

Supplementary 1. FFSV Estimation Guide in R

Step 1 – Shapley-value covariate selection in R for panel models

This step loads the core functions, reads one panel, and runs `fe_shapley_covariate_selection()` to obtain (a) the Shapley values of the covariates which decompose the adjusted R^2 of the full model exactly (the efficiency gap is of order 10^{-16}); (b) efficiency, superadditivity and core-membership check consistent with the cooperative game; and (c) the selection rule in which Rule A chooses covariates with positive Shapley value, and Rule B chooses the penalty-corrected rule as $J(k) = v(M_k) - \gamma k$ with default $\gamma = 0.01$ which can be identified by users.

R Code for FESV stepup

```
#####
# panel_shapley_selection.R
## FIXED EFFECTS COVARIATE SELECTION USING THE SHAPLEY VALUE
## Idea: in a panel y_it, the unit effects (alpha_i) and time effects (gamma_t)
# are HELD FIXED -- they are part of every model and are never selected.
# The candidate covariates are the PLAYERS of a cooperative game, and the best
# covariate set is the best coalition of that game.
## Cooperative game (N, v):
# * Players N : the candidate covariates
# * Fixed effects : unit and time effects are ALWAYS included (they are
# partialled out of y and X before the game is played,
# which is numerically equivalent to putting unit and
# time dummies in every coalition model); they are NOT
# players.
# * Characteristic fn : v(S) = fit(S | FE) - fit(empty | FE), v(empty) = 0
# method = "adj_r2" (DEFAULT) : adjusted R^2 (net of fixed effects).
# Because the adjustment penalises degrees of freedom, adding
# an irrelevant covariate LOWERS v, giving noise players
# NEGATIVE Shapley values -> enables sign-based selection
# (frequentist analogue of the Bayesian Occam penalty).
# method = "r2" : plain R^2 net of fixed effects (contributions >= 0)
# method = "neg_bic" : -BIC (Schwarz), relative to the FE-only model.
## Selection (data-driven, no fixed model size):
# Rule A : phi_i > 0 (positive marginal contribution)
# Rule B : argmax_k J(k) = v(M_k) - gamma*k (penalised nested optimisation)
## Authors: Htwe Ko and Pathairat Pastpipatkul (2026)
#####
## -----
## Fixing unit and time effects (partialling out)
## -----
# Partial the unit and time effects out of a numeric vector.
# Equivalent to regressing on unit + time dummies and keeping residuals
# (plus the grand mean, which cancels in every fit statistic).
fe_partial_vec <- function(v, unit, time) {
  gm <- mean(v)
  um <- tapply(v, unit, mean)
  tm <- tapply(v, time, mean)
  v - um[as.character(unit)] - tm[as.character(time)] + gm
}
# Partial the unit and time effects out of every column of a matrix
fe_partial_mat <- function(M, unit, time) {
  out <- apply(M, 2, fe_partial_vec, unit = unit, time = time)
  matrix(out, nrow = nrow(M), dimnames = list(NULL, colnames(M)))
}
## -----
## Model fit for a coalition S (fixed effects already held fixed)
## -----
# Goodness-of-fit for the FE model using covariates in S.
```

```

## y_fe, X_fe already have the unit and time effects partialled out.
## df_fe = NT - (N + T - 1): residual df of the FE-only model.
fe_model_fit <- function(y_fe, X_fe, S, df_fe, method = "adj_r2") {
  nT <- length(y_fe)
  ns <- length(S)
  tss <- sum(y_fe^2)
  if (ns == 0L) {
    ssr <- tss
  } else {
    Xs <- X_fe[, S, drop = FALSE]
    fit <- tryCatch(stats::lm.fit(Xs, y_fe), error = function(e) NULL)
    if (is.null(fit)) return(-Inf)
    ssr <- sum(fit$residuals^2)
  }
  if (method == "r2") {
    return(1 - ssr / tss)
  }
  if (method == "adj_r2") {
    df_resid <- df_fe - ns
    if (df_resid <= 0) return(-Inf)
    return(1 - (ssr / df_resid) / (tss / df_fe))
  }
  if (method == "neg_bic") {
    sigma2 <- ssr / nT
    if (sigma2 <= 0) sigma2 <- .Machine$double.eps
    ll <- -0.5 * nT * (log(2 * pi) + log(sigma2) + 1)
    bic <- -2 * ll + ns * log(nT)
    return(-bic)
  }
  stop("Unknown method: ", method)
}

## Characteristic function v(S) = fit(S) - fit(empty), so that v(empty) = 0.
fe_characteristic_function <- function(S, y_fe, X_fe, df_fe,
  method = "adj_r2", v_empty = NULL) {
  if (is.null(v_empty)) {
    v_empty <- fe_model_fit(y_fe, X_fe, integer(0), df_fe, method)
  }
  fe_model_fit(y_fe, X_fe, S, df_fe, method) - v_empty
}

## -----
## Coalition-value cache (exact enumeration)
## -----

## Binary expansion of coalition bitmask m over p players
## (analogue of intToBin() in the BGCS pipeline).
fe_intToBin <- function(m, p) {
  as.integer(intToBits(m))[1:p]
}

## Precompute v(S) for all 2^p coalitions. Returns numeric vector indexed by
## bitmask m = 0..(2^p - 1); bit (i-1) set <=> player i in S.
fe_coalition_values <- function(y_fe, X_fe, df_fe,
  method = "adj_r2", verbose = FALSE) {
  p <- ncol(X_fe)
  M <- 2^p
  v_empty <- fe_model_fit(y_fe, X_fe, integer(0), df_fe, method)
  vals <- numeric(M)
  for (m in 0:(M - 1)) {
    S <- which(fe_intToBin(m, p) == 1)
    vals[m + 1] <- fe_model_fit(y_fe, X_fe, S, df_fe, method) - v_empty
    if (verbose && m %% 256 == 0) cat(" coalition", m, "of", M - 1, "\n")
  }
  vals
}

```

```

}
# Print the characteristic function v(S) for all coalitions
# (mirror of the coalition listing in Example 5 of the BGCS demo).
fe_print_characteristic_function <- function(vals, p, names_p = NULL) {
  cat("\nCharacteristic function v(S) for all coalitions:\n")
  cat("-----\n")
  for (m in 0:(2^p - 1)) {
    S <- which(fe_intToBin(m, p) == 1)
    lab <- if (length(S) == 0) "{}" else {
      inner <- if (is.null(names_p)) paste(S, collapse = ",")
      else paste(names_p[S], collapse = ",")
      paste0("{", inner, "}")
    }
    cat(sprintf("v(%s) = %.4f\n", lab, vals[m + 1]))
  }
}
## -----
## Exact Shapley values
## -----
fe_shapley_exact <- function(y_fe, X_fe, df_fe, method = "adj_r2",
  verbose = FALSE, vals = NULL) {
  p <- ncol(X_fe)
  names_p <- colnames(X_fe)
  if (is.null(vals))
    vals <- fe_coalition_values(y_fe, X_fe, df_fe, method, verbose)
  # log combinatorial weights w(s) = s!(p-s-1)!/p!
  logw <- function(s) lfactorial(s) + lfactorial(p - s - 1) - lfactorial(p)
  phi <- numeric(p)
  powers <- 2^(0:(p - 1))
  for (i in 1:p) {
    bit_i <- powers[i]
    others <- setdiff(0:(p - 1), i - 1)
    acc <- 0
    # iterate over all subsets S of N\{i} via bitmask over 'others'
    n_oth <- length(others)
    for (mm in 0:(2^n_oth - 1)) {
      # build coalition bitmask without i
      sel_bits <- others[which(bitwAnd(mm, 2^(0:(n_oth - 1))) > 0)]
      base_mask <- if (length(sel_bits)) sum(2^sel_bits) else 0
      s <- length(sel_bits)
      v_without <- vals[base_mask + 1]
      v_with <- vals[bitwOr(base_mask, bit_i) + 1]
      acc <- acc + exp(logw(s)) * (v_with - v_without)
    }
    phi[i] <- acc
    if (verbose) cat(" phi_", i, " (", names_p[i], ") = ",
      round(phi[i], 4), "\n", sep = "")
  }
  names(phi) <- names_p
  phi
}
## -----
## Monte-Carlo Shapley values (permutation sampling) -- for larger p
## -----
fe_shapley_mc <- function(y_fe, X_fe, df_fe, method = "adj_r2",
  n_samples = 2000, verbose = FALSE) {
  p <- ncol(X_fe)
  names_p <- colnames(X_fe)
  v_empty <- fe_model_fit(y_fe, X_fe, integer(0), df_fe, method)
  vfun <- local({
    cache <- new.env(hash = TRUE, parent = emptyenv())

```

```

function(S) {
  key <- paste(sort(S), collapse = "-")
  hit <- cache[[key]]
  if (!is.null(hit)) return(hit)
  val <- fe_model_fit(y_fe, X_fe, S, df_fe, method) - v_empty
  cache[[key]] <- val
  val
}
})
phi <- numeric(p)
for (b in 1:n_samples) {
  perm <- sample.int(p)
  prev <- 0
  S <- integer(0)
  for (i in perm) {
    S_new <- c(S, i)
    v_new <- vfun(S_new)
    phi[i] <- phi[i] + (v_new - prev)
    prev <- v_new
    S <- S_new
  }
  if (verbose && b %% 500 == 0) cat(" MC sample", b, "of", n_samples, "\n")
}
phi <- phi / n_samples
names(phi) <- names_p
phi
}
## -----
## Cooperative game diagnostics (consistent with the BGCS pipeline:
## check_superadditivity / check_core_membership / efficiency identity)
## -----
# Superadditivity check:  $v(A \cup B) \geq v(A) + v(B)$  for all disjoint A, B.
# Under penalised fits (adj_r2, neg_bic) violations are EXPECTED for
# coalitions containing noise players -- the frequentist analogue of the
# Occam-penalty violations in the Bayesian evidence game.
fe_check_superadditivity <- function(vals, p, tol = 1e-10, verbose = TRUE) {
  full <- 2^p - 1
  n_pairs <- 0L; n_viol <- 0L; worst_gap <- 0
  for (a in 1:(full - 1)) {
    rem <- bitwAnd(full, bitwNot(a))
    b <- rem
    while (b > 0) {
      if (b > a) { # count each unordered disjoint pair once
        gap <- vals[bitwOr(a, b) + 1] - vals[a + 1] - vals[b + 1]
        n_pairs <- n_pairs + 1L
        if (gap < -tol) {
          n_viol <- n_viol + 1L
          if (gap < worst_gap) worst_gap <- gap
        }
      }
      b <- bitwAnd(b - 1L, rem)
    }
  }
  superadd <- (n_viol == 0L)
  if (verbose) {
    cat(sprintf("Superadditivity: %d of %d disjoint pairs violated (worst gap %.4f)\n",
      n_viol, n_pairs, worst_gap))
    if (!superadd)
      cat(" -> expected under the df-penalised characteristic function\n")
  }
  list(superadditive = superadd, n_pairs = n_pairs,

```

```

    n_violations = n_viol, worst_gap = worst_gap)
}
#' Core membership check: is phi in the core, i.e.  $\sum_{i \in S} \phi_i \geq v(S)$ 
#' for every coalition S (with efficiency  $\sum(\phi) = v(N)$ )?
fe_check_core_membership <- function(phi, vals, p, tol = 1e-8, verbose = TRUE) {
  powers <- 2^(0:(p - 1))
  min_slack <- Inf; worst_S <- integer(0)
  for (m in 1:(2^p - 1)) {
    S <- which(bitwAnd(m, powers) > 0)
    slack <- sum(phi[S]) - vals[m + 1]
    if (slack < min_slack) { min_slack <- slack; worst_S <- S }
  }
  in_core <- (min_slack >= -tol)
  if (verbose) {
    cat(sprintf("Core membership: %s (min slack = %.4f at S = {%s})\n",
      ifelse(in_core, "phi IS in the core", "phi NOT in the core"),
      min_slack,
      paste(names(phi)[worst_S], collapse = ", ")))
  }
  list(in_core = in_core, min_slack = min_slack, worst_coalition = worst_S)
}
## -----
## Nested model selection:  $J(k) = v(M_k) - \gamma * k$ 
## -----
fe_nested_selection <- function(phi, y_fe, X_fe, df_fe,
  method = "adj_r2", gamma = 0.01,
  verbose = TRUE) {
  p <- length(phi)
  ranking <- order(phi, decreasing = TRUE)
  names_p <- names(phi)
  v_empty <- fe_model_fit(y_fe, X_fe, integer(0), df_fe, method)
  J <- numeric(p)
  for (k in 1:p) {
    Mk <- ranking[1:k]
    vMk <- fe_model_fit(y_fe, X_fe, Mk, df_fe, method) - v_empty
    J[k] <- vMk - gamma * k
    if (verbose) cat(sprintf(" k=%2d: J(k) = %.4f [%s]\n",
      k, J[k], paste(names_p[Mk], collapse = ", ")))
  }
  k_star <- which.max(J)
  selected <- ranking[1:k_star]
  if (verbose) {
    cat("\nOptimal size k* =", k_star, "\n")
    cat("Selected:", paste(names_p[selected], collapse = ", "), "\n")
  }
  list(k_star = k_star, selected = selected,
    selected_names = names_p[selected], J = J, ranking = ranking)
}
## -----
## Master routine
## -----
#' Fixed effects covariate selection using the Shapley value.
#' Unit and time effects are held fixed; the covariates play the game.
#' @param y numeric response (length NT)
#' @param X numeric matrix of candidate covariates (NT x p), named columns
#' @param unit unit (individual) index, length NT
#' @param time time index, length NT
#' @param method characteristic-function type (see top of file)
#' @param exact TRUE for exact Shapley (p small), FALSE for Monte Carlo
#' @param n_samples MC permutations when exact = FALSE
#' @param gamma gamma penalty for nested selection J(k)

```

```

fe_shapley_covariate_selection <- function(y, X, unit, time,
                                          method = "adj_r2",
                                          exact = TRUE,
                                          n_samples = 2000,
                                          gamma = 0.01,
                                          verbose = TRUE) {
  stopifnot(is.matrix(X))
  N <- length(unique(unit)); T <- length(unique(time))
  nT <- length(y); p <- ncol(X)
  if (is.null(colnames(X))) colnames(X) <- paste0("V", 1:p)
  df_fe <- nT - (N + T - 1) # residual df of the FE-only model (excl. slopes)
  if (verbose) {
    cat("=====\\n")
    cat(" Fixed Effects Covariate Selection using the Shapley Value\\n")
    cat("=====\\n")
    cat(" N =", N, " units, T =", T, " periods, NT =", nT, "\\n")
    cat(" Players (p) =", p, ":", paste(colnames(X), collapse = ", "), "\\n")
    cat(" Fixed (not players): unit effects alpha_i, time effects gamma_t\\n")
    cat(" Method =", method, " | FE residual df =", df_fe, "\\n")
    cat(" Computation:", ifelse(exact, "Exact", "Monte Carlo"), "\\n\\n")
  }
  # Hold unit and time effects fixed (partial them out once)
  y_fe <- fe_partial_vec(y, unit, time)
  X_fe <- fe_partial_mat(X, unit, time)
  # Coalition values (exact enumeration reused by Shapley + diagnostics)
  vals <- NULL
  if (exact) {
    vals <- fe_coalition_values(y_fe, X_fe, df_fe, method, verbose = FALSE)
    # Sanity: v(empty) = 0 by construction (mirrors v(null model) normalisation)
    stopifnot(abs(vals[1]) < 1e-12)
    phi <- fe_shapley_exact(y_fe, X_fe, df_fe, method, verbose, vals = vals)
  } else {
    phi <- fe_shapley_mc(y_fe, X_fe, df_fe, method, n_samples, verbose)
  }
  # --- Cooperative game diagnostics (as in the BGCS pipeline) -----
  v_grand <- fe_characteristic_function(1:p, y_fe, X_fe, df_fe, method)
  superadd <- NULL; core <- NULL
  if (verbose) {
    cat("\\nGame-theoretic verification:\\n")
    cat("-----\\n")
    cat("v(empty) = 0 (by normalisation)\\n")
    cat("Efficiency check:\\n")
    cat(" v(N)          =", round(v_grand, 6), "\\n")
    cat(" Sum of Shapley values =", round(sum(phi), 6), "\\n")
    cat(" Difference      =",
        format(abs(v_grand - sum(phi)), digits = 3), "\\n")
  }
  if (exact) {
    superadd <- fe_check_superadditivity(vals, p, verbose = verbose)
    core <- fe_check_core_membership(phi, vals, p, verbose = verbose)
  }
  # Rule A: positive Shapley value
  sel_sign <- which(phi > 0)
  # Rule B: nested penalised selection
  if (verbose) cat("\\nNested model selection (J(k) = v(M_k) - gamma*k):\\n")
  nested <- fe_nested_selection(phi, y_fe, X_fe, df_fe, method, gamma, verbose)

  list(shapley_values = phi,
       selected_covariates = sel_sign, # naming aligned with BGCS pipeline
       selected_sign = sel_sign,
       selected_nested = nested$selected,

```

```

    nested = nested,
    coalition_values = vals,
    superadditivity = superadd,
    core = core,
    v_grand = v_grand,
    efficiency_gap = abs(v_grand - sum(phi)),
    df_fe = df_fe,
    method = method,
    N = N, T = T, p = p)
}
## -----
## Convenience: fixed-effects OLS estimate for a chosen covariate set
## -----
# Estimate the FE model (unit + time effects held fixed) for covariates in
# cols `sel`. Classic (non-robust) SEs, degrees of freedom corrected for
# the fixed effects.
fe_ols <- function(y, X, unit, time, sel) {
  N <- length(unique(unit)); T <- length(unique(time)); nT <- length(y)
  y_fe <- fe_partial_vec(y, unit, time)
  X_fe <- fe_partial_mat(X, unit, time)
  Xs <- X_fe[, sel, drop = FALSE]
  fit <- lm.fit(Xs, y_fe)
  resid <- fit$residuals
  ssr <- sum(resid^2)
  k <- length(sel)
  df <- nT - (N + T - 1) - k
  sigma2 <- ssr / df
  XtX_inv <- solve(crossprod(Xs))
  se <- sqrt(diag(XtX_inv) * sigma2)
  beta <- fit$coefficients
  tval <- beta / se
  pval <- 2 * pt(-abs(tval), df = df)
  tss <- sum(y_fe^2)
  r2 <- 1 - ssr / tss
  adj <- 1 - (ssr / df) / (tss / (nT - (N + T - 1)))
  tab <- data.frame(Estimate = beta, Std.Error = se,
                    t.value = tval, p.value = pval,
                    row.names = colnames(Xs))
  list(coefficients = tab, r2 = r2, adj_r2 = adj,
        sigma = sqrt(sigma2), df_residual = df, ssr = ssr)
}
## -----
## Evaluation metrics (project-consistent definitions)
## Recall = TP / (TP + FN) (completeness of ground-truth recovery)
## Precision = TP / (TP + FP)
## F1 = 2 PR / (P + R)
## -----
selection_metrics <- function(selected, active, p) {
  sel <- rep(FALSE, p); sel[selected] <- TRUE
  act <- rep(FALSE, p); act[active] <- TRUE
  TP <- sum(sel & act)
  FP <- sum(sel & !act)
  FN <- sum(!sel & act)
  TN <- sum(!sel & !act)
  precision <- if ((TP + FP) > 0) TP / (TP + FP) else NA_real_
  recall <- if ((TP + FN) > 0) TP / (TP + FN) else NA_real_
  f1 <- if (!is.na(precision) && !is.na(recall) && (precision + recall) > 0)
    2 * precision * recall / (precision + recall) else 0
  c(TP = TP, FP = FP, FN = FN, TN = TN,
    precision = precision, recall = recall, F1 = f1)
}

```

```
cat("panel_shapley_selection.R loaded (fixed effects Shapley selection).\n")
```

Step 2 – fit the model file and run the model

Once the selected covariates are known, Step 2 fits the actual fixed-effects regressions reported in the paper and produces the comparison between the full ten-covariate model and the FESV-selected model. The driver *fit model.R* runs the same three-stage workflow used in the paper.

The complete R codes for Step 2 model estimation outputs

```
#####
# all sample.R
## REAL-DATA APPLICATION: FIXED EFFECTS COVARIATE SELECTION USING THE SHAPLEY
# Data : developing.csv (25 developing countries x 24 years, 2000-2023)
# Model : Income_it = sum_j beta_j x_{jit} + alpha_i + gamma_t + e_it
# Country effects alpha_i and year effects gamma_t are held fixed
# (never selected); the 6 macro covariates are the players
# (p = 6, exact Shapley: 2^6 = 64 coalitions).
## Stage 1: Full FE OLS with all candidate covariates
# Stage 2: Cooperative game selection
# Rule A: phi_i > 0
# Rule B: argmax_k J(k) = v(M_k) - gamma*k
# + game-theoretic verification (efficiency, superadditivity, core)
# Stage 3: FE OLS on the selected coalition, comparison with Stage 1
#####
## place the panel_shapley_selection.R file in a working directory of your choice
source("D:/FESV/panel_shapley_selection.R")
set.seed(2024)
## -----
## Load and prepare the data
## -----
dat <- read.csv("C:/Users/User/OneDrive/Desktop/all.csv")
dat <- dat[order(dat$Id, dat$Year), ]
cat("Data: developing.csv\n")
cat(" Countries:", length(unique(dat$Id)),
    " | Years:", min(dat$Year), "-", max(dat$year),
    " | Rows:", nrow(dat), "\n\n")
covar_names <- c("IE", "IH", "Debt", "FDI", "INF", "POP", "TOP", "UR", "EXR", "GR")
y <- dat$Growth
X <- as.matrix(dat[, covar_names])
unit <- dat$Id
time <- dat$year
p <- ncol(X)
cat("Estimation sample: NT =", length(y),
    "(N =", length(unique(unit)), ", T =", length(unique(time))), "\n")
cat("Players (p =", p, "):", paste(colnames(X), collapse = ", "), "\n")
cat("Fixed (not players): country effects alpha_i, year effects gamma_t\n\n")
## -----
## STAGE 1: Full FE OLS (all candidate covariates)
## -----
cat("#####\n")
cat("# STAGE 1: FULL FE OLS (all", p, "covariates) #\n")
cat("#####\n\n")
fe_full <- fe_ols(y, X, unit, time, sel = 1:p)
print(round(fe_full$coefficients, 4))
cat(sprintf("\nR^2 (net of FE) = %.4f | Adj = %.4f | sigma = %.4f | df = %d\n",
    fe_full$r2, fe_full$adj_r2, fe_full$sigma, fe_full$df_residual))
## -----
## STAGE 2: Cooperative game covariate selection (exact Shapley, p = 6)
## -----
cat("\n#####\n")
```

```

cat("# STAGE 2: FE SHAPLEY GAME SELECTION (exact, 2^6 coalitions)      #\n")
cat("#####\n\n")
res <- fe_shapley_covariate_selection(
  y, X, unit, time,
  method = "adj_r2", exact = TRUE, gamma = 0.01, verbose = TRUE)
phi <- res$shapley_values
cat("\nShapley ranking (descending):\n")
print(round(sort(phi, decreasing = TRUE), 4))
selA <- res$selected_sign
selB <- res$selected_nested
cat("\nRule A (phi > 0) :", paste(colnames(X)[selA], collapse = ", "), "\n")
cat("Rule B (nested) :", paste(colnames(X)[selB], collapse = ", "), "\n")
cat("Agreement      :", ifelse(setequal(selA, selB), "YES", "NO"), "\n")
# Final coalition: Rule B, the penalised optimum; report Rule A alongside
# if it differs
sel_final <- selB
# Shapley plot
png("panel_shapley_values_developing.png", width = 900, height = 600)
sorted_idx <- order(phi, decreasing = TRUE)
barplot(phi[sorted_idx],
  main = "Shapley Values - Developing Countries (Fixed Effects)",
  names.arg = names(phi)[sorted_idx],
  col = ifelse(phi[sorted_idx] > 0, "steelblue", "indianred"),
  border = "white",
  ylab = "Shapley Value", las = 2, cex.names = 0.8)
grid()
dev.off()
cat("\nShapley values plot saved to panel_shapley_values_developing.png\n")
## -----
## STAGE 3: FE OLS on the selected coalition + comparison
## -----
cat("\n#####\n")
cat("# STAGE 3: FE OLS ON SELECTED COALITION      #\n")
cat("#####\n\n")
cat("Coalition (Rule B):", paste(colnames(X)[sel_final], collapse = ", "), "\n\n")
fe_sel <- fe_ols(y, X, unit, time, sel = sel_final)
print(round(fe_sel$coefficients, 4))
cat(sprintf("\nR^2 (net of FE) = %.4f | Adj = %.4f | sigma = %.4f | df = %d\n",
  fe_sel$r2, fe_sel$adj_r2, fe_sel$sigma, fe_sel$df_residual))
if (!setequal(selA, selB)) {
  cat("\n--- Rule A coalition for comparison ---\n")
  cat("Coalition (Rule A):", paste(colnames(X)[selA], collapse = ", "), "\n\n")
  fe_A <- fe_ols(y, X, unit, time, sel = selA)
  print(round(fe_A$coefficients, 4))
  cat(sprintf("\nR^2 (net of FE) = %.4f | Adj = %.4f | sigma = %.4f | df = %d\n",
    fe_A$r2, fe_A$adj_r2, fe_A$sigma, fe_A$df_residual))
}
## -----
## Model comparison: full vs selected coalition
## -----
cat("\n#####\n")
cat("# MODEL COMPARISON: FULL vs SELECTED COALITION      #\n")
cat("#####\n\n")
nT <- length(y)
bic_of <- function(fe, k) {
  sigma2 <- fe$ssr / nT
  ll <- -0.5 * nT * (log(2 * pi) + log(sigma2) + 1)
  -2 * ll + k * log(nT)
}
comp <- data.frame(
  Model = c("Full (all covariates)", "Selected coalition (Rule B)"),

```

```

k = c(p, length(sel_final)),
R2 = c(fe_full$r2, fe_sel$r2),
Adj_R2 = c(fe_full$adj_r2, fe_sel$adj_r2),
sigma = c(fe_full$sigma, fe_sel$sigma),
BIC = c(bic_of(fe_full, p), bic_of(fe_sel, length(sel_final))))
print(comp, row.names = FALSE, digits = 5)
cat("\nall-countries FE Shapley application completed successfully.\n")

```

Supplementary 2. BPVS-FE and BPVS-FE Estimation Guide in R

This is a step-by-step model computation guide for the BPVS models introduced in Pastpipatkul & Ko [8]. It uses two R scripts written by the authors, which reproduce every number reported in the manuscript.

Step 1 – Load the functions of the model setup

BPVS R code

```

# =====
# bpvs_model.R — Bayesian Panel Variable Selection (BPVS) Model
# Pathairat Pastpipatkul and Htwe Ko (2025)
# Revised: FE (unit dummies) and RE (random intercept) variants
# =====
#
# PROBABILITY MODEL:
#
# Likelihood (Panel):
# y_it ~ N(mu_it, sigma^2)
# mu_it = alpha_i + sum_{j=1}^p beta_j * x_{itj}
#
# BPVS-RE (random intercepts):
# alpha_i ~ N(0, tau_unit^2) # random effects (country effects)
# fitted with: y ~ x1 + ... + xp + (1 | id)
#
# BPVS-FE (fixed effects, unit dummies):
# alpha_i treated as free unit-specific intercepts
# fitted with nonlinear brms formula:
# y ~ alpha + eta
# alpha ~ 0 + id # unshrunk unit dummies
# eta ~ 0 + x1 + ... + xp # horseshoe only on slopes
#
# Prior on slopes (Regularized Horseshoe — continuous spike-and-slab):
# beta_j ~ N(0, tau^2 * lambda_j^2)
# lambda_j ~ Cauchy+(0, 1) # local shrinkage
# tau ~ Cauchy+(0, tau_0) # global shrinkage
# tau_0 = p0/(p - p0) * sigma / sqrt(N) # auto-scaled
#
# Prior on variance components:
# sigma ~ student_t(3, 0, 10)
# tau_unit ~ exponential(1)
#
# IDENTIFICATION:
# - FE: unit effects identified via dummy coefficients (proper wide prior)
# - RE: alpha_i identified via proper prior variance tau_unit
# - Horseshoe identified when p << N or sufficient MCMC iterations
# - PIP_j = Pr(|beta_j| > eps | data), eps = 0.01 * sd(y)
#
# OUTPUTS:
# fit_bpvs_fe() — fits BPVS-FE via brms (unit dummies + horseshoe)
# fit_bpvs_re() — fits BPVS-RE via brms (random intercept + horseshoe)
# fit_bpvs() — dispatcher: effects = "fe" | "re"
# compute_pip() — computes posterior inclusion probabilities
# rank_variables() — ranks predictors by PIP with selection labels
# extract_model_stats() — WAIC and Bayesian R2
# compare_bpvs() — FE vs RE comparison + Bayesian/Frequentist Hausman
# bpvs_pipeline() — fit + compute + rank + print in one call
# =====

```

```

required_packages <- c("brms", "tidyverse", "posterior", "bayesplot", "loo", "plm")
for (pkg in required_packages) {
  if (!requireNamespace(pkg, quietly = TRUE)) {
    install.packages(pkg, repos = "https://cloud.r-project.org")
  }
  library(pkg, character.only = TRUE)
}

backend <- if (requireNamespace("cmdstanr", quietly = TRUE)) "cmdstanr" else "rstan"

# -----
# extract_slope_terms: pull predictor names from a linear formula RHS
# -----
extract_slope_terms <- function(formula) {
  t <- terms(formula)
  attr(t, "term.labels")
}

# -----
# fit_bpvs_fe: Fit BPVS with FIXED EFFECTS (unit dummies)
# y ~ alpha + eta ; alpha ~ 0 + id ; eta ~ 0 + x1 + ... + xp
# Horseshoe applied ONLY to slope block (eta); dummies unshrunk.
# -----
fit_bpvs_fe <- function(formula, data, id_var = "id",
  chains = 4, iter = 4000, warmup = 2000, cores = 4,
  seed = 123, adapt_delta = 0.95, max_treedepth = 12,
  refresh = 500, ...) {
  if (!id_var %in% names(data)) {
    stop("id_var '", id_var, "' not found in data.")
  }
  data <- as.data.frame(data)
  data[[id_var]] <- factor(data[[id_var]])

  y_name <- all.vars(formula)[1]
  slopes <- extract_slope_terms(formula)
  if (length(slopes) == 0) stop("No predictors on RHS of formula.")

  nl_formula <- bf(
    as.formula(paste(y_name, "~ alpha + eta")),
    as.formula(paste("alpha ~ 0 +", id_var)),
    as.formula(paste("eta ~ 0 +", paste(slopes, collapse = " + "))),
    nl = TRUE
  )

  priors <- c(
    set_prior("horseshoe(df = 1, scale_global = 2, autoscale = TRUE)",
      nlpar = "eta", class = "b"),
    set_prior("normal(0, 10)", nlpar = "alpha", class = "b")
  )

  fit <- brm(
    formula = nl_formula,
    data = data,
    family = gaussian(),
    prior = priors,
    chains = chains,
    iter = iter,
    warmup = warmup,
    cores = cores,
    seed = seed,
    backend = backend,
    refresh = refresh,
    save_pars = save_pars(all = TRUE),
    control = list(adapt_delta = adapt_delta, max_treedepth = max_treedepth),
    ...
  )
  attr(fit, "bpvs_effects") <- "fe"
  attr(fit, "bpvs_slopes") <- slopes
  attr(fit, "bpvs_id_var") <- id_var

```

```

return(fit)
}

# -----
# fit_bpvs_re: Fit BPVS with RANDOM EFFECTS (random intercepts)
# -----
fit_bpvs_re <- function(formula, data, id_var = "id",
  chains = 4, iter = 4000, warmup = 2000, cores = 4,
  seed = 123, adapt_delta = 0.95, max_treedepth = 12,
  refresh = 500, ...) {

  if(!id_var %in% names(data)) {
    stop("id_var '", id_var, "' not found in data.")
  }
  data <- as.data.frame(data)
  data[[id_var]] <- factor(data[[id_var]])

  y_name <- all.vars(formula)[1]
  slopes <- extract_slope_terms(formula)
  re_formula <- as.formula(paste(
    y_name, "~", paste(slopes, collapse = " + "), "+ (1 |", id_var, ")")
  ))

  fit <- brm(
    formula = re_formula,
    data = data,
    family = gaussian(),
    prior = c(set_prior("horseshoe(df = 1, scale_global = 2, autoscale = TRUE)",
      class = "b")),
    chains = chains,
    iter = iter,
    warmup = warmup,
    cores = cores,
    seed = seed,
    backend = backend,
    refresh = refresh,
    save_pars = save_pars(all = TRUE),
    control = list(adapt_delta = adapt_delta, max_treedepth = max_treedepth),
    ...
  )
  attr(fit, "bpvs_effects") <- "re"
  attr(fit, "bpvs_slopes") <- slopes
  attr(fit, "bpvs_id_var") <- id_var
  return(fit)
}

# -----
# fit_bpvs: dispatcher
# -----
fit_bpvs <- function(formula, data, effects = c("re", "fe"), id_var = "id", ...) {
  effects <- match.arg(effects)
  if (effects == "fe") {
    fit_bpvs_fe(formula, data, id_var = id_var, ...)
  } else {
    fit_bpvs_re(formula, data, id_var = id_var, ...)
  }
}

# -----
# compute_pip: Compute Posterior Inclusion Probabilities
# PIP_j = Pr(|beta_j| > eps | data), eps = effect_threshold * sd(y)
# FE: slope block columns are b_eta_<var>
# RE: slope columns are b_<var>
# -----
compute_pip <- function(fit, data, outcome_var = "Income",
  effect_threshold = 0.01) {
  post <- as_draws_df(fit)
  eps <- effect_threshold * sd(data[[outcome_var]], na.rm = TRUE)
  effects <- attr(fit, "bpvs_effects") %||% "re"

```

```

pattern <- if (effects == "fe") "^b_eta_" else "^b_"

b_names <- grep(pattern, names(post), value = TRUE)
# RE: drop intercept; FE: eta block has no intercept
b_names <- setdiff(b_names, "b_Intercept")

pip <- sapply(b_names, function(col) {
  mean(abs(post[[col]]) > eps, na.rm = TRUE)
})
med <- sapply(b_names, function(col) median(post[[col]], na.rm = TRUE))
q025 <- sapply(b_names, function(col) quantile(post[[col]], 0.025, na.rm = TRUE))
q975 <- sapply(b_names, function(col) quantile(post[[col]], 0.975, na.rm = TRUE))

# Clean variable names: strip "b_eta_" / "b_" prefix
var_names <- gsub("^b_eta_|^b_", "", b_names)

data.frame(
  Variable = var_names,
  Estimate = round(med, 6),
  CI_lower = round(q025, 6),
  CI_upper = round(q975, 6),
  PIP = round(pip, 6),
  stringsAsFactors = FALSE
)
}

# -----
# rank_variables: Rank predictors by PIP with selection labels
# -----
rank_variables <- function(pip_df) {
  pip_df %>%
    arrange(desc(PIP)) %>%
    mutate(
      Rank = 1:n(),
      Selection = case_when(
        PIP > 0.5 ~ "Selected",
        PIP > 0.1 ~ "Weak",
        TRUE ~ "Excluded"
      )
    )
}

# -----
# extract_model_stats: Extract WAIC and Bayesian R2
# -----
extract_model_stats <- function(fit) {
  w <- tryCatch(waic(fit), error = function(e) NULL)
  r2 <- tryCatch(bayes_R2(fit)[1], error = function(e) NA)
  if (!is.null(w)) {
    waic_val <- w$estimates["waic", "Estimate"]
    waic_se <- w$estimates["waic", "SE"]
  } else {
    waic_val <- NA; waic_se <- NA
  }
  list(
    WAIC = round(waic_val, 2),
    WAIC_SE = round(waic_se, 2),
    Bayes_R2 = round(r2, 4)
  )
}

# -----
# print_bpvs_results: Pretty-print BPVS output
# -----
print_bpvs_results <- function(ranking, stats, title = "BPVS") {
  cat("\n=====\\n")
  cat(paste(title, "— BAYESIAN PANEL VARIABLE SELECTION RESULTS\\n"))
  cat("=====\\n")
  cat("\n--- Variable Ranking (by Posterior Inclusion Probability) ---\\n")
}

```

```

print(ranking %>% select(Rank, Variable, Estimate, PIP, Selection),
      row.names = FALSE)

cat(sprintf("\n--- Model Fit ---\n"))
cat(sprintf("  WAIC:    %.2f (se = %.2f)\n", stats$WAIC, stats$WAIC_SE))
cat(sprintf("  Bayes R2:  %.4f\n", stats$Bayes_R2))

selected <- ranking$Variable[ranking$Selection == "Selected"]
weak     <- ranking$Variable[ranking$Selection == "Weak"]

cat("\n--- Variable Selection Summary ---\n")
if (length(selected) > 0) {
  cat("  Selected (PIP > 0.50):", paste(selected, collapse = ", "), "\n")
} else {
  cat("  Selected (PIP > 0.50): (none)\n")
}
if (length(weak) > 0) {
  cat("  Weak   (PIP > 0.10):", paste(weak, collapse = ", "), "\n")
}
excluded <- ranking$Variable[ranking$Selection == "Excluded"]
if (length(excluded) > 0) {
  cat("  Excluded (PIP <= 0.10):", paste(excluded, collapse = ", "), "\n")
}
cat("===== \n")
}

# -----
# bayesian_hausman: compare FE vs RE slope posteriors
# For each variable, posterior of (beta_FE - beta_RE);
# if the 95% interval excludes 0 -> RE inconsistent for that variable.
# -----
baysian_hausman <- function(fit_fe, fit_re, data, outcome_var = "Income",
                           effect_threshold = 0.01) {
  slopes <- attr(fit_fe, "bpvs_slopes") %||% attr(fit_re, "bpvs_slopes")

  post_fe <- as_draws_df(fit_fe)
  post_re <- as_draws_df(fit_re)

  delta_list <- lapply(slopes, function(v) {
    fe_col <- paste0("b_eta_", v)
    re_col <- paste0("b_", v)
    delta <- post_fe[[fe_col]] - post_re[[re_col]]
    data.frame(
      Variable = v,
      Delta    = delta
    )
  })
  delta_df <- do.call(rbind, delta_list)

  delta_df %>%
    group_by(Variable) %>%
    summarise(
      Delta_median = median(Delta),
      Delta_q025   = quantile(Delta, 0.025),
      Delta_q975   = quantile(Delta, 0.975),
      Pr_delta_not0 = mean(abs(Delta) > effect_threshold * sd(data[[outcome_var]]), na.rm = TRUE)),
      groups = "drop"
    ) %>%
    mutate(
      RE_Consistent = ifelse(Delta_q025 > 0 | Delta_q975 < 0, "No (RE biased)", "Yes")
    )
  }

# -----
# hausman_plm: Frequentist Hausman test via plm (auxiliary)
# -----
hausman_plm <- function(formula, data, id_var = "id", time_var = "year") {
  if (!requireNamespace("plm", quietly = TRUE)) return(NULL)
  pdf <- tryCatch(
    plm::pdata.frame(as.data.frame(data), index = c(id_var, time_var)),

```

```

    error = function(e) NULL
  )
  if (is.null(pdf)) return(NULL)
  fe <- tryCatch(plm::plm(formula, data = pdf, model = "within"),
    error = function(e) NULL)
  re <- tryCatch(plm::plm(formula, data = pdf, model = "random"),
    error = function(e) NULL)
  if (is.null(fe) || is.null(re)) return(NULL)
  h <- tryCatch(plm::phtest(fe, re), error = function(e) NULL)
  if (is.null(h)) return(NULL)
  list(
    statistic = unname(h$statistic),
    p.value = h$p.value
  )
}

# -----
# compare_bpvs: FE vs RE comparison table + Hausman checks
# -----
compare_bpvs <- function(fit_fe, fit_re, data, outcome_var = "Income",
  formula = NULL, id_var = "id", time_var = "year",
  effect_threshold = 0.01) {

  fe_pip <- compute_pip(fit_fe, data, outcome_var, effect_threshold)
  re_pip <- compute_pip(fit_re, data, outcome_var, effect_threshold)

  fe_rank <- rank_variables(fe_pip) %>% select(Variable, FE_Estimate = Estimate, FE_PIP = PIP,
    FE_Selection = Selection)
  re_rank <- rank_variables(re_pip) %>% select(Variable, RE_Estimate = Estimate, RE_PIP = PIP,
    RE_Selection = Selection)

  comparison <- merge(fe_rank, re_rank, by = "Variable", all = TRUE) %>%
    arrange(desc(FE_PIP))

  stats_fe <- extract_model_stats(fit_fe)
  stats_re <- extract_model_stats(fit_re)

  bh <- bayesian_hausman(fit_fe, fit_re, data, outcome_var, effect_threshold)
  comparison <- merge(comparison, bh, by = "Variable", all.x = TRUE)

  haus <- hausman_plm(formula, data, id_var, time_var)

  list(
    comparison = comparison,
    stats_fe = stats_fe,
    stats_re = stats_re,
    bayesian_hausman = bh,
    hausman_plm = haus
  )
}

# -----
# print_compare: print FE vs RE comparison
# -----
print_compare <- function(cmp) {
  cat("\n=====\\n")
  cat("BPVS — FE vs RE COMPARISON\\n")
  cat("=====\\n")

  cat("\n--- Model Fit ---\\n")
  cat(sprintf(" FE: WAIC = %.2f (se %.2f), Bayes R2 = %.4f\\n",
    cmp$stats_fe$WAIC, cmp$stats_fe$WAIC_SE, cmp$stats_fe$Bayes_R2))
  cat(sprintf(" RE: WAIC = %.2f (se %.2f), Bayes R2 = %.4f\\n",
    cmp$stats_re$WAIC, cmp$stats_re$WAIC_SE, cmp$stats_re$Bayes_R2))

  cat("\n--- FE vs RE: Estimates and PIPs ---\\n")
  print(cmp$comparison %>% select(Variable, FE_Estimate, RE_Estimate,
    FE_PIP, RE_PIP, FE_Selection, RE_Selection),
    row.names = FALSE)
}

```

```

cat("\n--- Bayesian Hausman (posterior of beta_FE - beta_RE) ---\n")
print(cmp$bayesian_hausman %>% select(Variable, Delta_median, Delta_q025,
  Delta_q975, Pr_delta_not0, RE_Consistent),
  row.names = FALSE)
n_biased <- sum(cmp$bayesian_hausman$RE_Consistent == "No (RE biased)")
if (n_biased > 0) {
  cat(sprintf(" %d variable(s) show FE/RE disagreement -> RE potentially biased there.\n", n_biased))
} else {
  cat(" No FE/RE disagreement -> RE consistent with FE.\n")
}

if (!is.null(cmp$hausman_plm)) {
  cat("\n--- Frequentist Hausman (plm, auxiliary) ---\n")
  cat(sprintf(" Statistic = %.3f, p-value = %.4f\n",
    cmp$hausman_plm$statistic, cmp$hausman_plm$p.value))
  if (cmp$hausman_plm$p.value < 0.05) {
    cat(" p < 0.05 -> RE inconsistent; prefer BPVS-FE.\n")
  } else {
    cat(" p >= 0.05 -> no evidence against RE; BPVS-RE acceptable.\n")
  }
}
cat("===== \n")
}

# -----
# bpvs_pipeline: fit (FE and/or RE) + compute + rank + print
# -----
bpvs_pipeline <- function(formula, data, effects = c("fe", "re"),
  outcome_var = "Income", id_var = "id",
  time_var = "year", chains = 4, iter = 4000,
  warmup = 2000, cores = 4, seed = 123,
  adapt_delta = 0.95, max_treedepth = 12, ...) {

  effects <- match.arg(effects, several.ok = TRUE)

  results <- list()

  if ("fe" %in% effects) {
    cat("\n--- Fitting BPVS-FE (fixed effects, unit dummies) ---\n")
    fit_fe <- fit_bpvs_fe(formula, data, id_var = id_var, chains = chains,
      iter = iter, warmup = warmup, cores = cores,
      seed = seed, adapt_delta = adapt_delta,
      max_treedepth = max_treedepth, ...)
    fe_pip <- compute_pip(fit_fe, data, outcome_var)
    fe_rank <- rank_variables(fe_pip)
    fe_stat <- extract_model_stats(fit_fe)
    print_bpvs_results(fe_rank, fe_stat, title = "BPVS-FE")
    results$fit_fe <- fit_fe
    results$rank_fe <- fe_rank
    results$pip_fe <- fe_pip
    results$stats_fe <- fe_stat
  }

  if ("re" %in% effects) {
    cat("\n--- Fitting BPVS-RE (random intercepts) ---\n")
    fit_re <- fit_bpvs_re(formula, data, id_var = id_var, chains = chains,
      iter = iter, warmup = warmup, cores = cores,
      seed = seed, adapt_delta = adapt_delta,
      max_treedepth = max_treedepth, ...)
    re_pip <- compute_pip(fit_re, data, outcome_var)
    re_rank <- rank_variables(re_pip)
    re_stat <- extract_model_stats(fit_re)
    print_bpvs_results(re_rank, re_stat, title = "BPVS-RE")
    results$fit_re <- fit_re
    results$rank_re <- re_rank
    results$pip_re <- re_pip
    results$stats_re <- re_stat
  }

  if (all(c("fe", "re") %in% effects)) {

```

```

    cmp <- compare_bpvs(results$fit_fe, results$fit_re, data, outcome_var,
                        formula = formula, id_var = id_var, time_var = time_var)
    print_compare(cmp)
    results$comparison <- cmp
  }

invisible(results)
}

`%||` <- function(a, b) if (is.null(a)) b else a

```

Step 2 – Obtain BPVS estimation outputs

```

# =====
# bpvs_empirical.R — BPVS Real Data Application (FE and RE)
# Pathairat Pastpipatkul and Htwe Ko (2025)
# =====
#
# USAGE:
#   source("bpvs_model.R")
#   source("bpvs_empirical.R")
#
# OUTPUTS:
#   Console — FE & RE rankings, PIPs, selection, comparison + Hausman
#   bpvs_emp_results/ — PNG figures + saved models (.rds)
# =====

cat("=====\n")
cat("BPVS — EMPIRICAL APPLICATION (FE and RE)\n")
cat("=====\n")
cat("\nSources: bpvs_model.R\n\n")

source("bpvs_model.R")
library(patchwork)
library(ggrepel)

set.seed(2025123)

# -----
# 1. Load and prepare data
# -----

# --- EDIT THIS PATH to point to your CSV file ---
data_path <- "C:/Users/User/OneDrive/Desktop/developing.csv"

# --- EDIT THIS if your outcome column has a different name ---
outcome_var <- "Growth"
time_var <- "year"
id_var <- "id"

if (!file.exists(data_path)) {
  stop("Data file not found: ", data_path, "\n",
       "Please update 'data_path' in bpvs_empirical.R to point to your CSV.")
}

data_raw <- read.csv(data_path)
cat(sprintf("Loaded: %s\n", data_path))
cat(sprintf("Dimensions: %d rows x %d columns\n", nrow(data_raw), ncol(data_raw)))

# Drop character identifiers, ensure panel structure
data <- data_raw %>%
  dplyr::select(-any_of("Country")) %>%
  mutate(!sym(id_var) := factor(data[[id_var]]))

cat(sprintf("Panel: %d countries, %d time periods, %d total observations\n",
            n_distinct(data[[id_var]]), n_distinct(data[[time_var]]), nrow(data)))

```

```

if(!outcome_var %in% names(data)) {
  stop("Outcome variable '", outcome_var, "' not found. Adjust 'outcome_var'.")
}

# Identify numeric predictors (exclude id, year, outcome)
predictors <- setdiff(names(data), c(id_var, time_var, outcome_var))
cat("Predictors:", paste(predictors, collapse = ", "), "\n")

# Scale predictors for horseshoe prior
data_scaled <- data
for (v in predictors) {
  m <- mean(data_scaled[[v]], na.rm = TRUE)
  s <- sd(data_scaled[[v]], na.rm = TRUE)
  data_scaled[[v]] <- (data_scaled[[v]] - m) / s
}

# -----
# 2. Build formula
# -----
formula <- as.formula(paste(outcome_var, "~", paste(predictors, collapse = " + ")))
cat("Model formula:\n")
print(formula)

# -----
# 3. Fit BPVS-FE and BPVS-RE
# -----
cat("\nFitting BPVS-FE and BPVS-RE (horseshoe prior)...\n")

res <- bpvs_pipeline(
  formula = formula,
  data = data_scaled,
  effects = c("fe", "re"),
  outcome_var = outcome_var,
  id_var = id_var,
  time_var = time_var,
  chains = 4,
  iter = 4000,
  warmup = 2000,
  cores = 4,
  seed = 789,
  adapt_delta = 0.95,
  max_treedepth = 12,
  refresh = 1000
)

fit_fe <- res$fit_fe
fit_re <- res$fit_re
rank_fe <- res$rank_fe
rank_re <- res$rank_re
cmp <- res$comparison

# -----
# 4. Save models and results
# -----
output_dir <- "bpvs_emp_results"
if (!dir.exists(output_dir)) dir.create(output_dir)

saveRDS(fit_fe, file.path(output_dir, "bpvs_fit_fe.rds"))
saveRDS(fit_re, file.path(output_dir, "bpvs_fit_re.rds"))
write.csv(rank_fe, file.path(output_dir, "bpvs_ranking_fe.csv"), row.names = FALSE)
write.csv(rank_re, file.path(output_dir, "bpvs_ranking_re.csv"), row.names = FALSE)
write.csv(cmp$comparison, file.path(output_dir, "bpvs_comparison.csv"), row.names = FALSE)
write.csv(cmp$bayesian_hausman, file.path(output_dir, "bpvs_bayes_hausman.csv"), row.names = FALSE)
if (!is.null(cmp$hausman_plm)) {
  write.csv(as.data.frame(cmp$hausman_plm), file.path(output_dir, "bpvs_hausman_plm.csv"),
    row.names = FALSE)
}

# -----
# 5. Visualizations

```

```

# -----
# 5a. Inclusion probability bar chart (FE vs RE)
make_inclusion_plot <- function(ranking, title, filename) {
  ranking$Variable <- factor(ranking$Variable, levels = rev(ranking$Variable))
  p <- ggplot(ranking, aes(x = Variable, y = PIP)) +
    geom_segment(aes(xend = Variable, yend = 0), color = "gray60", linewidth = 0.5) +
    geom_point(aes(color = Selection), size = 4) +
    geom_hline(yintercept = 0.5, linetype = "dashed", color = "red", alpha = 0.5) +
    geom_hline(yintercept = 0.1, linetype = "dotted", color = "orange", alpha = 0.5) +
    scale_color_manual(values = c("Selected" = "#1b9e77", "Weak" = "#d95f02",
                                   "Excluded" = "#7570b3")) +
    labs(title = title,
         subtitle = "Dashed red = 0.5 (Selected), dotted orange = 0.1 (Weak signal)",
         x = "Predictor", y = "PIP = Pr(|beta| > eps | data)",
         color = "Decision") +
    ylim(0, 1.05) + coord_flip() +
    theme_minimal() + theme(legend.position = "bottom")
  ggsave(file.path(output_dir, filename), p, width = 8, height = max(4, 0.5 * nrow(ranking)))
  p
}

p_incl_fe <- make_inclusion_plot(rank_fe, "BPVS-FE: Posterior Inclusion Probabilities",
                                "inclusion_probabilities_FE.png")
p_incl_re <- make_inclusion_plot(rank_re, "BPVS-RE: Posterior Inclusion Probabilities",
                                "inclusion_probabilities_RE.png")

# 5b. Coefficient plot with 95% CIs
make_coef_plot <- function(ranking, title, filename) {
  ranking$Variable <- factor(ranking$Variable, levels = ranking$Variable)
  p <- ggplot(ranking, aes(x = Estimate, y = Variable)) +
    geom_vline(xintercept = 0, linetype = "dashed", alpha = 0.3) +
    geom_pointrange(aes(xmin = CI_lower, xmax = CI_upper, color = Selection),
                   size = 0.9, fatten = 2.5) +
    scale_color_manual(values = c("Selected" = "#1b9e77", "Weak" = "#d95f02",
                                   "Excluded" = "#7570b3")) +
    labs(title = title,
         subtitle = "Horseshoe prior shrinks irrelevant coefficients toward zero",
         x = "Posterior Median (95% CI)", y = "Predictor", color = "Decision") +
    theme_minimal() + theme(legend.position = "bottom")
  ggsave(file.path(output_dir, filename), p, width = 8, height = max(4, 0.6 * nrow(ranking)))
  p
}

p_coef_fe <- make_coef_plot(rank_fe, "BPVS-FE: Coefficient Estimates (95% CI)",
                             "coefficient_estimates_FE.png")
p_coef_re <- make_coef_plot(rank_re, "BPVS-RE: Coefficient Estimates (95% CI)",
                             "coefficient_estimates_RE.png")

# 5c. PIP comparison FE vs RE
pip_compare <- merge(
  rank_fe %>% select(Variable, PIP_FE = PIP),
  rank_re %>% select(Variable, PIP_RE = PIP),
  by = "Variable", all = TRUE
)
p_pip_compare <- ggplot(pip_compare, aes(x = PIP_FE, y = PIP_RE, label = Variable)) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "gray50") +
  geom_point(size = 4, color = "steelblue", alpha = 0.8) +
  geom_text_repel(size = 3.5) +
  coord_fixed(xlim = c(0, 1), ylim = c(0, 1)) +
  labs(title = "BPVS: Posterior Inclusion Probabilities — FE vs RE",
       x = "BPVS-FE PIP", y = "BPVS-RE PIP") +
  theme_minimal()
ggsave(file.path(output_dir, "pip_FE_vs_RE.png"), p_pip_compare, width = 6, height = 6)

# 5d. Trace plots for convergence (slopes)
beta_params_fe <- grep("^b_eta_", variables(fit_fe), value = TRUE)
if (length(beta_params_fe) > 12) beta_params_fe <- beta_params_fe[1:12]
p_trace_fe <- mcmc_trace(fit_fe, pars = beta_params_fe, facet_args = list(ncol = 3))
ggsave(file.path(output_dir, "trace_plots_FE.png"), p_trace_fe,

```

```

width = 12, height = 3 * ceiling(length(beta_params_fe) / 3))

beta_params_re <- setdiff(grep("^b_", variables(fit_re), value = TRUE), "b_Intercept")
if (length(beta_params_re) > 12) beta_params_re <- beta_params_re[1:12]
p_trace_re <- mcmc_trace(fit_re, pars = beta_params_re, facet_args = list(ncol = 3))
ggsave(file.path(output_dir, "trace_plots_RE.png"), p_trace_re,
        width = 12, height = 3 * ceiling(length(beta_params_re) / 3))

# Display
print(p_incl_fe)
print(p_incl_re)
print(p_coef_fe)
print(p_coef_re)
print(p_pip_compare)

cat(sprintf("\nResults saved to: %s\n", output_dir))
cat("=====\n")
cat("EMPIRICAL APPLICATION COMPLETE\n")
cat("=====\n")

```