars_imputed <- impute_vars
chain_diag_stats$n_imputed_vars <- length(vars_imputed)
keep_vars <- intersect(vars_imputed, var_names)
missing_in_chain <- setdiff(vars_imputed, var_names)
if (length(missing_in_chain)) {
  missing_df <- data.frame(
    variable = missing_in_chain,
    stringsAsFactors = FALSE
)
  write_csv_safely(missing_df, results_path("mice_chain_diagnostics_missing_vars.csv"), row.names = FALSE)
}
if (length(vars_imputed) && length(keep_vars) == 0L) {
  warn_df <- data.frame(
    time = as.character(Sys.time()),
    stage = "MI",
    component = "chain_diagnostics",
    analysis_variant = NA_character_,
    model_type = NA_character_,
    group = NA_character_,
    outcome = NA_character_,
    imputation = NA_integer_,
    batch = NA_integer_,
    message = "Chain diagnostics: no imputed variables matched chainMean names; skipping drift metrics.",
    stringsAsFactors = FALSE
)
  append_warnings(warn_df)
mean_chain <- mean_chain[, 0, drop = FALSE]
var_names <- character()
} else if (length(keep_vars)) {
  mean_chain <- mean_chain[, keep_vars, drop = FALSE]
  var_names <- keep_vars
}

```

```

}

safe_sd <- function(x) if (sum(is.finite(x)) < 2) NA_real_ else stats::sd(x, na.rm = TRUE)
safe_maxdiff <- function(x) {
  x <- x[is.finite(x)]
  if (length(x) < 2) return(NA_real_)
  max(abs(diff(x)))
}
safe_slope <- function(x, iter) {
  ok <- is.finite(x)
  if (sum(ok) < 2) return(NA_real_)
  coef(stats::lm(x[ok] ~ iter[ok]))[2]
}
tail_finite <- function(x, k) tail(x[is.finite(x)], k)
safe_maxdiff_tail <- function(x, k) {
  xf <- tail_finite(x, k)
  if (length(xf) < 2) return(NA_real_)
  max(abs(diff(xf)))
}

n_vars <- ncol(mean_chain)
n_finite <- integer(n_vars)
drift_all <- numeric(n_vars)
drift_tail <- numeric(n_vars)
tail_n_finite <- integer(n_vars)
tail_window_na_frac <- numeric(n_vars)
slope <- numeric(n_vars)
sd_chain <- numeric(n_vars)
overall_reason <- character(n_vars)
tail_reason <- character(n_vars)
flag_reason <- character(n_vars)
flag <- logical(n_vars)
diagnostic_available <- logical(n_vars)
sd_obs <- rep(NA_real_, n_vars)

chain_src_df <- mi_df_run
stopifnot(is.data.frame(chain_src_df))

```

```

sd_obs <- vapply(var_names, function(v) {
  stopifnot(v %in% names(chain_src_df))
  x <- chain_src_df[[v]]
  if (inherits(x, "haven_labelled")) x <- suppressWarnings(as.numeric(x))
  if (!is.numeric(x)) return(NA_real_)
  safe_sd(x)
}, numeric(1))

for (j in seq_len(n_vars)) {
  x <- mean_chain[, j]
  n_finite[j] <- sum(is.finite(x))
  drift_all[j] <- safe_maxdiff(x)
  slope[j] <- safe_slope(x, iter_idx)
  sd_chain[j] <- safe_sd(x)
  tail_vals <- tail_finite(x, TAIL_NFINITE)
  tail_n_finite[j] <- length(tail_vals)
  drift_tail[j] <- safe_maxdiff_tail(x, TAIL_NFINITE)
  tail_window <- tail(x, min(TAIL_WINDOW_ITERS, length(x)))
  tail_window_na_frac[j] <- mean(!is.finite(tail_window))
  overall_reason[j] <- if (n_finite[j] >= 2) "ok" else "insufficient_finite_overall"
  tail_reason[j] <- if (tail_n_finite[j] >= 2) "ok" else "insufficient_finite_tail"
}

drift_all_scaled <- drift_all / sd_obs
drift_tail_scaled <- drift_tail / sd_obs
slope_scaled <- slope / sd_obs

for (j in seq_len(n_vars)) {
  if (!is.finite(sd_obs[j]) || sd_obs[j] <= 0) {
    flag[j] <- NA
    flag_reason[j] <- "missing_scale"
    diagnostic_available[j] <- FALSE
    next
  }
  if (tail_n_finite[j] < 2) {
    flag[j] <- NA
    flag_reason[j] <- "tail_insufficient"
  }
}

```

```

  diagnostic_available[j] <- FALSE
  next
}
if (!is.finite(drift_tail_scaled[j]) || !is.finite(slope_scaled[j])) {
  flag[j] <- NA
  flag_reason[j] <- "insufficient_data"
  diagnostic_available[j] <- FALSE
  next
}
flag_tail <- drift_tail_scaled[j] > 0.01
flag_slope <- abs(slope_scaled[j]) > 0.001
flag[j] <- flag_tail | flag_slope
flag_reason[j] <- if (flag_tail & flag_slope) {
  "both"
} else if (flag_tail) {
  "tail_drift"
} else if (flag_slope) {
  "slope"
} else {
  "none"
}
diagnostic_available[j] <- TRUE
}

chain_diag <- data.frame(
  variable = var_names,
  method = imp$method[var_names],
  n_finite = n_finite,
  drift_all = drift_all,
  drift_tail = drift_tail,
  drift_all_scaled = drift_all_scaled,
  drift_tail_scaled = drift_tail_scaled,
  tail_n_finite = tail_n_finite,
  tail_window_na_frac = tail_window_na_frac,
  slope = slope,
  slope_scaled = slope_scaled,
  sd_chain = sd_chain,

```

```

sd_obs = sd_obs,
overall_reason = overall_reason,
tail_reason = tail_reason,
flag_reason = flag_reason,
diagnostic_available = diagnostic_available,
flag = flag,
stringsAsFactors = FALSE
)

chain_diag_stats$n_with_chainMean <- nrow(chain_diag)
chain_diag_stats$n_with_drift_tail <- sum(is.finite(chain_diag$drift_tail))
if (chain_diag_stats$n_with_chainMean > 0) {
  chain_diag_stats$drift_tail_na_frac <- 1 - (chain_diag_stats$n_with_drift_tail /
                                                chain_diag_stats$n_with_chainMean)
  chain_diag_stats$tail_window_na_mean <- mean(chain_diag$tail_window_na_frac, na.rm = TRUE)
}
write_csv_safely(chain_diag, results_path("mice_chain_diagnostics.csv"), row.names = FALSE)
if (any(isTRUE(chain_diag$flag), na.rm = TRUE)) {
  frac_flag <- mean(chain_diag$flag, na.rm = TRUE)
  if (is.finite(frac_flag) && frac_flag < 0.05) {
    message("Chain diagnostics: low drift flags (", round(frac_flag * 100, 1),
           "%). MAXIT_MI may be reduced safely (consider 10 or 5).")
  }
}
}

# quick sanity: these must exist and be numeric in completed data
imp_n <- imp$m
get_imp_stats <- list(count = 0L, seconds = 0)
get_imp <- function(i, imp_obj = imp) {
  t0 <- Sys.time()
  d <- normalize_types(mice:::complete(imp_obj, action = i), levels_ref)
  get_imp_stats$count <-> get_imp_stats$count + 1L
  get_imp_stats$seconds <-> get_imp_stats$seconds +
    as.numeric(difftime(Sys.time(), t0, units = "secs"))
d
}

```

```

}

d1 <- get_imp(1)
stopifnot(all(c("paco2", "vbg_co2") %in% names(d1)))
stopifnot(is.numeric(d1$paco2), is.numeric(d1$vbg_co2))

# post-MICE sanity: no remaining NA in covars_ps
covars_check <- intersect(covars_ps, names(d1))
na_counts <- vapply(d1[, covars_check, drop = FALSE], function(x) sum(is.na(x)), numeric(1))
na_counts <- na_counts[na_counts > 0]
if (length(na_counts)) {
  message("Post-MICE NA counts (covars_ps): ",
         paste(names(na_counts), na_counts, collapse = ", "))
  ev_sum <- summarize_logged_events(imp)
  if (nrow(ev_sum)) {
    ev_sub <- ev_sum[ev_sum$variable %in% names(na_counts), , drop = FALSE]
    if (nrow(ev_sub)) {
      print(utils::head(ev_sub, 10))
    } else {
      message("No loggedEvents entries for covars_ps with NA.")
    }
  } else {
    message("No loggedEvents recorded.")
  }
  stop("Post-MICE check failed: remaining NA in covars_ps. See loggedEvents summary above.")
}

# Purpose: mi mcerror.
stopifnot(exists("imp"))

# Representative logistic model (unweighted) to assess Monte Carlo error
mc_diag_fml <- imv_proc ~ has_abg + age_at_encounter + curr_bmi + sex + encounter_type
register_model_diagram("MI MC error: IMV ~ ABG + covariates", mc_diag_fml, width = 10, height = 6)
cap_mc <- capture_warnings(
  tryCatch(
    with(
      imp,
      glm(imv_proc ~ has_abg + age_at_encounter + curr_bmi + sex + encounter_type,

```

```

    family = binomial(), x = FALSE, y = FALSE, model = FALSE)
),
error = function(e) e
),
context = make_context(
  stage = "diagnostics",
  component = "mi_mcerror_glm",
  analysis_variant = "mi",
  model_type = "glm",
  group = NA_character_,
  outcome = "imv_proc",
  imputation = NA_integer_,
  batch = NA_integer_
)
)
append_warnings(cap_mc$warnings)
mc_fit <- cap_mc$value

use_compact <- inherits(mc_fit, "error")
if (use_compact) {
  message("MC error: fallback to compact per-imputation fits (", conditionMessage(mc_fit), ".)")
}

mc_pool <- NULL
mc_sum <- NULL
mc_results <- NULL
mc_variances <- NULL
if (!use_compact) {
  mc_pool <- pool(mc_fit)
  mc_sum <- summary(mc_pool, conf.int = TRUE)
}

compute_mc_error <- function(mc_pool, mc_fit, m) {
  stopifnot(!is.null(mc_fit$analyses), length(mc_fit$analyses) >= 2)
  coefs <- lapply(mc_fit$analyses, coef)
  common_terms <- Reduce(intersect, lapply(coefs, names))
  stopifnot(length(common_terms) > 0)
}

```

```

Q <- do.call(cbind, lapply(coefs, function(v) v[common_terms]))
B <- apply(Q, 1, stats::var, na.rm = TRUE)
out <- sqrt(B / ncol(Q))
names(out) <- common_terms
out
}

compute_mc_error_from_results <- function(results_list) {
  coefs <- results_list
  common_terms <- Reduce(intersect, lapply(coefs, names))
  if (!length(common_terms)) return(NA_real_)
  Q <- do.call(cbind, lapply(coefs, function(v) v[common_terms]))
  B <- apply(Q, 1, stats::var, na.rm = TRUE)
  out <- sqrt(B / ncol(Q))
  names(out) <- common_terms
  out
}

if (use_compact) {
  mc_terms <- all.vars(mc_diag_fml)
  mc_fit_list <- lapply(seq_len(imp$m), function(i) {
    d <- mice::complete(imp, action = i)
    d <- d[, mc_terms, drop = FALSE]
    cap_i <- capture_warnings(
      tryCatch(
        glm(mc_diag_fml, data = d, family = binomial(), x = FALSE, y = FALSE, model = FALSE),
        error = function(e) e
      ),
      context = make_context(
        stage = "diagnostics",
        component = "mi_mcerror_glm",
        analysis_variant = "mi",
        model_type = "glm",
        group = NA_character_,
        outcome = "imv_proc",
        imputation = i,
        batch = NA_integer_
      )
    )
    cap_i[[i]] <- d
  })
}

```

```

    )
  )
append_warnings(cap_i$warnings)
fit_i <- cap_i$value
if (inherits(fit_i, "error")) return(list(error = conditionMessage(fit_i)))
list(coef = coef(fit_i), vcov = vcov(fit_i))
})

ok <- vapply(mc_fit_list, function(x) is.list(x) && is.null(x$error), logical(1))
mc_fit_list <- mc_fit_list[ok]
if (!length(mc_fit_list)) stop("MC error fallback failed: no successful fits.")

mc_results <- lapply(mc_fit_list, function(x) x$coef)
mc_variances <- lapply(mc_fit_list, function(x) x$vcov)

mc_pool <- mitools::MIcombine(results = mc_results, variances = mc_variances)
est <- as.numeric(coef(mc_pool))
se <- sqrt(diag(mc_pool$variance))
mc_sum <- data.frame(
  term = names(coef(mc_pool)),
  estimate = est,
  std.error = se,
  conf.low = est - 1.96 * se,
  conf.high = est + 1.96 * se,
  stringsAsFactors = FALSE
)
}

m_mc <- if (!use_compact && !is.null(mc_fit$analyses)) length(mc_fit$analyses) else imp$m
mc_err_vec <- if (!use_compact) {
  compute_mc_error(mc_pool, mc_fit, m_mc)
} else {
  compute_mc_error_from_results(mc_results)
}

if (is.null(names(mc_err_vec)) || all(names(mc_err_vec) == "")) {
  names(mc_err_vec) <- mc_sum$term[seq_len(min(length(mc_err_vec), nrow(mc_sum)))]
}

```

```

}

mc_err_aligned <- mc_err_vec[mc_sum$term]
if (all(is.na(mc_err_aligned))) {
  message("MC error not computed (all NA). mice version: ", as.character(packageVersion("mice")))
  if (!is.null(mc_pool$pooled)) {
    message("mc_pool$pooled columns: ", paste(names(mc_pool$pooled), collapse = ", "))
  }
}

stopifnot(all(c("conf.low", "conf.high") %in% names(mc_sum)))

mc_tab <- mc_sum |>
  mutate(
    mc_error = mc_err_aligned,
    mc_err_over_se = mc_error / std.error
  ) |>
  select(term, estimate, std.error, mc_error, mc_err_over_se, conf.low, conf.high)

mc_model_desc <- "Diagnostic model: imv_proc ~ has_abg + age_at_encounter + curr_bmi + sex + encounter_type
  ↵ (unweighted)."
knitr::asis_output(paste0("**MC error diagnostic model:** ", mc_model_desc, "\n"))

MC error diagnostic model: Diagnostic model: imv_proc ~ has_abg + age_at_encounter + curr_bmi + sex + encounter_type (unweighted).

gt::gt(mc_tab) |>
  gt::tab_header(title = "Monte Carlo error vs SE (diagnostic only)") |>
  gt::cols_label(
    term      = "Term",
    estimate  = "Estimate",
    std.error = "SE",
    mc_error  = "MC error",
    mc_err_over_se = "MC error / SE",
    conf.low   = "2.5%",
    conf.high  = "97.5%"
  ) |>
  gt::fmt_number(columns = c(estimate, std.error, mc_error, mc_err_over_se), decimals = 3) |>
  gt::fmt_missing(columns = gt::everything(), missing_text = "-")

```

Monte Carlo error vs SE (diagnostic only)

Term	Estimate	SE	MC error	MC error / SE	2.5%	97.5%
(Intercept)	-3.948	0.410	0.027	0.067	-4.753766048	-3.143213659
has_abg	2.133	0.123	0.002	0.013	1.892069856	2.374000259
age_at_encounter	-0.003	0.003	0.000	0.012	-0.008447498	0.002916156
curr_bmi	-0.012	0.010	0.001	0.081	-0.031285028	0.006983105
sexMale	0.241	0.099	0.001	0.011	0.047718257	0.434390921
encounter_typeInpatient	1.184	0.165	0.001	0.006	0.859343534	1.508064673

```
# QC flag for key estimands only (diagnostic, not pass/fail for all coefficients)
mc_key_terms <- intersect(c("has_abg"), mc_tab$term)
if (length(mc_key_terms)) {
  mc_key <- mc_tab |>
    dplyr::filter(term %in% mc_key_terms) |>
    dplyr::mutate(qc_flag = mc_err_over_se > MC_ERR_RATIO_THRESH)
  render_table_pdf_maybe(
    mc_key,
    caption = paste0("MC error QC (key terms only; threshold = ", MC_ERR_RATIO_THRESH, ")"),
    file_stub = "mc_error_qc_key_terms",
    digits = 3,
    show = SHOW_LOW_VALUE_TABLES
  )
  if (any(mc_key$qc_flag, na.rm = TRUE)) {
    knitr::asis_output(
      paste0(
        "**QC note:** MC error/SE exceeds ",
        MC_ERR_RATIO_THRESH,
        " for at least one key term; consider larger m or report as a limitation."
      )
    )
  }
}
}

# Purpose: mi diagnostics.
```

```

stopifnot(exists("imp"))
imp_n <- imp$m
get_imp <- function(i, imp_obj = imp) {
  normalize_types(mice::complete(imp_obj, action = i), levels_ref)
}

# Guard against memory blow-up from complete("long")
N <- nrow(imp$data)
M <- imp$m
long_rows <- (M + 1L) * N
LONG_ROWS_MAX <- 2e6
imp_size <- as.numeric(utils::object.size(imp))
if (long_rows > LONG_ROWS_MAX || imp_size > 1e9) {
  message("Skipping mice::densityplot/stripplot (mids -> complete('long')) due to size; using memory-safe
  diagnostics.")
}

get_obs_imp_vectors <- function(imp_obj, var) {
  obs <- imp_obj$data[[var]]
  obs <- obs[!is.na(obs)]
  impv <- imp_obj$imp[[var]]
  stopifnot(!is.null(impv))
  imp_vals <- unlist(impv, use.names = FALSE)
  if (is.numeric(obs)) obs <- obs[is.finite(obs)]
  if (is.numeric(imp_vals)) imp_vals <- imp_vals[is.finite(imp_vals)]
  list(obs = obs, imp = imp_vals)
}

plot_obs_imp <- function(imp_obj, var, fig_dir, n_obs = 50000, n_imp = 50000) {
  vecs <- get_obs_imp_vectors(imp_obj, var)
  obs <- vecs$obs
  imp_vals <- vecs$imp
  stopifnot(length(obs) + length(imp_vals) > 0)

  is_cat <- is.factor(obs) || is.character(obs) ||
  (is.numeric(obs) && length(unique(obs[!is.na(obs)])) <= 5)
}

```

```

if (!is_cat) {
  set.seed(MI_SEED)
  if (length(obs) > n_obs) obs <- sample(obs, n_obs)
  if (length(imp_vals) > n_imp) imp_vals <- sample(imp_vals, n_imp)

  df <- dplyr::tibble(
    value = c(obs, imp_vals),
    status = c(rep("Observed", length(obs)), rep("Imputed", length(imp_vals)))
  )
  p <- ggplot2::ggplot(df, ggplot2::aes(x = value, color = status, fill = status)) +
    ggplot2::geom_density(alpha = 0.2, na.rm = TRUE) +
    ggplot2::theme_minimal(base_size = 10) +
    ggplot2::labs(title = paste("Observed vs imputed:", var), x = NULL, y = "Density")
  out_file <- fs::path(fig_dir, paste0("diag-mi-density-", .make_safe_name(var), ".png"))
  ggplot2::ggsave(out_file, p, width = 7, height = 5, dpi = 200)

  stat_names <- c("mean", "sd", "p10", "p50", "p90", "n")
  obs_stats <- c(
    mean = mean(obs, na.rm = TRUE),
    sd = stats::sd(obs, na.rm = TRUE),
    p10 = stats::quantile(obs, 0.10, na.rm = TRUE),
    p50 = stats::quantile(obs, 0.50, na.rm = TRUE),
    p90 = stats::quantile(obs, 0.90, na.rm = TRUE),
    n = length(obs)
  )
  imp_stats <- c(
    mean = if (length(imp_vals)) mean(imp_vals, na.rm = TRUE) else NA_real_,
    sd = if (length(imp_vals)) stats::sd(imp_vals, na.rm = TRUE) else NA_real_,
    p10 = if (length(imp_vals)) stats::quantile(imp_vals, 0.10, na.rm = TRUE) else NA_real_,
    p50 = if (length(imp_vals)) stats::quantile(imp_vals, 0.50, na.rm = TRUE) else NA_real_,
    p90 = if (length(imp_vals)) stats::quantile(imp_vals, 0.90, na.rm = TRUE) else NA_real_,
    n = length(imp_vals)
  )
  return(dplyr::tibble(
    variable = var,
    type = "numeric",
    stat = stat_names,
  ))
}

```

```

    observed = uname(obs_stats),
    imputed = uname(imp_stats)
  )))
}

obs_chr <- as.character(obs)
imp_chr <- as.character(imp_vals)
levels_all <- sort(unique(c(obs_chr, imp_chr)))
prop_obs <- if (length(obs_chr)) {
  as.numeric(table(factor(obs_chr, levels = levels_all))) / length(obs_chr)
} else {
  rep(0, length(levels_all))
}
prop_imp <- if (length(imp_chr)) {
  as.numeric(table(factor(imp_chr, levels = levels_all))) / length(imp_chr)
} else {
  rep(0, length(levels_all))
}
prop_df <- dplyr::tibble(
  level = levels_all,
  observed = prop_obs,
  imputed = prop_imp
)
miss_plot_df <- tidyr::pivot_longer(prop_df, c(observed, imputed),
                                      names_to = "status", values_to = "prop")
p <- ggplot2::ggplot(miss_plot_df, ggplot2::aes(x = level, y = prop, fill = status)) +
  ggplot2::geom_col(position = "dodge", width = 0.7) +
  ggplot2::theme_minimal(base_size = 10) +
  ggplot2::theme(axis.text.x = ggplot2::element_text(angle = 20, hjust = 1)) +
  ggplot2::labs(title = paste("Observed vs imputed:", var), x = NULL, y = "Proportion")
out_file <- fs::path(fig_dir, paste0("diag-mi-bar-", .make_safe_name(var), ".png"))
ggplot2::ggsave(out_file, p, width = 7, height = 5, dpi = 200)

return(prop_df |>
  tidyr::pivot_longer(c(observed, imputed), names_to = "status", values_to = "prop") |>
  dplyr::mutate(variable = var, type = "categorical", stat = "prop"))
}

```

```

# Throttle diagnostics to avoid memory blow-up
trace_vars <- intersect(c("curr_bmi", "serum_hco3", "hr"), names(imp$imp))
if (length(trace_vars)) {
  trace_list <- lapply(trace_vars, function(v) {
    imp_mat <- imp$imp[[v]]
    stopifnot(!is.null(imp_mat))
    means <- colMeans(imp_mat, na.rm = TRUE)
    dplyr::tibble(variable = v, imputation = seq_along(means), mean_imputed = means)
  })
  trace_df <- dplyr::bind_rows(trace_list)
  if (nrow(trace_df)) {
    p_trace <- ggplot2::ggplot(trace_df, ggplot2::aes(x = imputation, y = mean_imputed)) +
      ggplot2::geom_line() +
      ggplot2::geom_point(size = 0.6) +
      ggplot2::facet_wrap(~variable, scales = "free_y") +
      ggplot2::theme_minimal(base_size = 10) +
      ggplot2::labs(title = "Mean imputed value by imputation", x = "Imputation", y = "Mean")
    print(p_trace)
  }
}

vars_show <- trace_vars[seq_len(min(2, length(trace_vars)))]
dens_list <- lapply(vars_show, function(v) {
  obs <- imp$data[[v]]
  imp1 <- get_imp(1)[[v]]
  obs_vals <- obs[!is.na(obs)]
  imp_vals <- imp1[is.na(obs)]
  if (length(obs_vals) > 5000) obs_vals <- sample(obs_vals, 5000)
  if (length(imp_vals) > 5000) imp_vals <- sample(imp_vals, 5000)
  dplyr::tibble(
    variable = v,
    value = c(obs_vals, imp_vals),
    status = c(rep("Observed", length(obs_vals)),
              rep("Imputed (imp1)", length(imp_vals)))
  )
})
dens_df <- dplyr::bind_rows(dens_list)

```

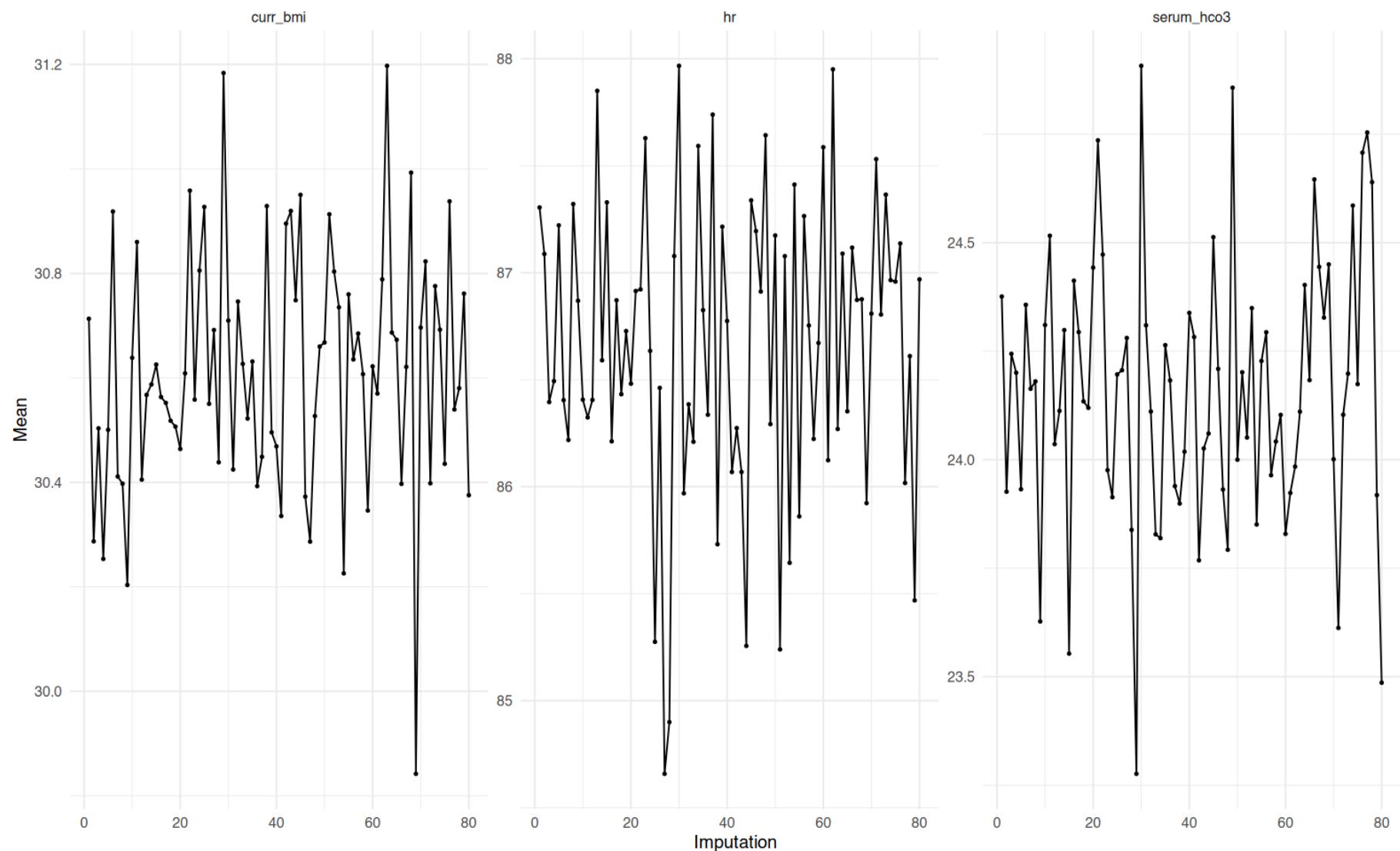
```

p_dens <- ggplot2::ggplot(dens_df, ggplot2::aes(x = value, color = status, fill = status)) +
  ggplot2::geom_density(alpha = 0.2, na.rm = TRUE) +
  ggplot2::facet_wrap(~variable, scales = "free") +
  ggplot2::theme_minimal(base_size = 10) +
  ggplot2::labs(title = "Observed vs imputed distributions (imp1)", x = NULL, y = "Density")
print(p_dens)

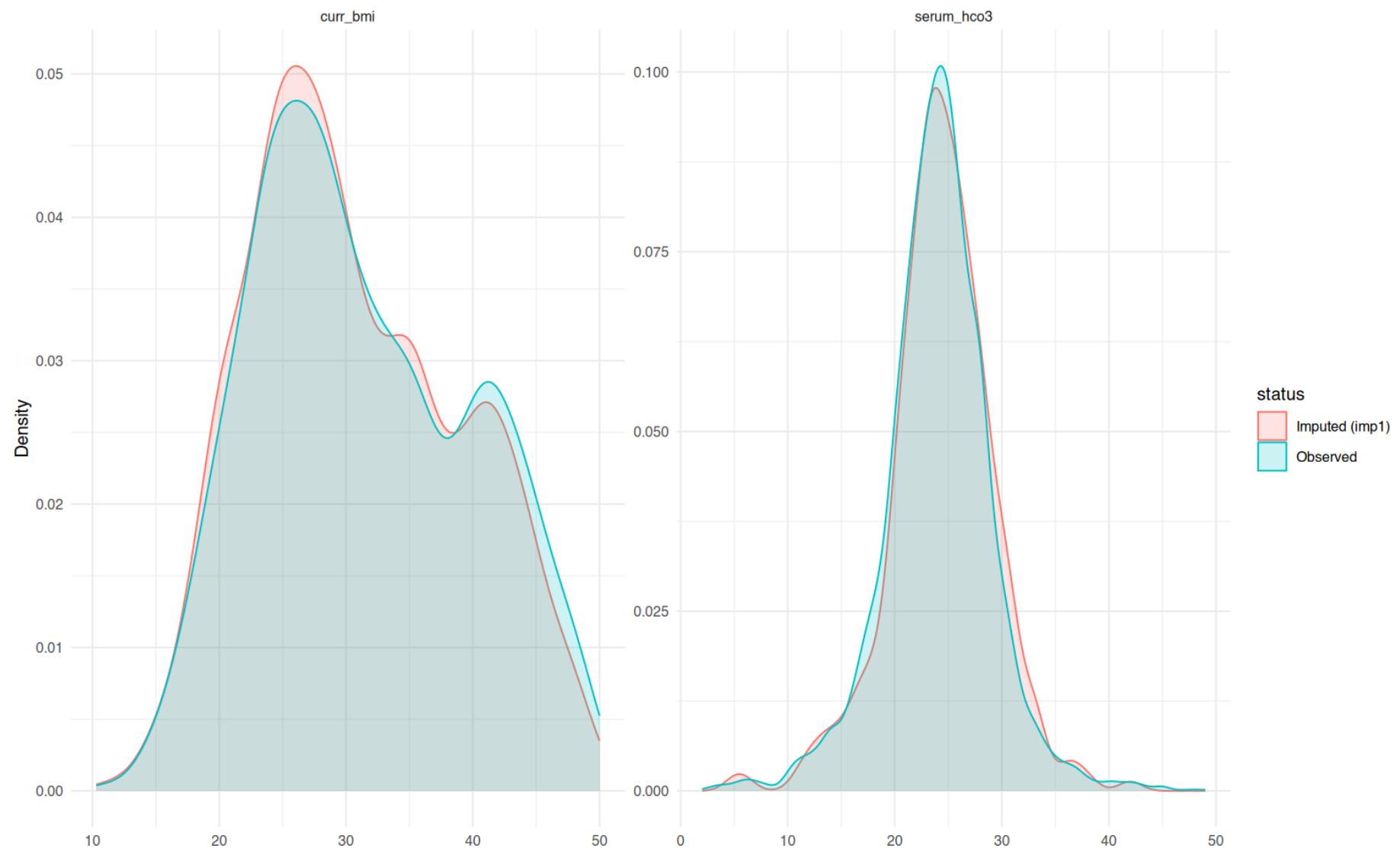
p_box <- ggplot2::ggplot(dens_df, ggplot2::aes(x = status, y = value, fill = status)) +
  ggplot2::geom_boxplot(outlier.size = 0.5, na.rm = TRUE) +
  ggplot2::facet_wrap(~variable, scales = "free") +
  ggplot2::theme_minimal(base_size = 10) +
  ggplot2::theme(axis.text.x = ggplot2::element_text(angle = 20, hjust = 1)) +
  ggplot2::labs(title = "Observed vs imputed (imp1)", x = NULL, y = NULL)
print(p_box)
}

```

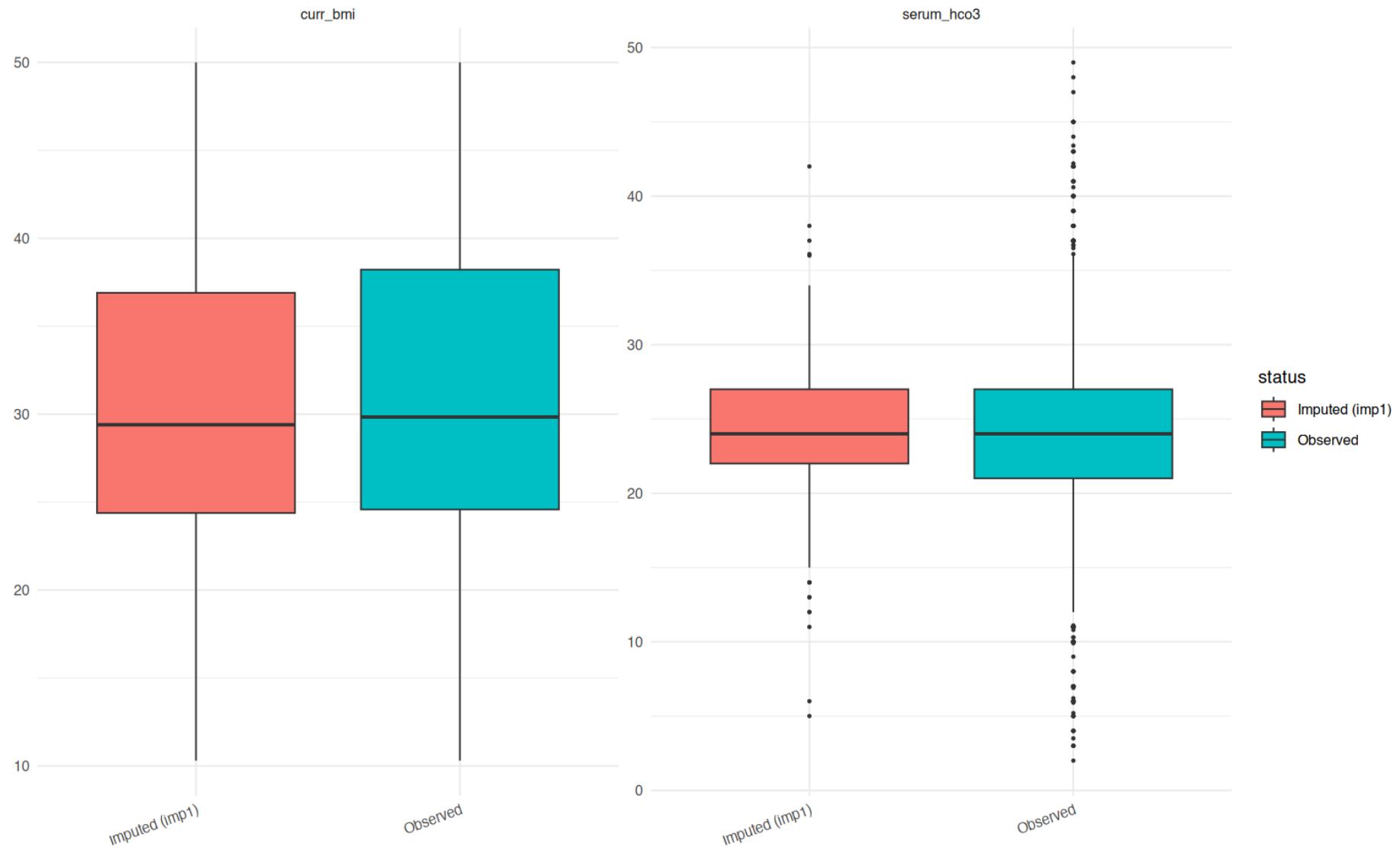
Mean imputed value by imputation



Observed vs imputed distributions (imp1)



Observed vs imputed (imp1)



```
# Observed vs imputed summaries (memory-safe)
diag_vars <- intersect(
  c("age_at_encounter", "curr_bmi", "temp_new", "sbp", "dbp", "hr",
    "sodium", "serum_cr", "serum_hco3", "serum_lac", "wbc", "plt",
```

```

"sex", "race_ethnicity", "location", "encounter_type",
"copd", "asthma", "chf", "dm"),
names(imp$data)
)
diag_vars <- intersect(diag_vars, names(imp$imp))
if (length(diag_vars)) {
  miss_counts <- colSums(is.na(imp$data[diag_vars]))
  diag_vars <- diag_vars[miss_counts > 0]
}

mi_obs_imp_summary <- dplyr::bind_rows(lapply(diag_vars, function(v) {
  out <- plot_obs_imp(imp, v, results_path("figs"))
  invisible(gc())
  out
}))

mi_obs_imp_summary_file <- results_path("mi_obs_vs_imp_summary.csv")
if (nrow(mi_obs_imp_summary)) {
  write_csv_safely(mi_obs_imp_summary, mi_obs_imp_summary_file, row_names = FALSE)
  render_table_pdf_maybe(
    mi_obs_imp_summary,
    caption = "Observed vs imputed summaries",
    file_stub = "mi_obs_vs_imp_summary",
    digits = 3,
    show = SHOW_LOW_VALUE_TABLES
  )
} else {
  mi_obs_imp_stub <- data.frame(
    variable = character(),
    n_obs = numeric(),
    mean_obs = numeric(),
    sd_obs = numeric(),
    mean_imp = numeric(),
    sd_imp = numeric(),
    stringsAsFactors = FALSE
  )
  write_csv_safely(mi_obs_imp_stub, mi_obs_imp_summary_file, row_names = FALSE)
  message("No variables with missingness available for observed vs imputed summaries.")
}

```

```

}

# Purpose: mi missing pattern.
md_pat <- NULL
invisible(utils::capture.output(
  md_pat <- mice::md.pattern(imp$data, plot = FALSE)
))

md_pat_file <- results_path("missingness-pattern.csv")
write_csv_safely(md_pat, md_pat_file, row_names = TRUE)

md_fig_file <- results_path("figs", "missingness-pattern.png")
grDevices::png(md_fig_file, width = 1800, height = 1200, res = 200)
mice::md.pattern(imp$data, plot = TRUE)
grDevices::dev.off()

# Purpose: mi imputation diagnostics.
stopifnot(exists("imp"))
imp_n <- imp$m
get_imp <- function(i, imp_obj = imp) { normalize_types(mice::complete(imp_obj, action = i), levels_ref) }

# Choose a manageable set of incomplete variables
miss_overall <- naniar::miss_var_summary(subset_data) %>% arrange(desc(pct_miss))
vars_incomplete <- miss_overall$variable[miss_overall$n_miss > 0]
per_imp_n <- min(500, nrow(imp$data))
dat_obs <- imp$data %>%
  dplyr::slice_sample(n = min(per_imp_n, nrow(imp$data))) %>%
  dplyr::mutate(.imp = 0L, .imp_label = "Observed")
dat_imp <- purrr::map_dfr(seq_len(imp_n), function(i) {
  di <- get_imp(i)
  di <- dplyr::slice_sample(di, n = min(per_imp_n, nrow(di)))
  di$.imp <- i
  di$.imp_label <- "Imputed"
  di
})
dat_long <- dplyr::bind_rows(dat_obs, dat_imp)
vars_show <- head(intersect(vars_incomplete, names(dat_long))), 6) # limit for plotting & ensure present in imp

```

```

# Density plots by imputation status
if (length(vars_show)) {
  strata_candidates <- intersect(c("has_abg", "has_vbg", "imv_proc", "death_60d"), names(dat_long))
  printed_counts <- FALSE

  for (v in vars_show) {
    if (is.numeric(dat_long[[v]])) {
      df_plot <- dat_long |>
        dplyr::filter(is.finite(.data[[v]]))
      p <- ggplot(df_plot, aes(x = .data[[v]], fill = .imp_label)) +
        geom_density(alpha = 0.4, adjust = 1, na.rm = TRUE) +
        labs(title = paste("Observed vs imputed density:", v), x = v, fill = "Source") +
        theme_minimal()
    } else {
      df_plot <- dat_long |>
        dplyr::filter(!is.na(.data[[v]]))
      p <- ggplot(df_plot, aes(x = .data[[v]], fill = .imp_label)) +
        geom_bar(position = "fill", na.rm = TRUE) +
        labs(title = paste("Observed vs imputed proportions:", v), x = v, y = "Proportion", fill = "Source") +
        theme_minimal() +
        coord_flip()
    }
    print(p)
  }

# Box/violin plots by strata (numeric vars only)
if (length(strata_candidates)) {
  pd <- position_dodge(width = 0.8)
  for (sv in strata_candidates) {
    for (v in vars_show) {
      if (!is.numeric(dat_long[[v]])) next
      df_plot <- dat_long |>
        dplyr::filter(!is.na(.data[[sv]]), is.finite(.data[[v]])) |>
        dplyr::mutate(strata = factor(.data[[sv]]))
      if (nrow(df_plot) == 0L) next

      if (!printed_counts) {

```

```

    invisible(table(strata = df_plot$.strata, source = df_plot$.imp_label))
    printed_counts <- TRUE
  }

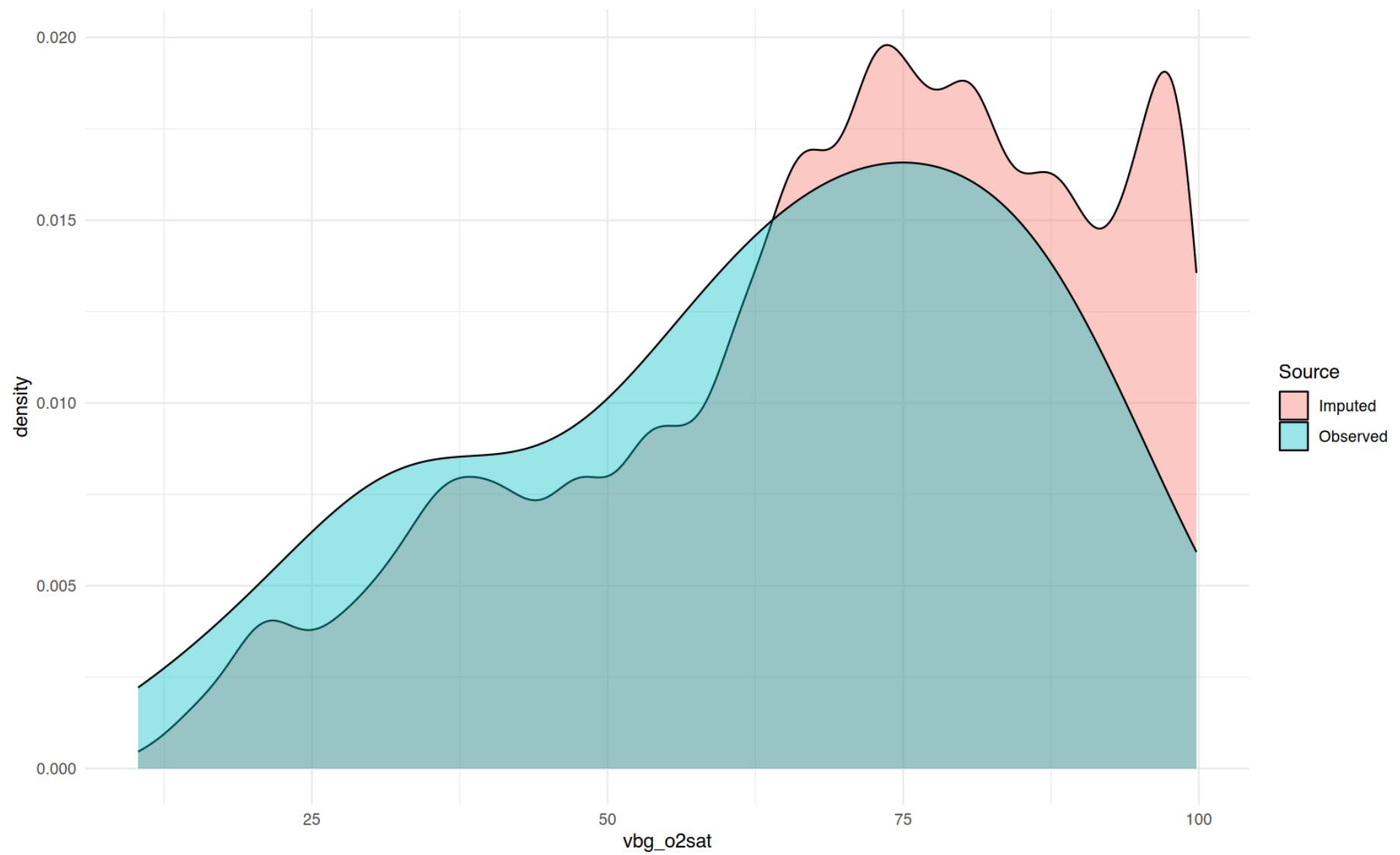
  if (all(levels(df_plot$.strata) %in% c("0", "1"))) {
    x_scale <- ggplot2::scale_x_discrete(labels = c("0" = "No", "1" = "Yes"))
  } else {
    x_scale <- ggplot2::scale_x_discrete(drop = FALSE)
  }

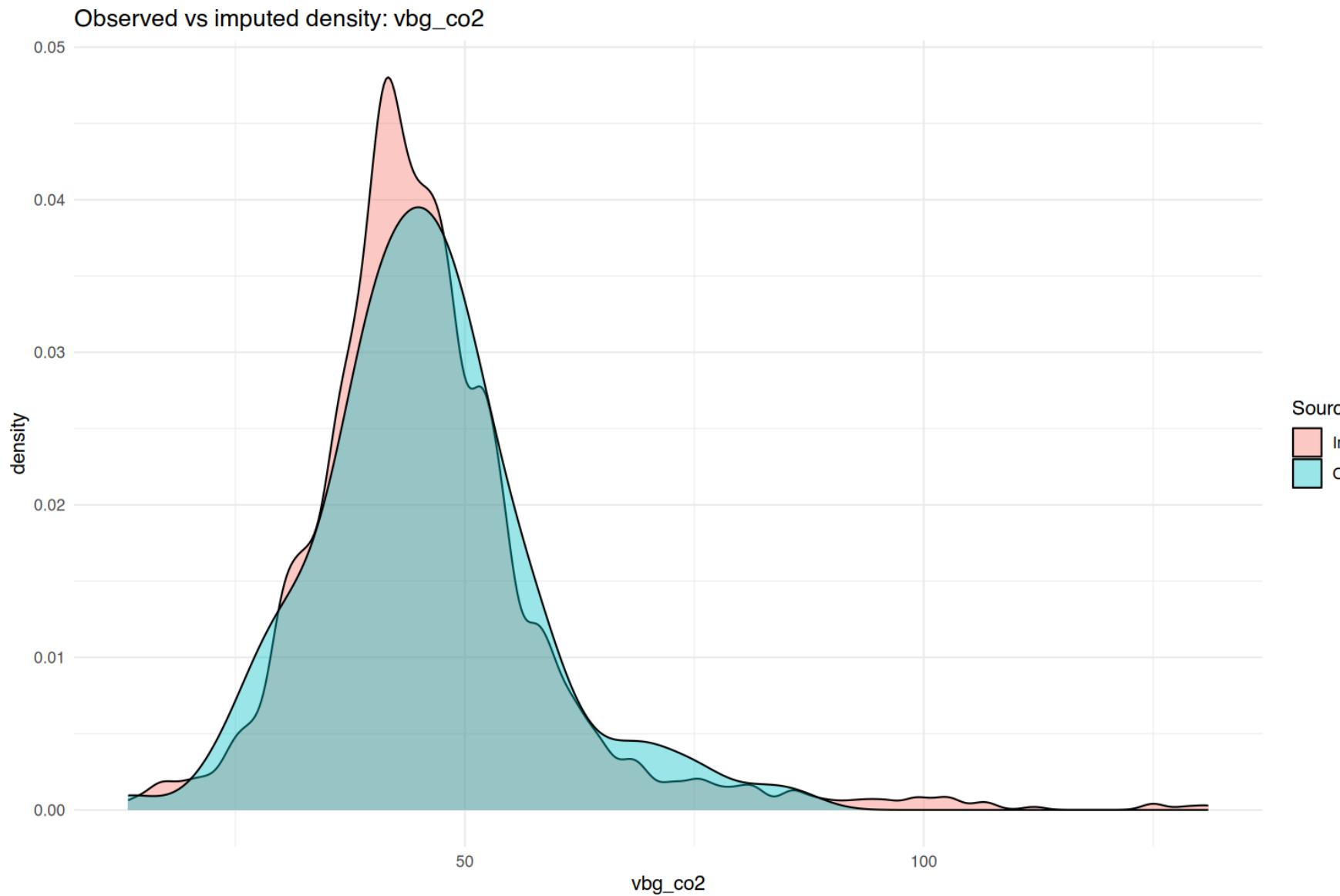
  p <- ggplot(df_plot, aes(
    x = .strata,
    y = .data[[v]],
    fill = .imp_label,
    group = interaction(.imp_label, .strata)
  )) +
    geom_violin(alpha = 0.5, scale = "width", trim = TRUE, position = pd, na.rm = TRUE) +
    geom_boxplot(width = 0.2, outlier.size = 0.6, position = pd, na.rm = TRUE) +
    x_scale +
    labs(title = paste("Observed vs imputed:", v, "by", sv),
        x = sv, y = v, fill = "Source") +
    theme_minimal()
  print(p)
  out_file <- results_path(
    "figs",
    paste0("diag-mi-obs-imp-by-", .make_safe_name(sv), "-", .make_safe_name(v), ".png")
  )
  save_diag_plot(p, out_file, width = 7, height = 5)
}

}
}
}

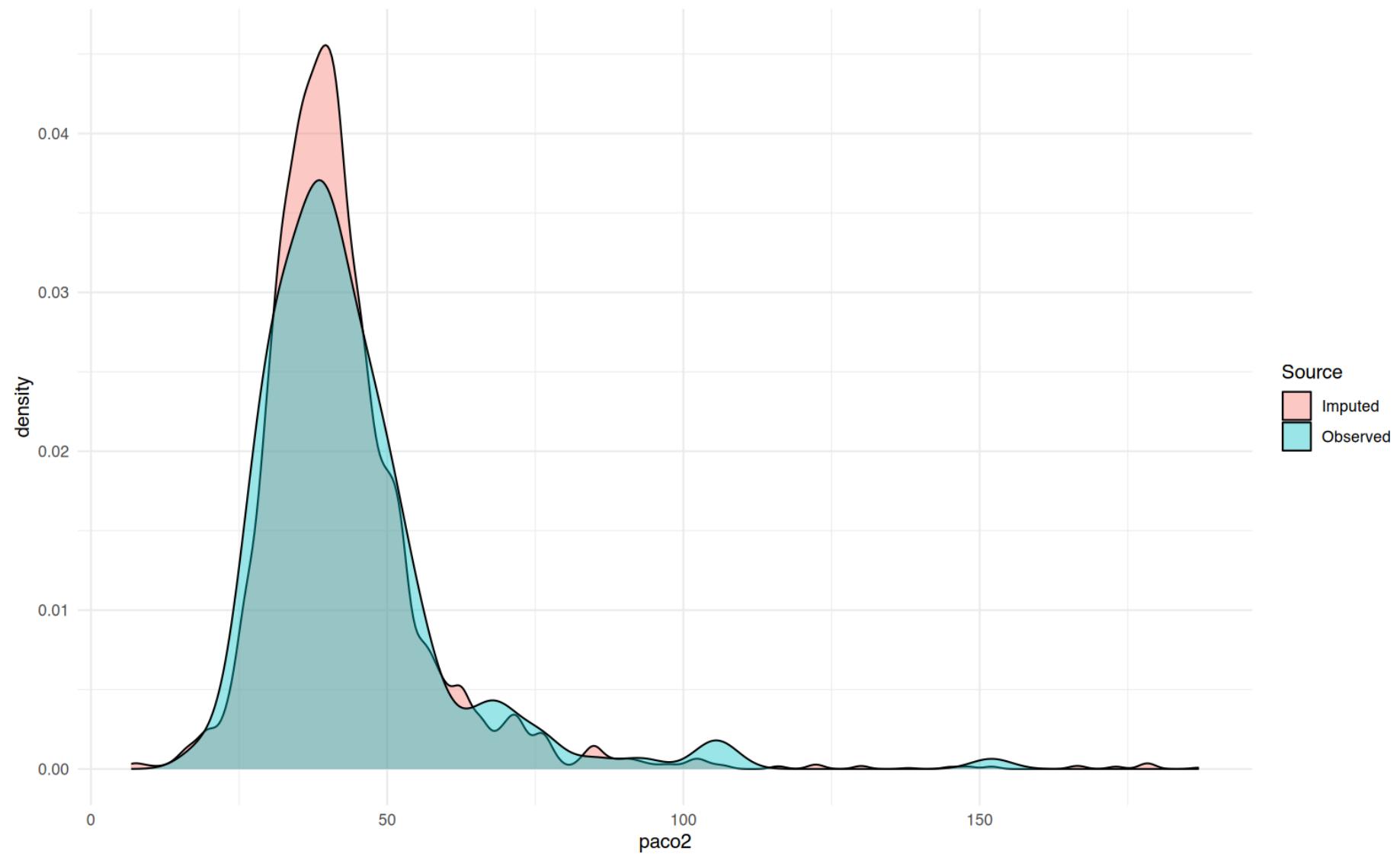
```

Observed vs imputed density: vbg_o2sat

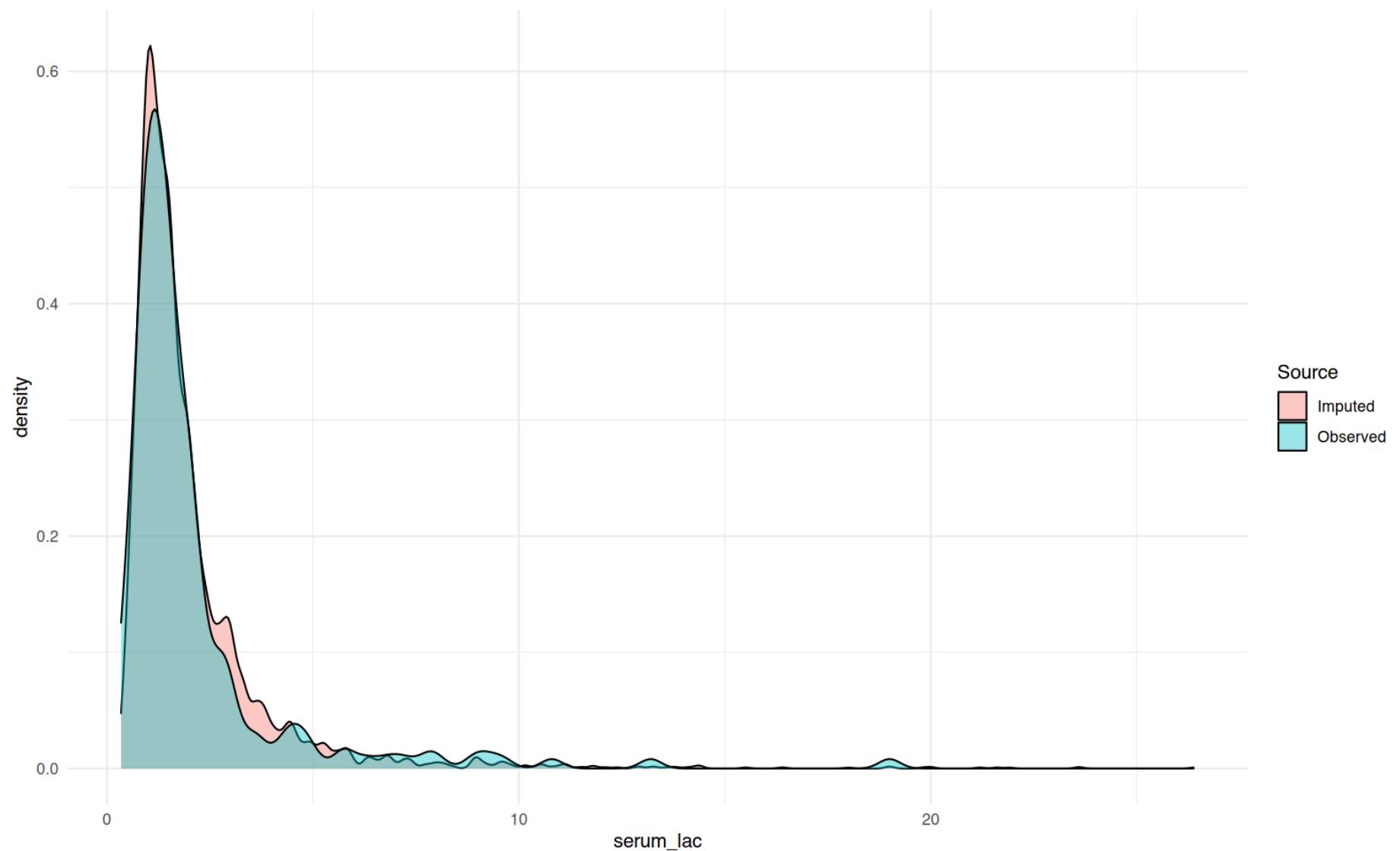




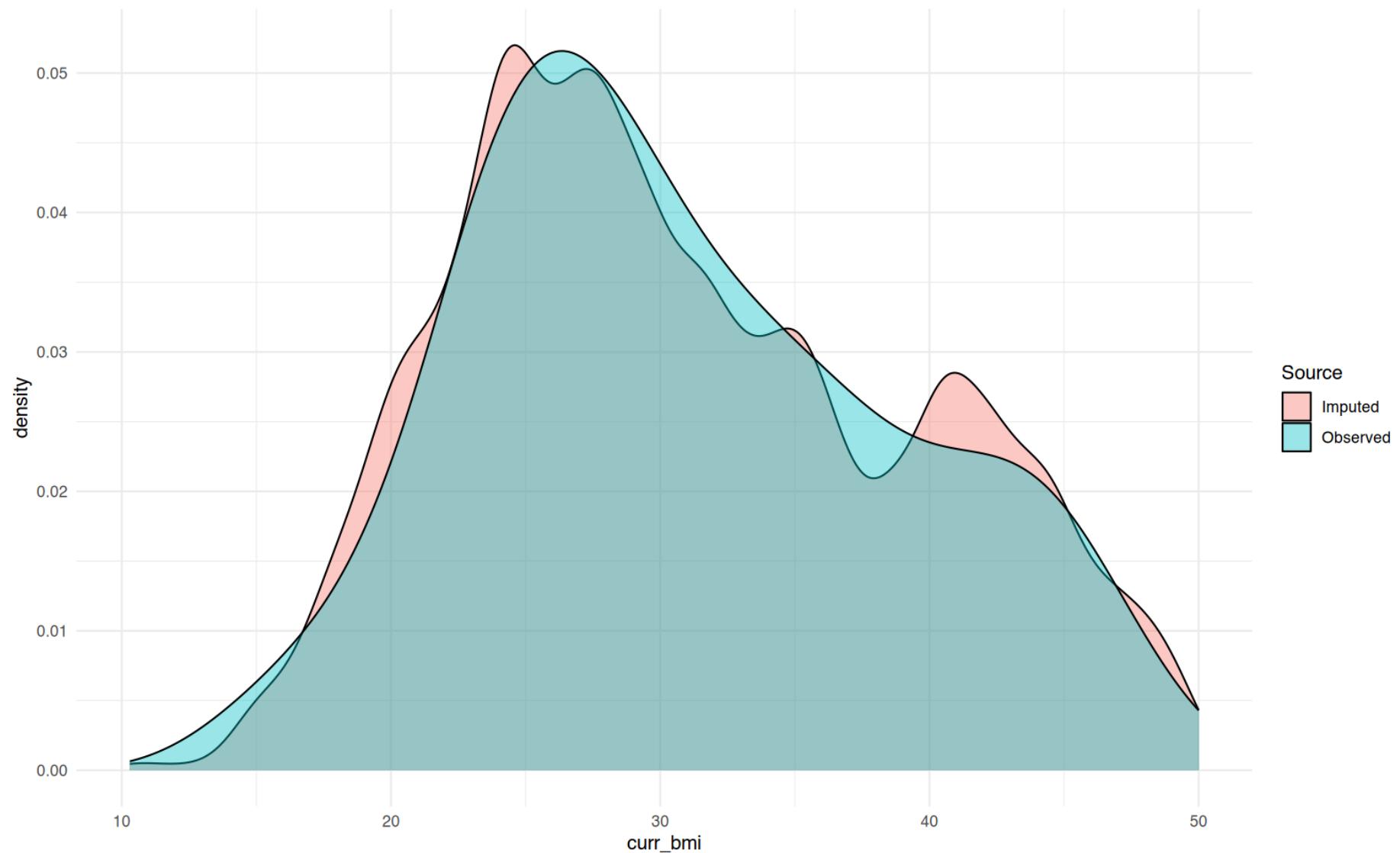
Observed vs imputed density: paco2



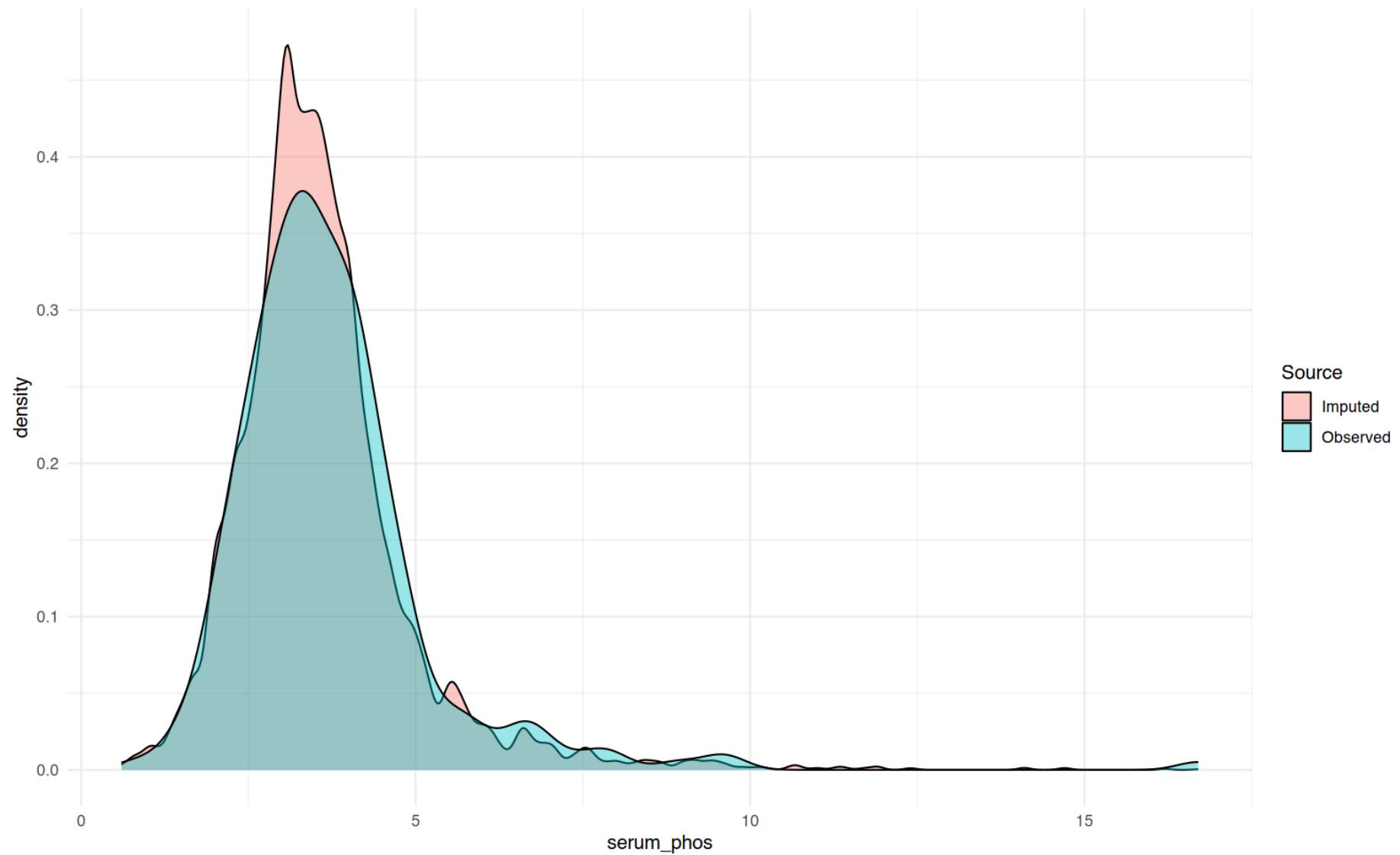
Observed vs imputed density: serum_lac



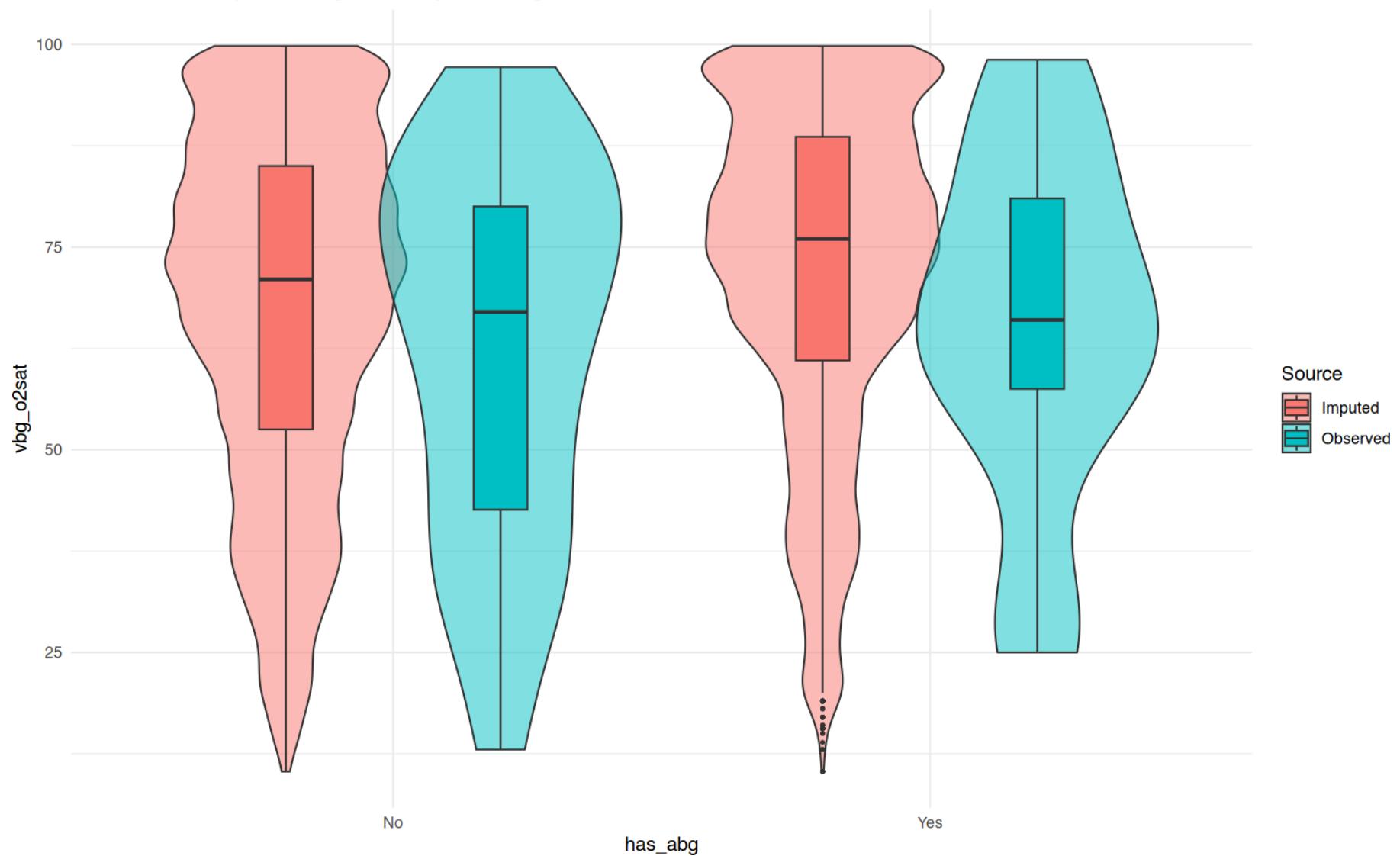
Observed vs imputed density: curr_bmi



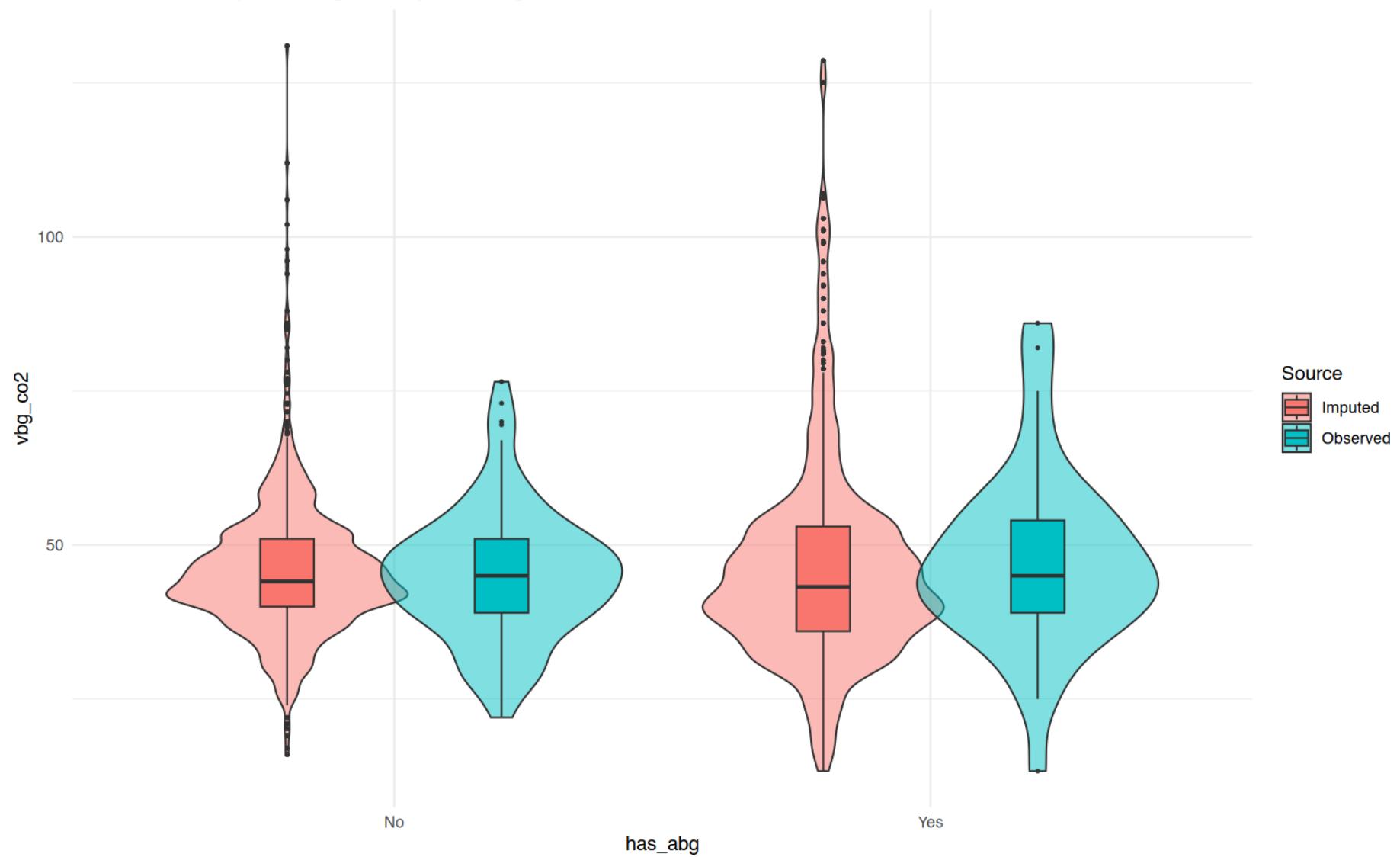
Observed vs imputed density: serum_phos



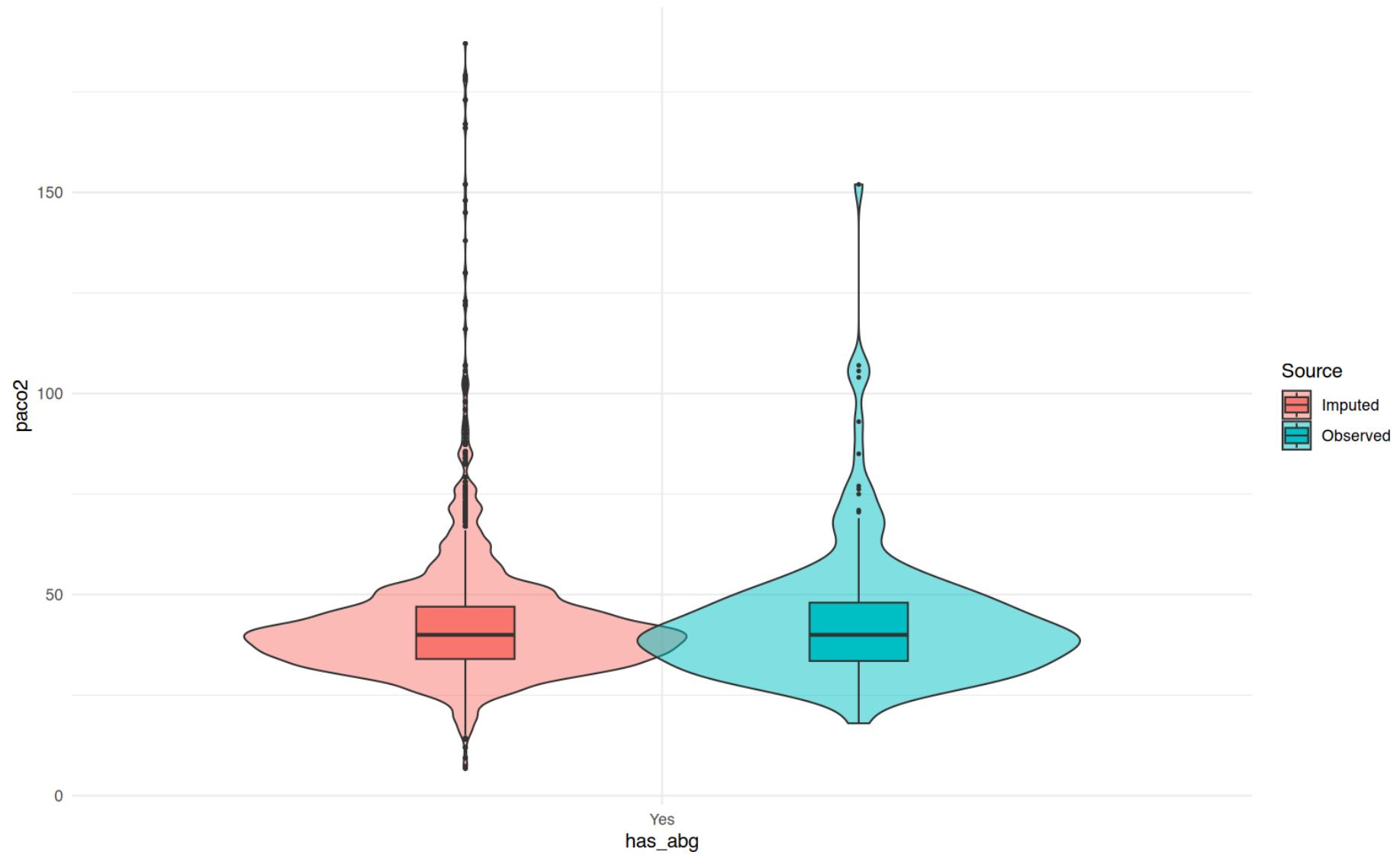
Observed vs imputed: vbg_o2sat by has_abg



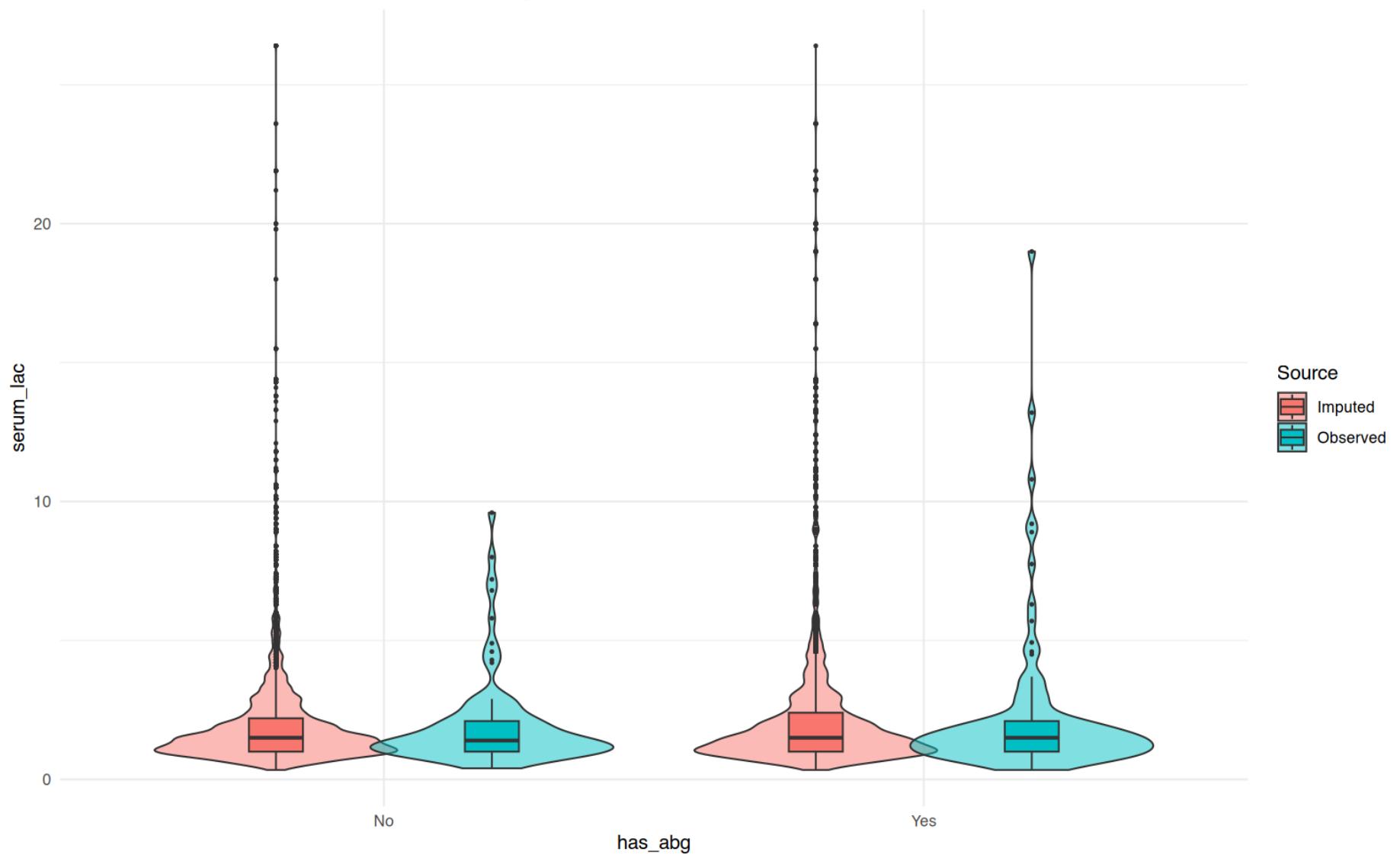
Observed vs imputed: vbg_co2 by has_abg



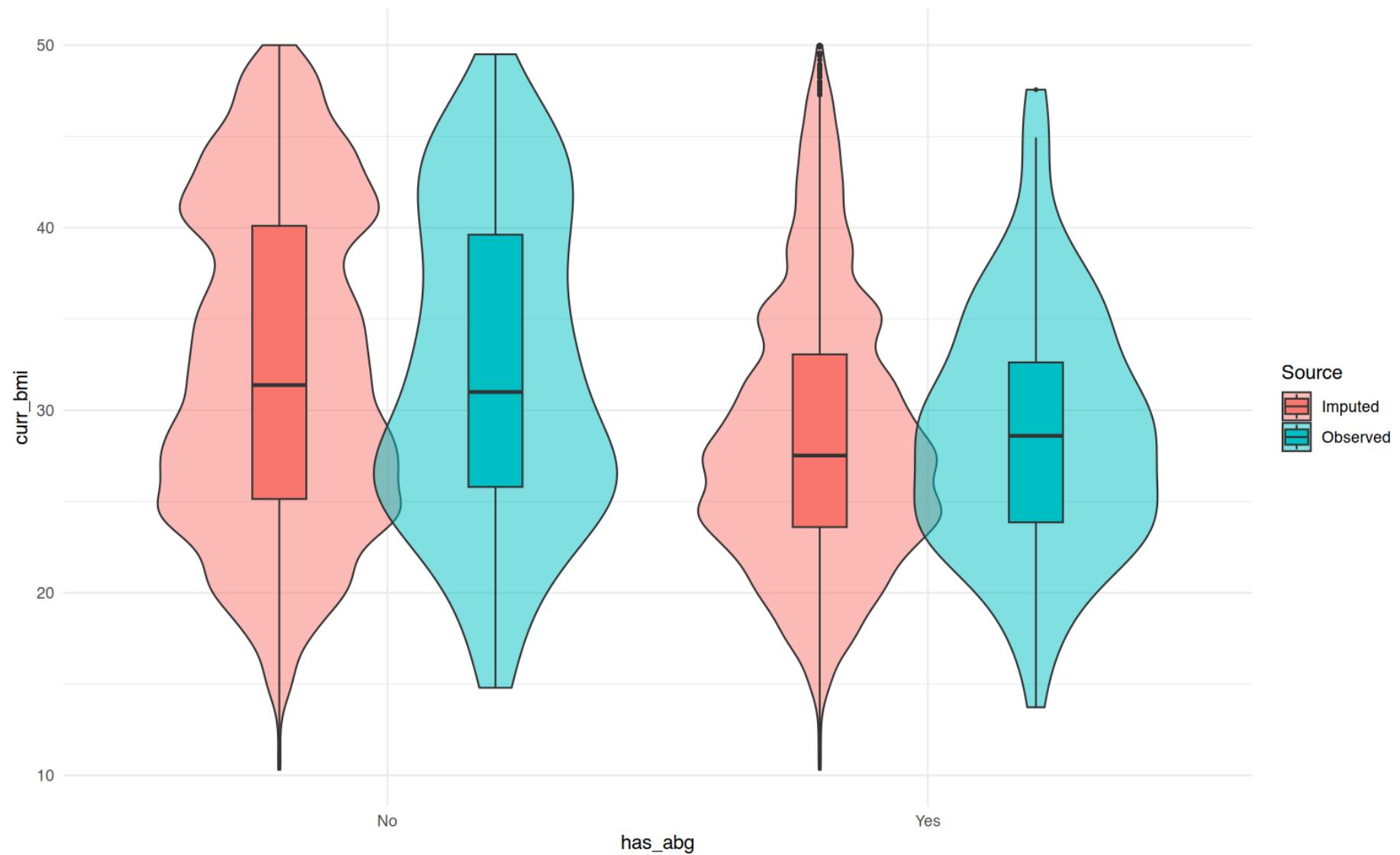
Observed vs imputed: paco2 by has_abg



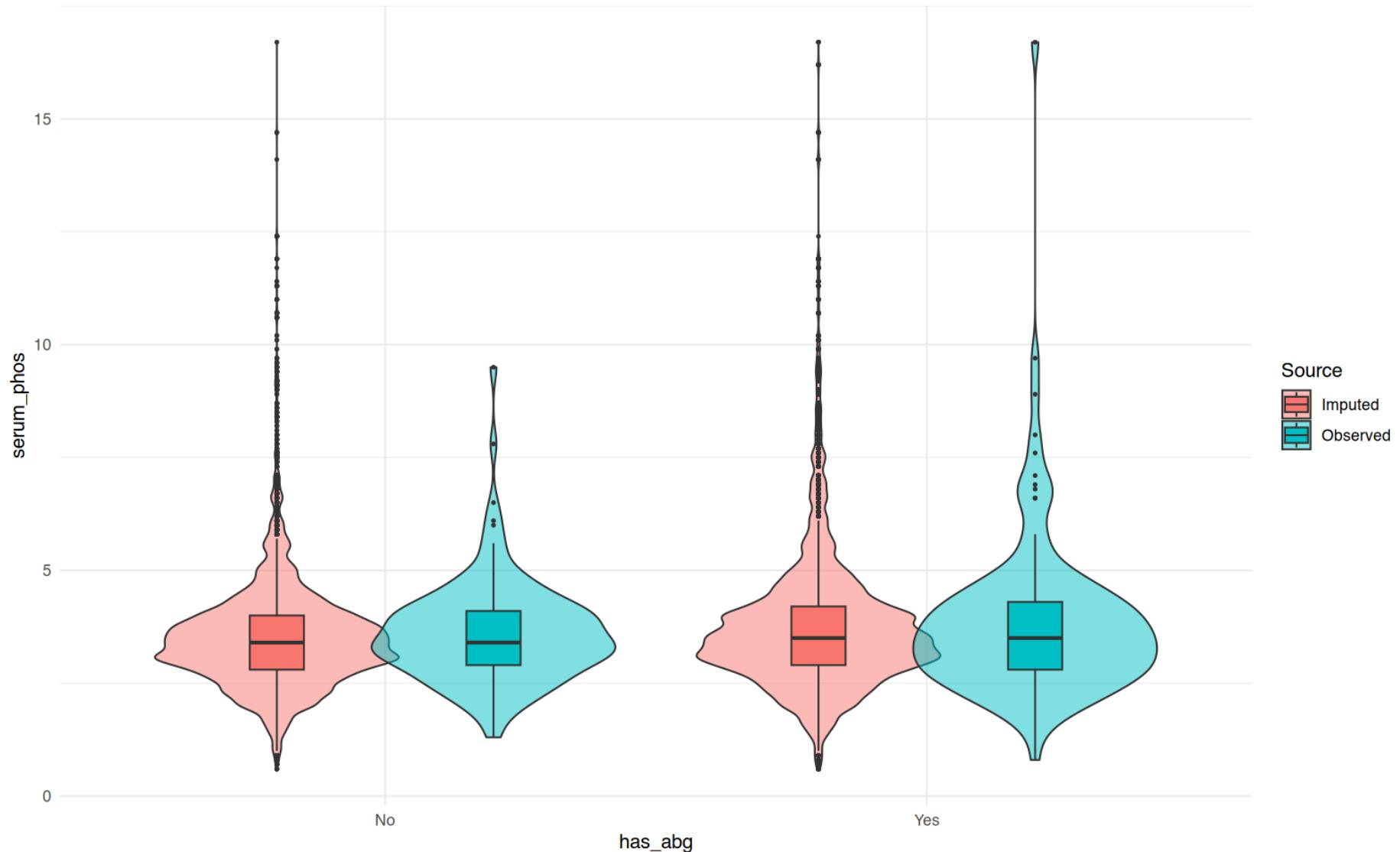
Observed vs imputed: serum_lac by has_abg



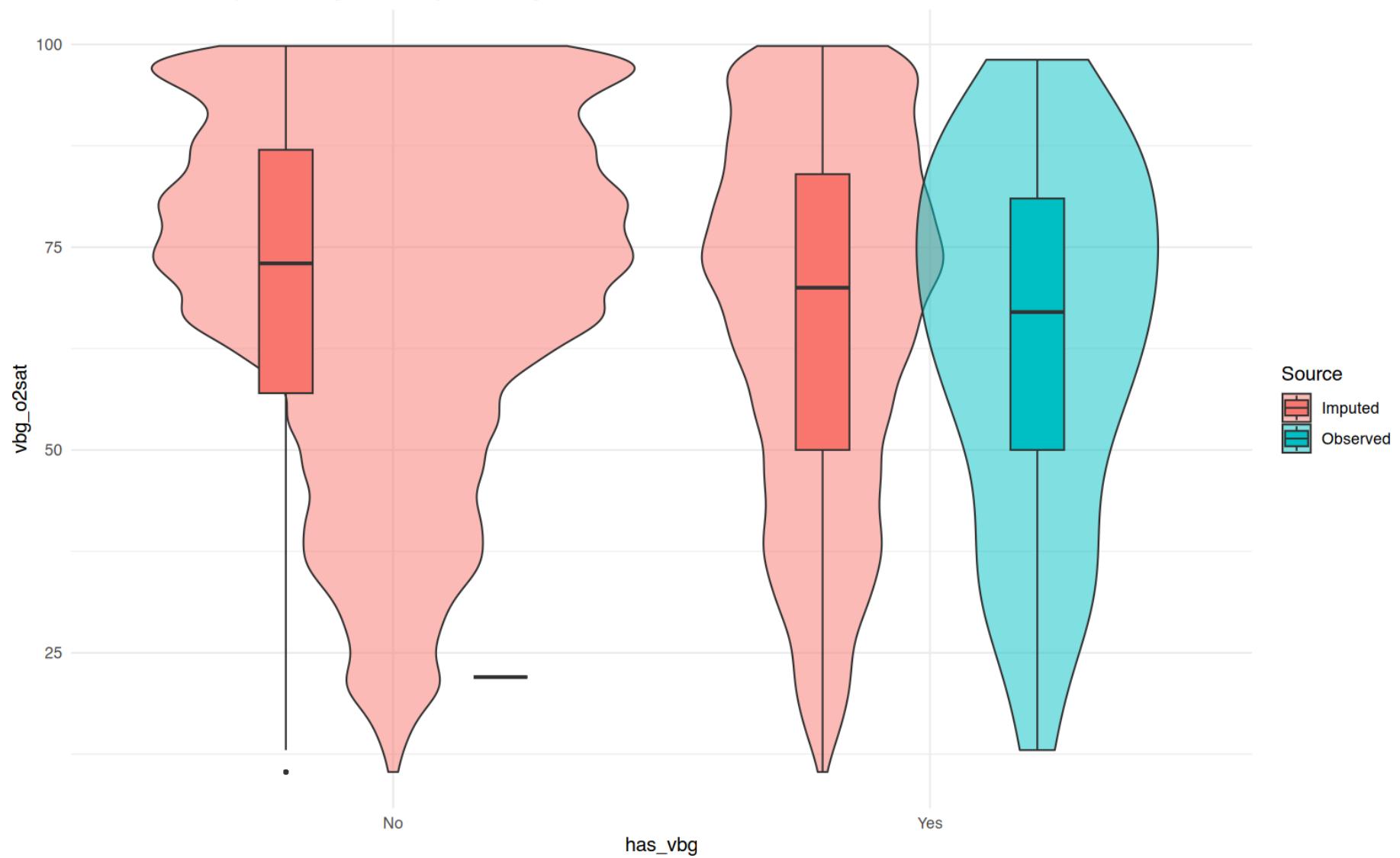
Observed vs imputed: curr_bmi by has_abg



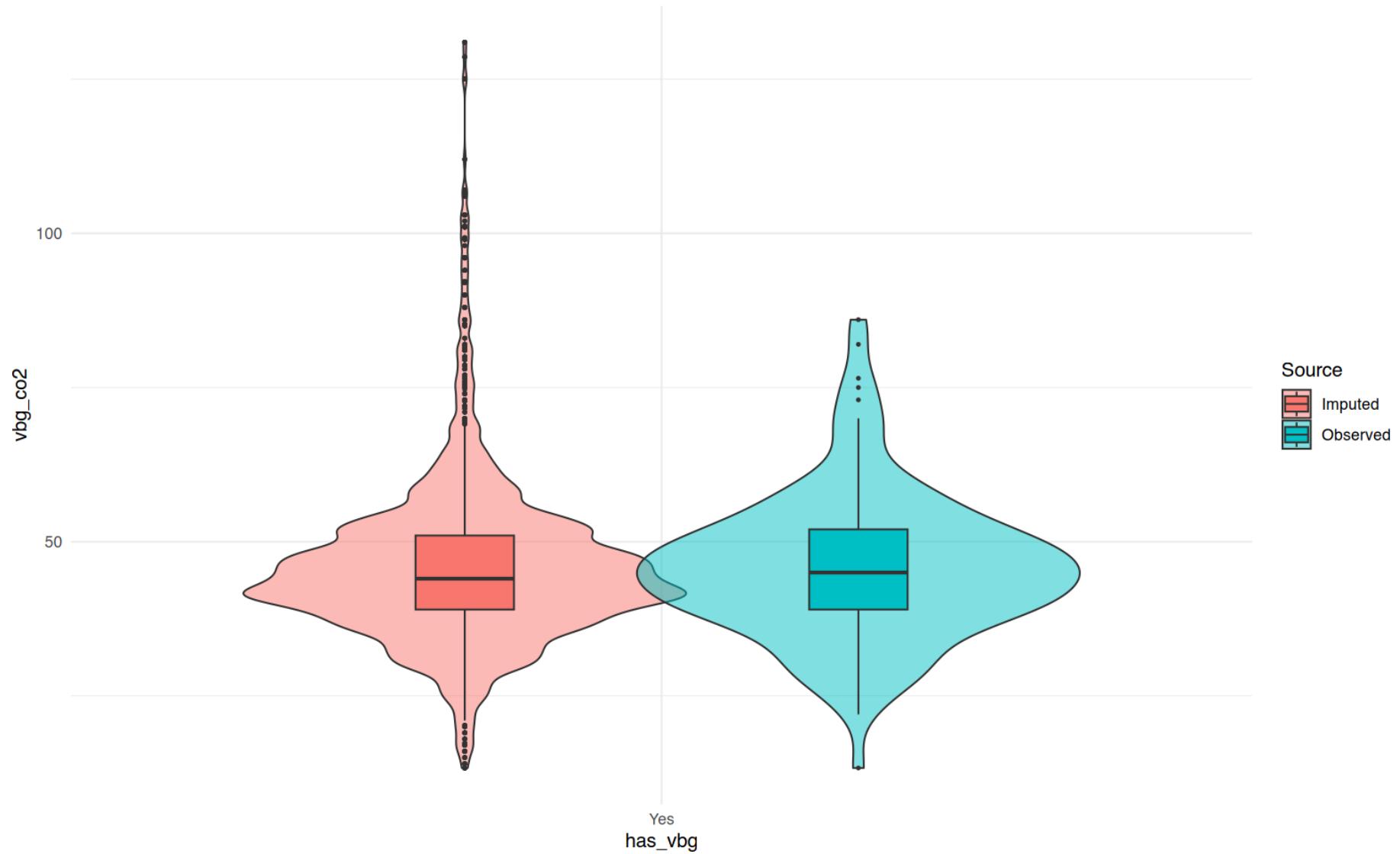
Observed vs imputed: serum_phos by has_abg



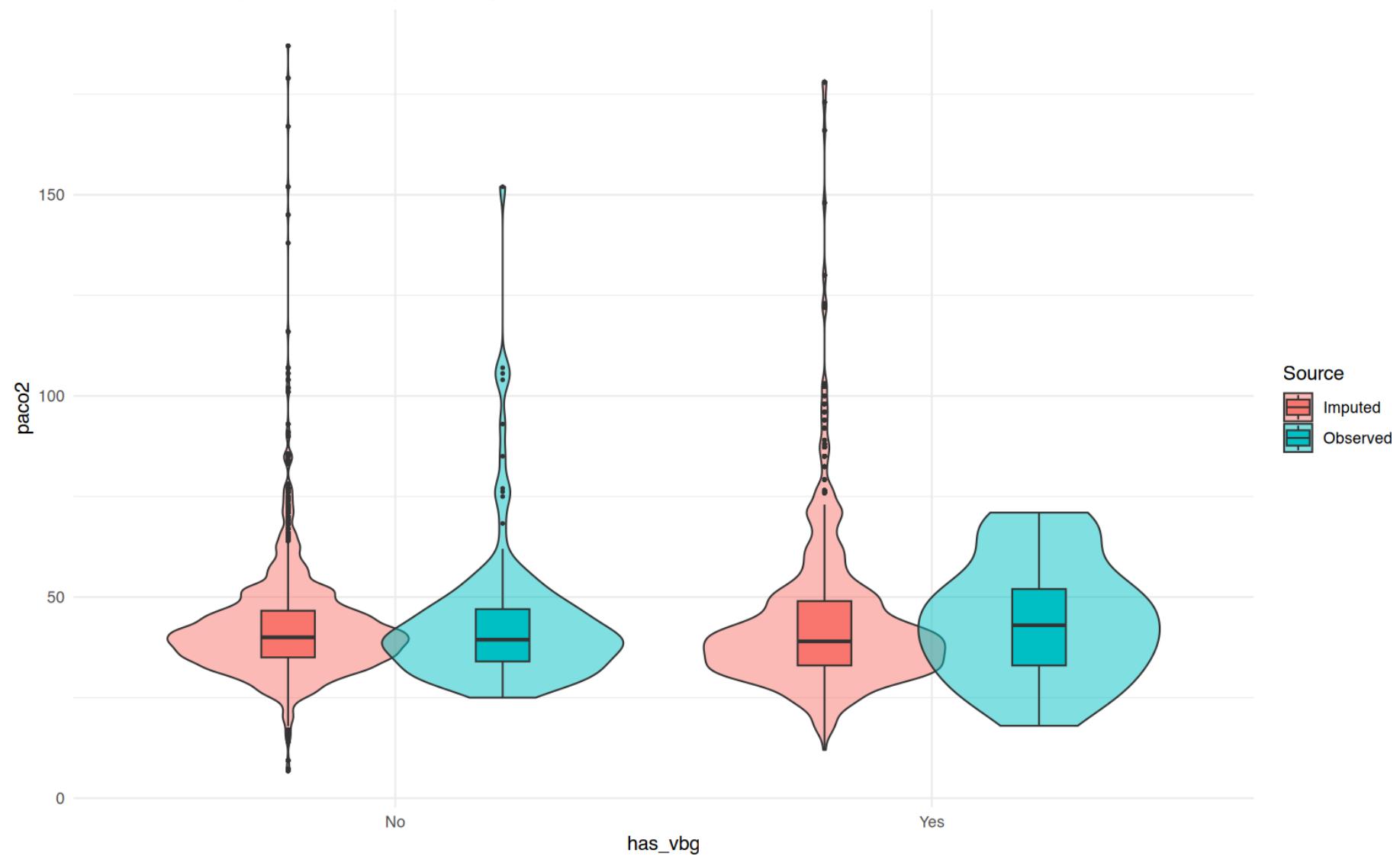
Observed vs imputed: vbg_o2sat by has_vbg



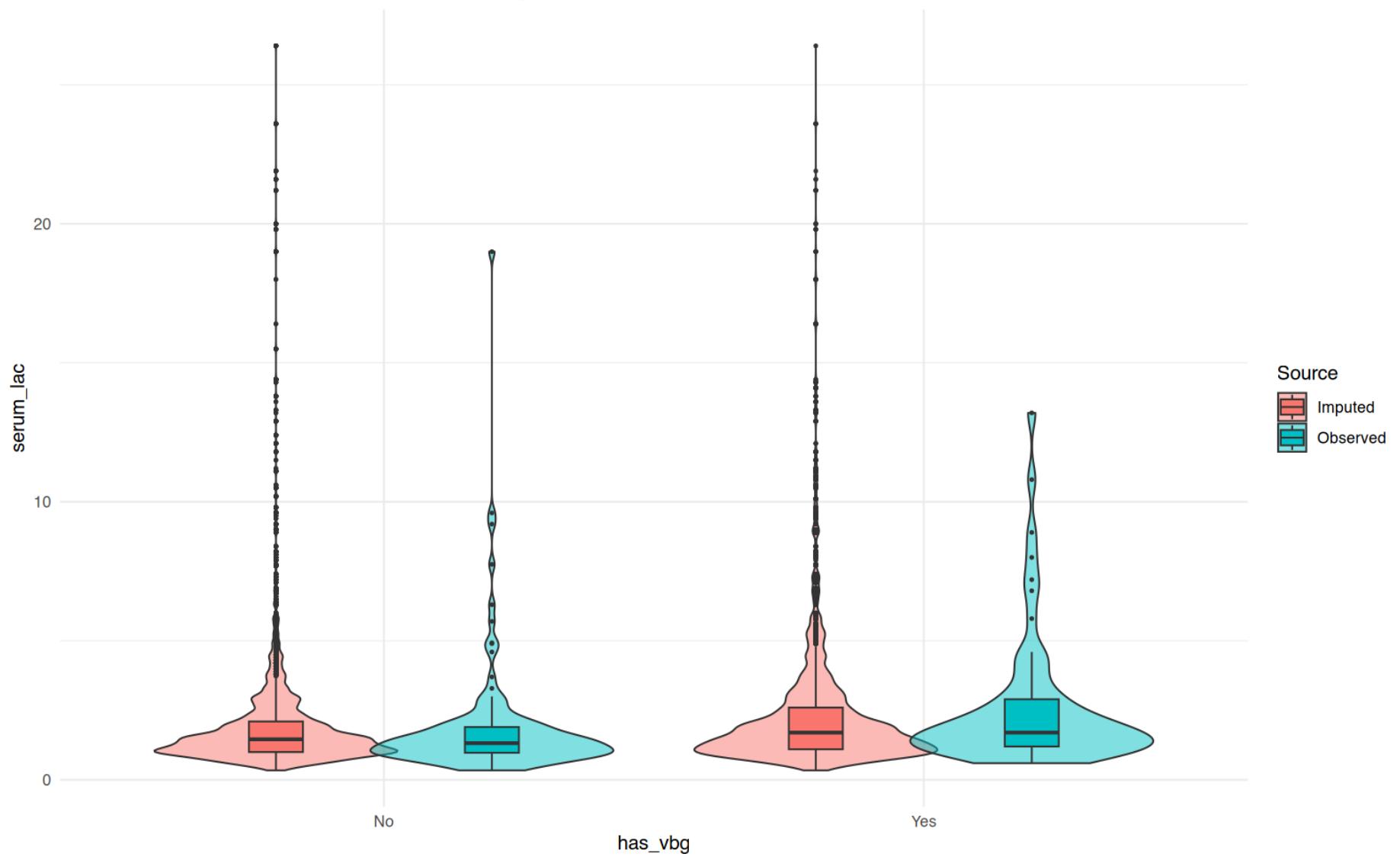
Observed vs imputed: vbg_co2 by has_vbg



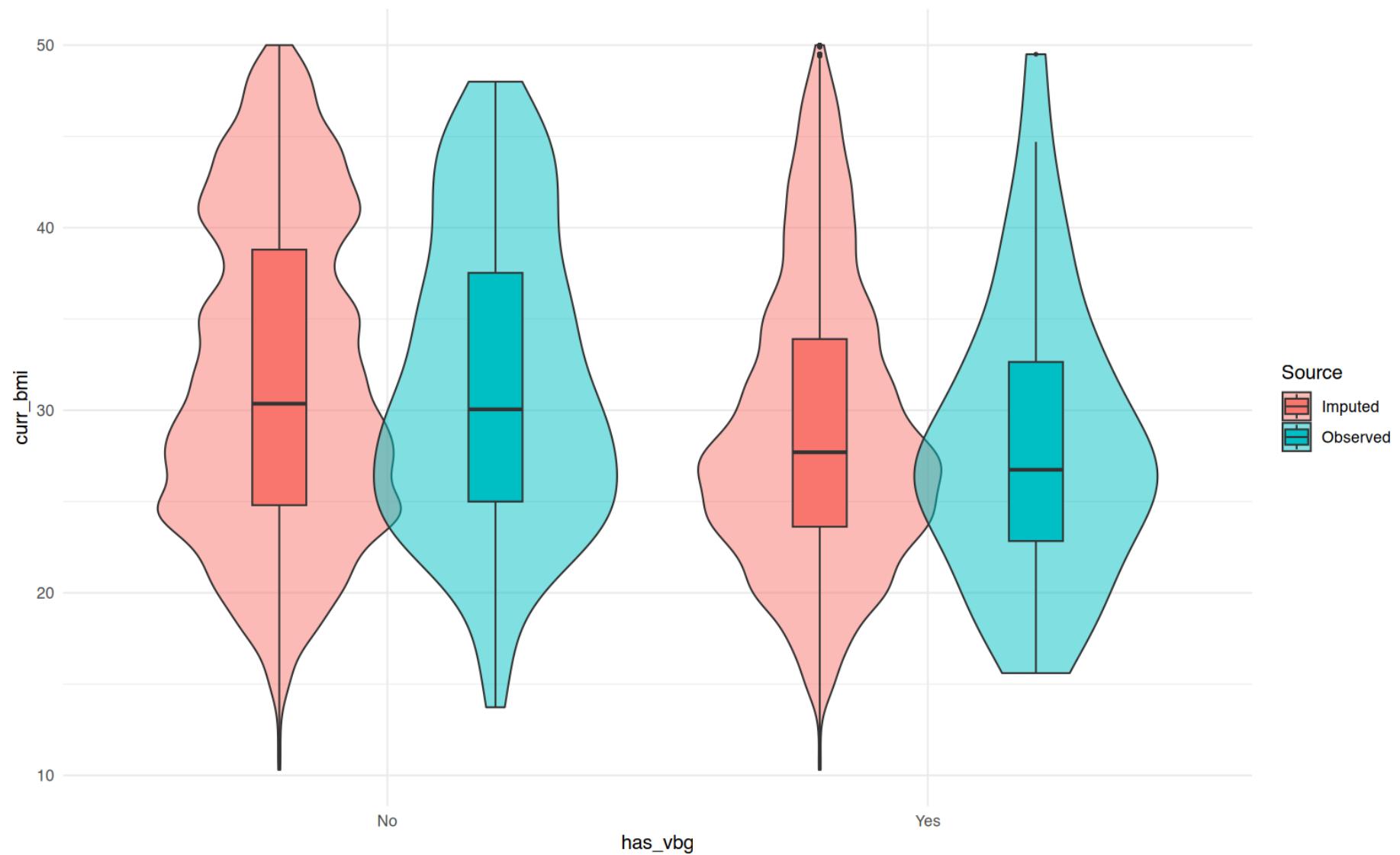
Observed vs imputed: paco2 by has_vbg



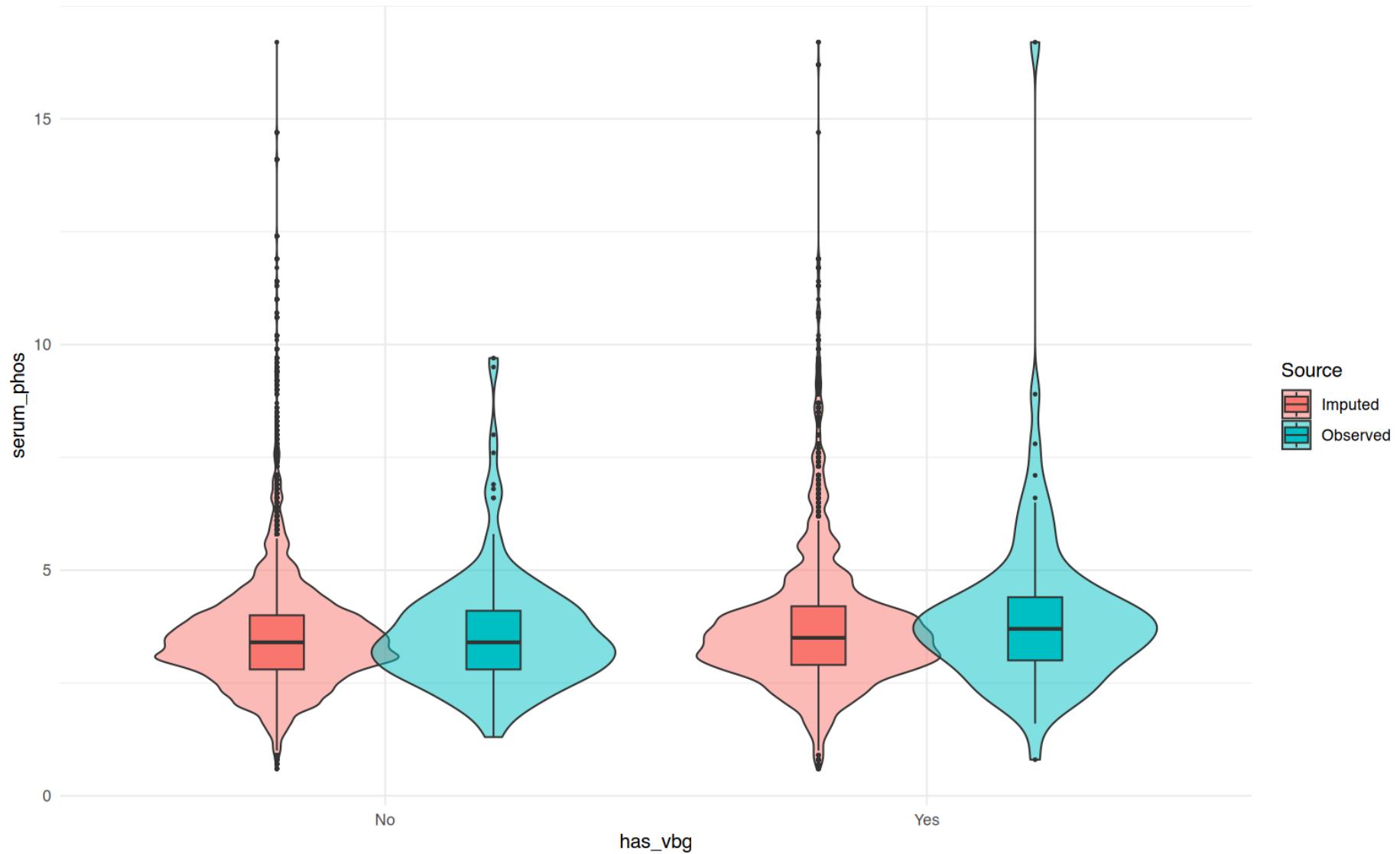
Observed vs imputed: serum_lac by has_vbg



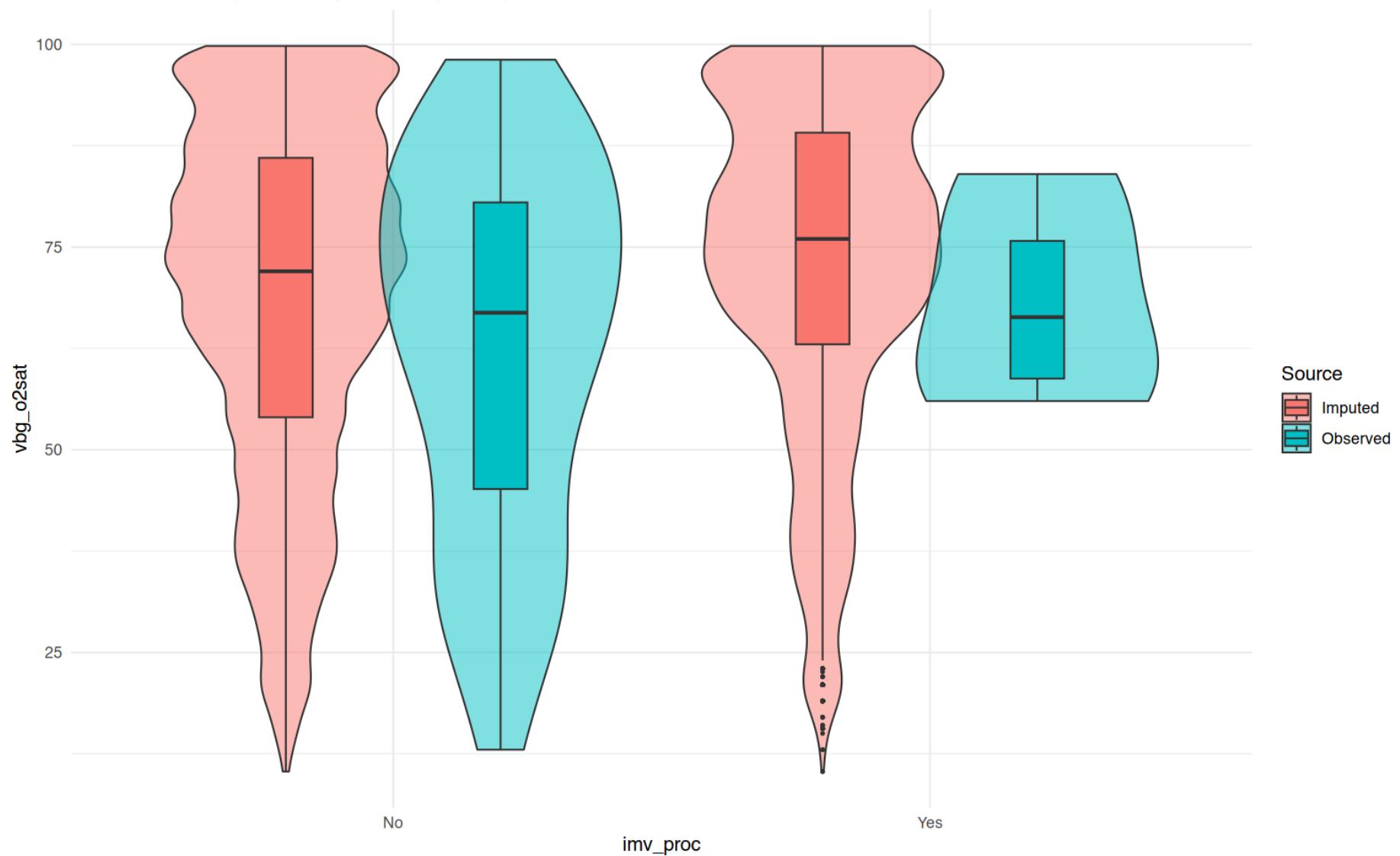
Observed vs imputed: curr_bmi by has_vbg



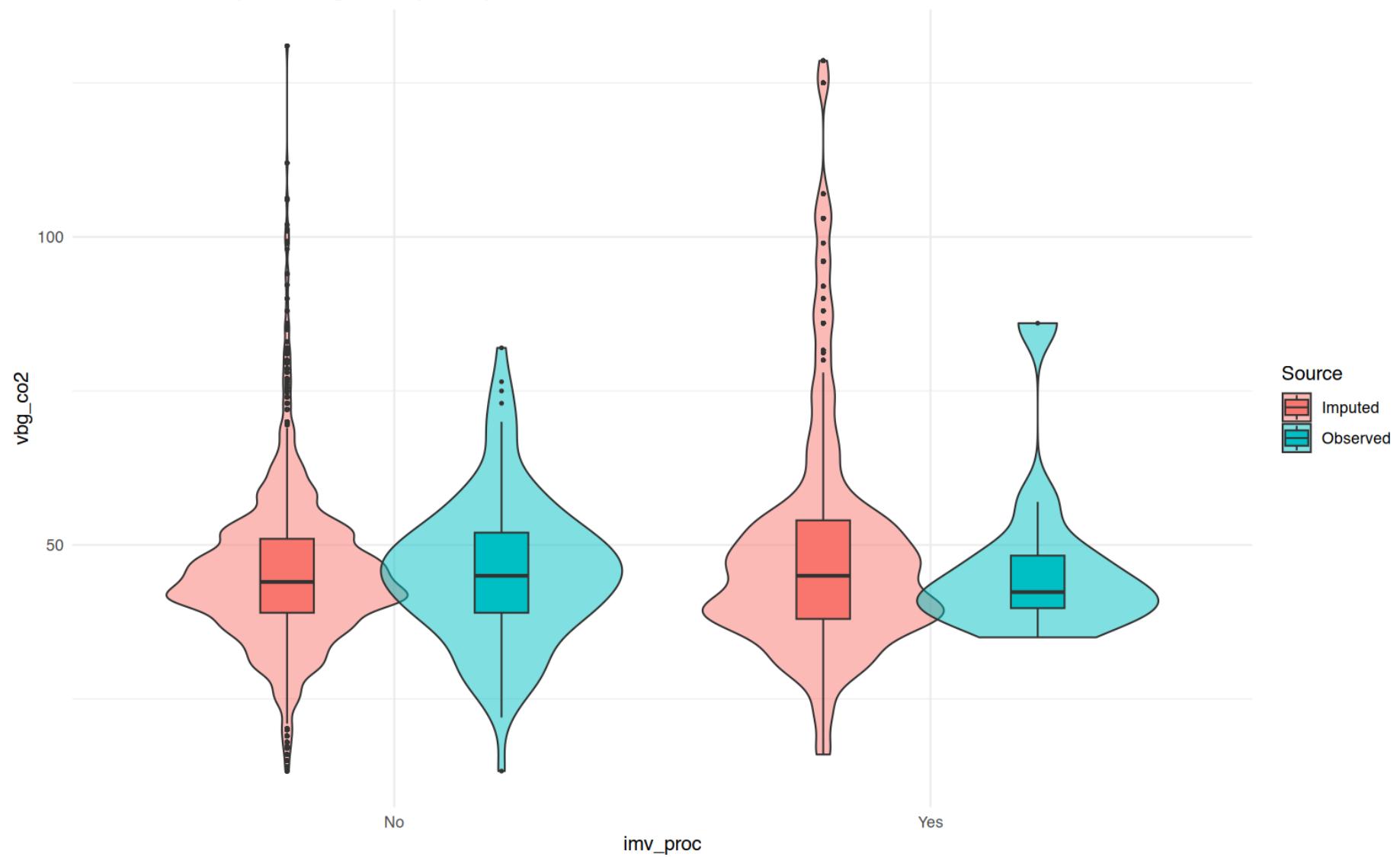
Observed vs imputed: serum_phos by has_vbg



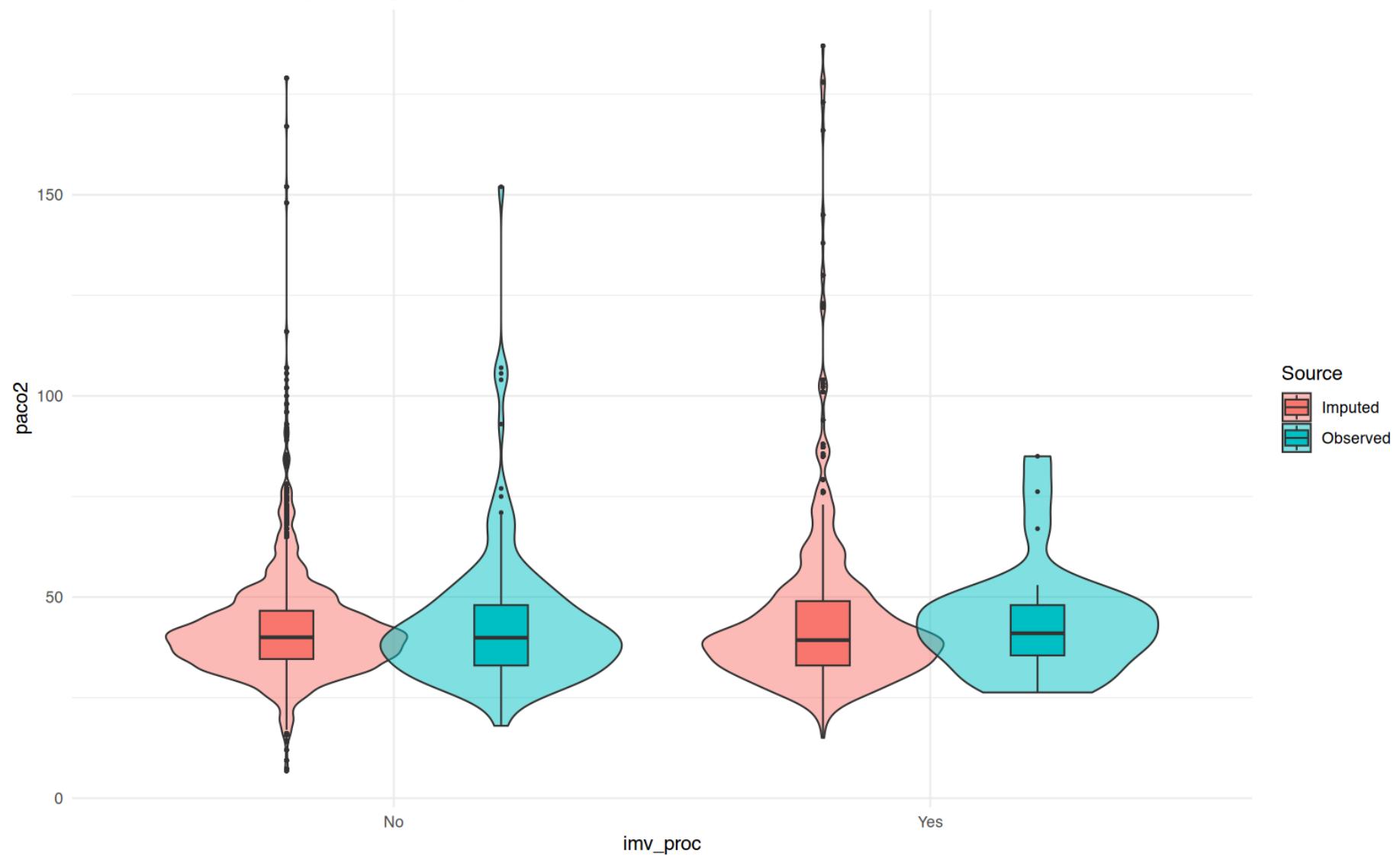
Observed vs imputed: vbg_o2sat by imv_proc



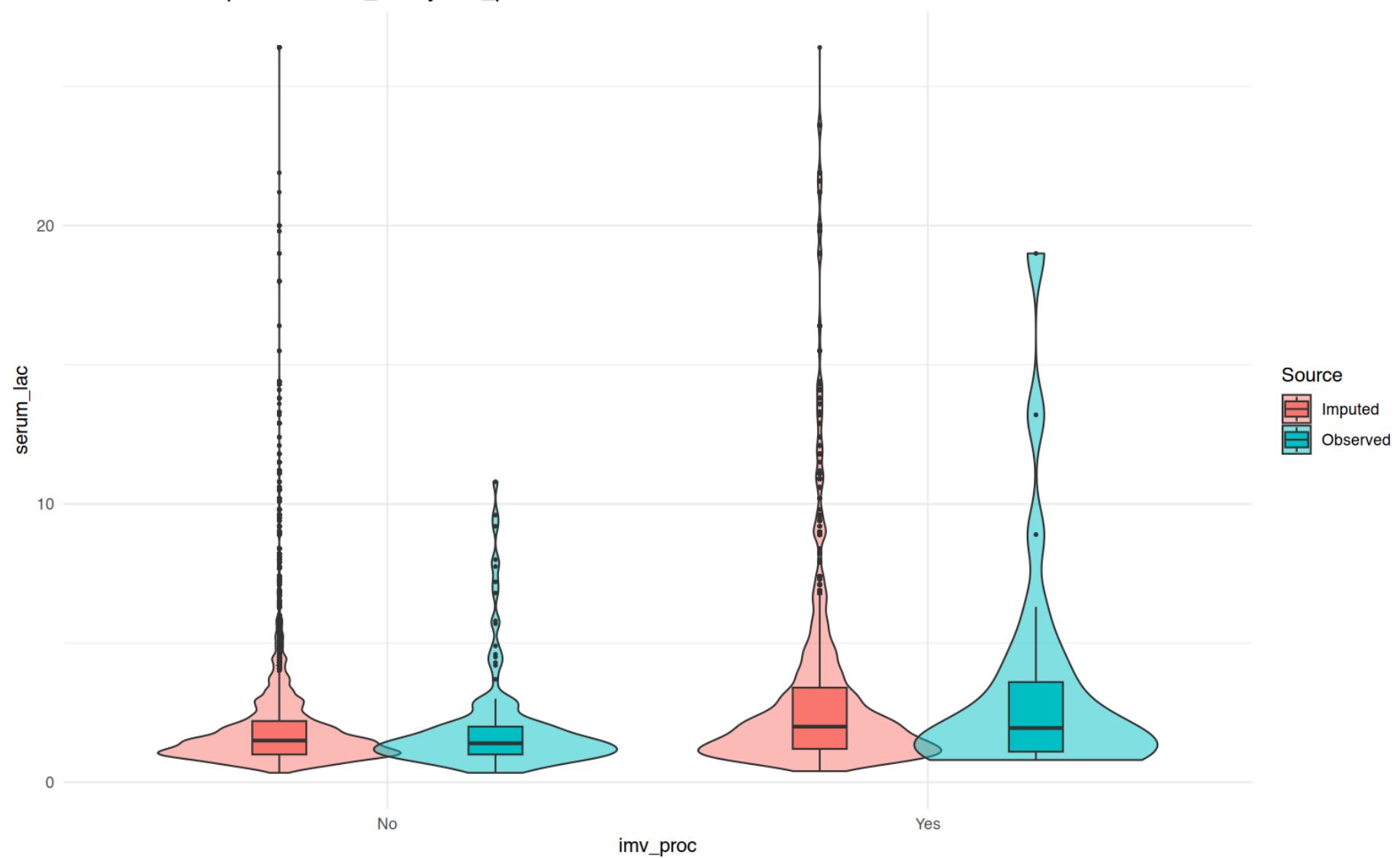
Observed vs imputed: vbg_co2 by imv_proc



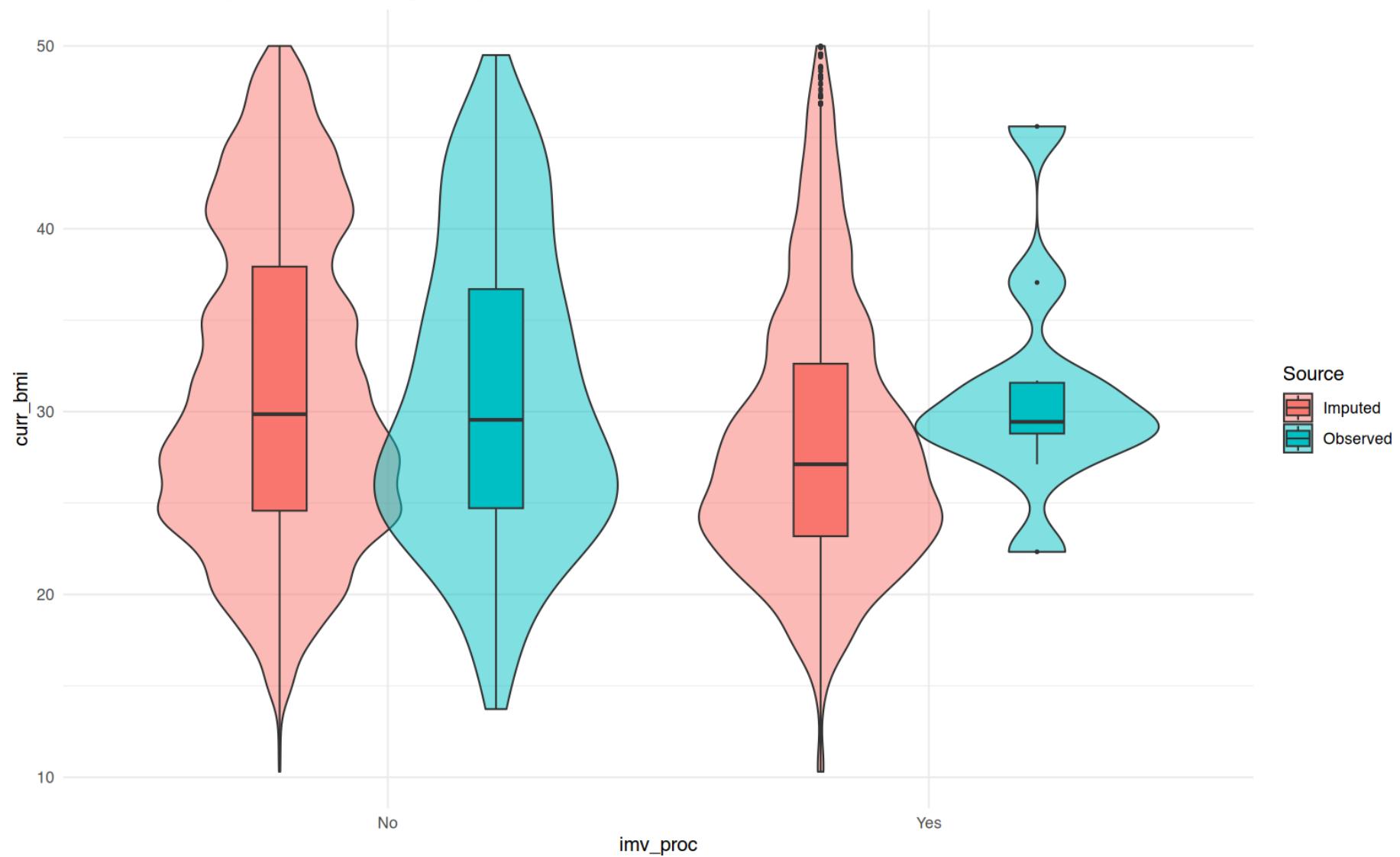
Observed vs imputed: paco2 by imv_proc



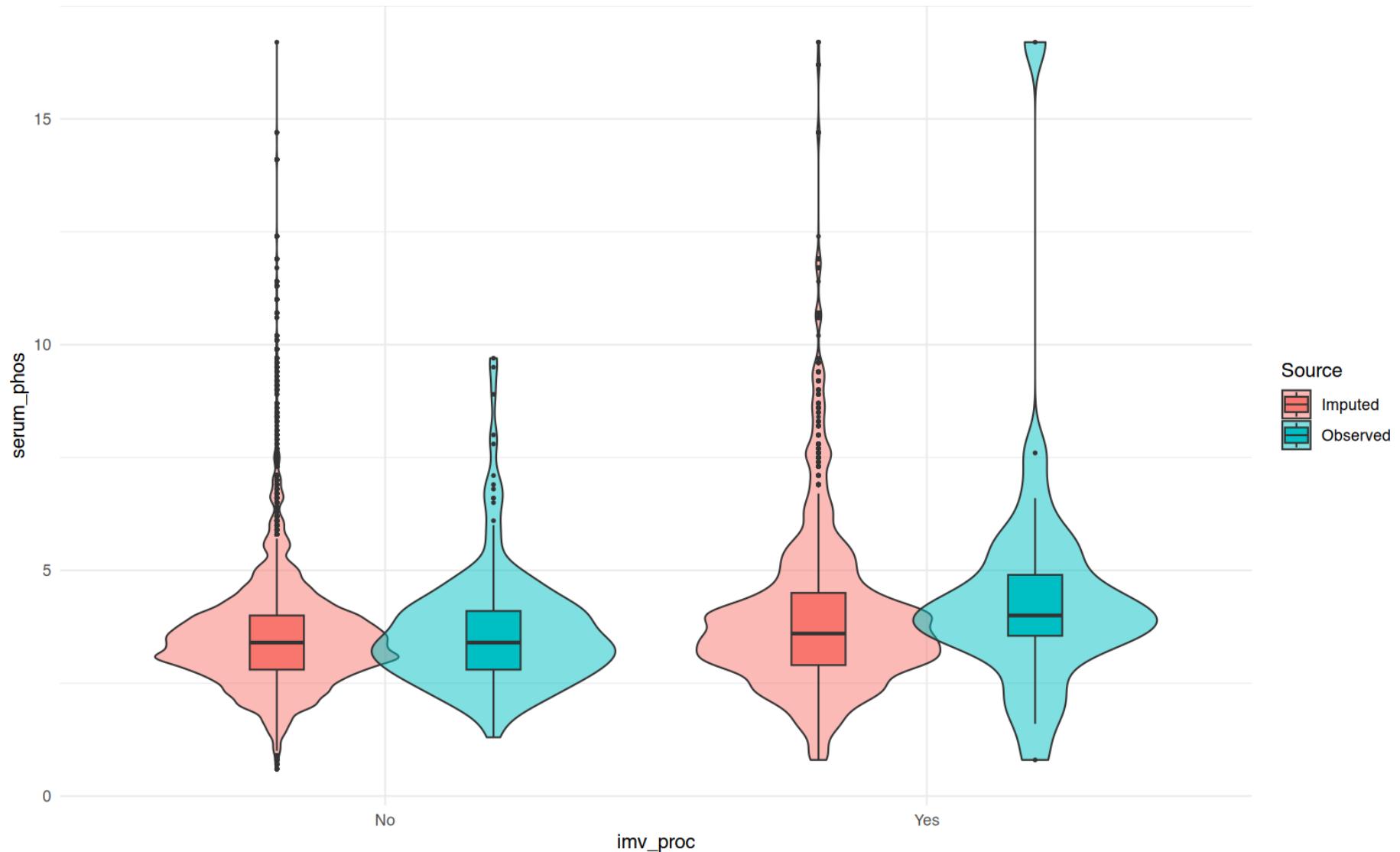
Observed vs imputed: serum_lac by imv_proc



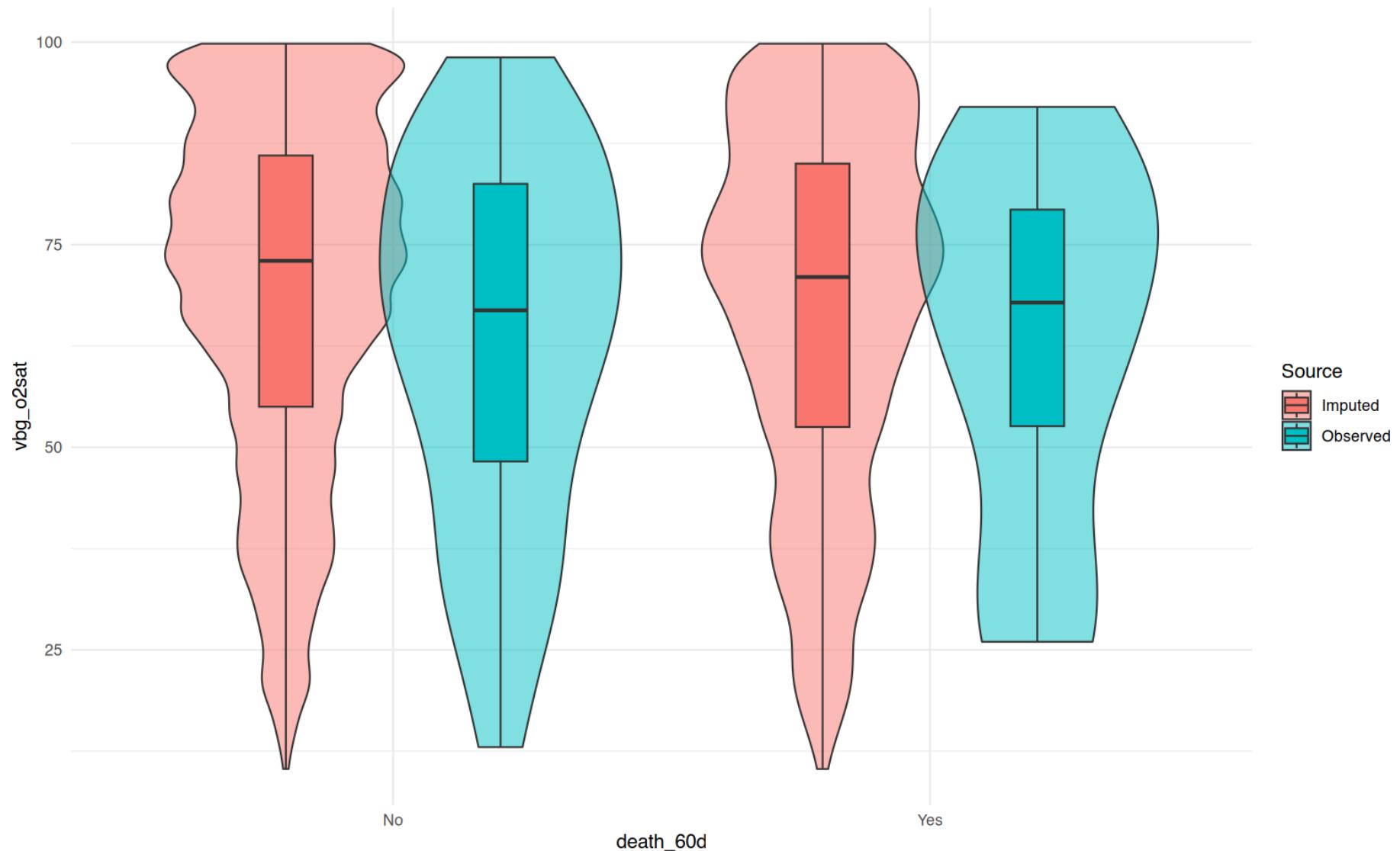
Observed vs imputed: curr_bmi by imv_proc



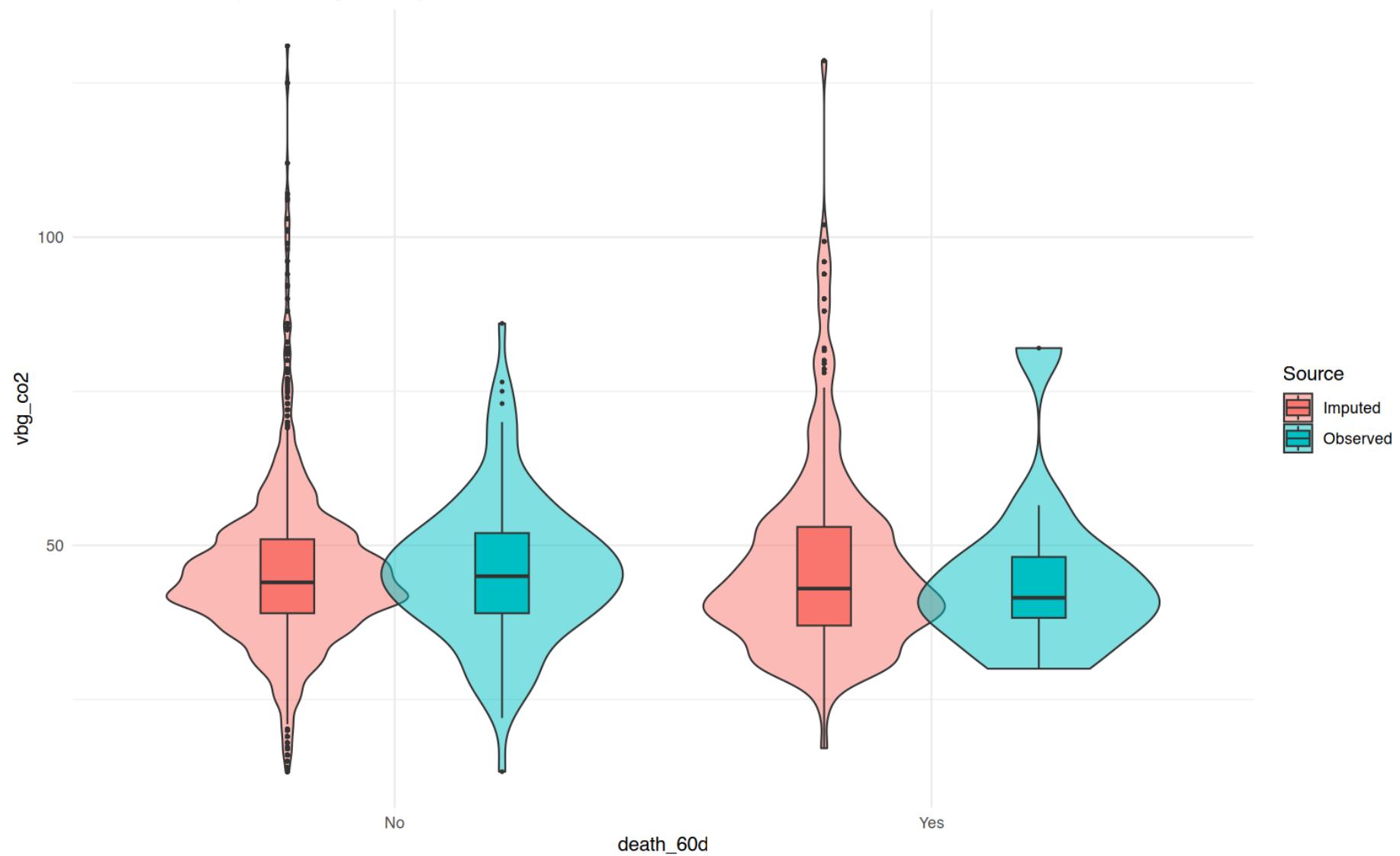
Observed vs imputed: serum_phos by imv_proc



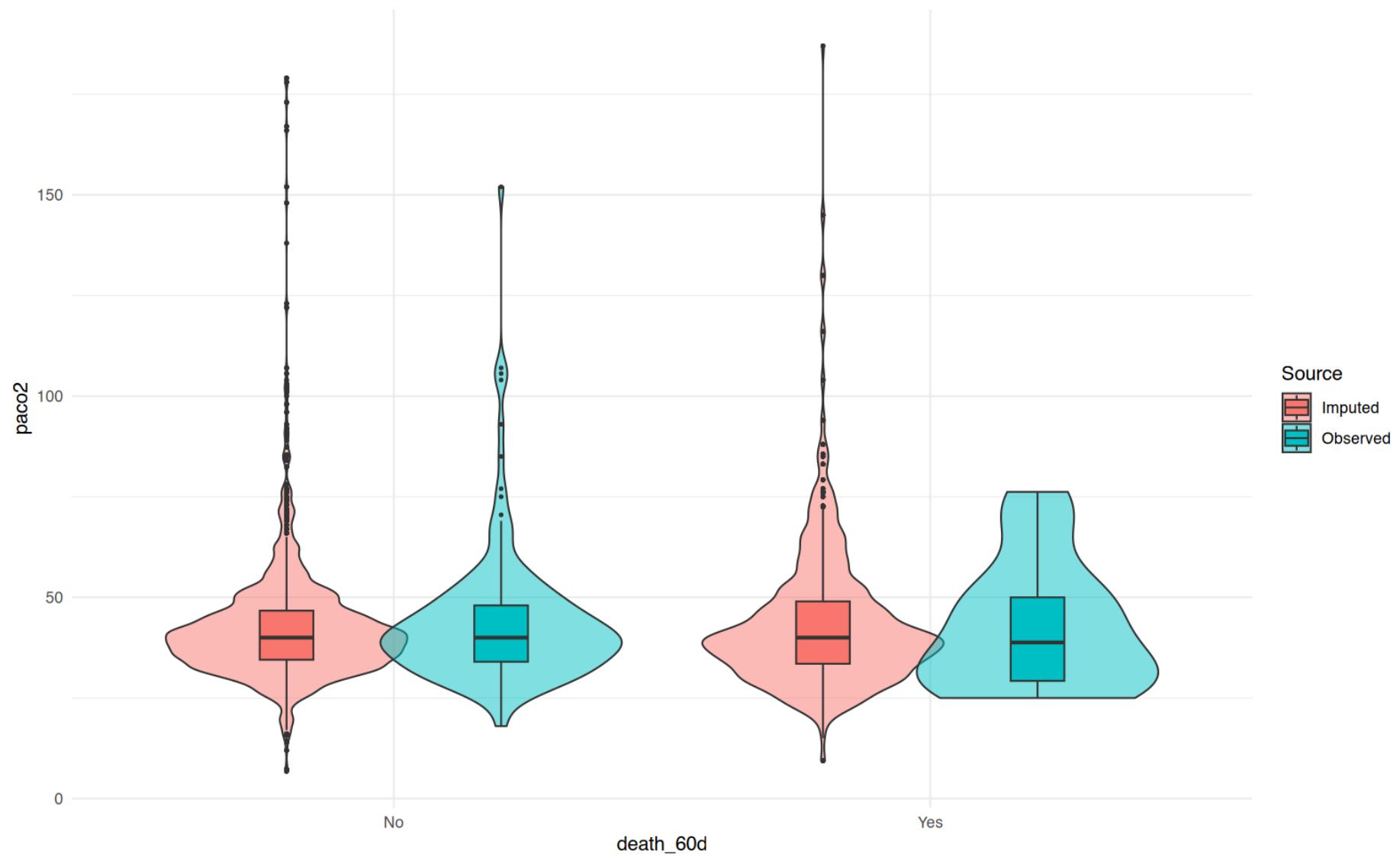
Observed vs imputed: vbg_o2sat by death_60d



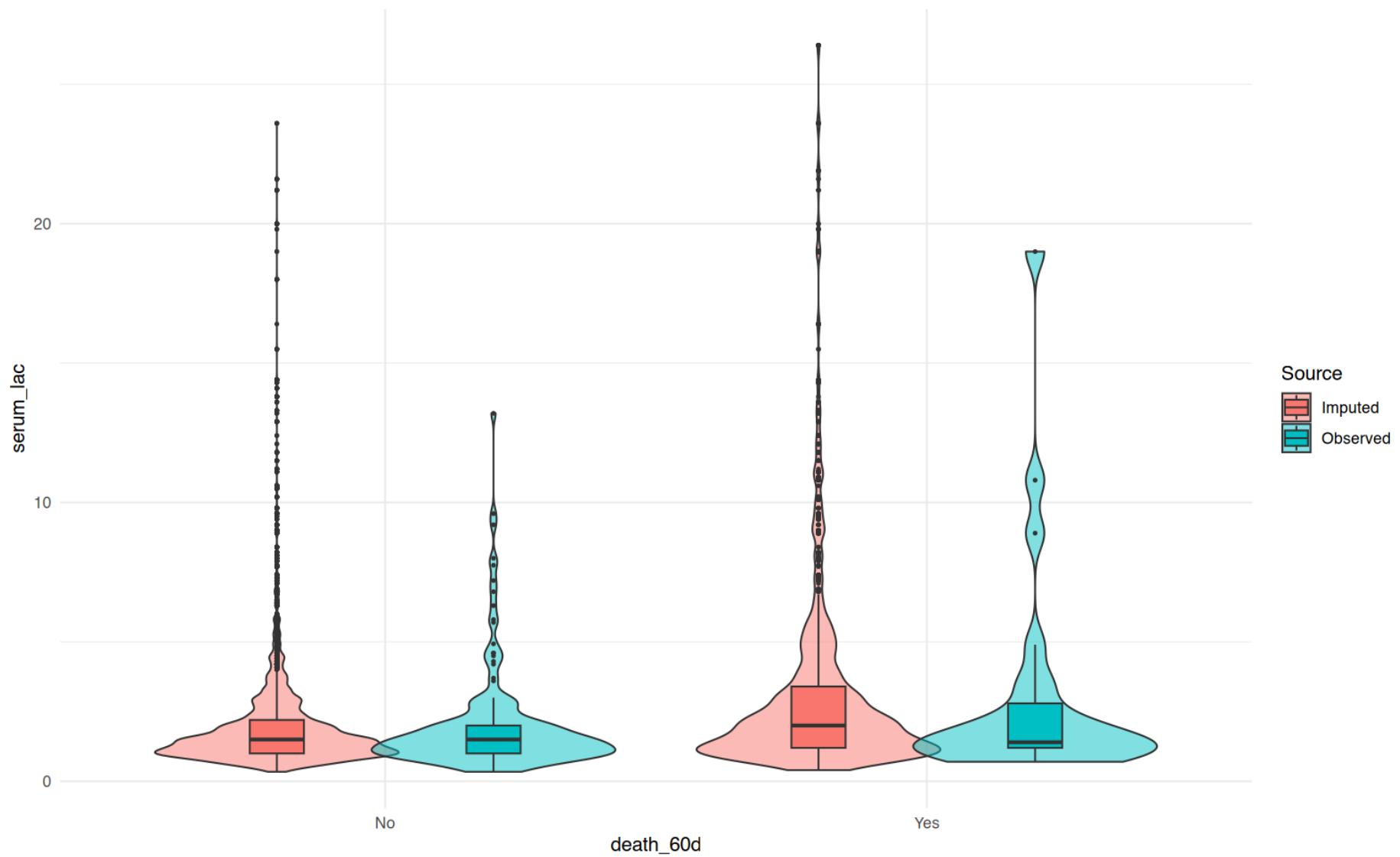
Observed vs imputed: vbg_co2 by death_60d



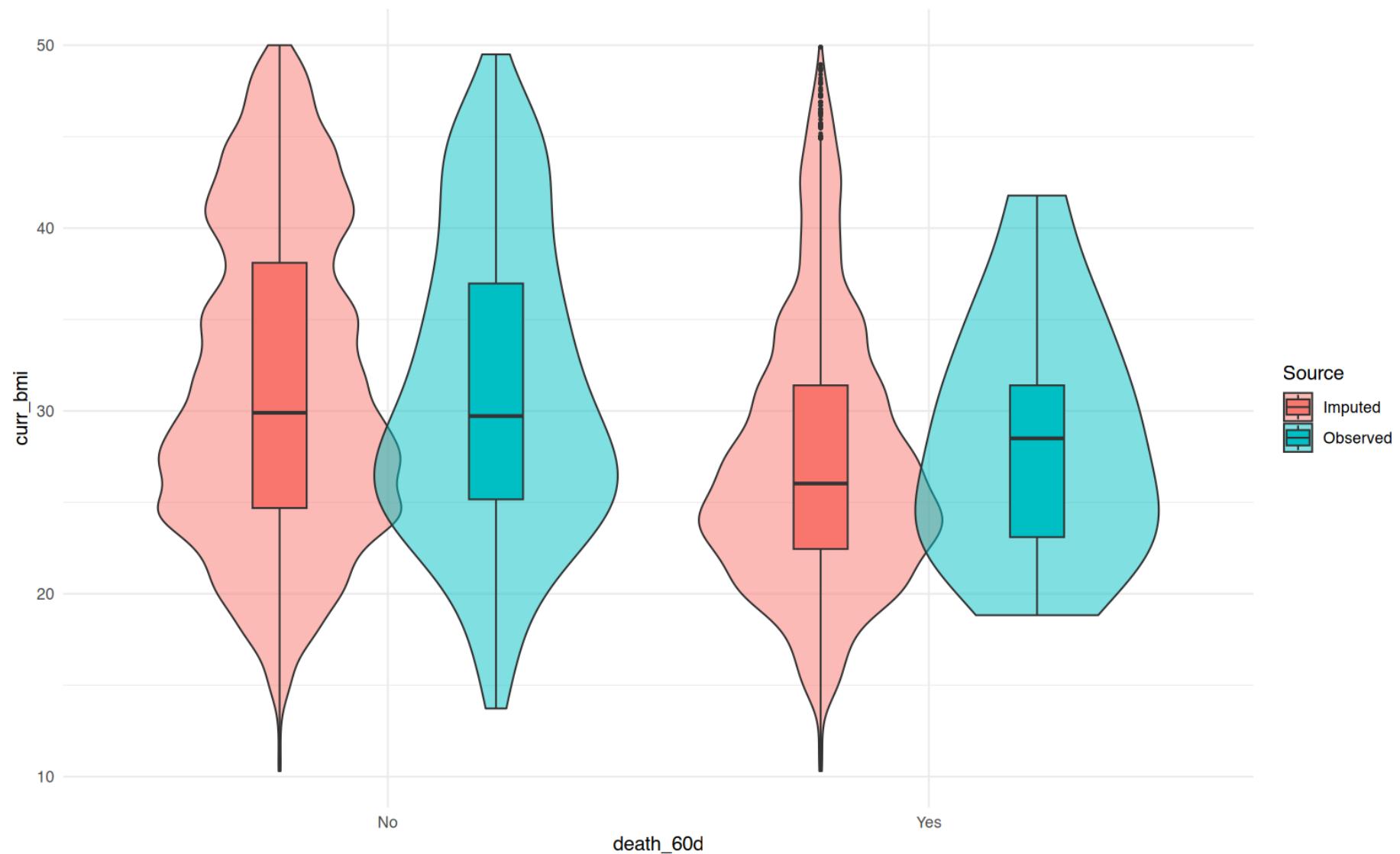
Observed vs imputed: paco2 by death_60d



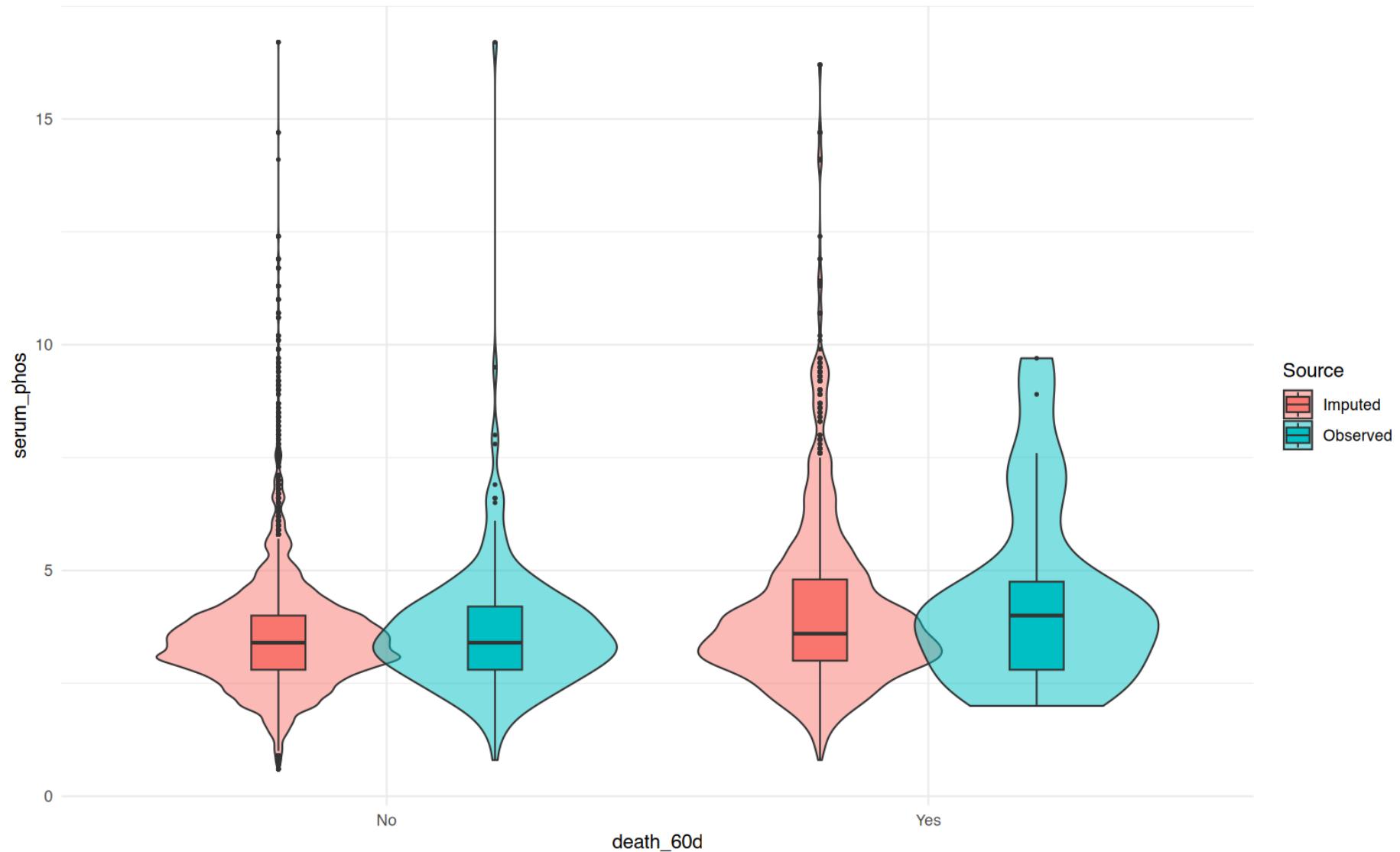
Observed vs imputed: serum_lac by death_60d



Observed vs imputed: curr_bmi by death_60d



Observed vs imputed: serum_phos by death_60d



```
# Purpose: mi mcerr progress.  
mc_file <- results_path("mi_mcerr_progress.csv")  
stopifnot(file.exists(mc_file))  
mc_prog <- utils::read.csv(mc_file)
```

```

render_table_pdf_maybe(
  mc_prog,
  caption = "MC error progress by m (early-stop diagnostic)",
  file_stub = "mi_mcerr_progress",
  digits = 3,
  show = SHOW_LOW_VALUE_TABLES
)

# --- Lean missingness audit (memory-safe) -----
library(dplyr)
library(ggplot2)
library(naniar)

stopifnot(exists("imp"))

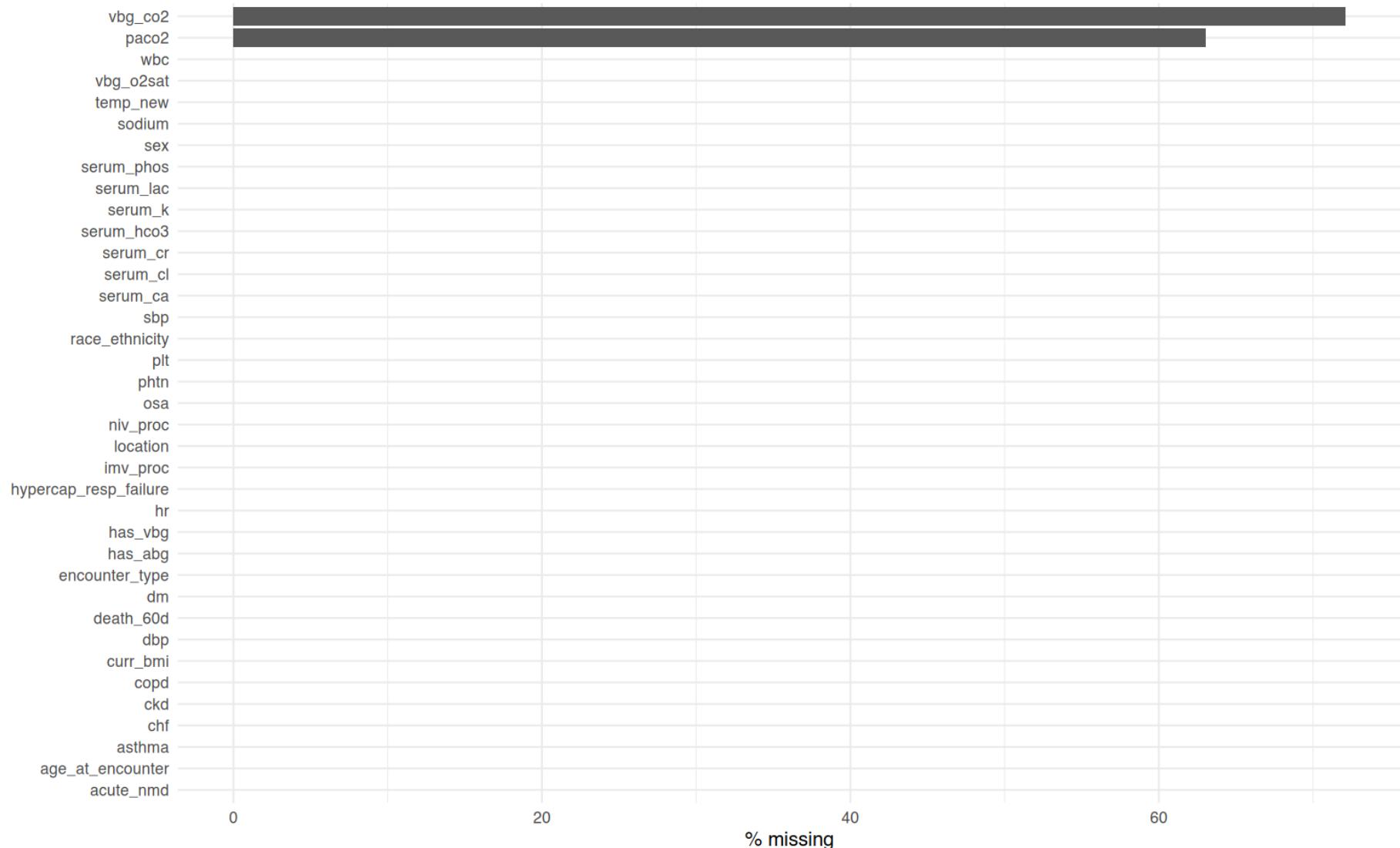
dat_imp <- mice::complete(imp, action = 1, include = FALSE)

# 1) Tabular summary on completed data (should be near 0% by design)
miss_tbl <- naniar::miss_var_summary(dat_imp) %>% arrange(desc(pct_miss))
render_table_pdf_maybe(
  miss_tbl,
  caption = "Missingness after MI (completed data)",
  file_stub = "missingness_after_mi",
  digits = 2,
  show = SHOW_LOW_VALUE_TABLES
)

# 2) Bar plot of top-K (mostly zeros after imputation)
K <- 40
top_vars <- miss_tbl$variable[seq_len(min(K, nrow(miss_tbl)))]
p_top <- ggplot(miss_tbl[miss_tbl$variable %in% top_vars, ],
                 aes(x = reorder(variable, pct_miss), y = pct_miss)) +
  geom_col() +
  coord_flip() +
  labs(title = "Top missing variables (after MI)", x = NULL, y = "% missing") +
  theme_minimal()
print(p_top)

```

Top missing variables (after MI)



```
# 3) Small heatmap on imputed data (rows/cols sampled)
M <- 60
R <- min(1500, nrow(dat_imp))
cols_heat <- head(top_vars, M)
```

```
rows_heat <- dplyr::slice_sample(dat_imp, n = R)

p_heat <- naniar::vis_miss(rows_heat[, cols_heat, drop = FALSE]) +
  labs(title = sprintf("Missingness heatmap on imputed data (%d rows × %d cols)", R, length(cols_heat)))
print(p_heat)
```

Missingness heatmap on imputed data (1500 rows x 37 cols)



```
# 4) Optional, but often heavy: UpSet of co-missingness - skip by default.  
# If you really want it, do it for top 6-10 variables only:  
# naniar::gg_miss_upset(dat_imp[, head(top_vars, 8), drop = FALSE])
```

3.3 Refit propensity models within each imputation

MI propensity scores use logistic regression with restricted cubic splines (`rms::rcs`, 4 knots by default) for continuous covariates; the same covariate set used in non-MI models is reused here (`covars_ps`). IPSW truncation rules are unchanged.

Note: the MI computations below run in a single pass per imputation (weights, balance, cat3, spline). Subsequent MI sections reuse those outputs and will stop if they are missing.

3.3.1 FAIL-FAST CHECKS

```
# Purpose: fail fast checks.
# These checks intentionally stop early if MI state is not valid.
# Failing here is cheaper than failing deep in long modeling loops.
if (M_IMP < 50) {
  stop("M_IMP must be >= 50. Current M_IMP = ", M_IMP)
}

stopifnot(exists("imp"))

if (imp$m < 50) {
  stop("imp$m must be >= 50. Current imp$m = ", imp$m)
}

# Ensure no missing covariates in each imputation for WeightIt
for (i in seq_len(imp$m)) {
  di <- get_imp(i)
  # Weighting models require complete covariates after MI.
  assert_no_na_covars(di, covars_ps, context = paste0("imputation ", i))
}

# This single-pass loop computes MI weights, target balance, 3-level outcomes,
# and spline outcomes in one traversal of imputations to avoid repeated
# mice::complete() calls and re-normalization.

stopifnot(exists("imp"))
imp_n <- imp$m
mi_single_pass_t0 <- Sys.time()
```

```

# --- Helper: fit ABG/VBG weights for a single imputation -----
fit_abg_one <- function(d, imp_index) {
  # Build PS design frame for this imputation.
  d_ps <- d[, c("has_abg", "has_vbg", covars_ps), drop = FALSE]
  d_ps <- normalize_types(d_ps, levels_ref)
  d_ps <- droplevels_all(d_ps)

  # Write one factor-level diagnostic file (first imputation only).
  if (imp_index == 1L) {
    write_factor_levels_diag(d_ps, covars_ps, "abg", file_prefix = "mi_logistic_ps_factor_levels")
  }

  # Guardrail: covariates must be complete before fitting propensity model.
  assert_no_na_covars(d_ps, covars_ps, context = "ABG MI PS (glm)")

  # Fit MI logistic propensity model with spline terms for continuous predictors.
  ps_fit <- fit_mi_ps_glm(
    d_ps, "has_abg", covars_ps,
    k = MI_PS_SPLINE_K, maxit = MI_GLM_MAXIT,
    context = make_context(
      stage = "MI", component = "mi_ps_glm",
      analysis_variant = "weighted_imputed",
      model_type = "ps",
      group = "ABG",
      outcome = NA_character_,
      imputation = imp_index,
      batch = NA_integer_
    )
  )
  if (!is.null(ps_fit$error)) {
    stop("MI PS model failed (ABG, imp ", imp_index, "): ", ps_fit$error)
  }

  # Convert propensity scores to one-sided IPSW according to project rules.
  ipow <- compute_ipow_weights(
    ps_fit,

```

```

treat = to01(d_ps$has_abg),
ps_floor_quantile = ps_trunc_quantile,
stabilize = TRUE
)
assert_finite_weights(ipow$weights[d_ps$has_abg == 1], "w_abg")

# Return compact metadata only (no large model object retained).
list(
  weights = ipow$weights,
  ps = ipow$ps,
  ipow_info = list(
    ps_floor = ipow$ps_floor,
    cap      = ipow$cap,
    trunc_rate = mean(ipow$truncated, na.rm = TRUE)
  ),
  method = ps_fit$method,
  fit_ok = isTRUE(ps_fit$fit_ok),
  converged = isTRUE(ps_fit$converged),
  n = ps_fit$n,
  p = ps_fit$p,
  formula = ps_fit$formula,
  basis = ps_fit$basis,
  coef = ps_fit$coef,
  vcov_diag = ps_fit$vcov_diag
)
}

fit_vbg_one <- function(d, imp_index) {
  # Same logic as ABG helper, using has_vbg as treatment indicator.
  d_ps <- d[, c("has_abg", "has_vbg", covars_ps), drop = FALSE]
  d_ps <- normalize_types(d_ps, levels_ref)
  d_ps <- droplevels_all(d_ps)
  if (imp_index == 1L) {
    write_factor_levels_diag(d_ps, covars_ps, "vbg", file_prefix = "mi_logistic_ps_factor_levels")
  }
  assert_no_na_covars(d_ps, covars_ps, context = "VBG MI PS (glm)")
  ps_fit <- fit_mi_ps_glm(

```

```

d_ps, "has_vbg", covars_ps,
k = MI_PS_SPLINE_K, maxit = MI_GLM_MAXIT,
context = make_context(
  stage = "MI", component = "mi_ps_glm",
  analysis_variant = "weighted_imputed",
  model_type = "ps",
  group = "VBG",
  outcome = NA_character_,
  imputation = imp_index,
  batch = NA_integer_
)
)
if (!is.null(ps_fit$error)) {
  stop("MI PS model failed (VBG, imp ", imp_index, "): ", ps_fit$error)
}

# Apply one-sided IPSW transformation to VBG propensity scores.
ipow <- compute_ipow_weights(
  ps_fit,
  treat = to01(d_ps$has_vbg),
  ps_floor_quantile = ps_trunc_quantile,
  stabilize = TRUE
)
assert_finite_weights(ipow$weights[d_ps$has_vbg == 1], "w_vbg")

# Keep only lightweight fields needed downstream.
list(
  weights = ipow$weights,
  ps = ipow$ps,
  ipow_info = list(
    ps_floor = ipow$ps_floor,
    cap      = ipow$cap,
    trunc_rate = mean(ipow$truncated, na.rm = TRUE)
  ),
  method = ps_fit$method,
  fit_ok = isTRUE(ps_fit$fit_ok),
  converged = isTRUE(ps_fit$converged),

```

```

n = ps_fit$n,
p = ps_fit$p,
formula = ps_fit$formula,
basis = ps_fit$basis,
coef = ps_fit$coef,
vcov_diag = ps_fit$vcov_diag
)
}

# --- Helper: target balance (treated vs target) -----
target_balance_table <- function(data, treat_var, weights, covars, levels_ref = NULL) {
  # Computes standardized mean differences before and after weighting,
  # comparing treated rows to the target population distribution.
  stopifnot(length(weights) == nrow(data))
  treat <- to01(data[[treat_var]]) == 1L
  treat[is.na(treat)] <- FALSE
  w <- as.numeric(weights)

  w_mean <- function(x, wts) {
    # Small weighted-mean helper that tolerates missing/non-finite values.
    ok <- is.finite(x) & is.finite(wts)
    if (!any(ok)) return(NA_real_)
    sum(wts[ok] * x[ok]) / sum(wts[ok])
  }

  out <- lapply(covars, function(v) {
    x <- data[[v]]
    if (is.character(x)) {
      if (!is.null(levels_ref) && !is.null(levels_ref[[v]])) {
        x <- factor(x, levels = levels_ref[[v]])
      } else {
        x <- factor(x)
      }
    }

    if (is.factor(x)) {
      # Factor covariates: evaluate SMD at each level indicator.

```

```

levs <- levels(x)
lapply(levs, function(lv) {
  ind <- as.integer(x == lv)
  p_target <- mean(ind, na.rm = TRUE)
  sd_target <- sqrt(p_target * (1 - p_target))
  ind_treat <- ind[treat]
  w_treat <- w[treat]
  pre <- mean(ind_treat, na.rm = TRUE)
  post <- w_mean(ind_treat, w_treat)
  smd_pre <- if (is.finite(sd_target) && sd_target > 0) (pre - p_target) / sd_target else NA_real_
  smd_post <- if (is.finite(sd_target) && sd_target > 0) (post - p_target) / sd_target else NA_real_
  data.frame(
    variable = v,
    level = lv,
    type = "factor",
    smd_pre = smd_pre,
    smd_post = smd_post,
    stringsAsFactors = FALSE
  )
}) |> dplyr::bind_rows()
} else {
  # Numeric covariates: compare means scaled by target SD.
  x_num <- suppressWarnings(as.numeric(x))
  mean_target <- mean(x_num, na.rm = TRUE)
  sd_target <- stats::sd(x_num, na.rm = TRUE)
  x_treat <- x_num[treat]
  w_treat <- w[treat]
  mean_pre <- mean(x_treat, na.rm = TRUE)
  mean_post <- w_mean(x_treat, w_treat)
  smd_pre <- if (is.finite(sd_target) && sd_target > 0) (mean_pre - mean_target) / sd_target else NA_real_
  smd_post <- if (is.finite(sd_target) && sd_target > 0) (mean_post - mean_target) / sd_target else NA_real_
  data.frame(
    variable = v,
    level = NA_character_,
    type = "numeric",
    smd_pre = smd_pre,
    smd_post = smd_post,

```

```

        stringsAsFactors = FALSE
    )
}
})
dplyr::bind_rows(out)
}

# --- Helper: fit 3-level outcome per imputation -----
fit_cat3_imp <- function(d, weights, outcome_var, co2_var,
                           treat_var,
                           low_cut, high_cut, group_label, imp_index) {
  # Fits one weighted categorical-CO2 outcome model for one imputation.
  stopifnot(co2_var %in% names(d))
  d[[co2_var]] <- coerce_num(d[[co2_var]])

  # Restrict to treated cohort with observed CO2.
  g <- d[[treat_var]] == 1 & is.finite(d[[co2_var]])
  if (!any(g)) {
    return(list(error = "No treated rows with finite CO2"))
  }

  d2 <- d[g, , drop = FALSE]
  w <- weights[g]
  w[!is.finite(w)] <- NA_real_
  ok <- is.finite(w)
  if (!all(ok)) {
    d2 <- d2[ok, , drop = FALSE]
    w <- w[ok]
    if (nrow(d2) == 0L) return(list(error = "All weights non-finite"))
  }

  d2$co2_cat <- make_co2_cat3(d2[[co2_var]], low_cut, high_cut)
  d2$co2_cat <- stats::relevel(base::droplevels(d2$co2_cat), ref = "Normal")
  if (nlevels(d2$co2_cat) < 2) {
    return(list(error = "co2_cat has <2 levels"))
  }

  d2[[outcome_var]] <- to01(d2[[outcome_var]])
}

```

```

des <- survey::svydesign(ids = ~1, weights = ~w, data = d2)
fml <- stats::reformulate(c("co2_cat", adj_core), response = outcome_var)

# Fit model with diagnostics wrapper so warnings/errors are tracked in logs.
fit_res <- fit_with_diagnostics(
  function() survey::svyglm(fml, design = des, family = quasibinomial(),
    control = stats::glm.control(maxit = 50)),
  context = make_context(
    stage = "outcome",
    component = "cat3",
    analysis_variant = "mi_ipw",
    model_type = "cat3",
    group = group_label,
    outcome = outcome_var,
    imputation = imp_index,
    batch = NA_integer_
  )
)
diag <- fit_res$diag
if (!is.null(diag) && nrow(diag)) {
  diag$n_used <- nrow(d2)
  diag$events <- sum(d2[[outcome_var]] == 1, na.rm = TRUE)
}
fit <- fit_res$fit
fit_compact <- NULL
if (!is.null(fit) && inherits(fit, "svyglm")) {
  # Store only coef/vcov to avoid carrying heavy fit objects in memory.
  fit_compact <- list(
    coef = stats::coef(fit),
    vcov = stats::vcov(fit)
  )
}
list(fit = fit_compact, diag = diag, warnings = fit_res$warnings)
}

# --- Pool terms across imputations (robust svyglm) -----
pool_terms <- function(fits, term_prefix = NULL, term_pattern = NULL, min_ok_frac = 0.9) {

```

```

fit_compact <- lapply(fits, function(x) {
  if (inherits(x, "svyglm")) {
    return(list(coef = stats::coef(x), vcov = stats::vcov(x)))
  }
  if (is.list(x) && !is.null(x$coef) && !is.null(x$vcov)) {
    return(list(coef = x$coef, vcov = x$vcov))
  }
  NULL
})
ok <- !vapply(fit_compact, is.null, logical(1))
fit_compact <- fit_compact[ok]
m_tot <- length(ok)
m_ok <- length(fit_compact)
if (m_ok == 0L) {
  stop("pool_terms: no successful fits; check per-imputation diagnostics.")
}
if (m_ok < ceiling(min_ok_frac * m_tot)) {
  warning("pool_terms: only ", m_ok, " / ", m_tot,
         " fits succeeded (below min_ok_frac).", call. = FALSE)
}
results <- lapply(fit_compact, `[[`, "coef")
variances <- lapply(fit_compact, `[[`, "vcov")
pooled <- mitools::MIcombine(results = results, variances = variances)
pooled_mi_vcov_check(pooled)
est <- as.numeric(coef(pooled))
se <- sqrt(diag(pooled$variance))
term <- names(coef(pooled))
out <- data.frame(
  term = term,
  logOR = est,
  estimate = exp(est),
  SE = se,
  LCL = exp(est - 1.96 * se),
  UCL = exp(est + 1.96 * se),
  stringsAsFactors = FALSE
)
if (!is.null(term_prefix)) {

```

```

    out <- out[grep(paste0("^", term_prefix), out$term), , drop = FALSE]
}
if (!is.null(term_pattern)) {
  out <- out[grep(term_pattern, out$term), , drop = FALSE]
}
out
}

# --- Fit spline model on treated cohort only (IPSW) -----
fit_spline_imp <- function(data, weights, outcome, co2_var, treat_var,
                            adj_vars = NULL,
                            spline_df = SPLINE_DF,
                            spline_basis = SPLINE_BASIS,
                            grid_df = NULL,
                            ref_idx = NULL,
                            imp_index = NA_integer_) {
  # Fits one weighted spline model for one imputation and (optionally)
  # returns grid-level log-OR predictions for pooling.
  spline_basis <- match.arg(spline_basis, c("ns", "rcs"))
  stopifnot(is.data.frame(data))
  stopifnot(length(weights) == nrow(data))
  stopifnot(all(c(outcome, co2_var, treat_var) %in% names(data)))
  if (!is.null(adj_vars) && length(adj_vars)) {
    missing_adj <- setdiff(adj_vars, names(data))
    if (length(missing_adj)) {
      return(list(error = paste0("Missing adj_core vars: ", paste(missing_adj, collapse = ", "))))
    }
  }

  data[[treat_var]] <- to01(data[[treat_var]])
  data[[outcome]] <- to01(data[[outcome]])
  data[[co2_var]] <- coerce_num(data[[co2_var]])
  w <- suppressWarnings(as.numeric(weights))

  keep <- (data[[treat_var]] == 1L) &
    is.finite(data[[co2_var]]) &
    !is.na(data[[outcome]]) &

```

```

is.finite(w) & (w > 0)

if (!any(keep)) {
  return(list(error = "No eligible treated rows after filtering (treated==1, finite CO2, non-missing outcome, finite
  ↵ positive weights)."))
}

d2 <- data[keep, , drop = FALSE]
w2 <- w[keep]
d2$w <- w2

if (length(unique(d2[[outcome]])) < 2L) {
  return(list(error = paste0("Outcome has one level in treated sample after filtering: outcome=",
                             outcome, ", n=", nrow(d2),
                             ", events=", sum(d2[[outcome]] == 1, na.rm = TRUE))))
}

des <- survey::svydesign(ids = ~1, weights = ~w, data = d2)
if (spline_basis == "ns") {
  # Natural spline basis for CO2 plus fixed adjustment covariates.
  rhs_terms <- c(sprintf("splines::ns(%s, %d)", co2_var, spline_df), adj_vars)
  fml <- stats::as.formula(paste(outcome, "~", paste(rhs_terms, collapse = " + ")))
} else {
  # rms::rcs option retained for compatibility with previous runs.
  dd <- rms::datadist(d2)
  old_opt <- options(datadist = ".__dd_tmp__")
  assign(".__dd_tmp__", dd, envir = .GlobalEnv)
  on.exit({
    options(old_opt)
    rm(list = ".__dd_tmp__", envir = .GlobalEnv)
  }, add = TRUE)
  rhs_terms <- c(sprintf("rms::rcs(%s, %d)", co2_var, spline_df), adj_vars)
  fml <- stats::as.formula(paste(outcome, "~", paste(rhs_terms, collapse = " + ")))
}

group_label <- if (identical(treat_var, "has_abg")) "ABG" else if (identical(treat_var, "has_vbg")) "VBG" else
  ↵ treat_var

```

```

fit_res <- fit_with_diagnostics(
  function() survey::svyglm(fml, design = des, family = quasibinomial(),
                           control = stats::glm.control(maxit = 50)),
  context = make_context(
    stage = "outcome",
    component = "spline",
    analysis_variant = "mi_ipw",
    model_type = "spline",
    group = group_label,
    outcome = outcome,
    imputation = imp_index,
    batch = NA_integer_
  )
)
fit <- fit_res$fit
warns <- fit_res$warnings
diag <- fit_res$diag
if (!is.null(diag) && nrow(diag)) {
  diag$n_used <- nrow(d2)
  diag$events <- sum(d2[[outcome]] == 1, na.rm = TRUE)
}
if (is.null(fit)) {
  err_msg <- if (!is.null(diag$error_message) && is.finite(nchar(diag$error_message))) diag$error_message else
  "svyglm error"
  return(list(error = err_msg,
             warnings = warns,
             diag = diag,
             n_used = nrow(d2),
             events = sum(d2[[outcome]] == 1, na.rm = TRUE)))
}

fit_compact <- list(
  # Compact storage for Rubin pooling.
  coef = stats::coef(fit),
  vcov = stats::vcov(fit)
)

```

```

if (!is.null(grid_df)) {
  if (is.null(ref_idx)) {
    return(list(error = "ref_idx is required when grid_df is provided.",
               warnings = warns, diag = diag))
  }
  pr <- tryCatch(
    predict_or_curve_from_fit(fit, grid_df, ref_idx, co2_var),
    error = function(e) list(error = paste0("predict error: ", conditionMessage(e)))
  )
  if (!is.null(pr$error)) return(list(error = pr$error,
                                         warnings = warns,
                                         diag = diag,
                                         n_used = nrow(d2),
                                         events = sum(d2[[outcome]] == 1, na.rm = TRUE)))
  return(list(
    fit = fit_compact,
    logOR = pr$logOR,
    var_logOR = pr$var_logOR,
    warnings = warns,
    diag = diag,
    n_used = nrow(d2),
    events = sum(d2[[outcome]] == 1, na.rm = TRUE)
  ))
}
}

list(fit = fit_compact, warnings = warns, diag = diag,
      n_used = nrow(d2),
      events = sum(d2[[outcome]] == 1, na.rm = TRUE))
}

# --- Pool spline coefficients across imputations -----
pool_spline_coefs <- function(fit_list, min_ok_frac = 0.9) {
  fit_compact <- lapply(fit_list, function(x) {
    if (is.list(x) && !is.null(x$fit) && is.list(x$fit) &&
        !is.null(x$fit$coef) && !is.null(x$fit$vcov)) {
      return(x$fit)
    }
  })
}

```

```

if (is.list(x) && !is.null(x$coef) && !is.null(x$vcov)) {
  return(x)
}
if (inherits(x, "svyglm")) {
  return(list(coef = stats::coef(x), vcov = stats::vcov(x)))
}
NULL
})
ok <- !vapply(fit_compact, is.null, logical(1))
m_tot <- length(ok)
m_ok  <- sum(ok)
if (m_ok < ceiling(min_ok_frac * m_tot)) {
  stop("Too many failed spline fits: m_ok=", m_ok, " / m_total=", m_tot)
}
fit_compact <- fit_compact[ok]
results <- lapply(fit_compact, `[[~, "coef")
variances <- lapply(fit_compact, `[[~, "vcov")
pooled <- mitools::MIcombine(results = results, variances = variances)
pooled_mi_vcov_check(pooled)
est <- as.numeric(coef(pooled))
se  <- sqrt(diag(pooled$variance))
data.frame(
  term = names(coef(pooled)),
  estimate = est,
  SE = se,
  LCL = est - 1.96 * se,
  UCL = est + 1.96 * se,
  m_used = m_ok,
  m_total = m_tot,
  row.names = NULL
)
}

# --- Pool spline curves (pointwise Rubin pooling on log-OR scale) -----
pool_spline_curve <- function(fit_list, grid_df, ref_idx, co2_ref, min_ok_frac = 0.9) {
  n_grid <- nrow(grid_df)
  ok <- vapply(fit_list, function(x) {

```

```

is.list(x) && is.null(x$error) &&
  !is.null(x$logOR) && !is.null(x$var_logOR) &&
    length(x$logOR) == n_grid && length(x$var_logOR) == n_grid
}, logical(1))
m_tot <- length(ok)
m_ok  <- sum(ok)
if (m_ok < ceiling(min_ok_frac * m_tot)) {
  stop("Too many failed spline fits: m_ok=", m_ok, " / m_total=", m_tot)
}

logOR_mat <- matrix(NA_real_, nrow = n_grid, ncol = m_ok)
var_mat <- matrix(NA_real_, nrow = n_grid, ncol = m_ok)
ok_idx <- which(ok)
for (i in seq_along(ok_idx)) {
  fit_i <- fit_list[[ok_idx[i]]]
  logOR_mat[, i] <- fit_i$logOR
  var_mat[, i] <- fit_i$var_logOR
}
Qbar <- rowMeans(logOR_mat, na.rm = TRUE)
Ubar <- rowMeans(var_mat, na.rm = TRUE)
B    <- apply(logOR_mat, 1, stats::var, na.rm = TRUE)
Tvar <- Ubar + (1 + 1 / m_ok) * B
se   <- sqrt(Tvar)
if (!is.na(ref_idx) && is.finite(ref_idx)) {
  if (abs(exp(Qbar[ref_idx]) - 1) > 1e-6) {
    warning("Pooled spline OR at reference differs from 1 (check ref_idx/co2_ref).", call. = FALSE)
  }
}
data.frame(
  grid_df,
  logOR = Qbar,
  SE_logOR = se,
  OR = exp(Qbar),
  LCL = exp(Qbar - 1.96 * se),
  UCL = exp(Qbar + 1.96 * se),
  co2_ref = co2_ref,
  m_used = m_ok,

```

```

    m_total = m_tot,
    row.names = NULL
  )
}

# --- Single-pass loop -----
cat3_outcomes <- c("imv_proc", "niv_proc", "death_60d", "hypercap_resp_failure")
cat3_labels <- c(
  imv_proc = "IMV",
  niv_proc = "NIV",
  death_60d = "Death60d",
  hypercap_resp_failure = "HCRF"
)
abg_co2_obs <- subset_data$paco2[subset_data$has_abg == 1 & !is.na(subset_data$paco2)]
if (length(abg_co2_obs) < 10) stop("ABG spline: not enough observed PaCO2 values.")
q_abg <- stats::quantile(abg_co2_obs, probs = c(0.02, 0.98), na.rm = TRUE)
grid_abg_info_mi <- make_co2_grid_ref(
  "paco2",
  seq(q_abg[1], q_abg[2], length.out = SPLINE_GRID_N),
  x_ref_abg,
  ABG_CO2_REF
)
grid_abg_mi <- grid_abg_info_mi$grid
ref_idx_abg_mi <- grid_abg_info_mi$ref_idx
co2_ref_abg_mi <- grid_abg_info_mi$co2_ref

vbg_co2_obs <- subset_data$vbg_co2[subset_data$has_vbg == 1 & !is.na(subset_data$vbg_co2)]
if (length(vbg_co2_obs) < 10) stop("VBG spline: not enough observed VBG CO2 values.")
q_vbg <- stats::quantile(vbg_co2_obs, probs = c(0.02, 0.98), na.rm = TRUE)
grid_vbg_info_mi <- make_co2_grid_ref(
  "vbg_co2",
  seq(q_vbg[1], q_vbg[2], length.out = SPLINE_GRID_N),
  x_ref_vbg,
  VBG_CO2_REF
)
grid_vbg_mi <- grid_vbg_info_mi$grid

```

```

ref_idx_vbg_mi <- grid_vbg_info_mi$ref_idx
co2_ref_vbg_mi <- grid_vbg_info_mi$co2_ref

covars_use <- intersect(covars_ps, names(subset_data))
covars_use_abg <- covars_use
covars_use_vbg <- covars_use

# MI PS covariate typing (continuous vs categorical) for transparency
mi_logistic_ps_type_file <- results_path("mi_logistic_ps_covariate_types.csv")
d_ps_tmp <- get_imp(1)
d_ps_tmp <- normalize_types(d_ps_tmp, levels_ref)
d_ps_tmp <- d_ps_tmp[, c("has_abg", "has_vbg", covars_ps), drop = FALSE]
d_ps_tmp <- droplevels_all(d_ps_tmp)
cont_vars <- covars_ps[vapply(covars_ps, ps_is_continuous, logical(1), df = d_ps_tmp)]
cat_vars_ps <- setdiff(covars_ps, cont_vars)
mi_logistic_ps_types <- dplyr::bind_rows(
  data.frame(variable = cont_vars, type = "continuous", stringsAsFactors = FALSE),
  data.frame(variable = cat_vars_ps, type = "categorical", stringsAsFactors = FALSE)
)
write_csv_safely(mi_logistic_ps_types, mi_logistic_ps_type_file, row_names = FALSE)
rm(d_ps_tmp, cont_vars, cat_vars_ps, mi_logistic_ps_types)
gc()

```

	used (Mb)	gc trigger (Mb)	limit (Mb)	max used (Mb)
Ncells	7027904	375.4	13923887	743.7
Vcells	500776259	3820.7	1121083665	8553.2
			16384	1121055634
				8553.0

```

mi_logistic_ps_abg_list <- vector("list", imp_n)
mi_logistic_ps_vbg_list <- vector("list", imp_n)
mi_weight_dir <- results_path("mi_weights")
if (!dir.exists(mi_weight_dir)) dir.create(mi_weight_dir, recursive = TRUE)
bal_rows_abg <- vector("list", imp_n)
bal_rows_vbg <- vector("list", imp_n)
cat3.fits_abg <- setNames(lapply(cat3_outcomes, function(x) vector("list", imp_n)), cat3_outcomes)
cat3.fits_vbg <- setNames(lapply(cat3_outcomes, function(x) vector("list", imp_n)), cat3_outcomes)
spline.fits_abg <- setNames(lapply(cat3_outcomes, function(x) vector("list", imp_n)), cat3_outcomes)
spline.fits_vbg <- setNames(lapply(cat3_outcomes, function(x) vector("list", imp_n)), cat3_outcomes)

```

```

cat3_diag_rows <- list()
spline_diag_rows <- list()
weight_diag_abg <- vector("list", imp_n)
weight_diag_vbg <- vector("list", imp_n)
mi_imp_perf <- vector("list", imp_n)

message("MI single-pass: running ", imp_n, " imputations sequentially.")
for (i in seq_len(imp_n)) {
  # Per-imputation telemetry (time + memory) supports full-run troubleshooting.
  imp_label <- sprintf("imp_%03d", i)
  append_mem_snapshot("mi_single_pass", imp_label, "pre")
  mem_pre <- get_vcells_stats()
  imp_t0 <- Sys.time()
  d <- get_imp(i)

  # Fit ABG and VBG propensity models for this imputation.
  set.seed(20251206 + i)
  fit_abg <- fit_abg_one(d, i)
  set.seed(30251206 + i)
  fit_vbg <- fit_vbg_one(d, i)

  w_abg_i <- fit_abg$weights
  w_vbg_i <- fit_vbg$weights
  ps_abg_i <- fit_abg$ps
  ps_vbg_i <- fit_vbg$ps
  w_path_abg <- file.path(mi_weight_dir, sprintf("w_abg_imp_%03d.rds", i))
  w_path_vbg <- file.path(mi_weight_dir, sprintf("w_vbg_imp_%03d.rds", i))
  ps_path_abg <- file.path(mi_weight_dir, sprintf("ps_abg_imp_%03d.rds", i))
  ps_path_vbg <- file.path(mi_weight_dir, sprintf("ps_vbg_imp_%03d.rds", i))
  saveRDS(w_abg_i, w_path_abg)
  saveRDS(w_vbg_i, w_path_vbg)
  saveRDS(ps_abg_i, ps_path_abg)
  saveRDS(ps_vbg_i, ps_path_vbg)

  # Persist compact PS metadata so downstream chunks can reload on demand.
  mi_logistic_ps_abg_list[[i]] <- list(

```

```

weights_path = w_path_abg,
ps_path = ps_path_abg,
ipow_info = fit_abg$ipow_info,
method = fit_abg$method,
formula = fit_abg$formula,
basis = fit_abg$basis,
coef = fit_abg$coef,
vcov_diag = fit_abg$vcov_diag,
fit_ok = fit_abg$fit_ok,
converged = fit_abg$converged,
n = fit_abg$n,
p = fit_abg$p
)
mi_logistic_ps_vbg_list[[i]] <- list(
  weights_path = w_path_vbg,
  ps_path = ps_path_vbg,
  ipow_info = fit_vbg$ipow_info,
  method = fit_vbg$method,
  formula = fit_vbg$formula,
  basis = fit_vbg$basis,
  coef = fit_vbg$coef,
  vcov_diag = fit_vbg$vcov_diag,
  fit_ok = fit_vbg$fit_ok,
  converged = fit_vbg$converged,
  n = fit_vbg$n,
  p = fit_vbg$p
)

# Target-balance summaries for this imputation and group.
bal_rows_abg[[i]] <- target_balance_table(
  d, "has_abg", w_abg_i, covars_use_abg, levels_ref
) |>
  dplyr::mutate(group = "ABG", imp = i)
bal_rows_vbg[[i]] <- target_balance_table(
  d, "has_vbg", w_vbg_i, covars_use_vbg, levels_ref
) |>
  dplyr::mutate(group = "VBG", imp = i)

```

```

# 3-level CO2 outcome models for each endpoint (ABG and VBG).
for (outcome_var in cat3_outcomes) {
  fit_abg_cat <- fit_cat3_imp(
    d, w_abg_i, outcome_var, "paco2", "has_abg",
    ABG_CO2_LOW, ABG_CO2_HIGH, "ABG", i
  )
  cat3_fits_abg[[outcome_var]][[i]] <- fit_abg_cat$fit
  if (!is.null(fit_abg_cat$diag)) cat3_diag_rows[[length(cat3_diag_rows) + 1L]] <- fit_abg_cat$diag

  fit_vbg_cat <- fit_cat3_imp(
    d, w_vbg_i, outcome_var, "vbg_co2", "has_vbg",
    VBG_CO2_LOW, VBG_CO2_HIGH, "VBG", i
  )
  cat3_fits_vbg[[outcome_var]][[i]] <- fit_vbg_cat$fit
  if (!is.null(fit_vbg_cat$diag)) cat3_diag_rows[[length(cat3_diag_rows) + 1L]] <- fit_vbg_cat$diag
}

# Spline outcome models for each endpoint (ABG and VBG).
for (outcome_var in cat3_outcomes) {
  spline_abg <- fit_spline_imp(
    d, w_abg_i, outcome_var, "paco2", "has_abg",
    adj_vars = adj_core, spline_df = SPLINE_DF, spline_basis = SPLINE_BASIS,
    grid_df = grid_abg_mi, ref_idx = ref_idx_abg_mi, imp_index = i
  )
  spline_fits_abg[[outcome_var]][[i]] <- spline_abg
  if (!is.null(spline_abg$diag)) spline_diag_rows[[length(spline_diag_rows) + 1L]] <- spline_abg$diag

  spline_vbg <- fit_spline_imp(
    d, w_vbg_i, outcome_var, "vbg_co2", "has_vbg",
    adj_vars = adj_core, spline_df = SPLINE_DF, spline_basis = SPLINE_BASIS,
    grid_df = grid_vbg_mi, ref_idx = ref_idx_vbg_mi, imp_index = i
  )
  spline_fits_vbg[[outcome_var]][[i]] <- spline_vbg
  if (!is.null(spline_vbg$diag)) spline_diag_rows[[length(spline_diag_rows) + 1L]] <- spline_vbg$diag
}

```

```

# Store compact weight diagnostics used in report tables.
weight_diag_abg[[i]] <- {
  w <- w_abg_i
  t <- d$has_abg
  w <- w[t == 1]
  c(
    n = length(w),
    min = min(w, na.rm = TRUE),
    p99 = stats::quantile(w, 0.99, na.rm = TRUE),
    max = max(w, na.rm = TRUE),
    ess = sum(w)^2 / sum(w^2)
  )
}
weight_diag_vbg[[i]] <- {
  w <- w_vbg_i
  t <- d$has_vbg
  w <- w[t == 1]
  c(
    n = length(w),
    min = min(w, na.rm = TRUE),
    p99 = stats::quantile(w, 0.99, na.rm = TRUE),
    max = max(w, na.rm = TRUE),
    ess = sum(w)^2 / sum(w^2)
  )
}

# Save per-imputation performance diagnostics.
mem_post <- get_vcells_stats()
imp_t1 <- Sys.time()
mi_imp_perf[[i]] <- data.frame(
  imp = i,
  seconds = as.numeric(difftime(imp_t1, imp_t0, units = "secs")),
  gc_vcells_used_mb_pre = mem_pre$gc_vcells_used_mb,
  gc_vcells_frac_pre = mem_pre$gc_vcells_frac,
  gc_vcells_used_mb_post = mem_post$gc_vcells_used_mb,
  gc_vcells_frac_post = mem_post$gc_vcells_frac,
  run_id = diag_run_id,

```

```

run_ts = as.character(Sys.time()),
stringsAsFactors = FALSE
)
append_mem_snapshot("mi_single_pass", imp_label, "post")

# Aggressive cleanup inside loop to reduce peak memory pressure.
rm(d, fit_abg, fit_vbg, w_abg_i, w_vbg_i, ps_abg_i, ps_vbg_i,
   w_path_abg, w_path_vbg, ps_path_abg, ps_path_vbg,
   mem_pre, mem_post, imp_t0, imp_t1, imp_label)
invisible(gc())
}

mi_imp_perf_df <- dplyr::bind_rows(mi_imp_perf)
write_csv_safely(mi_imp_perf_df, results_path("mi_single_pass_imp_diagnostics.csv"), row_names = FALSE)
saveRDS(mi_logistic_ps_abg_list, mi_logistic_ps_abg_file)
saveRDS(mi_logistic_ps_vbg_list, mi_logistic_ps_vbg_file)

get_mi_weight <- function(group, i) {
  stopifnot(i >= 1L, i <= imp_n)
  grp <- toupper(group)
  meta <- if (grp == "ABG") {
    mi_logistic_ps_abg_list[[i]]
  } else if (grp == "V рг") {
    mi_logistic_ps_vbg_list[[i]]
  } else {
    stop("Unknown group for get_mi_weight: ", group)
  }
  stopifnot(!is.null(meta$weights_path), file.exists(meta$weights_path))
  readRDS(meta$weights_path)
}

get_mi_ps <- function(group, i) {
  stopifnot(i >= 1L, i <= imp_n)
  grp <- toupper(group)
  meta <- if (grp == "ABG") {
    mi_logistic_ps_abg_list[[i]]
  } else if (grp == "V рг") {

```

```

    mi_logistic_ps_vbg_list[[i]]
} else {
  stop("Unknown group for get_mi_ps: ", group)
}
stopifnot(!is.null(meta$ps_path), file.exists(meta$ps_path))
readRDS(meta$ps_path)
}

append_outcome_diag(dplyr::bind_rows(cat3_diag_rows, spline_diag_rows))

bal_imp_abg <- list(
  bal_long = dplyr::bind_rows(bal_rows_abg)
)
bal_imp_abg$bal_imp_summary <- bal_imp_abg$bal_long |>
  dplyr::group_by(group, imp) |>
  dplyr::summarise(
    max_abs_post = max(abs(smd_post), na.rm = TRUE),
    mean_abs_post = mean(abs(smd_post), na.rm = TRUE),
    .groups = "drop"
  )
bal_imp_abg$worst_rows_overall <- bal_imp_abg$bal_long |>
  dplyr::mutate(abs_post = abs(smd_post)) |>
  dplyr::arrange(desc(abs_post)) |>
  dplyr::ungroup()
bal_imp_abg$worst_by_imp <- bal_imp_abg$bal_long |>
  dplyr::mutate(term = ifelse(is.na(level), variable, paste0(variable, ":", level)),
    abs_post = abs(smd_post)) |>
  dplyr::group_by(group, imp) |>
  dplyr::slice_max(abs_post, n = 1, with_ties = FALSE) |>
  dplyr::ungroup() |>
  dplyr::select(group, imp, term, smd_pre, smd_post, abs_post)
bal_imp_abg$worst_terms_by_imp <- bal_imp_abg$bal_long |>
  dplyr::mutate(term = ifelse(is.na(level), variable, paste0(variable, ":", level)),
    abs_post = abs(smd_post)) |>
  dplyr::group_by(group, imp) |>
  dplyr::slice_max(abs_post, n = 10, with_ties = FALSE) |>
  dplyr::ungroup() |>

```

```

dplyr::select(group, imp, term, smd_pre, smd_post, abs_post)

bal_imp_vbg <- list(
  bal_long = dplyr::bind_rows(bal_rows_vbg)
)
bal_imp_vbg$bal_imp_summary <- bal_imp_vbg$bal_long |>
  dplyr::group_by(group, imp) |>
  dplyr::summarise(
    max_abs_post = max(abs(smd_post), na.rm = TRUE),
    mean_abs_post = mean(abs(smd_post), na.rm = TRUE),
    .groups = "drop"
)
bal_imp_vbg$worst_rows_overall <- bal_imp_vbg$bal_long |>
  dplyr::mutate(abs_post = abs(smd_post)) |>
  dplyr::arrange(desc(abs_post)) |>
  dplyr::ungroup()
bal_imp_vbg$worst_by_imp <- bal_imp_vbg$bal_long |>
  dplyr::mutate(term = ifelse(is.na(level), variable, paste0(variable, ":", level)),
    abs_post = abs(smd_post)) |>
  dplyr::group_by(group, imp) |>
  dplyr::slice_max(abs_post, n = 1, with_ties = FALSE) |>
  dplyr::ungroup() |>
  dplyr::select(group, imp, term, smd_pre, smd_post, abs_post)
bal_imp_vbg$worst_terms_by_imp <- bal_imp_vbg$bal_long |>
  dplyr::mutate(term = ifelse(is.na(level), variable, paste0(variable, ":", level)),
    abs_post = abs(smd_post)) |>
  dplyr::group_by(group, imp) |>
  dplyr::slice_max(abs_post, n = 10, with_ties = FALSE) |>
  dplyr::ungroup() |>
  dplyr::select(group, imp, term, smd_pre, smd_post, abs_post)

mi_weight_diag_abg <- dplyr::bind_rows(lapply(weight_diag_abg, function(x) {
  as.data.frame(as.list(x), stringsAsFactors = FALSE)
}))
mi_weight_diag_vbg <- dplyr::bind_rows(lapply(weight_diag_vbg, function(x) {
  as.data.frame(as.list(x), stringsAsFactors = FALSE)
}))

```

```

abg_cat_results <- dplyr::bind_rows(lapply(cat3_outcomes, function(outcome_var) {
  pool_terms(cat3_fits_abg[[outcome_var]], term_prefix = "co2_cat") |>
    dplyr::mutate(
      outcome = cat3_labels[[outcome_var]],
      group = "ABG",
      OR = estimate
    )
  }))
vbg_cat_results <- dplyr::bind_rows(lapply(cat3_outcomes, function(outcome_var) {
  pool_terms(cat3_fits_vbg[[outcome_var]], term_prefix = "co2_cat") |>
    dplyr::mutate(
      outcome = cat3_labels[[outcome_var]],
      group = "VBG",
      OR = estimate
    )
  }))
abg_spline <- list(
  curves = dplyr::bind_rows(lapply(cat3_outcomes, function(outcome_var) {
    pool_spline_curve(spline.fits_abg[[outcome_var]], grid_abg_mi, ref_idx_abg_mi, co2.ref_abg_mi) |>
      dplyr::mutate(outcome = outcome_var, group = "ABG")
  })),
  coefs = dplyr::bind_rows(lapply(cat3_outcomes, function(outcome_var) {
    pool_spline_coefs(spline.fits_abg[[outcome_var]]) |>
      dplyr::mutate(outcome = outcome_var, group = "ABG")
  }))
)
vbg_spline <- list(
  curves = dplyr::bind_rows(lapply(cat3_outcomes, function(outcome_var) {
    pool_spline_curve(spline.fits_vbg[[outcome_var]], grid_vbg_mi, ref_idx_vbg_mi, co2.ref_vbg_mi) |>
      dplyr::mutate(outcome = outcome_var, group = "VBG")
  })),
  coefs = dplyr::bind_rows(lapply(cat3_outcomes, function(outcome_var) {
    pool_spline_coefs(spline.fits_vbg[[outcome_var]]) |>
      dplyr::mutate(outcome = outcome_var, group = "VBG")
  }))
)

```

```

)
abg_curves <- abg_spline$curves
abg_coefs  <- abg_spline$coefs
vbg_curves <- vbg_spline$curves
vbg_coefs  <- vbg_spline$coefs

write_csv_safely(abg_curves, results_path("mi_spline_curve_abg.csv"), row_names = FALSE)
write_csv_safely(abg_coefs, results_path("mi_spline_coef_abg.csv"), row_names = FALSE)
write_csv_safely(vbg_curves, results_path("mi_spline_curve_vbg.csv"), row_names = FALSE)
write_csv_safely(vbg_coefs, results_path("mi_spline_coef_vbg.csv"), row_names = FALSE)

mi_single_pass_t1 <- Sys.time()
stopifnot(exists("runtime_log"))
runtime_log <- dplyr::bind_rows(
  runtime_log,
  data.frame(
    step_name = "mi_single_pass",
    seconds = as.numeric(difftime(mi_single_pass_t1, mi_single_pass_t0, units = "secs")),
    start_time = as.character(mi_single_pass_t0),
    end_time = as.character(mi_single_pass_t1),
    notes = paste0("m=", imp_n),
    run_id = runtime_run_id,
    run_mode = RUN_MODE,
    n_subset = nrow(subset_data),
    stringsAsFactors = FALSE
  )
)
mi_single_pass_done <- TRUE

```

3.3.2 ABG propensity (has_abg)

```

# Purpose: mi propensity abg.
stopifnot(exists("mi_single_pass_done"), isTRUE(mi_single_pass_done))
stopifnot(exists("mi_logistic_ps_abg_list"))

```

```
message("ABG MI weights were computed in the single-pass MI loop above.")
```

3.3.3 Balance diagnostics across imputations

```
# Purpose: target balance helpers.  
stopifnot(exists("target_balance_table"))  
  
# Use a fixed x-axis max so ABG/VBG balance plots are directly comparable.  
  
# Purpose: mi balance abg.  
stopifnot(exists("bal_imp_abg"))  
  
bal_abg_pooled <- bal_imp_abg$bal_long |>  
  mutate(label = ifelse(is.na(level), variable, paste0(variable, ":", level))) |>  
  group_by(label) |>  
  summarise(  
    pre_med = median(abs(smd_pre), na.rm = TRUE),  
    post_med = median(abs(smd_post), na.rm = TRUE),  
    pre_mean = mean(abs(smd_pre), na.rm = TRUE),  
    post_mean = mean(abs(smd_post), na.rm = TRUE),  
    post_max = max(abs(smd_post), na.rm = TRUE),  
    .groups = "drop"  
)  
  
bal_abg_plot <- bal_abg_pooled |>  
  mutate(label = factor(label, levels = label[order(post_med, decreasing = TRUE)])) |>  
  pivot_longer(c(pre_med, post_med), names_to = "type", values_to = "smd") |>  
  mutate(type = recode(type, pre_med = "Pre", post_med = "Post"))  
  
p_abg <- ggplot(bal_abg_plot, aes(x = smd, y = label, shape = type)) +  
  geom_vline(xintercept = 0.1, linetype = 2, linewidth = 0.3) +  
  geom_point(size = 1.2) +  
  labs(title = "MI target balance (ABG): pooled |SMD|", x = "|Target SMD|", y = NULL, shape = "Stage") +  
  scale_x_continuous(limits = c(0, BAL_XLIM_MAX),  
                     expand = expansion(mult = c(0, 0.02))) +  
  theme_minimal(base_size = 10)  
save_diag_plot(p_abg, results_path("figs", "diag-mi-balance-pooled-abg.png"), width = 7, height = 6)
```

```

render_table_pdf_maybe(
  bal_imp_abg$worst_rows_overall,
  caption = "ABG: target SMD rows across imputations (sorted by |SMD|)",
  file_stub = "abg_worst_target_smd_rows",
  digits = 3,
  show = SHOW_LOW_VALUE_TABLES
)

abg_imp_summary <- bal_imp_abg$bal_imp_summary |>
  summarise(
    med = median(max_abs_post, na.rm = TRUE),
    p90 = quantile(max_abs_post, 0.9, na.rm = TRUE),
    max = max(max_abs_post, na.rm = TRUE)
  )
  render_table_pdf_maybe(
    abg_imp_summary,
    caption = "ABG: max |Target SMD| summary across imputations",
    file_stub = "abg_target_smd_summary",
    digits = 3,
    show = SHOW_LOW_VALUE_TABLES
)

```

3.3.4 VBG propensity (has_vbg)

```

# Purpose: mi propensity vbg.
stopifnot(exists("mi_single_pass_done"), isTRUE(mi_single_pass_done))
stopifnot(exists("mi_logistic_ps_vbg_list"))
message("VBG MI weights were computed in the single-pass MI loop above.")

```

3.3.5 VBG balance

```

# Purpose: mi balance vbg.
stopifnot(exists("bal_imp_vbg"))

bal_vbg_pooled <- bal_imp_vbg$bal_long |>

```

```

mutate(label = ifelse(is.na(level), variable, paste0(variable, ":", level))) |>
group_by(label) |>
summarise(
  pre_med = median(abs(smd_pre), na.rm = TRUE),
  post_med = median(abs(smd_post), na.rm = TRUE),
  pre_mean = mean(abs(smd_pre), na.rm = TRUE),
  post_mean = mean(abs(smd_post), na.rm = TRUE),
  post_max = max(abs(smd_post), na.rm = TRUE),
  .groups = "drop"
)

bal_vbg_plot <- bal_vbg_pooled |>
  mutate(label = factor(label, levels = label[order(post_med, decreasing = TRUE)])) |>
  pivot_longer(c(pre_med, post_med), names_to = "type", values_to = "smd") |>
  mutate(type = recode(type, pre_med = "Pre", post_med = "Post"))

p_vbg <- ggplot(bal_vbg_plot, aes(x = smd, y = label, shape = type)) +
  geom_vline(xintercept = 0.1, linetype = 2, linewidth = 0.3) +
  geom_point(size = 1.2) +
  labs(title = "MI target balance (VBG): pooled |SMD|", x = "|Target SMD|", y = NULL, shape = "Stage") +
  scale_x_continuous(limits = c(0, BAL_XLIM_MAX),
                     expand = expansion(mult = c(0, 0.02))) +
  theme_minimal(base_size = 10)
save_diag_plot(p_vbg, results_path("figs", "diag-mi-balance-pooled-vbg.png"), width = 7, height = 6)

render_table_pdf_maybe(
  bal_imp_vbg$worst_rows_overall,
  caption = "VBG: target SMD rows across imputations (sorted by |SMD|)",
  file_stub = "vbg_worst_target_smd_rows",
  digits = 3,
  show = SHOW_LOW_VALUE_TABLES
)

vbg_imp_summary <- bal_imp_vbg$bal_imp_summary |>
  summarise(
    med = median(max_abs_post, na.rm = TRUE),
    p90 = quantile(max_abs_post, 0.9, na.rm = TRUE),

```

```

    max = max(max_abs_post, na.rm = TRUE)
  )
render_table_pdf_maybe(
  vbg_imp_summary,
  caption = "VBG: max |Target SMD| summary across imputations",
  file_stub = "vbg_target_smd_summary",
  digits = 3,
  show = SHOW_LOW_VALUE_TABLES
)

```

3.4 Weighted outcome models within each imputation + pooling

Within each imputation, fit covariate-adjusted CO₂ spline outcome models **only in the measured cohort** (has_abg==1 for PaCO₂; has_vbg==1 for VBG CO₂), using IPSW weights to address nonrandom testing. Curves are pooled pointwise across imputations (Rubin's rules on the log-OR scale) and displayed as odds ratios relative to CO₂_ref at a reference covariate profile.

3.4.1 Helper: fit + extract log-OR and SE from svyglm

```

# Purpose: mi pool helpers.
stopifnot(exists("fit_spline_imp"), exists("pool_spline_curve"),
          exists("pool_spline_coefs"), exists("pool_terms"))

# Purpose: mi abg outcomes.
stopifnot(exists("mi_single_pass_done"), isTRUE(mi_single_pass_done))
stopifnot(exists("abg_spline"), exists("abg_curves"), exists("abg_coefs"))
message("ABG MI spline results were computed in the single-pass MI loop above.")

```

3.4.2 VBG: MI pooled spline models (treated cohort only)

```

# Purpose: mi vbg outcomes.
stopifnot(exists("mi_single_pass_done"), isTRUE(mi_single_pass_done))
stopifnot(exists("vbg_spline"), exists("vbg_curves"), exists("vbg_coefs"))
message("VBG MI spline results were computed in the single-pass MI loop above.")

```

3.5 Explainability

```
# Purpose: shap top10 mi logistic.  
# MI logistic models are linear on the log-odds scale, so we compute  
# contribution magnitudes from centered design-matrix terms and aggregate  
# to parent features.  
stopifnot(exists("mi_logistic_ps_abg_list"), exists("mi_logistic_ps_vbg_list"))  
  
mi_shap_abg_file <- results_path("shap_top10_mi_logistic_abg.csv")  
mi_shap_vbg_file <- results_path("shap_top10_mi_logistic_vbg.csv")  
mi_shap_stability_file <- results_path("diag-ps-shap-stability.csv")  
mi_shap_error_file <- results_path("diag_mi_shap_errors.csv")  
mi_shap_method_file <- results_path("diag_mi_shap_method_used.csv")  
mi_shap_diag_file <- results_path("mi_shap_imp_diagnostics.csv")  
  
make_empty_mi_shap <- function() {  
  data.frame(feature = character(), mean_abs_shap = numeric(), stringsAsFactors = FALSE)  
}  
  
calc_mi_glm_contrib <- function(meta, d_ps, feature_names, imp_index, group_label,  
                                   sample_n = MI_SHAP_SAMPLE_N,  
                                   max_terms = MI_SHAP_MAX_TERMS) {  
  # Memory-safe contribution approximation for GLM PS:  
  # sample rows first, build sparse model matrix, then aggregate |contribution|  
  # by feature without constructing a full-N dense matrix.  
  diag_row <- data.frame(  
    imp = imp_index,  
    group = group_label,  
    sample_n = NA_integer_,  
    n_terms_used = NA_integer_,  
    seconds = NA_real_,  
    vcells_pre_mb = NA_real_,  
    vcells_post_mb = NA_real_,  
    method_used = "sparse_sampled_glm_link",  
    ok = FALSE,  
    error_msg = NA_character_,  
    stringsAsFactors = FALSE
```

```

)
t0 <- Sys.time()
mem_pre <- get_vcells_stats()
diag_row$vcells_pre_mb <- mem_pre$gc_vcells_used_mb

err_msg <- NA_character_
out <- tryCatch({
  if (is.null(meta$formula) || is.null(meta$coef) || !length(meta$coef)) {
    stop("Missing formula/coefficients in MI PS metadata.")
  }

  n_all <- nrow(d_ps)
  n_use <- min(as.integer(sample_n), n_all)
  if (!is.finite(n_use) || n_use < 2L) {
    stop("Insufficient rows available for MI SHAP sample.")
  }
  seed_offset <- if (toupper(group_label) == "ABG") 10000L else 20000L
  set.seed(20251206L + as.integer(imp_index) + seed_offset)
  idx <- if (n_all > n_use) sample.int(n_all, size = n_use, replace = FALSE) else seq_len(n_all)
  d_sample <- d_ps[idx, , drop = FALSE]
  diag_row$sample_n <- n_use

  form <- stats::as.formula(meta$formula)
  mm_model <- NULL
  mm_mode <- "dense_model_matrix"
  if (requireNamespace("Matrix", quietly = TRUE)) {
    mm_sparse_try <- tryCatch(
      Matrix::sparse.model.matrix(form, data = d_sample),
      error = function(e) NULL
    )
    if (!is.null(mm_sparse_try)) {
      mm_model <- mm_sparse_try
      mm_mode <- "sparse_sampled_glm_link"
    }
  }
  if (is.null(mm_model)) {
    mm_model <- stats::model.matrix(form, data = d_sample)
  }
})

```

```

    mm_mode <- "dense_sampled_glm_link"
}
diag_row$method_used <- mm_mode

coef_vec <- meta$coef
canon_term <- function(x) {
  x <- gsub(``, "", x, fixed = TRUE)
  gsub("\\\\s+", "", x, perl = TRUE)
}
mm_terms <- colnames(mm_model)
coef_terms <- names(coef_vec)
mm_key <- canon_term(mm_terms)
coef_key <- canon_term(coef_terms)
coef_pos <- setNames(seq_along(coef_terms), coef_key)
keep_mm <- which(mm_key %in% names(coef_pos) & mm_terms != "(Intercept)")
if (!length(keep_mm)) {
  stop("No overlapping terms between sparse design matrix and coefficient vector.")
}

mm_model <- mm_model[, keep_mm, drop = FALSE]
diag_row$n_terms_used <- ncol(mm_model)
if (is.finite(max_terms) && ncol(mm_model) > as.integer(max_terms)) {
  stop(
    "MI SHAP term count (", ncol(mm_model), ") exceeded MI_SHAP_MAX_TERMS (",
    as.integer(max_terms), ")."
  )
}

mm_terms_kept <- colnames(mm_model)
mm_keys_kept <- canon_term(mm_terms_kept)
coef_idx <- coef_pos[mm_keys_kept]
beta <- as.numeric(coef_vec[coef_idx])
names(beta) <- mm_terms_kept
mu <- colMeans(mm_model, na.rm = TRUE)
abs_beta <- abs(beta)
term_abs <- vapply(
  seq_len(ncol(mm_model)),

```

```

function(j) {
  xj <- mm_model[, j]
  if (!is.numeric(xj)) xj <- as.numeric(xj)
  mean(abs(xj - mu[j]), na.rm = TRUE) * abs_beta[j]
},
numeric(1)
)
names(term_abs) <- colnames(mm_model)
rm(mm_model, mu, abs_beta)
invisible(gc())

out_tbl <- data.frame(
  term = names(term_abs),
  mean_abs = as.numeric(term_abs),
  stringsAsFactors = FALSE
) |>
  dplyr::mutate(feature = vapply(term, term_to_parent_feature, character(1), feature_names = feature_names)) |>
  dplyr::filter(!is.na(feature)) |>
  dplyr::group_by(feature) |>
  dplyr::summarise(mean_abs_shap = sum(mean_abs, na.rm = TRUE), .groups = "drop")
if (!nrow(out_tbl)) {
  stop("No feature-mapped terms remained after contribution aggregation.")
}
out_tbl
}, error = function(e) {
  err_msg <- conditionMessage(e)
  make_empty_mi_shap()
})

mem_post <- get_vcells_stats()
diag_row$vcells_post_mb <- mem_post$gc_vcells_used_mb
diag_row$seconds <- as.numeric(difftime(Sys.time(), t0, units = "secs"))
diag_row$ok <- nrow(out) > 0
if (!isTRUE(diag_row$ok) && is.na(diag_row$error_msg)) {
  diag_row$error_msg <- err_msg
}

```

```

    list(data = out, diag = diag_row)
}

calc_mi_coefFallback <- function(meta_list, feature_names, group_label, top_n = SHAP_TOP_N) {
  # Fallback ranking using absolute coefficients if contribution matrix
  # cannot be computed (keeps figure/table generation robust).
  coef_rows <- lapply(seq_along(meta_list), function(i) {
    coef_vec <- meta_list[[i]]$coef
    if (is.null(coef_vec) || !length(coef_vec)) return(NULL)
    data.frame(
      imp = i,
      term = names(coef_vec),
      mean_abs = abs(as.numeric(coef_vec)),
      stringsAsFactors = FALSE
    )
  })
  coef_df <- dplyr::bind_rows(coef_rows)
  if (!nrow(coef_df)) {
    return(data.frame(feature = character(), mean_abs_shap = numeric(), group = character(), stringsAsFactors =
      FALSE))
  }
  coef_df |>
    dplyr::filter(term != "(Intercept)") |>
    dplyr::mutate(feature = vapply(term, term_to_parent_feature, character(1), feature_names = feature_names)) |>
    dplyr::filter(!is.na(feature)) |>
    dplyr::group_by(feature) |>
    dplyr::summarise(mean_abs_shap = mean(mean_abs, na.rm = TRUE), .groups = "drop") |>
    dplyr::arrange(dplyr::desc(mean_abs_shap)) |>
    dplyr::slice_head(n = top_n) |>
    dplyr::mutate(group = group_label)
}

if (RUN_SHAP) {
  # Streaming accumulator avoids storing per-imputation SHAP tables in memory.
  update_feature_accum <- function(accum, contrib_df) {
    if (!nrow(contrib_df)) return(accum)
    vals <- stats::setNames(contrib_df$mean_abs_shap, contrib_df$feature)
}

```

```

feats <- names(vals)
new_feats <- setdiff(feats, names(accum$sum))
if (length(new_feats)) {
  accum$sum[new_feats] <- 0
  accum$count[new_feats] <- 0
}
accum$sum[feats] <- accum$sum[feats] + vals
accum$count[feats] <- accum$count[feats] + 1L
accum
}
finalize_feature_accum <- function(accum, group_label) {
  if (!length(accum$sum)) {
    return(data.frame(feature = character(), mean_abs_shap = numeric(), group = character(), stringsAsFactors =
      FALSE))
  }
  keep <- accum$count > 0
  data.frame(
    feature = names(accum$sum)[keep],
    mean_abs_shap = as.numeric(accum$sum[keep] / accum$count[keep]),
    group = group_label,
    stringsAsFactors = FALSE
  ) |>
    dplyr::arrange(dplyr::desc(mean_abs_shap)) |>
    dplyr::slice_head(n = SHAP_TOP_N)
}

accum_abg <- list(sum = numeric(0), count = numeric(0))
accum_vbg <- list(sum = numeric(0), count = numeric(0))
mi_shap_diag_rows <- list()
mi_shap_error_rows <- list()

for (i in seq_len(imp$m)) {
  d_i <- get_imp(i)
  d_ps <- d_i[, c("has_abg", "has_vbg", "covars_ps"), drop = FALSE]
  d_ps <- normalize_types(d_ps, levels_ref)
  d_ps <- droplevels_all(d_ps)
}

```

```

abg_res <- calc_mi_glm_contrib(
  mi_logistic_ps_abg_list[[i]], d_ps, covars_ps,
  imp_index = i, group_label = "ABG",
  sample_n = MI_SHAP_SAMPLE_N, max_terms = MI_SHAP_MAX_TERMS
)
mi_shap_diag_rows[[length(mi_shap_diag_rows) + 1L]] <- abg_res$diag
if (!isTRUE(abg_res$diag$ok)) {
  mi_shap_error_rows[[length(mi_shap_error_rows) + 1L]] <- data.frame(
    imp = i, group = "ABG", error_msg = abg_res$diag$error_msg,
    stringsAsFactors = FALSE
  )
}
accum_abg <- update_feature_accum(accum_abg, abg_res$data)

vbg_res <- calc_mi_glm_contrib(
  mi_logistic_ps_vbg_list[[i]], d_ps, covars_ps,
  imp_index = i, group_label = "VBG",
  sample_n = MI_SHAP_SAMPLE_N, max_terms = MI_SHAP_MAX_TERMS
)
mi_shap_diag_rows[[length(mi_shap_diag_rows) + 1L]] <- vbg_res$diag
if (!isTRUE(vbg_res$diag$ok)) {
  mi_shap_error_rows[[length(mi_shap_error_rows) + 1L]] <- data.frame(
    imp = i, group = "VBG", error_msg = vbg_res$diag$error_msg,
    stringsAsFactors = FALSE
  )
}
accum_vbg <- update_feature_accum(accum_vbg, vbg_res$data)

rm(d_i, d_ps, abg_res, vbg_res)
invisible(gc())
}

mi_shap_top_abg <- finalize_feature_accum(accum_abg, "ABG")
mi_shap_top_vbg <- finalize_feature_accum(accum_vbg, "VBG")

used_method_abg <- "sparse_sampled_glm_link"
used_method_vbg <- "sparse_sampled_glm_link"

```

```

if (nrow(mi_shap_top_abg) == 0L && isTRUE(MI_SHAP_FAIL_OPEN)) {
  mi_shap_top_abg <- calc_mi_coef_fallback(mi_logistic_ps_abg_list, covars_ps, "ABG")
  used_method_abg <- "coef_fallback"
}
if (nrow(mi_shap_top_vbg) == 0L && isTRUE(MI_SHAP_FAIL_OPEN)) {
  mi_shap_top_vbg <- calc_mi_coef_fallback(mi_logistic_ps_vbg_list, covars_ps, "VBG")
  used_method_vbg <- "coef_fallback"
}

mi_shap_diag_df <- dplyr::bind_rows(mi_shap_diag_rows)
mi_shap_error_df <- dplyr::bind_rows(mi_shap_error_rows)
infer_primary_method <- function(diag_df, grp, fallback_method) {
  if (identical(fallback_method, "coef_fallback")) return("coef_fallback")
  diag_grp <- diag_df |>
    dplyr::filter(group == grp, ok %in% TRUE, !is.na(method_used), nzchar(method_used))
  if (!nrow(diag_grp)) return(fallback_method)
  names(sort(table(diag_grp$method_used), decreasing = TRUE))[1]
}
used_method_abg <- infer_primary_method(mi_shap_diag_df, "ABG", used_method_abg)
used_method_vbg <- infer_primary_method(mi_shap_diag_df, "VBG", used_method_vbg)
mi_shap_method_df <- data.frame(
  group = c("ABG", "VBG"),
  method_used = c(used_method_abg, used_method_vbg),
  n_success = c(
    sum(mi_shap_diag_df$group == "ABG" & (mi_shap_diag_df$ok %in% TRUE), na.rm = TRUE),
    sum(mi_shap_diag_df$group == "V ро" & (mi_shap_diag_df$ok %in% TRUE), na.rm = TRUE)
  ),
  fallback_used = c(used_method_abg == "coef_fallback", used_method_vbg == "coef_fallback"),
  stringsAsFactors = FALSE
)
write_csv_safely(mi_shap_diag_df, mi_shap_diag_file, row_names = FALSE)
write_csv_safely(mi_shap_error_df, mi_shap_error_file, row_names = FALSE)
write_csv_safely(mi_shap_method_df, mi_shap_method_file, row_names = FALSE)
write_csv_safely(mi_shap_top_abg, mi_shap_abg_file, row_names = FALSE)
write_csv_safely(mi_shap_top_vbg, mi_shap_vbg_file, row_names = FALSE)

```

```

mi_shap_stability <- dplyr::bind_rows(mi_shap_top_abg, mi_shap_top_vbg) |>
  dplyr::transmute(
    cohort = group,
    feature = feature,
    mean_abs_seed1 = mean_abs_shap,
    mean_abs_seed2 = mean_abs_shap,
    run_id = diag_run_id
  )
write_csv_safely(mi_shap_stability, mi_shap_stability_file, row_names = FALSE)

mi_shap_plot_df <- dplyr::bind_rows(mi_shap_top_abg, mi_shap_top_vbg)
if (nrow(mi_shap_plot_df) > 0) {
  p_shap_mi <- plot_shap_top10_two_panel(
    mi_shap_plot_df,
    "MI logistic IPSW: top 10 features by mean absolute contribution",
    "Mean absolute contribution on link scale"
  )
  shap_file <- results_path("figs", "shap-top10-mi-logistic-abg-vbg.png")
  save_diag_plot(p_shap_mi, shap_file, width = 9, height = 5, dpi = 200)
  register_plot_file("shap-top10-mi-logistic-abg-vbg", shap_file)
}

rm(
  accum_abg, accum_vbg, mi_shap_diag_rows, mi_shap_error_rows,
  mi_shap_diag_df, mi_shap_error_df, mi_shap_method_df,
  mi_shap_plot_df, mi_shap_stability, mi_shap_top_abg, mi_shap_top_vbg
)
invisible(gc())
} else {
  write_csv_safely(data.frame(), mi_shap_diag_file, row_names = FALSE)
  write_csv_safely(data.frame(), mi_shap_error_file, row_names = FALSE)
  write_csv_safely(data.frame(), mi_shap_method_file, row_names = FALSE)
  write_csv_safely(data.frame(), mi_shap_abg_file, row_names = FALSE)
  write_csv_safely(data.frame(), mi_shap_vbg_file, row_names = FALSE)
  write_csv_safely(data.frame(), mi_shap_stability_file, row_names = FALSE)
}

```

3.6 MI three-level PCO2 helpers and checks

```
# Purpose: chunk.  
stopifnot(exists("pool_terms"))  
message("MI 3-level helpers defined in the single-pass MI section.")
```

3.7 MI + IPW three-level PCO2 (ABG & VBG)

3.7.1 ABG: MI + IPW, three-level PCO2 outcomes

```
# Purpose: chunk.  
stopifnot(exists("mi_single_pass_done"), isTRUE(mi_single_pass_done))  
stopifnot(exists("abg_cat_results"))  
  
mi_abg_cat_forms <- list(  
  "MI IPW ABG 3-level: IMV ~ CO2 category + X"      = reformulate(c("co2_cat", adj_core), response = "imv_proc"),  
  "MI IPW ABG 3-level: NIV ~ CO2 category + X"      = reformulate(c("co2_cat", adj_core), response = "niv_proc"),  
  "MI IPW ABG 3-level: Death60d ~ CO2 category + X" = reformulate(c("co2_cat", adj_core), response = "death_60d"),  
  "MI IPW ABG 3-level: HCRF ~ CO2 category + X"     = reformulate(c("co2_cat", adj_core), response =  
    ↪ "hypercap_resp_failure")  
)  
register_model_diagrams(mi_abg_cat_forms)  
message("ABG MI 3-level results were computed in the single-pass MI loop above.")
```

3.7.2 VBG: MI + IPW, three-level PCO2 outcomes

```
# Purpose: chunk.  
stopifnot(exists("mi_single_pass_done"), isTRUE(mi_single_pass_done))  
stopifnot(exists("vbg_cat_results"))  
  
mi_vbg_cat_forms <- list(  
  "MI IPW VBG 3-level: IMV ~ CO2 category + X"      = reformulate(c("co2_cat", adj_core), response = "imv_proc"),  
  "MI IPW VBG 3-level: NIV ~ CO2 category + X"      = reformulate(c("co2_cat", adj_core), response = "niv_proc"),  
  "MI IPW VBG 3-level: Death60d ~ CO2 category + X" = reformulate(c("co2_cat", adj_core), response = "death_60d"),  
  "MI IPW VBG 3-level: HCRF ~ CO2 category + X"     = reformulate(c("co2_cat", adj_core), response =  
    ↪ "hypercap_resp_failure")
```

```
)
register_model_diagrams(mi_vbg_cat_forms)
message("VBG MI 3-level results were computed in the single-pass MI loop above.")
```

3.7.3 MI-pooled IPW associations (3-level CO2)

```
# Purpose: table3 ipw mi co2.
stopifnot(exists("abg_cat_results"), exists("vbg_cat_results"))

mi_threoplevel_results <- dplyr::bind_rows(
  dplyr::mutate(abg_cat_results, group = "ABG"),
  dplyr::mutate(vbg_cat_results, group = "VBG")
) |>
  dplyr::mutate(method = "IPW + MI adjusted")

table3_df <- dplyr::bind_rows(
  dplyr::mutate(abg_cat_results, group = "ABG"),
  dplyr::mutate(vbg_cat_results, group = "VBG")
) |>
  dplyr::mutate(
    exposure = gsub("^co2_cat", "", term),
    contrast = dplyr::recode(exposure,
      Low = "Low vs normal",
      High = "High vs normal",
      .default = exposure),
    outcome_label = dplyr::recode(
      outcome,
      IMV = "IMV",
      NIV = "NIV",
      Death60d = "Death (60d)",
      HCRF = "Hypercapnic RF"
    ),
    or_ci = sprintf("%.2f (%.2f, %.2f)", OR, LCL, UCL)
) |>
  dplyr::select(group, outcome_label, contrast, or_ci)
```

```

table3_wide <- table3_df |>
  tidyverse::pivot_wider(names_from = contrast, values_from = or_ci) |>
  dplyr::arrange(
    factor(group, levels = c("ABG", "VBG")),
    factor(outcome_label, levels = c("IMV", "NIV", "Death (60d)", "Hypercapnic RF"))
  )

gt::gt(table3_wide) |>
  gt::tab_header(title = "MI-pooled IPW associations between CO2 category and outcomes (adjusted)") |>
  gt::cols_label(
    group = "Cohort",
    outcome_label = "Outcome",
    `Low vs normal` = "Low vs normal OR (95% CI)",
    `High vs normal` = "High vs normal OR (95% CI)"
  ) |>
  gt::fmt_missing(columns = gt::everything(), missing_text = "-") |>
  gt::tab_source_note(
    paste0(
      "Weighted survey GLMs adjusted for baseline covariates; weights = MI-specific GLM (RCS) IPW; m = ", M_IMP,
      " imputations (seed ", MI_SEED, "); reference = Normal."
    )
  )
)

```

3.7.4 Summary: adjusted CO2-category associations across analysis tracks

```

# Purpose: table summary adjusted threelevel.
# Build a unified comparison table across three analysis tracks:
# Unweighted adjusted, IPW adjusted, and IPW + MI adjusted.
label_outcome <- function(x) {
  dplyr::recode(
    x,
    imv_proc = "IMV",
    niv_proc = "NIV",
    death_60d = "Death (60d)",
    hypercap_resp_failure = "Hypercapnic RF",
    IMV = "IMV",

```

MI-pooled IPW associations between CO2 category and outcomes (adjusted)

Cohort	Outcome	Low vs normal OR (95% CI)	High vs normal OR (95% CI)
ABG	IMV	1.41 (1.04, 1.90)	1.26 (0.93, 1.72)
ABG	NIV	1.04 (0.63, 1.74)	3.05 (1.98, 4.70)
ABG	Death (60d)	1.57 (1.09, 2.26)	1.52 (1.07, 2.15)
ABG	Hypercapnic RF	1.28 (0.71, 2.31)	9.24 (5.76, 14.83)
VBG	IMV	1.26 (0.78, 2.04)	1.64 (1.04, 2.59)
VBG	NIV	0.78 (0.39, 1.59)	2.71 (1.47, 4.98)
VBG	Death (60d)	1.08 (0.67, 1.74)	1.11 (0.70, 1.77)
VBG	Hypercapnic RF	0.71 (0.31, 1.63)	6.36 (3.51, 11.51)

Weighted survey GLMs adjusted for baseline covariates; weights = MI-specific GLM (RCS) IPW; m = 80 imputations (seed 20251206); reference = Normal.

```

NIV = "NIV",
Death60d = "Death (60d)",
HCRF = "Hypercapnic RF"
)
}

map_contrast <- function(term) {
  dplyr::case_when(
    grepl("Low", term) ~ "Low vs normal",
    grepl("High", term) ~ "High vs normal",
    TRUE ~ NA_character_
  )
}

prep_threelvel <- function(df, estimate_col, lcl_col, ucl_col, method_label) {
  # Standardize column names from each model output into a common schema.
  df |>
    dplyr::mutate(
      contrast = map_contrast(term),
      outcome_label = label_outcome(outcome),

```

```

    OR  = .data[[estimate_col]],
    LCL = .data[[lcl_col]],
    UCL = .data[[ucl_col]],
    method = method_label
) |>
dplyr::filter(!is.na(contrast)) |>
dplyr::select(method, group, outcome_label, contrast, OR, LCL, UCL)
}

count_model <- function(data, outcome, exposure, adj_vars,
                        treat_var = NULL, weight_var = NULL) {
# Counts complete-case model rows/events under each analysis variant.
d <- data
if (!is.null(treat_var)) d <- d[d[[treat_var]] == 1, , drop = FALSE]
vars <- c(outcome, exposure, adj_vars)
if (!is.null(weight_var)) vars <- c(vars, weight_var)
d <- d[, vars, drop = FALSE]
d <- d[complete.cases(d), , drop = FALSE]
if (!is.null(weight_var)) {
  w <- suppressWarnings(as.numeric(d[[weight_var]]))
  d <- d[is.finite(w) & w > 0, , drop = FALSE]
}
if (nrow(d) == 0L) {
  return(tibble::tibble(n_model = 0L, events = 0L))
}
d[[outcome]] <- to01(d[[outcome]])
tibble::tibble(
  n_model = nrow(d),
  events = sum(d[[outcome]] == 1, na.rm = TRUE)
)
}

count_group <- function(group, method) {
  exp_var   <- if (group == "ABG") "pco2_cat_abg" else "pco2_cat_vbg"
  treat_var <- if (group == "ABG") "has_abg" else "has_vbg"
  weight_var <- if (method == "IPW adjusted") {
    if (group == "ABG") "w_abg" else "w_vbg"

```

```

} else {
  NULL
}
outcomes <- c("imv_proc", "niv_proc", "death_60d", "hypercap_resp_failure")
purrr::map_dfr(outcomes, function(outcome) {
  cnt <- count_model(subset_data, outcome, exp_var, adj_core,
                      treat_var = treat_var, weight_var = weight_var)
  dplyr::mutate(cnt,
                method = method,
                group = group,
                outcome_label = label_outcome(outcome))
})
}

mi_counts_threelvel <- function(outcome, group) {
  # For MI models, report median sample size/events across imputations.
  imp_n <- imp$m
  counts <- lapply(seq_len(imp_n), function(i) {
    d <- get_imp(i)
    if (group == "ABG") {
      co2_var <- "paco2"
      treat_var <- "has_abg"
      w <- get_mi_weight("ABG", i)
      low_cut <- ABG_CO2_LOW
      high_cut <- ABG_CO2_HIGH
    } else {
      co2_var <- "vbg_co2"
      treat_var <- "has_vbg"
      w <- get_mi_weight("VBG", i)
      low_cut <- VBG_CO2_LOW
      high_cut <- VBG_CO2_HIGH
    }
    d[[co2_var]] <- suppressWarnings(as.numeric(d[[co2_var]]))
    keep <- d[[treat_var]] == 1 & is.finite(d[[co2_var]])
    if (!any(keep)) return(c(n_model = 0, events = 0))
    d2 <- d[keep, , drop = FALSE]
    w2 <- w[keep]
  })
}

```

```

d2$co2_cat <- make_co2_cat3(d2[[co2_var]], low_cut, high_cut)
keep2 <- !is.na(d2$co2_cat)
d2 <- d2[keep2, , drop = FALSE]
w2 <- w2[keep2]
if (nrow(d2) == 0L) return(c(n_model = 0, events = 0))
d2[[outcome]] <- to01(d2[[outcome]])
complete_ok <- complete.cases(d2[, c(outcome, adj_core, "co2_cat")], drop = FALSE)
w_ok <- is.finite(w2) & w2 > 0
d2 <- d2[complete_ok & w_ok, , drop = FALSE]
if (nrow(d2) == 0L) return(c(n_model = 0, events = 0))
c(n_model = nrow(d2), events = sum(d2[[outcome]] == 1, na.rm = TRUE))
})

n_vals <- vapply(counts, `[[`, numeric(1), "n_model")
e_vals <- vapply(counts, `[[`, numeric(1), "events")
tibble::tibble(
  n_model = round(stats::median(n_vals, na.rm = TRUE)),
  events = round(stats::median(e_vals, na.rm = TRUE))
)
}

mi_counts <- function(group) {
  outcomes <- c("imv_proc", "niv_proc", "death_60d", "hypercap_resp_failure")
  purrr::map_dfr(outcomes, function(outcome) {
    cnt <- mi_counts_threelevel(outcome, group)
    dplyr::mutate(cnt,
      method = "IPW + MI adjusted",
      group = group,
      outcome_label = label_outcome(outcome))
  })
}

counts_tbl <- dplyr::bind_rows(
  count_group("ABG", "Unweighted adjusted"),
  count_group("VBG", "Unweighted adjusted"),
  count_group("ABG", "IPW adjusted"),
  count_group("VBG", "IPW adjusted"),
  mi_counts("ABG"),

```

```

    mi_counts("VBG")
)

or_all <- dplyr::bind_rows(
  prep_threelvel(unw_threelvel_results, "estimate", "conf.low", "conf.high", "Unweighted adjusted"),
  prep_threelvel(ipw_threelvel_results, "estimate", "conf.low", "conf.high", "IPW adjusted"),
  prep_threelvel(mi_threelvel_results, "OR", "LCL", "UCL", "IPW + MI adjusted")
) |>
  dplyr::left_join(counts_tbl, by = c("method", "group", "outcome_label"))

table_summary <- or_all |>
  dplyr::mutate(or_ci = sprintf("%.2f (%.2f, %.2f)", OR, LCL, UCL)) |>
  dplyr::select(method, group, outcome_label, n_model, events, contrast, or_ci) |>
  tidyr::pivot_wider(names_from = contrast, values_from = or_ci) |>
  dplyr::arrange(
    factor(method, levels = c("Unweighted adjusted", "IPW adjusted", "IPW + MI adjusted")),
    factor(group, levels = c("ABG", "VBG")),
    factor(outcome_label, levels = c("IMV", "NIV", "Death (60d)", "Hypercapnic RF"))
  )

write_csv_safely(table_summary, results_path("table_summary_adjusted_threelvel.csv"), row_names = FALSE)

table_summary_abg <- table_summary |>
  dplyr::filter(group == "ABG")
table_summary_vbg <- table_summary |>
  dplyr::filter(group == "VBG")

render_table_pdf(
  table_summary_abg,
  caption = "Adjusted odds ratios (low/high vs normal) across analysis tracks for ABG cohort; n/events reflect model
  ↵ sample size (median across imputations for MI).",
  file_stub = "table_summary_adjusted_threelvel_abg",
  digits = 2
)

```

Table 12: Adjusted odds ratios (low/high vs normal) across analysis tracks for ABG cohort; n/events reflect model sample size (median across imputations for MI). (Part A)

method	group	outcome_label	n_model	events	Low vs normal
Unweighted adjusted	ABG	IMV	1911	463	1.39 (1.07, 1.80)
Unweighted adjusted	ABG	NIV	1911	183	0.99 (0.64, 1.52)
Unweighted adjusted	ABG	Death (60d)	1911	322	1.76 (1.29, 2.39)
Unweighted adjusted	ABG	Hypercapnic RF	1911	206	1.37 (0.81, 2.30)
IPW adjusted	ABG	IMV	1911	463	1.40 (1.06, 1.86)
IPW adjusted	ABG	NIV	1911	183	1.04 (0.65, 1.69)
IPW adjusted	ABG	Death (60d)	1911	322	1.52 (1.08, 2.16)
IPW adjusted	ABG	Hypercapnic RF	1911	206	1.54 (0.84, 2.82)
IPW + MI adjusted	ABG	IMV	1911	463	1.41 (1.04, 1.90)
IPW + MI adjusted	ABG	NIV	1911	183	1.04 (0.63, 1.74)
IPW + MI adjusted	ABG	Death (60d)	1911	322	1.57 (1.09, 2.26)
IPW + MI adjusted	ABG	Hypercapnic RF	1911	206	1.28 (0.71, 2.31)

Table 13: Adjusted odds ratios (low/high vs normal) across analysis tracks for ABG cohort; n/events reflect model sample size (median across imputations for MI). (Part B)

High vs normal
1.25 (0.96, 1.61)
2.62 (1.83, 3.78)
1.54 (1.14, 2.08)
8.50 (5.76, 12.89)
1.14 (0.85, 1.52)
2.82 (1.90, 4.17)
1.46 (1.05, 2.02)
9.39 (5.99, 14.73)
1.26 (0.93, 1.72)
3.05 (1.98, 4.70)
1.52 (1.07, 2.15)
9.24 (5.76, 14.83)

```
render_table_pdf(  
  table_summary_vbg,  
  caption = "Adjusted odds ratios (low/high vs normal) across analysis tracks for VBG cohort; n/events reflect model  
  ↵ sample size (median across imputations for MI).",  
  file_stub = "table_summary_adjusted_threellevel_vbg",  
  digits = 2  
)
```

Table 14: Adjusted odds ratios (low/high vs normal) across analysis tracks for VBG cohort; n/events reflect model sample size (median across imputations for MI). (Part A)

method	group	outcome_label	n_model	events	Low vs normal
Unweighted adjusted	VBG	IMV	1445	208	1.27 (0.87, 1.85)
Unweighted adjusted	VBG	NIV	1445	100	1.00 (0.56, 1.77)
Unweighted adjusted	VBG	Death (60d)	1445	200	1.53 (1.04, 2.25)
Unweighted adjusted	VBG	Hypercapnic RF	1445	133	0.84 (0.43, 1.58)
IPW adjusted	VBG	IMV	1445	208	1.34 (0.89, 2.03)
IPW adjusted	VBG	NIV	1445	100	0.88 (0.47, 1.67)
IPW adjusted	VBG	Death (60d)	1445	200	1.38 (0.91, 2.10)
IPW adjusted	VBG	Hypercapnic RF	1445	133	0.76 (0.37, 1.56)
IPW + MI adjusted	VBG	IMV	1445	208	1.26 (0.78, 2.04)
IPW + MI adjusted	VBG	NIV	1445	100	0.78 (0.39, 1.59)
IPW + MI adjusted	VBG	Death (60d)	1445	200	1.08 (0.67, 1.74)
IPW + MI adjusted	VBG	Hypercapnic RF	1445	133	0.71 (0.31, 1.63)

Table 15: Adjusted odds ratios (low/high vs normal) across analysis tracks for VBG cohort; n/events reflect model sample size (median across imputations for MI). (Part B)

High vs normal
1.61 (1.11, 2.33)
2.80 (1.71, 4.68)
1.29 (0.87, 1.89)
7.09 (4.49, 11.55)
1.58 (1.06, 2.37)
2.77 (1.62, 4.75)
1.31 (0.87, 1.98)
7.58 (4.54, 12.67)
1.64 (1.04, 2.59)
2.71 (1.47, 4.98)
1.11 (0.70, 1.77)
6.36 (3.51, 11.51)

```

# Purpose: stage1 cleanup.
append_mem_snapshot("stage1", "end", "post")
stage1_rm <- c(
  "unw_results_crude", "unw_results_adj", "unw_threellevel_results",
  "unw_combined_or_df", "unw_plot_df", "unw_p_or", "unw_axis_spec", "outcomes_unw"
)
missing_stage1 <- setdiff(stage1_rm, ls())
stopifnot(length(missing_stage1) == 0)
rm(list = stage1_rm)
invisible(gc())
append_mem_snapshot("stage1", "cleanup", "post")

```

3.8 Observed outcome rates by CO2 bin (MI + IPSW)

```

# Purpose: pool weighted 1-mmHg bin rates across imputations using summary-only objects.
stopifnot(exists("imp"), imp$m > 0)
stopifnot(exists("get_imp"), exists("get_mi_weight"))

co2_rate_mi_rows_abg <- vector("list", imp$m)
co2_rate_mi_rows_vbg <- vector("list", imp$m)

for (i in seq_len(imp$m)) {
  d_i <- get_imp(i)
  w_abg_i <- get_mi_weight("ABG", i)
  w_vbg_i <- get_mi_weight("VBG", i)

  d_abg_i <- d_i[, c("has_abg", "paco2", CO2_RATE_OUTCOMES), drop = FALSE]
  d_abg_i$w_rate <- w_abg_i
  d_vbg_i <- d_i[, c("has_vbg", "vbg_co2", CO2_RATE_OUTCOMES), drop = FALSE]
  d_vbg_i$w_rate <- w_vbg_i

  co2_rate_mi_rows_abg[[i]] <- compute_co2_bin_rates(
    df = d_abg_i |> dplyr::filter(has_abg == 1),
    co2_var = "paco2",
    outcomes = CO2_RATE_OUTCOMES,
    outcome_labels = CO2_RATE_OUTCOME_LABELS,
  )
}

```

```

weight_var = "w_rate",
q = c(0.02, 0.98),
bin_width = 1,
cohort_label = "ABG"
) |> dplyr::mutate(imp = i)

co2_rate_mi_rows_vbg[[i]] <- compute_co2_bin_rates(
  df = d_vbg_i |> dplyr::filter(has_vbg == 1),
  co2_var = "vbg_co2",
  outcomes = CO2_RATE_OUTCOMES,
  outcome_labels = CO2_RATE_OUTCOME_LABELS,
  weight_var = "w_rate",
  q = c(0.02, 0.98),
  bin_width = 1,
  cohort_label = "VBG"
) |> dplyr::mutate(imp = i)

rm(d_i, d_abg_i, d_vbg_i, w_abg_i, w_vbg_i)
invisible(gc(FALSE))
}

co2_rate_mi_abg <- pool_mi_bin_rates(dplyr::bind_rows(co2_rate_mi_rows_abg))
co2_rate_mi_vbg <- pool_mi_bin_rates(dplyr::bind_rows(co2_rate_mi_rows_vbg))

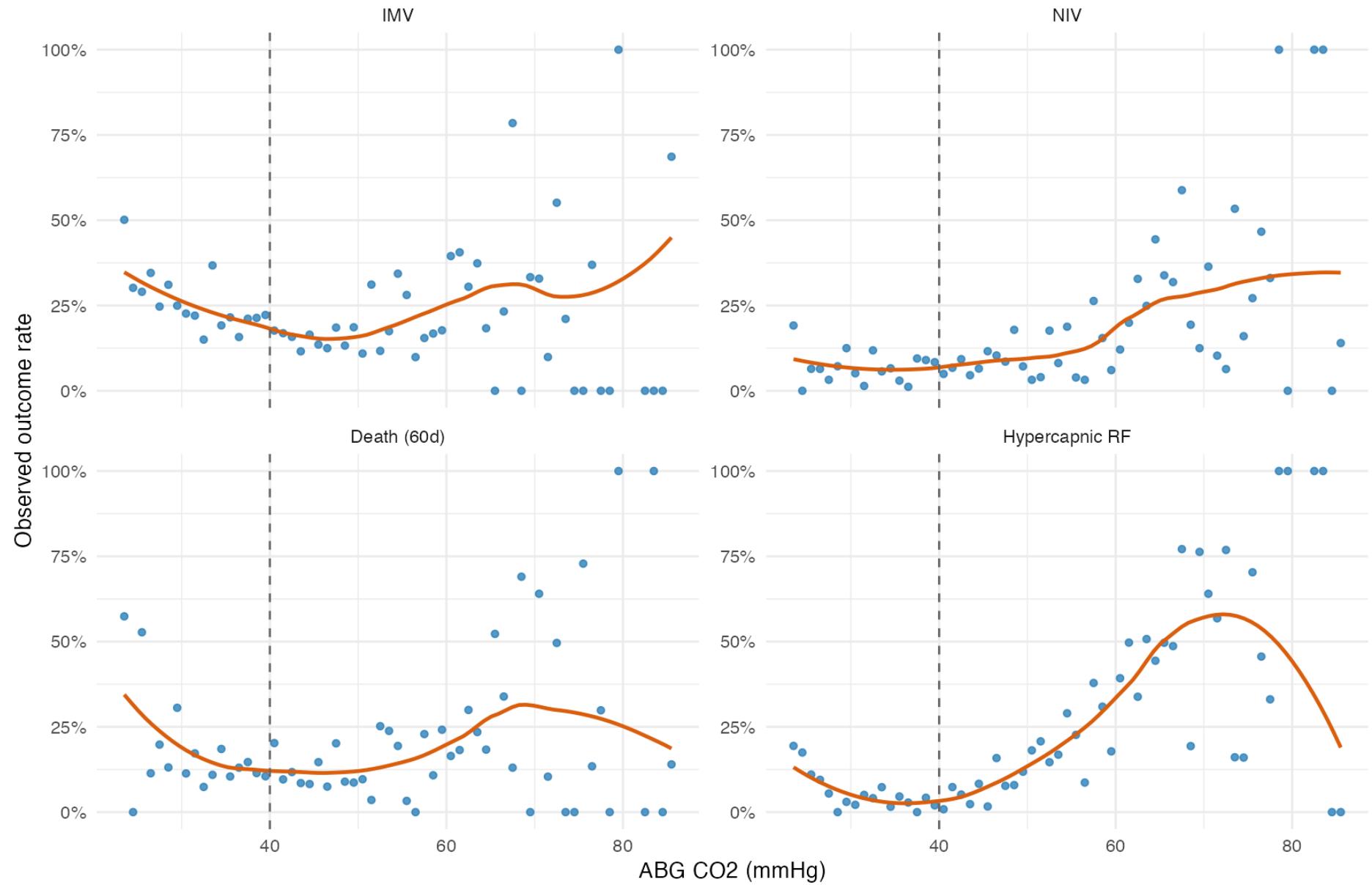
write_csv_safely(co2_rate_mi_abg, results_path("co2_binned_rates_mi_ipw_abg.csv"), row_names = FALSE)
write_csv_safely(co2_rate_mi_vbg, results_path("co2_binned_rates_mi_ipw_vbg.csv"), row_names = FALSE)

p_co2_rate_mi_abg <- plot_co2_bin_rate_panel(
  co2_rate_mi_abg,
  title = paste0("Observed outcome rates by ABG CO2 bin (MI + IPSW, m = ", imp$m, ")"),
  x_label = "ABG CO2 (mmHg)",
  vline_at = ABG_CO2_REF
)
p_co2_rate_mi_vbg <- plot_co2_bin_rate_panel(
  co2_rate_mi_vbg,
  title = paste0("Observed outcome rates by VBG CO2 bin (MI + IPSW, m = ", imp$m, ")"),
  x_label = "VBG CO2 (mmHg)",

```

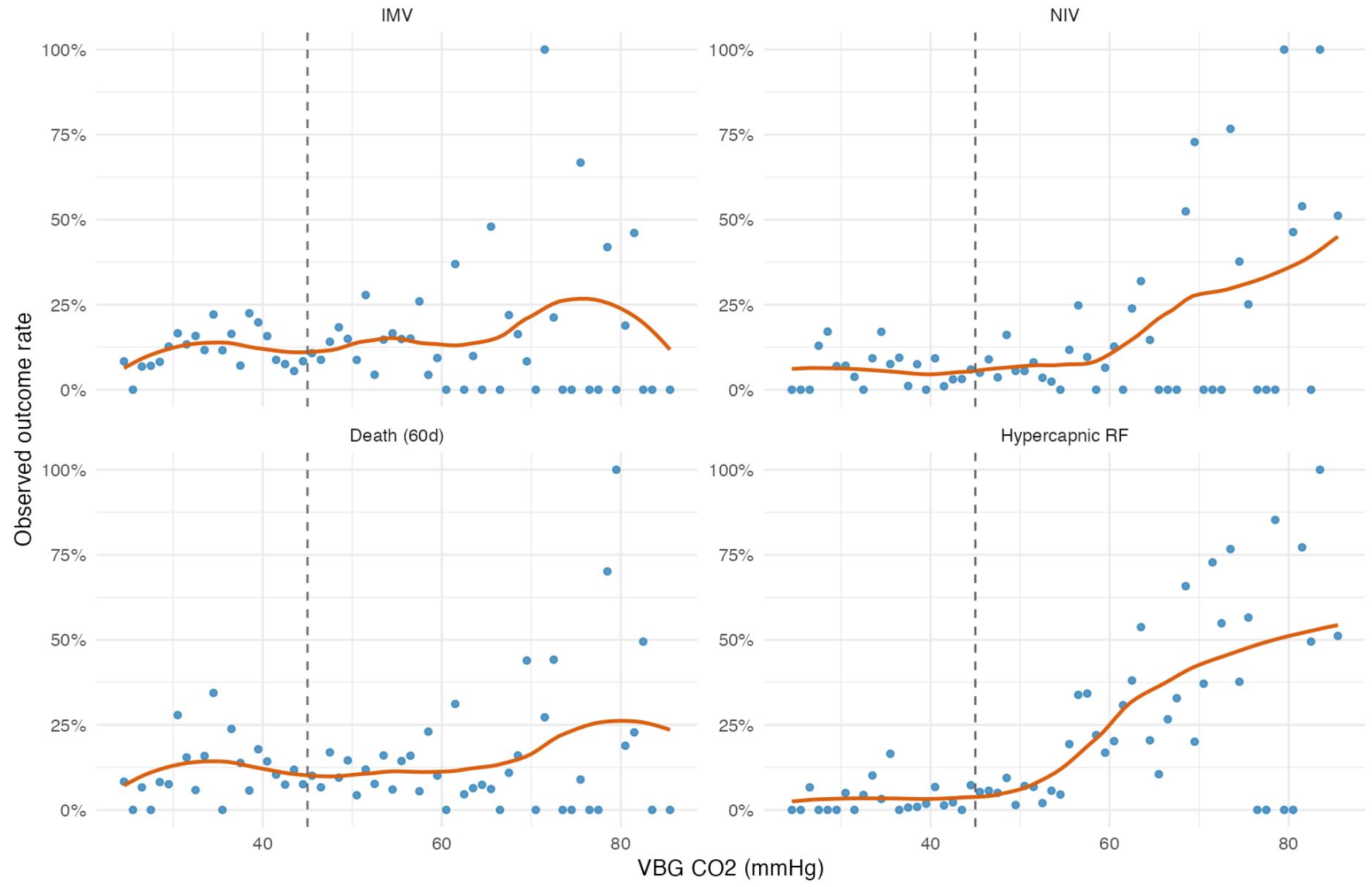
```
vline_at = VBG_CO2_REF  
)  
  
print_plot_once(p_co2_rate_mi_abg, "co2-binned-rates-mi-ipw-abg", width = 8.5, height = 6.0)
```

Observed outcome rates by ABG CO₂ bin (MI + IPSW, m = 80)



```
print_plot_once(p_co2_rate_mi_vbg, "co2-binned-rates-mi-ipw-vbg", width = 8.5, height = 6.0)
```

Observed outcome rates by VBG CO₂ bin (MI + IPSW, m = 80)



```

rm(co2_rate_mi_rows_abg, co2_rate_mi_rows_vbg, co2_rate_mi_abg, co2_rate_mi_vbg,
  p_co2_rate_mi_abg, p_co2_rate_mi_vbg)
invisible(gc(FALSE))

```

3.9 Manuscript outputs summary

```

# Cohort flow / sample sizes
flow_tbl <- tibble::tibble(
  metric = c(
    "Full cohort (raw)",
    "Analytic subset",
    "ABG tested",
    "ABG with PaCO2",
    "VBG tested",
    "VBG with VBG CO2"
  ),
  n = c(
    nrow(stata_data),
    nrow(subset_data),
    sum(subset_data$has_abg == 1, na.rm = TRUE),
    sum(subset_data$has_abg == 1 & !is.na(subset_data$paco2), na.rm = TRUE),
    sum(subset_data$has_vbg == 1, na.rm = TRUE),
    sum(subset_data$has_vbg == 1 & !is.na(subset_data$vbg_co2), na.rm = TRUE)
  )
)
render_table_pdf_maybe(flow_tbl, "Cohort flow summary", "cohort_flow_summary",
                       digits = 0, show = SHOW_LOW_VALUE_TABLES)

# Event counts by cohort
outcome_vars <- c("imv_proc", "niv_proc", "death_60d", "hypercap_resp_failure")
outcome_labels <- c(
  imv_proc = "IMV",
  niv_proc = "NIV",
  death_60d = "Death (60d)",
  hypercap_resp_failure = "Hypercapnic RF"
)

```

```

event_tbl <- dplyr::bind_rows(
  lapply(outcome_vars, function(o) {
    dplyr::tibble(
      outcome = outcome_labels[[o]],
      group = "ABG",
      n = sum(subset_data$has_abg == 1 & !is.na(subset_data[[o]]), na.rm = TRUE),
      events = sum(subset_data$has_abg == 1 & subset_data[[o]] == 1, na.rm = TRUE)
    )
  }),
  lapply(outcome_vars, function(o) {
    dplyr::tibble(
      outcome = outcome_labels[[o]],
      group = "VBG",
      n = sum(subset_data$has_vbg == 1 & !is.na(subset_data[[o]]), na.rm = TRUE),
      events = sum(subset_data$has_vbg == 1 & subset_data[[o]] == 1, na.rm = TRUE)
    )
  })
)
event_tbl <- event_tbl |>
  dplyr::mutate(pct = ifelse(n > 0, 100 * events / n, NA_real_))
render_table_pdf_maybe(event_tbl,
  "Outcome counts by cohort (ABG/VBG tested)",
  "outcome_counts_by_cohort",
  digits = 1, show = SHOW_LOW_VALUE_TABLES)

# Weighting diagnostics (non-MI weights)
stopifnot(all(c("w_abg", "w_vbg", "ps_abg", "ps_vbg") %in% names(subset_data)))
wt_abg <- subset_data$w_abg[subset_data$has_abg == 1]
wt_vbg <- subset_data$w_vbg[subset_data$has_vbg == 1]
ps_abg <- subset_data$ps_abg[subset_data$has_abg == 1]
ps_vbg <- subset_data$ps_vbg[subset_data$has_vbg == 1]
trunc_abg <- subset_data$trunc_abg[subset_data$has_abg == 1]
trunc_vbg <- subset_data$trunc_vbg[subset_data$has_vbg == 1]

wt_sum <- dplyr::bind_rows(
  weight_summary(wt_abg, ps = ps_abg, ps_floor = ps_floor_abg,
    truncated = trunc_abg) |>

```

```

    dplyr::mutate(group = "ABG"),
    weight_summary(wt_vbg, ps = ps_vbg, ps_floor = ps_floor_vbg,
                   truncated = trunc_vbg) |>
    dplyr::mutate(group = "VBG")
)
wt_sum_display <- wt_sum |>
  dplyr::select(group, n, ess, min, p99, max, trunc_rate, ps_p01) |>
  dplyr::rename(
    trunc = trunc_rate,
    p01_ps = ps_p01
)
render_table_pdf_maybe(wt_sum_display,
                       "Weighting diagnostics summary (non-MI)",
                       "weighting_diagnostics_non_mi",
                       wide = TRUE,
                       digits = 3,
                       show = SHOW_LOW_VALUE_TABLES)

# Missingness + MI spec summary
miss_vars <- c(covars_ps, "paco2", "vbg_co2",
                 "imv_proc", "niv_proc", "death_60d", "hypercap_resp_failure")
miss_vars <- intersect(miss_vars, names(subset_data_raw))
miss_tbl <- subset_data_raw |>
  dplyr::summarise(dplyr::across(dplyr::all_of(miss_vars), ~ mean(is.na(.)) * 100)) |>
  tidyr::pivot_longer(dplyr::everything(), names_to = "variable", values_to = "pct_missing") |>
  dplyr::arrange(dplyr::desc(pct_missing))
render_table_pdf_maybe(miss_tbl,
                       "Variables by missingness (pre-imputation)",
                       "missingness_all",
                       digits = 1,
                       show = SHOW_LOW_VALUE_TABLES)

mi_spec_tbl <- tibble::tibble(
  m = imp$m,
  maxit = MAXIT_MI,
  methods = paste(unique(imp$method[imp$method != ""])), collapse = ", "),
  ps_model = MI_PS_METHOD,

```

```

ps_spline_k = MI_PS_SPLINE_K,
ps_glm_maxit = MI_GLM_MAXIT
)
render_table_pdf_maybe(mi_spec_tbl,
                      "MI specification (methods used)",
                      "mi_specification",
                      show = SHOW_LOW_VALUE_TABLES)

# Purpose: chunk.
stopifnot(exists("imp"))
imp_n <- imp$m
get_imp <- function(i, imp_obj = imp) { normalize_types(mice::complete(imp_obj, action = i), levels_ref) }

# 1) must exist and be numeric
d1 <- get_imp(1)
stopifnot(all(c("paco2", "vbg_co2") %in% names(d1)))
stopifnot(is.numeric(d1$paco2), is.numeric(d1$vbg_co2))

# 2) confirm at least two PaCO2 levels among those with ABG in each imputation
table(vapply(seq_len(imp_n), function(i) {
  d <- get_imp(i)
  dplyr::n_distinct(d$paco2[d$has_abg == 1 & is.finite(d$paco2)])
}, integer(1)) > 1)

```

TRUE

80

```
# 3) light sanity check: ensure pooled MI cat3 results exist
stopifnot(exists("abg_cat_results"), nrow(abg_cat_results) > 0)
```

3.9.1 Visualization: pooled three-level ORs

```
# Purpose: chunk.
stopifnot(exists("abg_cat_results"), exists("vbg_cat_results"))
```

```

mi_combined_or_df <- dplyr::bind_rows(
  dplyr::mutate(abg_cat_results, group = "ABG"),
  dplyr::mutate(vbg_cat_results, group = "VBG")
) |>
  dplyr::mutate(
    outcome = factor(outcome,
                      levels = c("IMV", "NIV", "Death60d", "HCRF"),
                      labels = c("IMV", "NIV", "Death (60d)", "Hypercapnic RF")),
    group = factor(group, levels = c("ABG", "VBG"))
  )

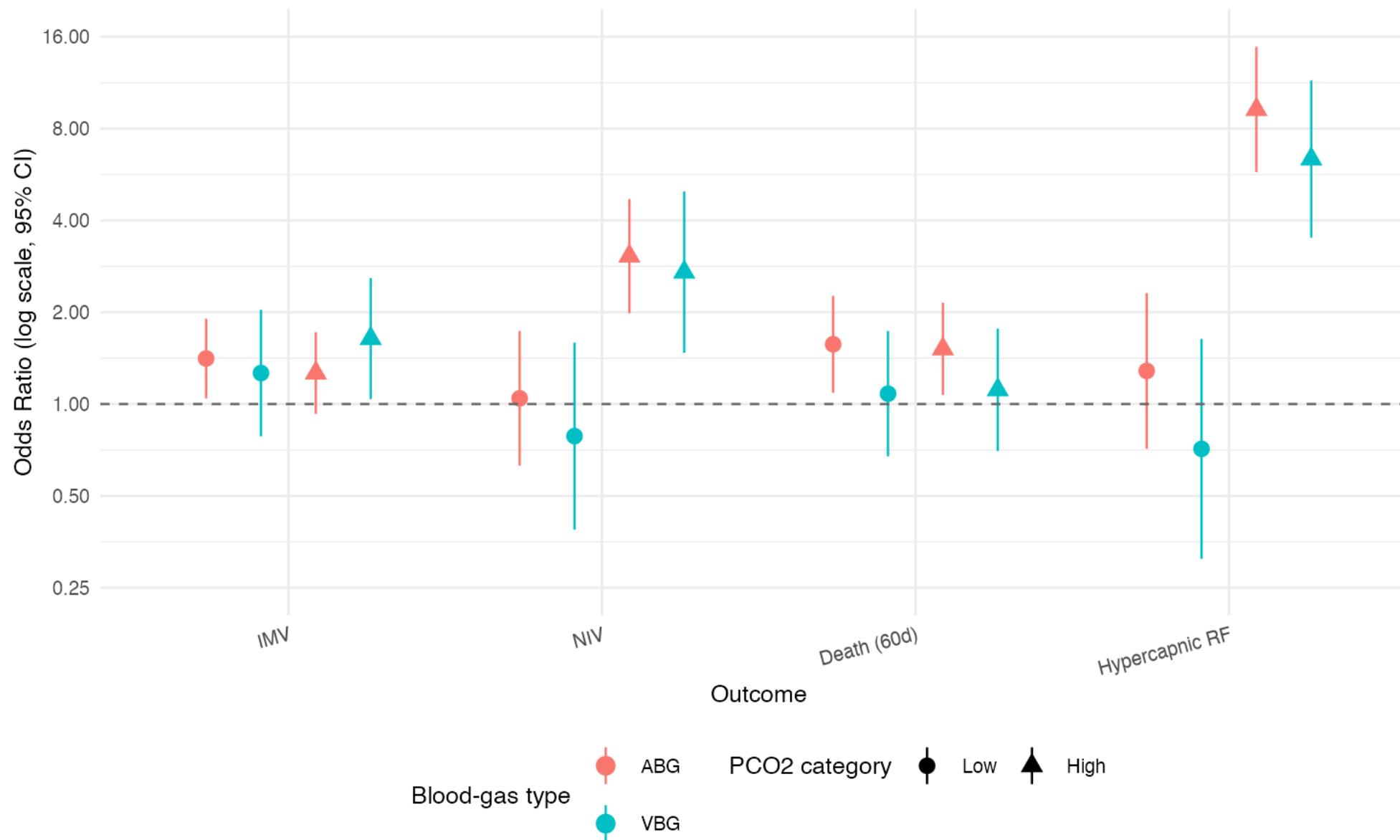
mi_combined_or_df <- map_or_exposure(mi_combined_or_df, "or-plot-mi-weighted")
mi_plot_df <- mi_combined_or_df |>
  dplyr::mutate(estimate = OR, conf.low = LCL, conf.high = UCL) |>
  dplyr::select(-OR, -LCL, -UCL)

mi_plot_df <- build_or_plot_df(
  mi_plot_df,
  "or-plot-mi-weighted",
  expected_exposure_levels = CO2_CAT_CONTRAST_LEVELS
)
mi_axis_spec <- compute_or_axis_spec(list(mi_plot_df), lo_col = "conf.low", hi_col = "conf.high")

mi_p_or <- plot_or_safe(
  mi_plot_df,
  plot_name = "or-plot-mi-weighted",
  axis_spec = mi_axis_spec,
  title = "MI-pooled, IPW-adjusted odds ratios by PCO2 category (ABG vs VBG)"
)
print_plot_once(mi_p_or, "or-plot-mi-weighted", width = 7.5, height = 4.8)

```

MI-pooled, IPW-adjusted odds ratios by PCO₂ category (ABG vs VBG)



```

# Purpose: mi propensity histograms logistic.
# Build an imputation-averaged propensity histogram for ABG and VBG.
# This mirrors the non-MI propensity display while respecting MI variability.
stopifnot(exists("imp"))
stopifnot(exists("mi_logistic_ps_abg_list"), exists("mi_logistic_ps_vbg_list"))

bin_edges <- seq(0, 1, length.out = 31)
bin_width <- diff(bin_edges)[1]
bin_centers <- head(bin_edges, -1) + bin_width / 2
treat_abg_all <- to01(imp$data$has_abg)
treat_vbg_all <- to01(imp$data$has_vbg)

mi_ps_hist_one <- function(ps_vec, treat_vec, score_type, imp_index) {
  # Use integer bin IDs to avoid floating-point matching issues during complete().
  df <- data.frame(
    ps = as.numeric(ps_vec),
    treat = as.integer(treat_vec),
    stringsAsFactors = FALSE
  ) |>
    dplyr::filter(is.finite(ps), !is.na(treat), ps >= 0, ps <= 1) |>
    dplyr::mutate(
      treat = factor(treat, levels = c(0L, 1L), labels = c("No Test", "Test")),
      bin_id = cut(
        ps,
        breaks = bin_edges,
        include.lowest = TRUE,
        right = FALSE,
        labels = FALSE
      )
    )
  }

  if (nrow(df) == 0 || all(is.na(df$bin_id))) return(data.frame())

  df |>
    dplyr::filter(!is.na(bin_id)) |>
    dplyr::count(treat, bin_id, .drop = FALSE, name = "n") |>
    tidyr::complete(treat, bin_id = seq_along(bin_centers), fill = list(n = 0L)) |>

```

```

dplyr::group_by(treat) |>
dplyr::mutate(
  n_total = sum(n, na.rm = TRUE),
  density = dplyr::if_else(n_total > 0, n / (n_total * bin_width), 0)
) |>
dplyr::ungroup() |>
dplyr::mutate(
  ScoreType = score_type,
  imp = imp_index,
  bin_mid = bin_centers[bin_id]
) |>
dplyr::select(ScoreType, imp, treat, bin_mid, density)
}

mi_ps_hist_rows <- vector("list", imp$m * 2L)
row_idx <- 0L
for (i in seq_len(imp$m)) {
  # Aggregate ABG and VBG bin-level densities for each imputation.
  row_idx <- row_idx + 1L
  mi_ps_hist_rows[[row_idx]] <- mi_ps_hist_one(get_mi_ps("ABG", i), treat_abg_all, "ABG", i)
  row_idx <- row_idx + 1L
  mi_ps_hist_rows[[row_idx]] <- mi_ps_hist_one(get_mi_ps("VBG", i), treat_vbg_all, "VBG", i)
}

# Average densities across imputations to avoid overplotting 80+ curves.
mi_ps_hist <- dplyr::bind_rows(mi_ps_hist_rows) |>
  dplyr::group_by(ScoreType, treat, bin_mid) |>
  dplyr::summarise(density = mean(density, na.rm = TRUE), .groups = "drop") |>
  dplyr::mutate(ScoreType = factor(ScoreType, levels = c("ABG", "VBG")))

if (nrow(mi_ps_hist) > 0 && any(is.finite(mi_ps_hist$density) & mi_ps_hist$density > 0)) {
  # Render side-by-side ABG/VBG density histograms with shared x-range [0,1].
  mi_ps_hist_plot <- ggplot(mi_ps_hist, aes(x = bin_mid, y = density, fill = treat)) +
    geom_col(position = "identity", alpha = 0.5, width = bin_width * 0.95) +
    facet_wrap(~ ScoreType, scales = "free_y") +
    coord_cartesian(xlim = c(0, 1)) +
    scale_fill_manual(values = c("No Test" = "steelblue", "Test" = "tomato")) +

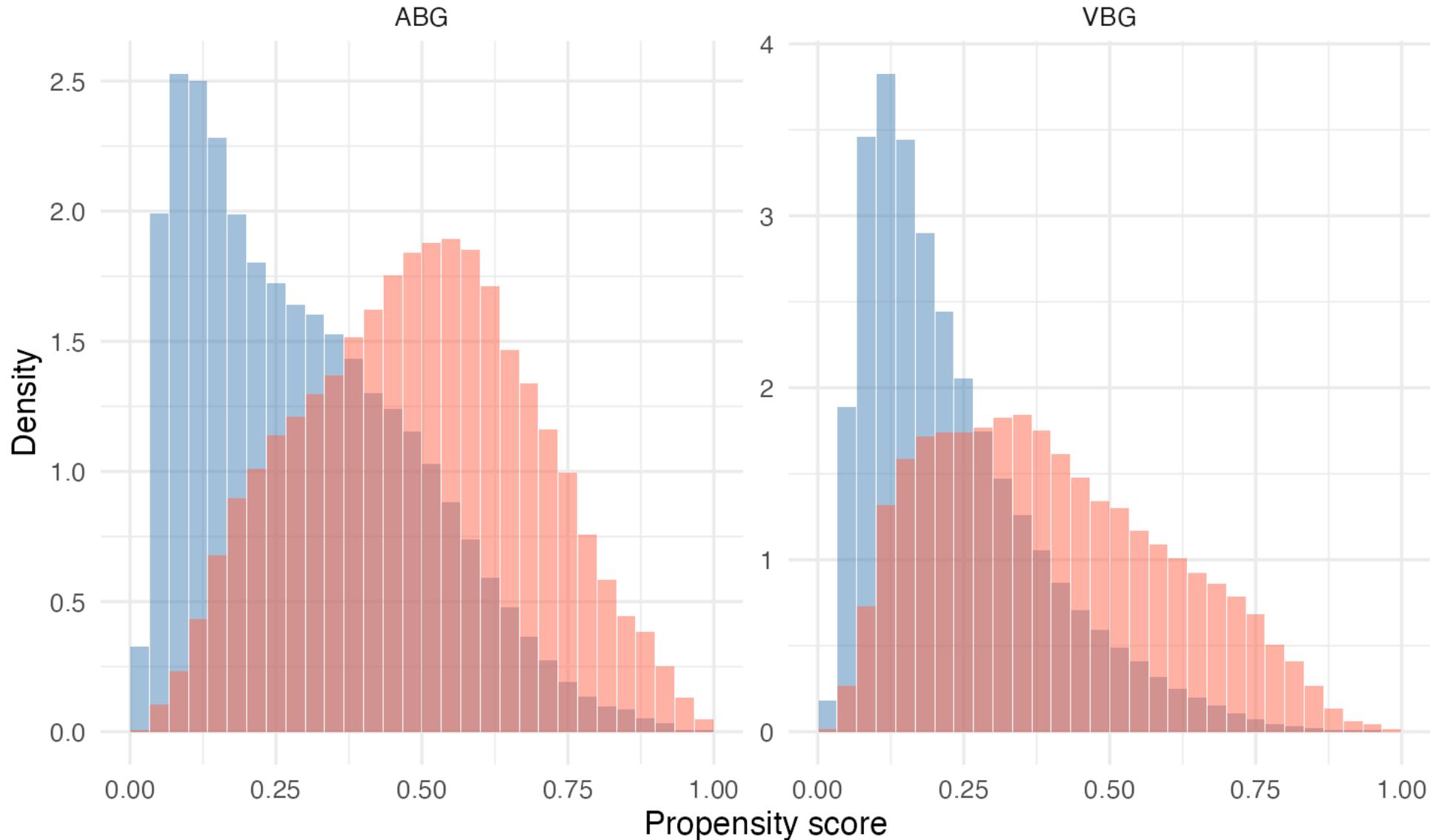
```

```
  labs(
    title = "MI logistic propensity score distributions",
    subtitle = paste0("Averaged across ", imp$m, " imputations"),
    x = "Propensity score",
    y = "Density",
    fill = "Group"
  ) +
  theme_minimal(base_size = 12)

  print_plot_once(mi_ps_hist_plot, "propensity-histograms-mi-logistic", width = 8.5, height = 5)
} else {
  warning("MI propensity histogram has no positive density values; inspect mi_logistic_ps_* artifacts.", call. =
    FALSE)
}
```

MI logistic propensity score distributions

Averaged across 80 imputations



```

# Purpose: loveplot mi logistic.
# Build pooled (median across imputations) raw vs weighted |SMD|
# so MI balance can be read with the same visual grammar as non-MI.
stopifnot(exists("bal_imp_abg"), exists("bal_imp_vbg"))

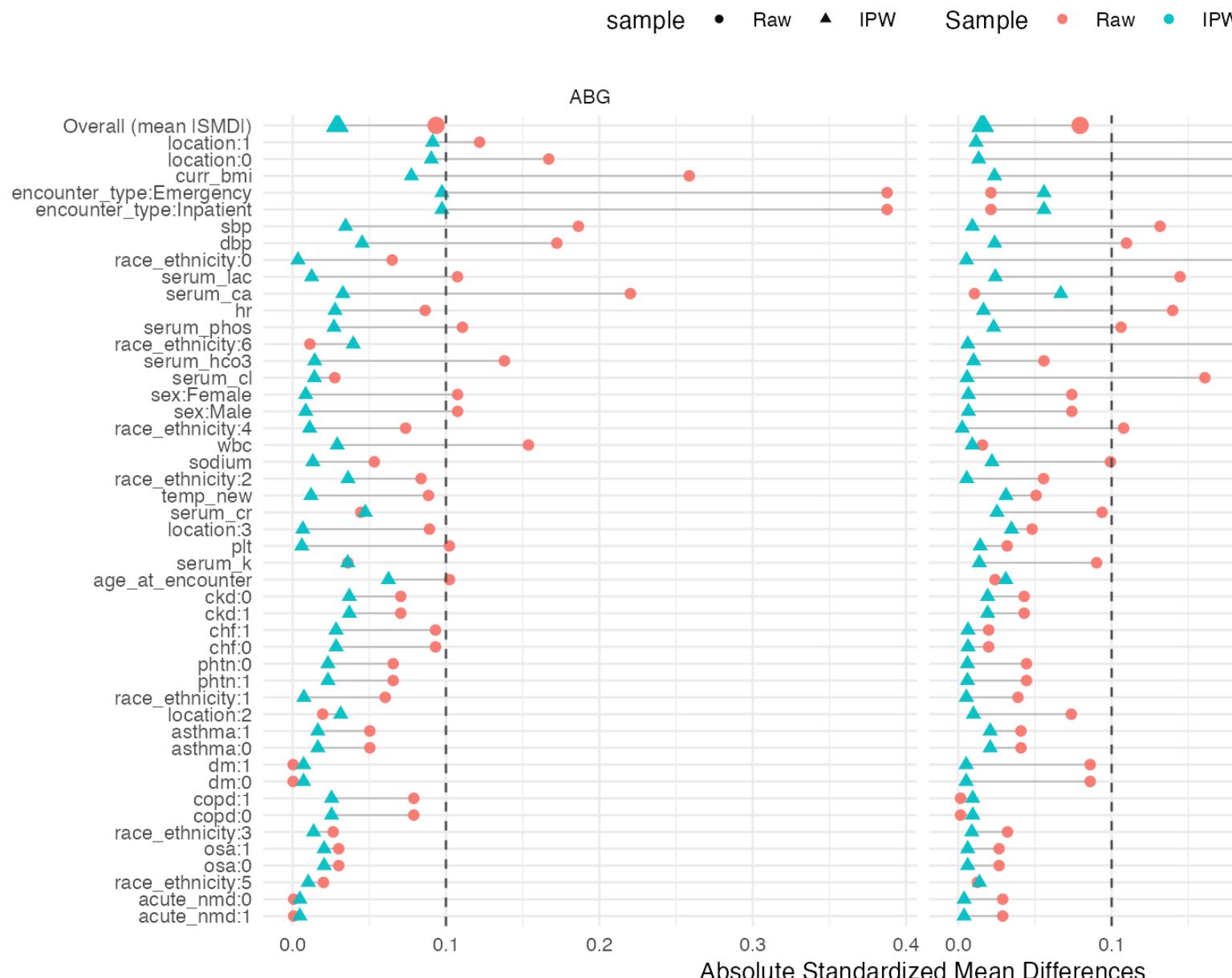
lov_mi <- dplyr::bind_rows(bal_imp_abg$bal_long, bal_imp_vbg$bal_long) |>
  dplyr::mutate(
    term = ifelse(is.na(level), variable, paste0(variable, ":", level)),
    abs_smd_raw = abs(smd_pre),
    abs_smd_ipw = abs(smd_post)
  ) |>
  dplyr::group_by(group, term) |>
  dplyr::summarise(
    abs_smd_raw = median(abs_smd_raw, na.rm = TRUE),
    abs_smd_ipw = median(abs_smd_ipw, na.rm = TRUE),
    .groups = "drop"
  )

lov_mi_long <- dplyr::bind_rows(
  lov_mi |>
    dplyr::transmute(group, term, sample = "Raw", abs_smd = abs_smd_raw, abs_smd_raw),
  lov_mi |>
    dplyr::transmute(group, term, sample = "IPW", abs_smd = abs_smd_ipw, abs_smd_raw)
)
lov_mi_long <- add_overall_love_rows(lov_mi_long)

# Export plotting data and then draw the two-panel love plot.
write_csv_safely(lov_mi_long, results_path("loveplot_mi_logistic_data.csv"), row_names = FALSE)
p_lov_mi <- loveplot_style(lov_mi_long, "MI logistic IPSW covariate balance (median across imputations)")
print_plot_once(p_lov_mi, "loveplot-mi-logistic-abg-vbg", width = 9, height = 6)

```

MI logistic IPSW covariate balance (median across imputations)



```

# Purpose: render SHAP top-10 directly from CSV to ensure PDF display.
# Read precomputed top-10 SHAP values and re-plot deterministically.
shap_abg_file <- results_path("shap_top10_mi_logistic_abg.csv")
shap_vbg_file <- results_path("shap_top10_mi_logistic_vbg.csv")

shap_abg <- utils::read.csv(shap_abg_file, stringsAsFactors = FALSE)
shap_vbg <- utils::read.csv(shap_vbg_file, stringsAsFactors = FALSE)
shap_abg$group <- "ABG"
shap_vbg$group <- "VBG"

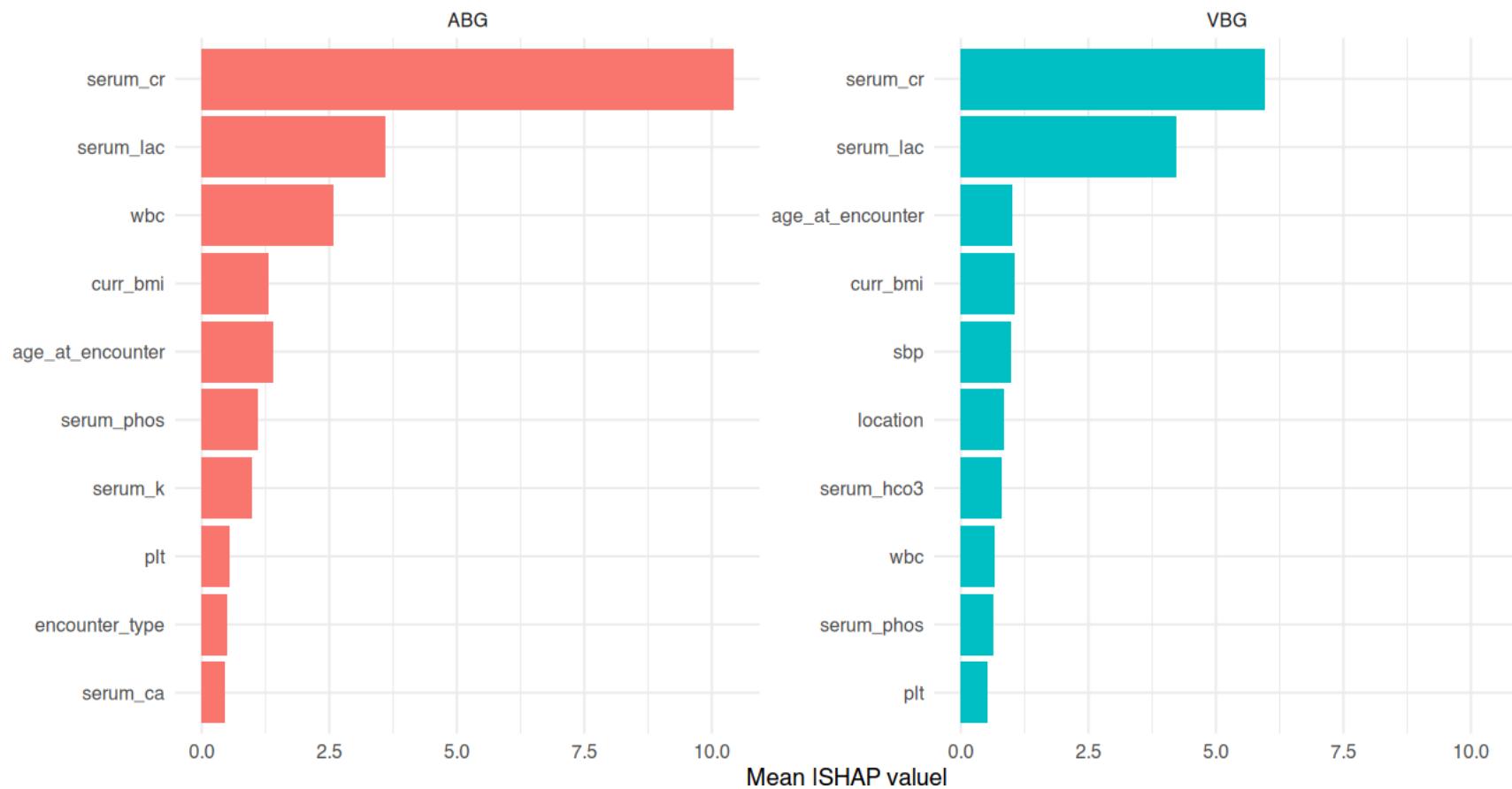
shap_plot_df <- dplyr::bind_rows(shap_abg, shap_vbg) |>
  dplyr::filter(is.finite(mean_abs_shap)) |>
  dplyr::group_by(group) |>
  dplyr::arrange(dplyr::desc(mean_abs_shap), .by_group = TRUE) |>
  dplyr::slice_head(n = 10L) |>
  dplyr::ungroup()

stopifnot(nrow(shap_plot_df) > 0)

p_shap <- plot_shap_top10_two_panel(
  shap_plot_df,
  "MI logistic IPSW SHAP top contributors (ABG vs VBG)",
  "Mean |SHAP value|"
)
print(p_shap)

```

MI logistic IPSW SHAP top contributors (ABG vs VBG)



3.9.2 Visualization

```
# Purpose: chunk.  
library(dplyr)  
library(ggplot2)  
library(patchwork)  
library(purrr)
```

```

mi_ipw_rcs_forms <- list(
  "MI IPW spline (adjusted) ABG: IMV ~ CO2 spline + X"      = make_spline_fml("imv_proc", "paco2", adj_core),
  "MI IPW spline (adjusted) ABG: NIV ~ CO2 spline + X"      = make_spline_fml("niv_proc", "paco2", adj_core),
  "MI IPW spline (adjusted) ABG: Death60d ~ CO2 spline + X" = make_spline_fml("death_60d", "paco2", adj_core),
  "MI IPW spline (adjusted) ABG: HCRF ~ CO2 spline + X"     = make_spline_fml("hypercap_resp_failure", "paco2",
    ↳ adj_core),
  "MI IPW spline (adjusted) VBG: IMV ~ CO2 spline + X"      = make_spline_fml("imv_proc", "vbg_co2", adj_core),
  "MI IPW spline (adjusted) VBG: NIV ~ CO2 spline + X"      = make_spline_fml("niv_proc", "vbg_co2", adj_core),
  "MI IPW spline (adjusted) VBG: Death60d ~ CO2 spline + X" = make_spline_fml("death_60d", "vbg_co2", adj_core),
  "MI IPW spline (adjusted) VBG: HCRF ~ CO2 spline + X"     = make_spline_fml("hypercap_resp_failure", "vbg_co2",
    ↳ adj_core)
)
register_model_diagrams(mi_ipw_rcs_forms)

stopifnot(exists("abg_curves"), exists("vbg_curves"))

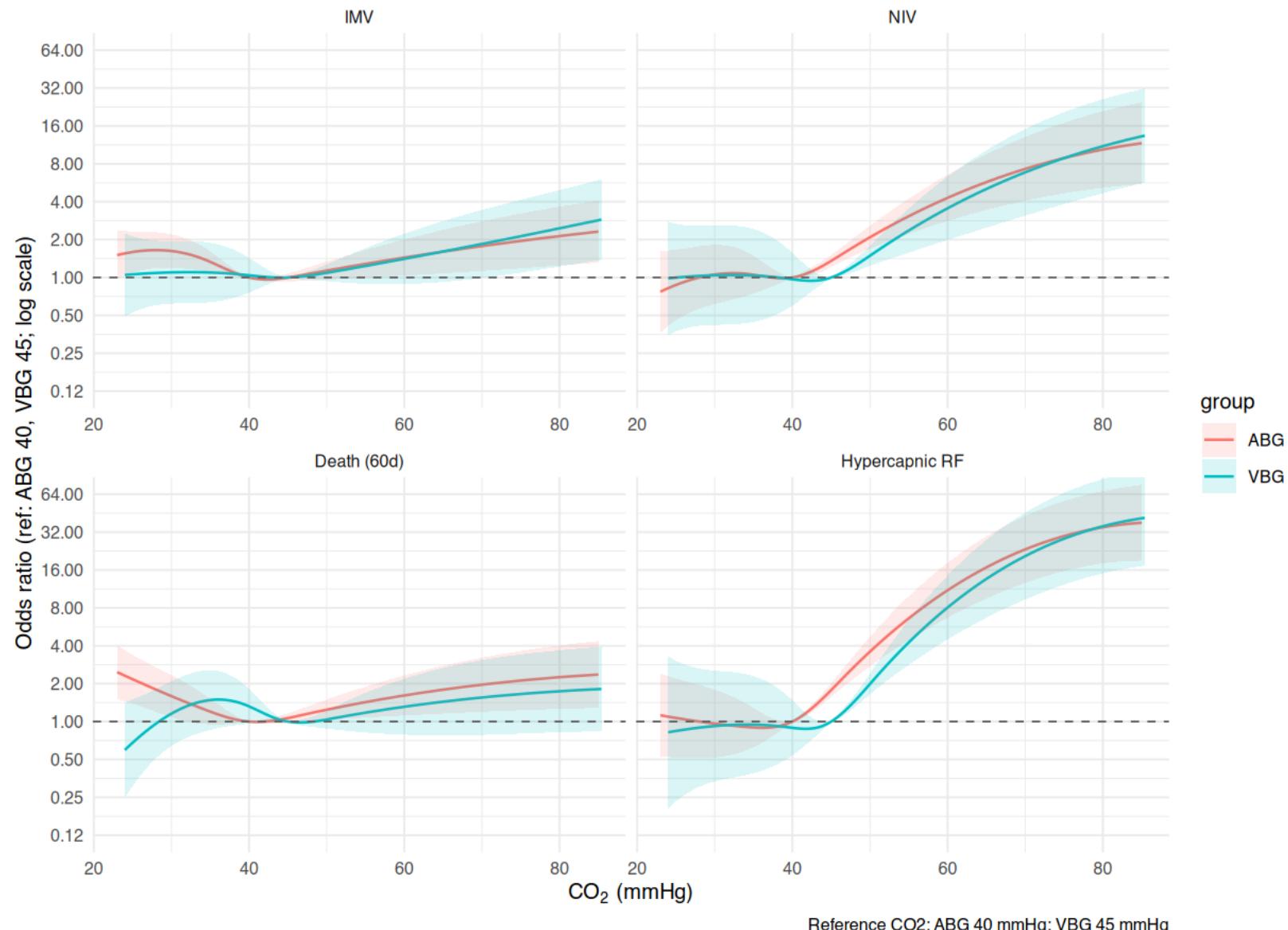
curve_abg <- abg_curves |>
  mutate(co2 = paco2) |>
  select(-paco2)
curve_vbg <- vbg_curves |>
  mutate(co2 = vbg_co2) |>
  select(-vbg_co2)
curve_all <- bind_rows(curve_abg, curve_vbg) |>
  mutate(outcome = factor(outcome,
    levels = c("imv_proc", "niv_proc", "death_60d", "hypercap_resp_failure"),
    labels = c("IMV", "NIV", "Death (60d)", "Hypercapnic RF")))
axis_mi_ipw <- compute_or_axis_spec(list(curve_abg, curve_vbg), lo_col = "LCL", hi_col = "UCL")

ggplot(curve_all, aes(x = co2, y = OR, color = group, fill = group)) +
  geom_line(linewidth = 0.6) +
  geom_ribbon(aes(ymin = LCL, ymax = UCL), alpha = 0.15, color = NA) +
  geom_hline(yintercept = 1, linetype = "dashed", color = "grey40") +
  or_axis_scale(axis_mi_ipw) +
  facet_wrap(~ outcome, scales = "free_x") +
  labs(
    title = expression(
      paste("MI-pooled, IPSW-adjusted spline odds ratios: ABG vs VBG CO"[2]))

```

```
),
x = expression(CO[2]~"(mmHg)" ),
y = paste0("Odds ratio (ref: ABG ", ABG_CO2_REF, ", VBG ", VBG_CO2_REF, "; log scale)"),
caption = paste0("Reference CO2: ABG ", ABG_CO2_REF, " mmHg; VBG ", VBG_CO2_REF, " mmHg")
) +
theme_minimal(base_size = 10)
```

MI-pooled, IPSW-adjusted spline odds ratios: ABG vs VBG CO₂



Purpose: ipw output parity.

```

non_mi_ipw_outputs <- tibble::tribble(
  ~parity_key, ~artifact_type, ~artifact_name,
  "threelevel_or", "figure", "or-plot-three-level-weighted.png",
  "abg_spline_panel", "figure", "spline-ipw-abg-trimmed.png",
  "vbg_spline_panel", "figure", "spline-ipw-vbg.png",
  "abg_shared_spline", "figure", "spline-ipw-abg-shared.png",
  "overlay_spline", "figure", "spline-ipw-overlay-abg-vbg.png",
  "propensity_hist_tested", "figure", "propensity-histograms-conditional.png",
  "shap_top10_plot", "figure", "shap-top10-ipw-gbm-abg-vbg.png",
  "shap_top10_abg", "table_csv", "shap_top10_ipw_gbm_abg.csv",
  "shap_top10_vbg", "table_csv", "shap_top10_ipw_gbm_vbg.csv",
  "love_plot", "figure", "loveplot-ipw-gbm-abg-vbg.png",
  "love_plot_data", "table_csv", "loveplot_ipw_gbm_data.csv",
  "threelevel_table", "table_csv", "table_summary_adjusted_threelevel.csv",
  "weighting_diag", "table_csv", "weighting_diagnostics_non_mi.csv"
)

mi_ipw_outputs <- tibble::tribble(
  ~parity_key, ~artifact_type, ~artifact_name,
  "threelevel_or", "figure", "or-plot-mi-weighted.png",
  "overlay_spline", "figure", "ipw-rcs-overlay-mi-abg-vbg*.png",
  "propensity_hist_tested", "figure", "propensity-histograms-mi-logistic.png",
  "shap_top10_plot", "figure", "shap-top10-mi-logistic-abg-vbg.png",
  "shap_top10_abg", "table_csv", "shap_top10_mi_logistic_abg.csv",
  "shap_top10_vbg", "table_csv", "shap_top10_mi_logistic_vbg.csv",
  "love_plot", "figure", "loveplot-mi-logistic-abg-vbg.png",
  "love_plot_data", "table_csv", "loveplot_mi_logistic_data.csv",
  "threelevel_table", "table_csv", "mi_threelevel_results.csv",
  "ps_model_store_abg", "diagnostic_rds", "mi_logistic_ps_abg_list.rds",
  "ps_model_store_vbg", "diagnostic_rds", "mi_logistic_ps_vbg_list.rds"
)

artifact_exists <- function(name, type) {
  if (is.na(name) || !nzchar(name)) return(FALSE)
  if (type == "figure") {
    fig_path <- results_path("figs", name)
    if (grepl("[*?\\[]", name)) return(length(Sys.glob(fig_path)) > 0)
  }
}

```

```

    return(file.exists(fig_path))
}
file.exists(results_path(name))
}

parity_tbl <- dplyr::full_join(
  dplyr::rename(non_mi_ipw_outputs, non_mi_artifact = artifact_name, artifact_type_non_mi = artifact_type),
  dplyr::rename(mi_ipw_outputs, mi_artifact = artifact_name, artifact_type_mi = artifact_type),
  by = "parity_key"
) |>
  dplyr::mutate(
    artifact_type = dplyr::coalesce(artifact_type_non_mi, artifact_type_mi),
    present_non_mi = mapply(artifact_exists, non_mi_artifact, artifact_type, USE.NAMES = FALSE),
    present_mi = mapply(artifact_exists, mi_artifact, artifact_type, USE.NAMES = FALSE),
    parity_status = dplyr::case_when(
      present_non_mi & present_mi ~ "both",
      present_non_mi & !present_mi ~ "non_mi_only",
      !present_non_mi & present_mi ~ "mi_only",
      TRUE ~ "unknown"
    ),
    recommended_action = dplyr::case_when(
      parity_status == "both" ~ "none",
      parity_status == "non_mi_only" ~ "add MI equivalent output",
      parity_status == "mi_only" ~ "add non-MI equivalent output",
      TRUE ~ "review manually"
    )
  ) |>
  dplyr::arrange(artifact_type, parity_key) |>
  dplyr::select(parity_key, artifact_type, non_mi_artifact, mi_artifact,
                present_non_mi, present_mi, parity_status, recommended_action)

write_csv_safely(parity_tbl, results_path("ipw_output_parity.csv"), row_names = FALSE)

parity_unknown <- parity_tbl |>
  dplyr::filter(parity_status == "unknown")
write_csv_safely(parity_unknown, results_path("ipw_output_parity_unknown.csv"), row_names = FALSE)
if (RUN_MODE == "full" && nrow(parity_unknown) > 0) {

```

```

  stop("Parity audit has unknown rows; see ", results_path("ipw_output_parity_unknown.csv"))
}

parity_one_sided <- parity_tbl |>
  dplyr::filter(parity_status != "both")

write_csv_safely(parity_one_sided, results_path("ipw_output_parity_diffs.csv"), row_names = FALSE)

render_table_pdf_maybe(
  parity_one_sided,
  caption = "IPSW output parity (artifacts present in only one track)",
  file_stub = "ipw_output_parity_one_sided",
  digits = 0,
  show = SHOW_LOW_VALUE_TABLES
)

```

4 Diagnostics and Exports

```

# Purpose: diagnostics inputs.
stopifnot(gbm_params$stop.method == "smd.max")
stopifnot(exists("M_IMP"), exists("MAXIT_MI"), exists("MI_SEED"))
stopifnot(exists("imp"))
stopifnot(exists("mi_logistic_ps_abg_list"), exists("mi_logistic_ps_vbg_list"))
meth <- imp$method

diag_fig_dir <- results_path("figs")
fs::dir_create(diag_fig_dir)

```

4.0.1 MI convergence and mixing

```

# Purpose: diagnostics mi convergence.
stopifnot(exists("imp"))

log_events_summary_file <- results_path("mice_logged_events_summary.csv")
stopifnot(file.exists(log_events_summary_file))

```

```

log_events_summary <- tryCatch(
  utils::read.csv(log_events_summary_file, stringsAsFactors = FALSE),
  error = function(e) data.frame()
)
if (nrow(log_events_summary) && !("empty" %in% names(log_events_summary) && all(log_events_summary$empty))) {
  render_table_pdf_maybe(
    log_events_summary,
    caption = "Logged events (MI)",
    file_stub = "mice_logged_events_summary",
    digits = 0,
    show = SHOW_LOW_VALUE_TABLES
)

stopifnot(all(c("variable", "n") %in% names(log_events_summary)))
total_events <- sum(log_events_summary$n, na.rm = TRUE)
if (is.finite(total_events) && total_events > 1000) {
  warning("MI loggedEvents count is high (", total_events, "). Review mice_logged_events_summary.csv.", call. =
    FALSE)
}
by_var <- log_events_summary |>
  dplyr::group_by(variable) |>
  dplyr::summarise(n = sum(n), .groups = "drop") |>
  dplyr::arrange(dplyr::desc(n))
if (nrow(by_var) && is.finite(total_events) && total_events > 0) {
  top_share <- by_var$n[1] / total_events
  if (top_share > 0.5) {
    warning("MI loggedEvents dominated by variable '", by_var$variable[1],
           "' (", round(100 * top_share, 1), "% of events).", call. = FALSE)
  }
}
} else {
  message("No logged events to summarize.")
}

stopifnot(exists("chain_diag"))
stopifnot(nrow(chain_diag) > 0)
n_imputed <- chain_diag_stats$n_imputed_vars

```

```

n_with_tail <- chain_diag_stats$n_with_drift_tail
message("Chain diagnostics: drift_tail available for ", n_with_tail, " / ", n_imputed, " imputed variables.")
stopifnot("drift_tail_scaled" %in% names(chain_diag))
frac_flagged <- mean(chain_diag$flag %in% TRUE, na.rm = TRUE)
med_scaled <- stats::median(chain_diag$drift_tail_scaled, na.rm = TRUE)
max_scaled <- max(chain_diag$drift_tail_scaled, na.rm = TRUE)
message("Chain diagnostics (scaled): flagged=", round(frac_flagged, 3),
       "; median drift_tail_scaled=", signif(med_scaled, 3),
       "; max drift_tail_scaled=", signif(max_scaled, 3))

top_flagged <- chain_diag |>
  dplyr::filter(flag %in% TRUE) |>
  dplyr::arrange(dplyr::desc(abs(drift_tail_scaled)))
if (nrow(top_flagged)) {
  render_table_pdf_maybe(
    top_flagged |>
      dplyr::select(variable, method, drift_tail, drift_tail_scaled, slope, slope_scaled,
                    tail_n_finite, tail_window_na_frac, flag_reason),
    caption = "MICE chain diagnostics (flagged drift)",
    file_stub = "mice_chain_diagnostics_flagged",
    digits = 3,
    show = SHOW_LOW_VALUE_TABLES
  )
} else {
  message("No flagged drift in chain diagnostics.")
}

top_tail_na <- chain_diag |>
  dplyr::arrange(dplyr::desc(tail_window_na_frac))
if (nrow(top_tail_na)) {
  render_table_pdf_maybe(
    top_tail_na,
    caption = "MICE chain diagnostics (tail NA fraction)",
    file_stub = "mice_chain_diagnostics_tail_na",
    digits = 3,
    show = SHOW_LOW_VALUE_TABLES
  )
}

```

```

}

trace_vars <- intersect(c("curr_bmi", "serum_hco3", "hr", "sodium", "serum_cr"),
                        names(imp$data))

imp_trace <- imp
if (imp$m > 10) {
  set.seed(MI_SEED)
  idx <- sort(sample(seq_len(imp$m), 10))
  imp_trace$imp <- lapply(imp$imp, function(x) x[, idx, drop = FALSE])
  imp_trace$m <- length(idx)
}

if (length(trace_vars)) {
  tr_file <- results_path("figs", "diag-mi-trace-selected.png")
  grDevices::png(tr_file, width = 1800, height = 1200, res = 200)
  plot(imp_trace, trace_vars)
  grDevices::dev.off()
}

png
2

message("Using memory-safe observed vs imputed plots from mi-diagnostics (no mids densityplot/stripplot).")

```

4.0.2 MI stability across m

```

# Purpose: diagnostics mi stability.
stopifnot(exists("imp"))
stopifnot(exists("mi_logistic_ps_abg_list"), exists("mi_logistic_ps_vbg_list"))
library(dplyr)

subset_mids <- function(imp_obj, m_keep) {
  if (imp_obj$m == m_keep) return(imp_obj)
  idx <- seq_len(m_keep)
  imp_new <- imp_obj
  imp_new$imp <- lapply(imp_obj$imp, function(x) x[, idx, drop = FALSE])

```

```

imp_new$m <- m_keep
imp_new
}

m_vals <- sort(unique(c(20, 50, M_IMP)))
m_vals <- m_vals[m_vals <= imp$m]

med_abg <- median(subset_data$paco2[subset_data$has_abg == 1 & !is.na(subset_data$paco2)], na.rm = TRUE)
med_vbg <- median(subset_data$vbg_co2[subset_data$has_vbg == 1 & !is.na(subset_data$vbg_co2)], na.rm = TRUE)
if (!is.finite(med_abg)) med_abg <- ABG_CO2_REF
if (!is.finite(med_vbg)) med_vbg <- VBG_CO2_REF
grid_abg_info_m <- make_co2_grid_ref("paco2", c(med_abg, ABG_CO2_REF), x_ref_abg, ABG_CO2_REF)
grid_vbg_info_m <- make_co2_grid_ref("vbg_co2", c(med_vbg, VBG_CO2_REF), x_ref_vbg, VBG_CO2_REF)
grid_abg_m <- grid_abg_info_m$grid
grid_vbg_m <- grid_vbg_info_m$grid
ref_idx_abg_m <- grid_abg_info_m$ref_idx
ref_idx_vbg_m <- grid_vbg_info_m$ref_idx
med_idx_abg <- match(med_abg, grid_abg_m$paco2)
med_idx_vbg <- match(med_vbg, grid_vbg_m$vbg_co2)
if (is.na(med_idx_abg)) med_idx_abg <- which.min(abs(grid_abg_m$paco2 - med_abg))
if (is.na(med_idx_vbg)) med_idx_vbg <- which.min(abs(grid_vbg_m$vbg_co2 - med_vbg))

stab_rows <- lapply(m_vals, function(mv) {
  imp_m <- subset_mids(imp, mv)
  get_imp_m <- function(i) normalize_types(mice::complete(imp_m, action = i), levels_ref)

  fits_abg <- lapply(seq_len(mv), function(i) {
    d <- get_imp_m(i)
    d <- d[, c("imv_proc", "has_abg", "paco2", "adj_core"), drop = FALSE]
    tryCatch(
      fit_spline_imp(
        d, get_mi_weight("ABG", i), "imv_proc", "paco2", "has_abg",
        adj_vars = adj_core,
        spline_df = SPLINE_DF, spline_basis = SPLINE_BASIS, grid_df = grid_abg_m,
        ref_idx = ref_idx_abg_m,
        imp_index = i
      ),

```

```

    error = function(e) list(error = conditionMessage(e))
  )
})
collect_warnings_from_list(fits_abg)
curve_abg <- pool_spline_curve(fits_abg, grid_abg_m, ref_idx_abg_m, ABG_CO2_REF,
                                min_ok_frac = 0.9)

fits_vbg <- lapply(seq_len(mv), function(i) {
  d <- get_imp_m(i)
  d <- d[, c("imv_proc", "has_vbg", "vbg_co2", adj_core), drop = FALSE]
  tryCatch(
    fit_spline_imp(
      d, get_mi_weight("V ро", i), "imv_proc", "vbg_co2", "has_vbg",
      adj_vars = adj_core,
      spline_df = SPLINE_DF, spline_basis = SPLINE_BASIS, grid_df = grid_vbg_m,
      ref_idx = ref_idx_vbg_m,
      imp_index = i
    ),
    error = function(e) list(error = conditionMessage(e))
  )
})
collect_warnings_from_list(fits_vbg)
curve_vbg <- pool_spline_curve(fits_vbg, grid_vbg_m, ref_idx_vbg_m, VBG_CO2_REF,
                                min_ok_frac = 0.9)

bind_rows(
  tibble::tibble(group = "ABG", m = mv, OR = curve_abg$OR[med_idx_abg]),
  tibble::tibble(group = "V ро", m = mv, OR = curve_vbg$OR[med_idx_vbg])
)
})

stab_df <- bind_rows(stab_rows)
ref <- stab_df |> filter(m == max(m_vals)) |> select(group, OR) |> rename(OR_ref = OR)
stab_df <- stab_df |>
  left_join(ref, by = "group") |>
  mutate(abs_diff = OR - OR_ref,
        pct_diff = 100 * (OR - OR_ref) / OR_ref)

```

```

stab_file <- results_path("mi_m_stability.csv")
write_csv_safely(stab_df, stab_file, row_names = FALSE)

# FMI / relative efficiency for a representative unweighted model
append_mem_snapshot("diagnostics", "mi_fmi_glm", "pre")
# This diagnostic previously used with(imp, glm(...)) which can materialize large
# completed-data objects. We pool a single coefficient via pool.scalar() instead.
fmi_formula <- imv_proc ~ has_abg + age_at_encounter + sex
fmi_vars <- all.vars(fmi_formula)
qhat <- rep(NA_real_, imp$m)
uhat <- rep(NA_real_, imp$m)
for (i in seq_len(imp$m)) {
  d_i <- mice:::complete(imp, action = i)
  d_i <- d_i[, intersect(fmi_vars, names(d_i)), drop = FALSE]
  if (!all(c("imv_proc", "has_abg", "age_at_encounter", "sex") %in% names(d_i))) {
    next
  }
  cap_fit <- capture_warnings(
    glm(
      formula = fmi_formula,
      data = d_i,
      family = binomial(),
      model = FALSE,
      x = FALSE,
      y = FALSE
    ),
    context = make_context(
      stage = "diagnostics",
      component = "mi_fmi_glm",
      analysis_variant = "mi",
      model_type = "glm",
      group = NA_character_,
      outcome = "imv_proc",
      imputation = i,
      batch = NA_integer_
    )
  )
}

```

```

)
append_warnings(cap_fit$warnings)
fit_i <- cap_fit$value
if (!is.null(fit_i)) {
  coef_i <- stats::coef(fit_i)
  vcov_i <- tryCatch(stats::vcov(fit_i), error = function(e) NULL)
  if (
    "has_abg" %in% names(coef_i) &&
    !is.null(vcov_i) &&
    "has_abg" %in% rownames(vcov_i) &&
    "has_abg" %in% colnames(vcov_i)
  ) {
    qhat[i] <- unname(coef_i[["has_abg"]])
    uhat[i] <- unname(vcov_i["has_abg", "has_abg"])
  }
}
rm(list = intersect(c("d_i", "cap_fit", "fit_i", "coef_i", "vcov_i"), ls()))
invisible(gc(FALSE))
}
ok <- is.finite(qhat) & is.finite(uhat) & uhat > 0
if (sum(ok) >= 2) {
  ps <- mice::pool.scalar(
    Q = qhat[ok],
    U = uhat[ok],
    n = sum(subset_data$has_abg == 1, na.rm = TRUE),
    k = 1
  )
  fmi_abg <- tibble::tibble(
    term = "has_abg",
    estimate = ps$qbar,
    std.error = sqrt(ps$t),
    fmi = ps$fmi,
    m_used = sum(ok)
  )
} else {
  warning("FMI not available from pooled scalar diagnostic; leaving FMI as NA.", call. = FALSE)
  fmi_abg <- tibble::tibble(

```

```

    term = "has_abg",
    estimate = NA_real_,
    std.error = NA_real_,
    fmi = NA_real_,
    m_used = sum(ok)
)
}

fmi_abg$rel_eff <- if (is.finite(fmi_abg$fmi[1])) 1 / (1 + fmi_abg$fmi[1] / M_IMP) else NA_real_
append_mem_snapshot("diagnostics", "mi_fmi_glm", "post")

```

4.0.3 MI maxit sensitivity (sampled)

```

# Purpose: diagnostics mi maxit.
run_maxit_sensitivity <- TRUE
if (run_maxit_sensitivity) {
  stopifnot(exists("mi_df"), exists("meth"), exists("pred"))
  set.seed(MI_SEED)
  idx <- sample(seq_len(nrow(mi_df)), min(2000, nrow(mi_df)))
  mi_df_sens <- mi_df[idx, , drop = FALSE]

  m_sens <- min(20, M_IMP)
  maxit_short <- max(5, floor(MAXIT_MI / 2))

  cap_short <- capture_warnings(
    mice::mice(
      data           = mi_df_sens,
      m              = m_sens,
      maxit         = maxit_short,
      predictorMatrix = pred,
      method        = meth,
      printFlag     = FALSE,
      seed          = MI_SEED
    ),
    context = make_context(
      stage = "MI",
      component = "mice_maxit_short",
    )
  )
}
```

```

analysis_variant = "mi",
model_type = "mice",
group = NA_character_,
outcome = NA_character_,
imputation = NA_integer_,
batch = NA_integer_
)
)
append_warnings(cap_short$warnings)
imp_short <- cap_short$value
le_short <- as.data.frame(imp_short$loggedEvents)
if (nrow(le_short)) {
  le_short$run_type <- "maxit_short"
}
write_csv_safely(le_short, results_path("mice_logged_events_maxit_short.csv"), row_names = FALSE)

cap_long <- capture_warnings(
  mice::mice(
    data           = mi_df_sens,
    m              = m_sens,
    maxit         = MAXIT_MI,
    predictorMatrix = pred,
    method        = meth,
    printFlag     = FALSE,
    seed          = MI_SEED + 1
  ),
  context = make_context(
    stage = "MI",
    component = "mice_maxit_long",
    analysis_variant = "mi",
    model_type = "mice",
    group = NA_character_,
    outcome = NA_character_,
    imputation = NA_integer_,
    batch = NA_integer_
  )
)

```

```

append_warnings(cap_long$warnings)
imp_long <- cap_long$value
le_long <- as.data.frame(imp_long$loggedEvents)
if (nrow(le_long)) {
  le_long$run_type <- "maxit_long"
}
write_csv_safely(le_long, results_path("mice_logged_events_maxit_long.csv"), row_names = FALSE)

mean_across_imps <- function(imp_obj, var) {
  means <- vapply(seq_len(imp_obj$m), function(i) {
    d <- mice::complete(imp_obj, action = i)
    mean(d[[var]], na.rm = TRUE)
  }, numeric(1))
  mean(means, na.rm = TRUE)
}

key_vars <- intersect(c("curr_bmi", "serum_hco3", "sodium", "serum_cr"), names(mi_df_sens))
sens_df <- lapply(key_vars, function(v) {
  m_short <- mean_across_imps(imp_short, v)
  m_long <- mean_across_imps(imp_long, v)
  data.frame(
    variable = v,
    mean_short = m_short,
    mean_long = m_long,
    diff = m_long - m_short,
    stringsAsFactors = FALSE
  )
}) |> bind_rows()

sens_file <- results_path("mi_maxit_sensitivity.csv")
write_csv_safely(sens_df, sens_file, row_names = FALSE)
}

# Purpose: diagnostics weights.
stopifnot(all(c("w_abg", "w_vbg") %in% names(subset_data)))

wt_abg <- subset_data$w_abg[subset_data$has_abg == 1]

```

```

wt_vbg <- subset_data$w_vbg[subset_data$has_vbg == 1]
ps_abg <- subset_data$ps_abg[subset_data$has_abg == 1]
ps_vbg <- subset_data$ps_vbg[subset_data$has_vbg == 1]
trunc_abg <- subset_data$trunc_abg[subset_data$has_abg == 1]
trunc_vbg <- subset_data$trunc_vbg[subset_data$has_vbg == 1]

wt_sum <- bind_rows(
  weight_summary(wt_abg, ps = ps_abg, ps_floor = ps_floor_abg,
                 truncated = trunc_abg) |>
    mutate(group = "ABG"),
  weight_summary(wt_vbg, ps = ps_vbg, ps_floor = ps_floor_vbg,
                 truncated = trunc_vbg) |>
    mutate(group = "VBG")
)
wt_sum_file <- results_path("weight_summary.csv")
write_csv_safely(wt_sum, wt_sum_file, row_names = FALSE)

ps_stat <- function(ps, group) {
  ps_ok <- ps[is.finite(ps)]
  if (!length(ps_ok)) {
    return(data.frame(
      group = group,
      ps_min = NA_real_,
      ps_p01 = NA_real_,
      ps_p50 = NA_real_,
      ps_p99 = NA_real_,
      ps_max = NA_real_,
      stringsAsFactors = FALSE
    ))
  }
  data.frame(
    group = group,
    ps_min = min(ps_ok, na.rm = TRUE),
    ps_p01 = stats::quantile(ps_ok, 0.01, na.rm = TRUE),
    ps_p50 = stats::median(ps_ok, na.rm = TRUE),
    ps_p99 = stats::quantile(ps_ok, 0.99, na.rm = TRUE),
    ps_max = max(ps_ok, na.rm = TRUE),
  )
}

```

```

    stringsAsFactors = FALSE
  )
}

ps_summary <- dplyr::bind_rows(
  ps_stat(ps_abg, "ABG"),
  ps_stat(ps_vbg, "VBG")
)
write_csv_safely(ps_summary, results_path("ps_overlap_summary.csv"), row_names = FALSE)
render_table_pdf_maybe(
  ps_summary,
  caption = "Propensity score overlap (tested cohort)",
  file_stub = "ps_overlap_summary",
  digits = 3,
  show = SHOW_LOW_VALUE_TABLES
)

wt_df <- bind_rows(
  tibble::tibble(group = "ABG", ps = ps_abg, weight = wt_abg),
  tibble::tibble(group = "VBG", ps = ps_vbg, weight = wt_vbg)
) |>
  filter(is.finite(weight), is.finite(ps))

plot_hist <- function(df, title) {
  ggplot(df, aes(x = weight)) +
    geom_histogram(bins = 40, fill = "grey70", color = "white") +
    labs(title = title, x = "Weight", y = "Count") +
    theme_minimal(base_size = 10)
}

p_hist_abg <- plot_hist(filter(wt_df, group == "ABG"), "ABG weight distribution")
p_hist_vbg <- plot_hist(filter(wt_df, group == "VBG"), "VBG weight distribution")
save_diag_plot(p_hist_abg, results_path("figs", "diag-wt-weights-hist-abg.png"), width = 7, height = 5)
save_diag_plot(p_hist_vbg, results_path("figs", "diag-wt-weights-hist-vbg.png"), width = 7, height = 5)

plot_scatter <- function(df, title) {
  top_wt <- df |>

```

```

    slice_max(order_by = weight, n = min(20, nrow(df)))
  ggplot(df, aes(x = ps, y = weight)) +
    geom_point(alpha = 0.3, size = 0.7) +
    geom_point(data = top_wt, color = "red", size = 1.2) +
    scale_y_log10() +
    labs(title = title, x = "Propensity score", y = "Weight (log10)") +
    theme_minimal(base_size = 10)
}

p_scatter_abg <- plot_scatter(filter(wt_df, group == "ABG"),
                               "ABG weights vs propensity (top 20 highlighted)")
p_scatter_vbg <- plot_scatter(filter(wt_df, group == "VBG"),
                               "VBG weights vs propensity (top 20 highlighted)")
save_diag_plot(p_scatter_abg, results_path("figs", "diag-wt-weights-vs-ps-abg.png"), width = 7, height = 5)
save_diag_plot(p_scatter_vbg, results_path("figs", "diag-wt-weights-vs-ps-vbg.png"), width = 7, height = 5)

```

4.0.4 Balance diagnostics

```

# Purpose: diagnostics balance.
stopifnot(exists("target_balance_table"))

covars_use <- intersect(covars_ps, names(subset_data))

bal_target_abg <- target_balance_table(subset_data, "has_abg", subset_data$w_abg, covars_use, levels_ref) |>
  mutate(group = "ABG")
bal_target_vbg <- target_balance_table(subset_data, "has_vbg", subset_data$w_vbg, covars_use, levels_ref) |>
  mutate(group = "VBG")

bal_target <- bind_rows(bal_target_abg, bal_target_vbg)
write_csv_safely(bal_target, results_path("balance_table.csv"), row_names = FALSE)

bal_target_sum <- bal_target |>
  group_by(group, variable) |>
  summarise(
    max_abs_pre = max(abs(smd_pre), na.rm = TRUE),
    max_abs_post = max(abs(smd_post), na.rm = TRUE),
    sum_abs = sum(abs(smd))
  )

```

```

.groups = "drop"
)
bal_worst <- bal_target_sum |>
  group_by(group) |>
  arrange(desc(max_abs_post)) |>
  ungroup()
write_csv_safely(bal_worst, results_path("balance_worst10.csv"), row_names = FALSE)
render_table_pdf_maybe(
  bal_worst,
  caption = "Target balance by covariate (sorted by max |SMD|)",
  file_stub = "balance_worst",
  digits = 3,
  show = SHOW_LOW_VALUE_TABLES
)

bal_plot_df <- bal_target |>
  mutate(label = ifelse(is.na(level), variable, paste0(variable, ":", level))) |>
  tidyr::pivot_longer(
    cols = c(smd_pre, smd_post),
    names_to = "stage",
    values_to = "smd"
  ) |>
  mutate(stage = recode(stage, smd_pre = "Pre", smd_post = "Post"))

p_bal_abg <- ggplot(filter(bal_plot_df, group == "ABG"),
  aes(x = reorder(label, abs(smd)), y = smd, color = stage, shape = stage)) +
  geom_point(size = 1) +
  geom_hline(yintercept = c(-0.1, 0.1), linetype = 2, linewidth = 0.3) +
  geom_hline(yintercept = c(-0.05, 0.05), linetype = 3, linewidth = 0.3) +
  coord_flip() +
  labs(x = NULL, y = "Target SMD", title = "ABG target balance") +
  theme_minimal(base_size = 10)

p_bal_vbg <- ggplot(filter(bal_plot_df, group == "VBG"),
  aes(x = reorder(label, abs(smd)), y = smd, color = stage, shape = stage)) +
  geom_point(size = 1) +
  geom_hline(yintercept = c(-0.1, 0.1), linetype = 2, linewidth = 0.3) +

```

```

geom_hline(yintercept = c(-0.05, 0.05), linetype = 3, linewidth = 0.3) +
coord_flip() +
labs(x = NULL, y = "Target SMD", title = "VBG target balance") +
theme_minimal(base_size = 10)

save_diag_plot(p_bal_abg, results_path("figs", "diag-balance-loveplot-abg.png"), width = 9, height = 7)
save_diag_plot(p_bal_vbg, results_path("figs", "diag-balance-loveplot-vbg.png"), width = 9, height = 7)

# MI target balance summaries across imputations
stopifnot(exists("bal_imp_abg"), exists("bal_imp_vbg"))
bal_imp <- bind_rows(bal_imp_abg$bal_long, bal_imp_vbg$bal_long)
bal_imp_summary <- bind_rows(bal_imp_abg$bal_imp_summary, bal_imp_vbg$bal_imp_summary)
worst_rows <- bind_rows(bal_imp_abg$worst_rows_overall, bal_imp_vbg$worst_rows_overall)
worst_by_imp <- bind_rows(bal_imp_abg$worst_by_imp, bal_imp_vbg$worst_by_imp)
worst_terms_by_imp <- bind_rows(bal_imp_abg$worst_terms_by_imp, bal_imp_vbg$worst_terms_by_imp)

write_csv_safely(bal_imp, results_path("balance_target_by_imp.csv"), row_names = FALSE)
write_csv_safely(bal_imp_summary, results_path("balance_target_imp_summary.csv"), row_names = FALSE)
write_csv_safely(worst_rows, results_path("balance_target_worst_rows.csv"), row_names = FALSE)
render_table_pdf_maybe(
  worst_rows,
  caption = "Target SMD rows across imputations (sorted by |SMD|)",
  file_stub = "balance_target_worst_rows",
  digits = 3,
  show = SHOW_LOW_VALUE_TABLES
)
write_csv_safely(worst_by_imp, results_path("balance_max_smd_by_imp.csv"), row_names = FALSE)
write_csv_safely(worst_terms_by_imp, results_path("balance_worst_terms.csv"), row_names = FALSE)
if (any(bal_imp_summary$max_abs_post > 0.10, na.rm = TRUE)) {
  warning("Target balance: max |SMD| > 0.10 in at least one imputation.", call. = FALSE)
}
if (nrow(bal_imp_summary)) {
  dist_tbl <- bal_imp_summary |>
    dplyr::group_by(group) |>
    dplyr::summarise(
      med = median(max_abs_post, na.rm = TRUE),
      iqr = IQR(max_abs_post, na.rm = TRUE),

```

```

    max = max(max_abs_post, na.rm = TRUE),
    .groups = "drop"
)
render_table_pdf_maybe(
  dist_tbl,
  caption = "Distribution of max |Target SMD| across imputations",
  file_stub = "balance_target_max_smd_distribution",
  digits = 3,
  show = SHOW_LOW_VALUE_TABLES
)
}

if (nrow(worst_terms_by_imp)) {
  worst_freq <- worst_terms_by_imp |>
    dplyr::count(group, term, sort = TRUE) |>
    dplyr::ungroup()
  render_table_pdf_maybe(
    worst_freq,
    caption = "Most frequent worst-balance terms",
    file_stub = "balance_worst_terms_freq",
    digits = 0,
    show = SHOW_LOW_VALUE_TABLES
  )
}

n_abg <- sum(subset_data$has_abg == 1, na.rm = TRUE)
n_vbg <- sum(subset_data$has_vbg == 1, na.rm = TRUE)

stopifnot(exists("wt_sum"))
ess_abg <- wt_sum$ess[wt_sum$group == "ABG"]
ess_vbg <- wt_sum$ess[wt_sum$group == "VBG"]

if (is.finite(ess_abg) && is.finite(n_abg) && ess_abg < 0.2 * n_abg) {
  warning("ABG balance: ESS < 0.2 * n_abg (", round(ess_abg, 1), " vs ", n_abg, ".)", call. = FALSE)
}
if (is.finite(ess_vbg) && is.finite(n_vbg) && ess_vbg < 0.2 * n_vbg) {
  warning("VBG balance: ESS < 0.2 * n_vbg (", round(ess_vbg, 1), " vs ", n_vbg, ".)", call. = FALSE)
}

```

4.0.5 Outcome diagnostics

```
# Purpose: diagnostics outcome.
stopifnot(exists("abg_curves"), exists("vbg_curves"))
curve_abg <- abg_curves |>
  mutate(co2 = paco2) |>
  select(-paco2)
curve_vbg <- vbg_curves |>
  mutate(co2 = vbg_co2) |>
  select(-vbg_co2)
curve_all <- bind_rows(curve_abg, curve_vbg) |>
  mutate(outcome = factor(outcome,
                          levels = c("imv_proc", "niv_proc", "death_60d", "hypercap_resp_failure"),
                          labels = c("IMV", "NIV", "Death (60d)", "Hypercapnic RF")))
axis_mi_outcome <- compute_or_axis_spec(list(curve_abg, curve_vbg), lo_col = "LCL", hi_col = "UCL")

p_outcome <- ggplot(curve_all, aes(x = co2, y = OR, color = group, fill = group)) +
  geom_line(linewidth = 0.6) +
  geom_ribbon(aes(ymin = LCL, ymax = UCL), alpha = 0.15, color = NA) +
  geom_hline(yintercept = 1, linetype = "dashed", color = "grey40") +
  or_axis_scale(axis_mi_outcome) +
  facet_wrap(~ outcome, scales = "free_x") +
  labs(
    title = "MI-pooled IPSW spline odds ratios: ABG vs VBG",
    x = expression(CO[2]~"(mmHg)"),
    y = paste0("Odds ratio (ref: ABG ", ABG_CO2_REF, ", VBG ", VBG_CO2_REF, "; log scale)"),
    caption = paste0("Reference CO2: ABG ", ABG_CO2_REF, " mmHg; VBG ", VBG_CO2_REF, " mmHg")
  ) +
  theme_minimal(base_size = 10)
save_diag_plot(p_outcome, results_path("figs", "diag-outcome-or-vs-paco2.png"),
               width = 10, height = 8)
```

4.0.6 Diagnostics summary and audit

```
# Purpose: diagnostics summary.
stopifnot(exists("bal_imp_summary"), exists("wt_sum"), exists("chain_diag_stats"))
```

```

target_abg_med <- median(bal_imp_summary$max_abs_post[bal_imp_summary$group == "ABG"], na.rm = TRUE)
target_abg_max <- max(bal_imp_summary$max_abs_post[bal_imp_summary$group == "ABG"], na.rm = TRUE)
target_vbg_med <- median(bal_imp_summary$max_abs_post[bal_imp_summary$group == "VBG"], na.rm = TRUE)
target_vbg_max <- max(bal_imp_summary$max_abs_post[bal_imp_summary$group == "VBG"], na.rm = TRUE)

runtime_proj_total_hrs <- NA_real_
if (is.finite(PILOT_FRAC) && PILOT_FRAC > 0 && PILOT_FRAC < 1) {
  runtime_log_curr <- runtime_log
  runtime_log_curr <- runtime_log_curr[runtime_log_curr$run_id == runtime_run_id, , drop = FALSE]
  scalable_steps <- c("mice_imputation", "mi_single_pass")
  scalable_secs <- runtime_log_curr |>
    dplyr::filter(step_name %in% scalable_steps) |>
    dplyr::summarise(total = sum(seconds, na.rm = TRUE)) |>
    dplyr::pull(total)
  if (length(scalable_secs) && is.finite(scalable_secs)) {
    runtime_proj_total_hrs <- scalable_secs / PILOT_FRAC / 3600
  }
}

get_wt_row <- function(group) {
  row <- wt_sum[wt_sum$group == group, , drop = FALSE]
  if (nrow(row) == 0L) {
    stop("Weight summary missing for group: ", group)
  }
  row[, c("ps_floor", "p01", "p05", "p95", "p99", "max", "sum_w", "ess",
         "top01_weight_share", "trunc_rate"), drop = FALSE]
}

wt_abg_row <- get_wt_row("ABG")
wt_vbg_row <- get_wt_row("VBG")

diag_summary <- bind_rows(
  tibble::tibble(
    block = "ABG weights",
    run_mode = RUN_MODE,
    m = M_IMP,
    maxit = MAXIT_MI,

```

```

pilot_frac = PILOT_FRAC,
stop_method = gbm_params$stop.method,
ps_floor_quantile = ps_trunc_quantile,
ps_floor = wt_abg_row$ps_floor,
weight_p01 = wt_abg_row$p01,
weight_p05 = wt_abg_row$p05,
weight_p95 = wt_abg_row$p95,
weight_p99 = wt_abg_row$p99,
weight_max = wt_abg_row$max,
sum_w = wt_abg_row$sum_w,
ess = wt_abg_row$ess,
top1_weight_share = wt_abg_row$top01_weight_share,
trunc_rate = wt_abg_row$trunc_rate,
target_max_smd_post_med = target_abg_med,
target_max_smd_post_max = target_abg_max,
runtime_proj_total_hrs = runtime_proj_total_hrs,
chain_diag_n_imputed_vars = chain_diag_stats$n_imputed_vars,
chain_diag_n_with_chainMean = chain_diag_stats$n_with_chainMean,
chain_diag_n_with_drift_tail = chain_diag_stats$n_with_drift_tail,
chain_diag_drift_tail_na_frac = chain_diag_stats$drift_tail_na_frac,
chain_diag_tail_window_na_mean = chain_diag_stats$tail_window_na_mean
),
tibble::tibble(
  block = "VBG weights",
  run_mode = RUN_MODE,
  m = M_IMP,
  maxit = MAXIT_MI,
  pilot_frac = PILOT_FRAC,
  stop_method = gbm_params$stop.method,
  ps_floor_quantile = ps_trunc_quantile,
  ps_floor = wt_vbg_row$ps_floor,
  weight_p01 = wt_vbg_row$p01,
  weight_p05 = wt_vbg_row$p05,
  weight_p95 = wt_vbg_row$p95,
  weight_p99 = wt_vbg_row$p99,
  weight_max = wt_vbg_row$max,
  sum_w = wt_vbg_row$sum_w,

```

```

ess = wt_vbg_row$ess,
top1_weight_share = wt_vbg_row$top01_weight_share,
trunc_rate = wt_vbg_row$trunc_rate,
target_max_smd_post_med = target_vbg_med,
target_max_smd_post_max = target_vbg_max,
runtime_proj_total_hrs = runtime_proj_total_hrs,
chain_diag_n_imputed_vars = chain_diag_stats$n_imputed_vars,
chain_diag_n_with_chainMean = chain_diag_stats$n_with_chainMean,
chain_diag_n_with_drift_tail = chain_diag_stats$n_with_drift_tail,
chain_diag_drift_tail_na_frac = chain_diag_stats$drift_tail_na_frac,
chain_diag_tail_window_na_mean = chain_diag_stats$tail_window_na_mean
),
tibble::tibble(
  block = "ABG outcomes",
  run_mode = RUN_MODE,
  m = M_IMP,
  maxit = MAXIT_MI,
  pilot_frac = PILOT_FRAC,
  stop_method = gbm_params$stop.method,
  ps_floor_quantile = ps_trunc_quantile,
  ps_floor = wt_abg_row$ps_floor,
  weight_p01 = wt_abg_row$p01,
  weight_p05 = wt_abg_row$p05,
  weight_p95 = wt_abg_row$p95,
  weight_p99 = wt_abg_row$p99,
  weight_max = wt_abg_row$max,
  sum_w = wt_abg_row$sum_w,
  ess = wt_abg_row$ess,
  top1_weight_share = wt_abg_row$top01_weight_share,
  trunc_rate = wt_abg_row$trunc_rate,
  target_max_smd_post_med = target_abg_med,
  target_max_smd_post_max = target_abg_max,
  runtime_proj_total_hrs = runtime_proj_total_hrs,
  chain_diag_n_imputed_vars = chain_diag_stats$n_imputed_vars,
  chain_diag_n_with_chainMean = chain_diag_stats$n_with_chainMean,
  chain_diag_n_with_drift_tail = chain_diag_stats$n_with_drift_tail,
  chain_diag_drift_tail_na_frac = chain_diag_stats$drift_tail_na_frac,
)

```

```

chain_diag_tail_window_na_mean = chain_diag_stats$tail_window_na_mean
),
tibble::tibble(
  block = "VBG outcomes",
  run_mode = RUN_MODE,
  m = M_IMP,
  maxit = MAXIT_MI,
  pilot_frac = PILOT_FRAC,
  stop_method = gbm_params$stop.method,
  ps_floor_quantile = ps_trunc_quantile,
  ps_floor = wt_vbg_row$ps_floor,
  weight_p01 = wt_vbg_row$p01,
  weight_p05 = wt_vbg_row$p05,
  weight_p95 = wt_vbg_row$p95,
  weight_p99 = wt_vbg_row$p99,
  weight_max = wt_vbg_row$max,
  sum_w = wt_vbg_row$sum_w,
  ess = wt_vbg_row$ess,
  top1_weight_share = wt_vbg_row$top01_weight_share,
  trunc_rate = wt_vbg_row$trunc_rate,
  target_max_smd_post_med = target_vbg_med,
  target_max_smd_post_max = target_vbg_max,
  runtime_proj_total_hrs = runtime_proj_total_hrs,
  chain_diag_n_imputed_vars = chain_diag_stats$n_imputed_vars,
  chain_diag_n_with_chainMean = chain_diag_stats$n_with_chainMean,
  chain_diag_n_with_drift_tail = chain_diag_stats$n_with_drift_tail,
  chain_diag_drift_tail_na_frac = chain_diag_stats$drift_tail_na_frac,
  chain_diag_tail_window_na_mean = chain_diag_stats$tail_window_na_mean
)
)

check_abg <- bal_imp_summary |>
  dplyr::filter(group == "ABG") |>
  dplyr::summarise(x = max(max_abs_post, na.rm = TRUE), .groups = "drop") |>
  dplyr::pull(x)
check_vbg <- bal_imp_summary |>
  dplyr::filter(group == "VBG") |>

```

```

dplyr::summarise(x = max(max_abs_post, na.rm = TRUE), .groups = "drop") |>
  dplyr::pull(x)
diag_abg <- diag_summary$target_max_smd_post_max[diag_summary$block == "ABG weights"]
diag_vbg <- diag_summary$target_max_smd_post_max[diag_summary$block == "VBG weights"]
if (length(diag_abg) && is.finite(check_abg)) {
  stopifnot(isTRUE(all.equal(check_abg, diag_abg, tolerance = 1e-8)))
}
if (length(diag_vbg) && is.finite(check_vbg)) {
  stopifnot(isTRUE(all.equal(check_vbg, diag_vbg, tolerance = 1e-8)))
}
stopifnot(exists("covars_use_abg"), exists("covars_use_vbg"))
stopifnot(setequal(covars_use_abg, covars_use_vbg))

diag_summary_file <- results_path("diagnostics_summary.csv")
write_csv_safely(diag_summary, diag_summary_file, row_names = FALSE)

diag_summary_display <- diag_summary |>
  dplyr::select(
    block,
    m,
    ess,
    trunc_rate,
    target_max_smd_post_med,
    target_max_smd_post_max
  ) |>
  dplyr::rename(
    `trunc` = trunc_rate,
    `med_max_smd` = target_max_smd_post_med,
    `max_max_smd` = target_max_smd_post_max
  )
render_table_pdf_maybe(diag_summary_display,
                      "Diagnostics summary (IPSW + MI)",
                      "diagnostics_summary_display",
                      wide = TRUE,
                      digits = 3,
                      show = SHOW_LOW_VALUE_TABLES)

```

```

audit_lines <- c(
  "# Diagnostics Audit",
  "",
  "## A1. MI workflow",
  "- Impute -> single-pass per-imputation loop (weights, target balance, 3-level outcomes, spline outcomes) -> pool
  ↵ curves and coefficients with Rubin's rules (chunk: `mi-single-pass`; downstream MI display chunks read those
  ↵ objects).",
  "## A2. MI settings",
  paste0("- m = ", M_IMP, ", maxit = ", MAXIT_MI, ", seed = ", MI_SEED,
         "; treatments/outcomes/PaC02/VBG C02 are not imputed but are predictors (`mi-exec`)."),
  "## A3. Propensity weighting",
  "- Unimputed weighting uses WeightIt with method = \"gbm\" and balance-based stopping (stop.method = \"smd.max\");
  ↵ no AUC-based tuning (`propensity-config`, `ipw-abg-weighting`, `ipw-vbg-workflow`).",
  "- MI weighting uses logistic PS with restricted cubic splines (glm + rcs); no SHAP is computed for MI.",
  "## A4. One-sided IPSW + truncation",
  "- Weights are 1/ps for observed tests (ABG or VBG), truncated only for very small propensities (ps floor = 1st
  ↵ percentile), then stabilized by the mean; controls receive weight 1 for balance diagnostics only.",
  "## A5. Robust variance",
  "- Outcome models are survey::svyglm with svydesign (robust SEs), using spline(C02) + X adjustment; ABG and VBG are
  ↵ fit separately within the measured cohort.",
  "## A6. Pooling",
  "- mitools::MIcombine pools coefficients and robust vcov from svyglm; spline curves are pooled pointwise on the
  ↵ log-OR scale relative to C02_ref (helpers defined in `mi-single-pass`).",
  "",
  "## Potential mismatches / risks",
  "- Target balance diagnostics compare weighted treated cohort to the full analytic sample (no treated-vs-control
  ↵ balance).",
  "- MI stability across m uses subsets of the first m imputations from the main mids object (not full re-imputation
  ↵ at each m).",
  "- Unweighted analyses remain earlier in the notebook for context; primary inference is based on weighted spline
  ↵ models."
)
audit_file <- results_path("diagnostics_audit.md")
write_diag_lines(audit_lines, audit_file)

```

4.0.7 Performance / runtime log

```
# Purpose: warnings summary.
stopifnot(exists("mi_warn_log"), exists("mi_info_log"))
warn_df <- mi_warn_log
write_csv_safely(warn_df, results_path("mi_warnings_log.csv"), row_names = FALSE)
write_csv_safely(warn_df, results_path("warnings_log.csv"), row_names = FALSE)
write_csv_safely(mi_info_log, results_path("mi_info_log.csv"), row_names = FALSE)

if (nrow(warn_df)) {
  msg_counts <- warn_df |>
    dplyr::count(stage, component, message, sort = TRUE)
  render_table_pdf_maybe(
    msg_counts,
    caption = "Warning messages by stage/component",
    file_stub = "warnings_by_stage_component",
    digits = 0,
    show = SHOW_LOW_VALUE_TABLES
  )

  nonconv <- warn_df |>
    dplyr::filter(grepl("glm.fit: algorithm did not converge", message, fixed = TRUE))
  if (nrow(nonconv)) {
    nonconv_ctx <- nonconv |>
      dplyr::count(stage, component, analysis_variant, model_type, outcome, group, imputation, sort = TRUE)
    render_table_pdf_maybe(
      nonconv_ctx,
      caption = "Nonconvergence contexts",
      file_stub = "nonconvergence_contexts",
      digits = 0,
      show = SHOW_LOW_VALUE_TABLES
    )
  }
} else {
  message("No captured warnings.")
}
```

```

write_csv_safely(plot_drop_log(), results_path("plot_drop_log.csv"), row_names = FALSE)

# Purpose: mi nonconvergence summary.
stopifnot(exists("mi_outcome_diag"))
if (nrow(mi_outcome_diag) == 0L) {
  stop("mi_outcome_diag is empty; outcome diagnostics were not captured.")
}
out_diag_file <- results_path("model_fit_diagnostics.csv")
write_csv_safely(mi_outcome_diag, out_diag_file)
write_csv_safely(mi_outcome_diag, results_path("mi_outcome_fit_diagnostics.csv"))

nonconv <- mi_outcome_diag |>
  dplyr::filter(isTRUE(nonconv_flag) | isFALSE(converged))
sep <- mi_outcome_diag |>
  dplyr::filter(isTRUE(sep_flag))

if (nrow(nonconv)) {
  nonconv_counts <- nonconv |>
    dplyr::count(analysis_variant, group, outcome, model_type, sort = TRUE)
  render_table_pdf_maybe(
    nonconv_counts,
    caption = "Nonconverged fits by variant/group/outcome",
    file_stub = "nonconverged_fits_by_variant",
    digits = 0,
    show = SHOW_LOW_VALUE_TABLES
  )

  worst_imps <- nonconv |>
    dplyr::count(imputation, sort = TRUE)
  render_table_pdf_maybe(
    worst_imps,
    caption = "Imputations with most nonconverged fits",
    file_stub = "nonconverged_imputations",
    digits = 0,
    show = SHOW_LOW_VALUE_TABLES
  )
} else {

```

```

    message("No nonconverged outcome fits recorded.")
}

if (nrow(sep)) {
  sep_counts <- sep |>
    dplyr::count(analysis_variant, group, outcome, model_type, sort = TRUE)
  render_table_pdf_maybe(
    sep_counts,
    caption = "Separation flags by variant/group/outcome",
    file_stub = "separation_flags_by_variant",
    digits = 0,
    show = SHOW_LOW_VALUE_TABLES
  )
} else {
  message("No separation flags recorded.")
}

worst_phat <- mi_outcome_diag |>
  dplyr::filter(is.finite(min_phat) | is.finite(max_phat)) |>
  dplyr::mutate(extreme = pmin(min_phat, 1 - max_phat, na.rm = TRUE)) |>
  dplyr::arrange(extreme)
if (nrow(worst_phat)) {
  render_table_pdf_maybe(
    worst_phat,
    caption = "Worst fitted-probability extremes",
    file_stub = "worst_phat_extremes",
    digits = 3,
    show = SHOW_LOW_VALUE_TABLES
  )
}

diag_df <- mi_outcome_diag
diag_df$sep_warn <- grep("fitted probabilities numerically 0 or 1",
                           diag_df$top_warning, fixed = TRUE) |
  grep("separat", diag_df$top_warning, ignore.case = TRUE)
diag_df$phat_extreme <- (diag_df$min_phat < PROB_EPS) |
  (diag_df$max_phat > 1 - PROB_EPS)

```

```

diag_df$sep_flag <- dplyr::coalesce(as.logical(diag_df$sep_flag), FALSE)
diag_df$nonconv_flag <- dplyr::coalesce(as.logical(diag_df$nonconv_flag), FALSE) |
  dplyr::coalesce(!as.logical(diag_df$converged), FALSE)

mi_variants <- intersect(c("mi_ipw", "weighted_imputed"), unique(diag_df$analysis_variant))
if (!length(mi_variants)) mi_variants <- unique(diag_df$analysis_variant)

issue_summary <- diag_df |>
  dplyr::filter(analysis_variant %in% mi_variants) |>
  dplyr::group_by(group, outcome, component, analysis_variant) |>
  dplyr::summarise(
    n_fits = dplyr::n(),
    n_nonconv = sum(nonconv_flag %in% TRUE, na.rm = TRUE),
    n_sep_warn = sum(sep_warn %in% TRUE, na.rm = TRUE),
    n_phat_extreme = sum(phat_extreme %in% TRUE, na.rm = TRUE),
    n_sep_flag = sum(sep_flag %in% TRUE, na.rm = TRUE),
    .groups = "drop"
  )
write_csv_safely(issue_summary, results_path("mi_fit_issue_summary.csv"))

# Top warning messages by stage/component
if (nrow(mi_warn_log)) {
  warn_top <- mi_warn_log |>
    dplyr::count(stage, component, analysis_variant, model_type, message, sort = TRUE)
  render_table_pdf_maybe(
    warn_top,
    caption = "Warnings by stage/component",
    file_stub = "warnings_by_stage_component_full",
    digits = 0,
    show = SHOW_LOW_VALUE_TABLES
  )
}

# Top MICE loggedEvents drivers
stopifnot(exists("log_events_summary"))
if (nrow(log_events_summary)) {
  render_table_pdf_maybe(

```

```

log_events_summary,
caption = "MICE loggedEvents drivers",
file_stub = "mice_logged_events_drivers",
digits = 0,
show = SHOW_LOW_VALUE_TABLES
)
}

# Replay debug mode (off by default)
MI_DEBUG_REPLAY <- FALSE
if (MI_DEBUG_REPLAY) {
  stopifnot(file.exists(results_path("model_fit_diagnostics.csv")))
  diag_df <- read.csv(results_path("model_fit_diagnostics.csv"), stringsAsFactors = FALSE)
  bad <- diag_df[isFALSE(diag_df$converged), , drop = FALSE]
  if (nrow(bad)) {
    row <- bad[1, , drop = FALSE]
    message("Replaying: group=", row$group, ", outcome=", row$outcome,
            ", model_type=", row$model_type, ", imputation=", row$imputation)
    options(warn = 1)
    stopifnot(exists("imp"))
    get_imp <- function(i, imp_obj = imp) {
      normalize_types(mice::complete(imp_obj, action = i), levels_ref)
    }
    d_dbg <- get_imp(row$imputation)
    if (row$group == "ABG") {
      keep_dbg <- d_dbg$has_abg == 1 & is.finite(d_dbg$paco2)
      w_dbg <- get_mi_weight("ABG", row$imputation)[keep_dbg]
      d_dbg <- d_dbg[keep_dbg, , drop = FALSE]
      if (row$model_type == "spline") {
        d_dbg <- d_dbg[, c(row$outcome, "has_abg", "paco2", adj_core), drop = FALSE]
        fit_spline_imp(d_dbg, w_dbg, row$outcome, "paco2", "has_abg",
                      adj_vars = adj_core, spline_df = SPLINE_DF, spline_basis = SPLINE_BASIS,
                      grid_df = NULL, ref_idx = NULL, imp_index = row$imputation)
      } else {
        d_dbg$co2_cat <- make_co2_cat3(d_dbg$paco2, ABG_CO2_LOW, ABG_CO2_HIGH)
        d_dbg$co2_cat <- stats::relevel(base::droplevels(d_dbg$co2_cat), ref = "Normal")
        d_dbg[[row$outcome]] <- to01(d_dbg[[row$outcome]])
      }
    }
  }
}

```

```

des <- survey::svydesign(ids = ~1, weights = ~w_dbg, data = d_dbg)
fml <- stats::reformulate(c("co2_cat", adj_core), response = row$outcome)
survey::svyglm(fml, design = des, family = quasibinomial(),
               control = stats::glm.control(maxit = 50))
}
} else if (row$group == "VBG") {
  keep_dbg <- d_dbg$has_vbg == 1 & is.finite(d_dbg$vbg_co2)
  w_dbg <- get_mi_weight("VBG", row$imputation)[keep_dbg]
  d_dbg <- d_dbg[keep_dbg, , drop = FALSE]
  if (row$model_type == "spline") {
    d_dbg <- d_dbg[, c(row$outcome, "has_vbg", "vbg_co2", adj_core), drop = FALSE]
    fit_spline_imp(d_dbg, w_dbg, row$outcome, "vbg_co2", "has_vbg",
                   adj_vars = adj_core, spline_df = SPLINE_DF, spline_basis = SPLINE_BASIS,
                   grid_df = NULL, ref_idx = NULL, imp_index = row$imputation)
  } else {
    d_dbg$co2_cat <- make_co2_cat3(d_dbg$vbg_co2, VBG_CO2_LOW, VBG_CO2_HIGH)
    d_dbg$co2_cat <- stats::relevel(base::droplevels(d_dbg$co2_cat), ref = "Normal")
    d_dbg[[row$outcome]] <- to01(d_dbg[[row$outcome]])
    des <- survey::svydesign(ids = ~1, weights = ~w_dbg, data = d_dbg)
    fml <- stats::reformulate(c("co2_cat", adj_core), response = row$outcome)
    survey::svyglm(fml, design = des, family = quasibinomial(),
                   control = stats::glm.control(maxit = 50))
  }
}
options(warn = 0)
} else {
  message("MI_DEBUG_REPLAY=TRUE but no nonconverged rows found.")
}
}

```

4.0.8 Performance / runtime log

```

# Purpose: runtime log export.
stopifnot(exists("runtime_log"), nrow(runtime_log) > 0)
runtime_log_curr <- runtime_log
runtime_log_curr <- runtime_log_curr[runtime_log_curr$run_id == runtime_run_id, , drop = FALSE]

```

```

runtime_log_file <- results_path("runtime_log.csv")
write_csv_safely(runtime_log_curr, runtime_log_file, row_names = FALSE)
top_steps <- runtime_log_curr |>
  arrange(desc(seconds))
top15_file <- results_path("runtime_summary_top15.csv")
write_csv_safely(top_steps, top15_file, row_names = FALSE)
render_table_pdf_maybe(
  top_steps,
  caption = "Runtime steps (seconds)",
  file_stub = "runtime_steps",
  digits = 3,
  show = SHOW_LOW_VALUE_TABLES
)

total_seconds <- sum(runtime_log_curr$seconds, na.rm = TRUE)
runtime_summary <- bind_rows(
  tibble::tibble(step_name = "TOTAL", seconds = total_seconds),
  runtime_log_curr |>
    arrange(desc(seconds)) |>
    select(step_name, seconds)
)
runtime_summary_file <- results_path("runtime_summary.csv")
write_csv_safely(runtime_summary, runtime_summary_file, row_names = FALSE)
render_table_pdf_maybe(
  runtime_summary,
  caption = "Runtime summary (total + all steps)",
  file_stub = "runtime_summary",
  digits = 3,
  show = SHOW_LOW_VALUE_TABLES
)

# GBM preflight diagnostics (CSV only)
write_csv_safely(memory_snapshots, results_path("memory_snapshots.csv"), row_names = FALSE)
write_csv_safely(gbm_preflight_design_dims, results_path("gbm_preflight_design_dims.csv"), row_names = FALSE)
write_csv_safely(gbm_preflight_warnings, results_path("gbm_preflight_warnings.csv"), row_names = FALSE)

if (is.finite(PILOT_FRAC) && PILOT_FRAC > 0 && PILOT_FRAC < 1) {

```

```

scalable_steps <- c("mice_imputation", "mi_single_pass")
proj <- runtime_log_curr |>
  filter(step_name %in% scalable_steps) |>
  mutate(projected_seconds = seconds / PILOT_FRAC,
         projected_hours = projected_seconds / 3600)
proj_total <- sum(proj$projected_hours, na.rm = TRUE)
render_table_pdf_maybe(
  proj |> select(step_name, seconds, projected_hours),
  caption = paste0("Runtime projection (scalable steps; pilot_frac = ", PILOT_FRAC, ")"),
  file_stub = "runtime_projection",
  digits = 3,
  show = SHOW_LOW_VALUE_TABLES
)
if (is.finite(proj_total) && proj_total > 10) {
  warning("Projected full-run time exceeds 10 hours (", round(proj_total, 1), "h).",
         call. = FALSE)
}
} else {
  message("Runtime projection skipped because PILOT_FRAC == 1 (full run).")
}

stopifnot(exists("get_imp_stats"))
get_imp_df <- data.frame(
  count = get_imp_stats$count,
  seconds = get_imp_stats$seconds,
  seconds_per_call = if (get_imp_stats$count > 0) get_imp_stats$seconds / get_imp_stats$count else NA_real_,
  run_id = runtime_run_id,
  run_mode = RUN_MODE,
  n_subset = nrow(subset_data),
  stringsAsFactors = FALSE
)
write_csv_safely(get_imp_df, results_path("get_imp_usage.csv"), row_names = FALSE)

# Purpose: diagnostics completeness.
expected_diag <- c(
  "runtime_log.csv", "runtime_summary.csv", "runtime_summary_top15.csv",
  "latex_codeblock_risk_scan.csv",

```

```

"warnings_log.csv", "mi_warnings_log.csv", "mi_info_log.csv",
"diagnostics_summary.csv", "diagnostics_missingness.csv",
"diagnostics_missingness-by-strata.csv",
"balance_target_imp_summary.csv", "balance_target_by_imp.csv",
"balance_target_worst_rows.csv", "balance_max_smd_by_imp.csv",
"balance_worst_terms.csv", "balance_worst10.csv", "balance_table.csv",
"weight_summary.csv", "ps_overlap_summary.csv",
"model_fit_diagnostics.csv", "mi_outcome_fit_diagnostics.csv",
"mi_fit_issue_summary.csv", "mi_m_stability.csv", "mi_maxit_sensitivity.csv",
"mi_obs_vs_imp_summary.csv", "mi_spline_curve_abg.csv",
"mi_spline_curve_vbg.csv", "mi_spline_coef_abg.csv",
"mi_spline_coef_vbg.csv", "diag-ps-shap-stability.csv",
"diag_mi_shap_errors.csv", "diag_mi_shap_method_used.csv", "mi_shap_imp_diagnostics.csv",
"mice_logged_events_raw.csv", "mice_logged_events_summary.csv",
"mice_pred_width_preflight.csv", "mice_chain_diagnostics.csv",
"mice_batches_log.csv", "missingness-by-strata.csv",
"missingness-drivers.csv", "missingness-pattern.csv", "plot_drop_log.csv",
"mice_smoketest.log"
)
missing_files <- expected_diag[!file.exists(results_path(expected_diag))]
stopifnot(length(missing_files) == 0)

run_id_mismatch <- character()
for (f in expected_diag) {
  path <- results_path(f)
  if (grepl("\\.csv$", f)) {
    df <- tryCatch(read.csv(path, nrows = 1), error = function(e) NULL)
    stopifnot(!is.null(df))
    stopifnot("run_id" %in% names(df))
    if (nrow(df) > 0 && !identical(as.character(df$run_id[1]), diag_run_id)) {
      run_id_mismatch <- c(run_id_mismatch, f)
    }
  }
}

if (length(missing_files) || length(run_id_mismatch)) {
  msg <- paste0(

```

```

"Diagnostics completeness check failed: missing files [",
paste(missing_files, collapse = ", "),
"]"; run_id mismatch [",
paste(run_id_mismatch, collapse = ", "),
"]."
)
stop(msg)
}

```

4.1 Save, export, and session info

```

# Purpose: diagnostics audit run.
audit_script <- here::here("R", "diagnostics_audit.R")
audit_md <- results_path("diagnostics_audit.md")
stopifnot(file.exists(audit_script))

audit_out <- tryCatch(
  system2(
    "Rscript",
    args = c(shQuote(audit_script)),
    env = c(paste0("DIAG_RESULTS_DIR=", results_dir)),
    stdout = TRUE,
    stderr = TRUE
  ),
  error = function(e) e
)
if (inherits(audit_out, "error")) {
  warning("diagnostics_audit.R failed: ", conditionMessage(audit_out), call. = FALSE)
} else {
  status <- attr(audit_out, "status")
  if (!is.null(status) && status != 0) {
    warning("diagnostics_audit.R exited with status ", status, ".", call. = FALSE)
  }
}

# Purpose: diagnostics audit summary.
audit_issues <- results_path("diagnostics_audit_issues.csv")

```

```

if (file.exists(audit_issues)) {
  issues_df <- read.csv(audit_issues)
  issues_df <- issues_df |>
    dplyr::arrange(factor(severity, levels = c("blocker", "high", "medium", "low")))
  render_table_pdf_maybe(
    issues_df,
    caption = "Diagnostics audit issues (see Results/diagnostics_audit.md for details)",
    file_stub = "diagnostics_audit_issues",
    digits = 2,
    show = SHOW_LOW_VALUE_TABLES
  )
} else {
  cat("Diagnostics audit summary not available.\n")
}

# Purpose: mi save exports.
stopifnot(exists("abg_curves"), exists("vbg_curves"), exists("abg_coefs"), exists("vbg_coefs"))
saveRDS(
  list(
    abg_curves = abg_curves,
    vbg_curves = vbg_curves,
    abg_coefs = abg_coefs,
    vbg_coefs = vbg_coefs
  ),
  mi_pooled_file
)

# Purpose: mi session.
writeLines(capture.output(sessionInfo()), results_path("sessionInfo.txt"))

# Purpose: software statement.
cat("Software: R ", as.character(getRversion()),
  "; key packages: mice, WeightIt, cobalt, survey, rms.\n")

Software: R 4.5.2 ; key packages: mice, WeightIt, cobalt, survey, rms.

# Purpose: pdf hygiene scan.
qmd_src <- resolve_current_qmd()

```

```

pdf_file <- file.path(
  dirname(qmd_src),
  paste0(tools::file_path_sans_ext(basename(qmd_src)), ".pdf")
)
pdf_file <- normalizePath(pdf_file, winslash = "/", mustWork = FALSE)
scan_out <- results_path("pdf_hygiene_scan.csv")
scanner <- "none"
scan_status <- "skipped_pdf_missing"
txt <- character()

if (file.exists(pdf_file)) {
  scan_status <- "scanned"
  # "pdftotext" is a system binary (Poppler), not an R package.
  # Fallback to pdftools::pdf_text() when the binary is unavailable.
  pdftotext_bin <- Sys.which("pdftotext")

  if (nzchar(pdftotext_bin)) {
    scanner <- "pdftotext"
    tmp_txt <- tempfile(fileext = ".txt")
    scan_ok <- tryCatch({
      system2(pdftotext_bin, c(pdf_file, tmp_txt), stdout = FALSE, stderr = FALSE)
      file.exists(tmp_txt)
    }, error = function(e) FALSE)
    if (isTRUE(scan_ok)) {
      txt <- readLines(tmp_txt, warn = FALSE)
    }
  }
}

if (length(txt) == 0 && requireNamespace("pdftools", quietly = TRUE)) {
  scanner <- "pdftools"
  txt <- tryCatch(
    unlist(pdftools::pdf_text(pdf_file), use.names = FALSE),
    error = function(e) character()
  )
}

if (length(txt) == 0) scan_status <- "skipped_no_scanner"

```

```

}

# Match real absolute paths and avoid false hits from the code regex itself.
path_pattern <- "(/Users/[A-Za-z0-9._-]+/[A-Za-z0-9._-/ -]+)|([A-Za-z]:\\\\\\[^[:space:]]+)"
bad <- if (length(txt)) txt[grepl(path_pattern, txt)] else character()
n_bad <- length(bad)
scan_df <- data.frame(
  scan_status = scan_status,
  scanner = scanner,
  line = if (n_bad > 0) bad else "",
  stringsAsFactors = FALSE
)
write_csv_safely(scan_df, scan_out, row_names = FALSE)

if (RUN_MODE == "full" && scan_status != "scanned") {
  stop("PDF hygiene scan unavailable (no scanner found); see ", scan_out)
}
if (RUN_MODE == "full" && n_bad > 0) {
  stop("PDF hygiene scan detected absolute paths; see ", scan_out)
}

# Purpose: numbering hygiene scan.
qmd_file <- resolve_current_qmd()
src_lines <- readLines(qmd_file, warn = FALSE)
pattern_manual <- "(^#+\s+[0-9]+(\.\.[0-9]+)*\s+[0-9]+\s+)|(Table\s+[0-9]+:\s+Table\s+[0-9]+)"
hits <- which(grepl(pattern_manual, src_lines))
scan_df <- data.frame(
  line_no = hits,
  line = src_lines[hits],
  stringsAsFactors = FALSE
)
write_csv_safely(scan_df, results_path("numbering_hygiene_scan.csv"), row_names = FALSE)

```