```r
### SCRIPT 12 - R ###
#
#  Análisis SOM no supervisado (5 variables) + K-means (K = 8)
#  Mapeo de variable objetivo Ahorro_final = (Presupuesto - Precio_final)/Presupuesto
#
# ================================================================

# ---- Paquetes ----------------------------------------------------

dep_pkgs <- c(
  "kohonen",
  "dplyr",
  "tidyr",
  "readxl",
  "readr",
  "ggplot2",
  "tibble",
  "stringr",
  "colorspace"
)

invisible(lapply(dep_pkgs, function(p) {
  if (!requireNamespace(p, quietly = TRUE)) install.packages(p)
}))

lapply(dep_pkgs, library, character.only = TRUE) |> invisible()

set.seed(321)
options(dplyr.summarise.inform = FALSE)

# ---- Parámetros y rutas ------------------------------------------

root_dir  <- "F:/Tesis Guillermo_casa/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/__final_Modelos"
file_train <- file.path(root_dir, "df_train_v5_completo.xlsx")
file_test  <- file.path(root_dir, "df_test_v5_completo.xlsx")

# Variables exactas del dataset
vars_som <- c(
  "Presupuesto_licitacion_lote_c",
  "N_ofertantes",
  "C_precio_p",
  "Plazo_m",
  "Baja_p"
)

# Objetivo a mapear en el SOM
obj_label <- "Ahorro_final"  # (Presupuesto - Precio_final) / Presupuesto
req_obj_cols <- c("Presupuesto_licitacion_lote_c", "Precio_final_e_c")

# Variables disponibles en licitación para XYF
vars_licit <- c(
  "N_lotes",
  "Intervalo_urge_d",
  "N_clasi_empresa",
  "N_CPV",
  "Tipo_de_Administracion_c",
  "Tipo_de_procedimiento_c"
)

k_som  <- 8
som_xy <- c(10, 10)

# ---- Utilidades --------------------------------------------------

check_required_cols <- function(df, cols, name) {
  miss <- setdiff(cols, names(df))
  if (length(miss) > 0) stop(sprintf("Faltan columnas en %s: %s", name, paste(miss, collapse = ", ")))
}

scale_fit <- function(x) {
  m <- scale(x)
```

```r
  list(x = m, params = list(center = attr(m, "scaled:center"), scale = attr(m,
  "scaled:scale")))
}

scale_apply <- function(x, params) {
  scale(x, center = params$center, scale = params$scale)
}

make_mm_licit <- function(df) {
  df2 <- df |>
    mutate(
      Tipo_de_Administracion_c = as.factor(Tipo_de_Administracion_c),
      Tipo_de_procedimiento_c  = as.factor(Tipo_de_procedimiento_c)
    )
  mm <- stats::model.matrix(
    ~ 0 + N_lotes + Intervalo_urge_d + N_clasi_empresa + N_CPV +
      Tipo_de_Administracion_c + Tipo_de_procedimiento_c,
    data = df2
  )
  storage.mode(mm) <- "double"
  mm
}

align_cols <- function(x_new, col_ref) {
  miss <- setdiff(col_ref, colnames(x_new))
  if (length(miss) > 0) x_new <- cbind(x_new, matrix(0, nrow = nrow(x_new), ncol =
  length(miss), dimnames = list(NULL, miss)))
  extra <- setdiff(colnames(x_new), col_ref)
  if (length(extra) > 0) x_new <- x_new[, setdiff(colnames(x_new), extra), drop =
  FALSE]
  x_new[, col_ref, drop = FALSE]
}

majority_label <- function(x) { x <- as.character(x); tab <- table(x); names(tab)[
which.max(tab)] }

pal_cf <- function(n, alpha = 1) rainbow(n, end = 4/6, alpha = alpha)[n:1]
hi_contrast <- function(n, alpha = 1) colorspace::qualitative_hcl(n, palette = "Set3"
, alpha = alpha)

# ---- Carga de datos ----------------------------------------------

message("Leyendo train/test desde: ", root_dir)

df_train <- readxl::read_xlsx(file_train) |> tibble::as_tibble()
df_test  <- readxl::read_xlsx(file_test)  |> tibble::as_tibble()

check_required_cols(df_train, c(vars_som, req_obj_cols, vars_licit), "train")
check_required_cols(df_test,  c(vars_som, req_obj_cols, vars_licit), "test")

# ---- Preparación para SOM (5 variables) ------------------------

x_train_raw <- df_train |>
  select(all_of(vars_som)) |>
  mutate(across(everything(), as.numeric))

x_test_raw <- df_test |>
  select(all_of(vars_som)) |>
  mutate(across(everything(), as.numeric))

keep_train <- stats::complete.cases(x_train_raw)
keep_test  <- stats::complete.cases(x_test_raw)

message(sprintf("SOM | Train completas: %d/%d | Test completas: %d/%d", sum(keep_train
), nrow(df_train), sum(keep_test), nrow(df_test)))

x_train_raw <- x_train_raw[keep_train, , drop = FALSE]
x_test_raw  <- x_test_raw[keep_test,  , drop = FALSE]

# Objetivo = Ahorro_final
pli  <- df_train$Presupuesto_licitacion_lote_c[keep_train]
pfin <- df_train$Precio_final_e_c[keep_train]
y_train_obj <- (pli - pfin) / pli
```

```r
y_train_obj[!is.finite(y_train_obj)] <- NA_real_

# Escalado SOM
sc_som <- scale_fit(as.matrix(x_train_raw))
x_train <- sc_som$x
x_test  <- scale_apply(as.matrix(x_test_raw), sc_som$params)

# ---- Entrenamiento SOM ----------------------------------------

som_grid <- kohonen::somgrid(xdim = som_xy[1], ydim = som_xy[2], topo = "hexagonal",
toroidal = FALSE)

set.seed(321)
som_model <- kohonen::som(
  X         = x_train,
  grid      = som_grid,
  rlen      = 10000,
  alpha     = c(0.05, 0.005),
  keep.data = TRUE
)

# K-means sobre códigos del SOM
set.seed(321)
codes_matrix <- som_model$codes[[1]]
km_codes     <- stats::kmeans(codes_matrix, centers = k_som, nstart = 50, iter.max =
100)
unit_to_cluster <- km_codes$cluster

eur_train    <- som_model$unit.classif
cl_train_som <- unit_to_cluster[eur_train]

# ---- Resúmenes (neurona/cluster) ------------------------------

obj_por_neur <- tapply(y_train_obj, eur_train, mean, na.rm = TRUE)
prop_vec <- rep(NA_real_, som_xy[1] * som_xy[2])
prop_vec[as.integer(names(obj_por_neur))] <- as.numeric(obj_por_neur)

cluster_mean_obj <- tibble(
  cluster   = factor(sort(unique(cl_train_som))),
  obj_medio = as.numeric(tapply(y_train_obj, cl_train_som, mean, na.rm = TRUE)[levels(
  factor(cl_train_som))])
)

resumen_clusters <- tibble(cluster = factor(cl_train_som), obj = y_train_obj) |>
  dplyr::group_by(cluster) |>
  dplyr::summarise(
    n_casos = dplyr::n(),
    obj_media = mean(obj, na.rm = TRUE),
    obj_sd = sd(obj, na.rm = TRUE)
  ) |>
  dplyr::ungroup() |>
  arrange(as.integer(as.character(cluster)))

# ---- Visualización --------------------------------------------

pdf(file.path(root_dir, "SOM_K8_Simple_Mapas.pdf"), width = 8.3, height = 11.7)
par(mfrow = c(4, 4), mar = c(2, 2, 3, 1))

plot(som_model, type = "dist.neighbours", main = "U-Matrix", palette.name = pal_cf,
shape = "straight")
add.cluster.boundaries(som_model, unit_to_cluster, lwd = 1.75, col = "white")

for (i in seq_len(ncol(x_train))) {
  plot(som_model, type = "property", property = som_model$codes[[1]][, i],
       main = colnames(x_train)[i], palette.name = pal_cf, shape = "straight")
  add.cluster.boundaries(som_model, unit_to_cluster, lwd = 1.75, col = "white")
}

plot(som_model, type = "count", main = "Casos por neurona", palette.name = pal_cf,
shape = "straight")
add.cluster.boundaries(som_model, unit_to_cluster, lwd = 1.75, col = "white")

plot(som_model, type = "property", property = prop_vec,
```

```r
      main = paste0("Media ", obj_label, " por neurona (train)"),
      palette.name = pal_cf, shape = "straight")
add.cluster.boundaries(som_model, unit_to_cluster, lwd = 1.75, col = "white")

dev.off()

# Pie chart de componentes
pdf(file.path(root_dir, "SOM_K8_Simple_Codes_PieChart.pdf"), width = 8.3, height = 11.7)
plot(som_model, type = "codes", main = "SOM K8 - Componentes (Pie Chart)", palette.name = hi_contrast, shape = "straight")
add.cluster.boundaries(som_model, unit_to_cluster, lwd = 1.75, col = "white")
dev.off()

# Fondo con objetivo
pdf(file.path(root_dir, "SOM_K8_Simple_Background_AhorroFinal.pdf"), width = 8.3, height = 11.7)
plot(som_model, type = "property", property = prop_vec, main = paste0("Media ", obj_label, " por neurona (train)"), palette.name = pal_cf, shape = "straight")
add.cluster.boundaries(som_model, unit_to_cluster, lwd = 1.75, col = "white")
dev.off()

# ---- Predicción con SOM ----------------------------------------

predict_som_cluster <- function(new_x_scaled) {
  if (is.null(dim(new_x_scaled)) || any(dim(new_x_scaled) == 0)) {
    return(tibble(neurona_bmu = integer(0), cluster_som = factor(character(0), levels = sort(unique(unit_to_cluster)))))
  }
  mapped <- predict(som_model, newdata = as.matrix(new_x_scaled))
  bmu <- as.integer(mapped$unit.classif)
  cl  <- unit_to_cluster[bmu]
  tibble(neurona_bmu = bmu, cluster_som = factor(cl, levels = sort(unique(unit_to_cluster))))
}

add_expected_obj <- function(tbl_clusters) {
  tbl_clusters |>
    left_join(cluster_mean_obj, by = c("cluster_som" = "cluster")) |>
    rename(esperado_cluster = obj_medio)
}

# ---- Clasificación con XYF (predict de Kohonen) ----------------

train_som_rows <- df_train[keep_train, , drop = FALSE]
keep_xyf_train <- stats::complete.cases(train_som_rows[, vars_licit])
message(sprintf("XYF | Train completas (licitación): %d/%d", sum(keep_xyf_train), sum(keep_train)))

if (sum(keep_xyf_train) > 0) {
  y_train_xyf <- factor(cl_train_som[keep_xyf_train])

  x_licit_train_mm <- make_mm_licit(train_som_rows[keep_xyf_train, vars_licit, drop = FALSE])
  sc_xyf <- scale_fit(x_licit_train_mm)
  x_licit_train <- sc_xyf$x
  mm_cols <- colnames(x_licit_train)

  xyf_grid <- kohonen::somgrid(xdim = som_xy[1], ydim = som_xy[2], topo = "hexagonal", toroidal = FALSE)

  set.seed(321)
  xyf_model <- kohonen::xyf(
    X    = x_licit_train,
    Y    = y_train_xyf,
    grid = xyf_grid,
    rlen = 5000,
    alpha = c(0.05, 0.005),
    keep.data = TRUE
  )

  unit_xyf_train <- xyf_model$unit.classif
  unit_labels <- tibble(unit = unit_xyf_train, label = y_train_xyf) |>
```

```r
    summarise(label = majority_label(label), .by = unit)

} else {
  message("Aviso: no hay filas completas en TRAIN para variables de licitación; no se entrena XYF.")
  xyf_model <- NULL
  sc_xyf <- list(params = list(center = NULL, scale = NULL))
  x_licit_train <- matrix(numeric(0), nrow = 0, ncol = 0)
  mm_cols <- character(0)
  xyf_grid <- NULL
  unit_labels <- tibble(unit = integer(0), label = factor(character(0)))
}

predict_xyf_cluster <- function(new_mm_scaled) {
  # Devuelve tantas filas como new_mm_scaled tenga; NAs si no se puede predecir
  if (is.null(xyf_model)) {
    n_rows <- if (!is.null(new_mm_scaled)) NROW(new_mm_scaled) else 0L
    return(tibble(
      neurona_bmu_xyf = rep(NA_integer_, n_rows),
      cluster_xyf = factor(rep(NA_character_, n_rows), levels = levels(y_train_xyf))
    ))
  }
  if (is.null(new_mm_scaled)) {
    return(tibble(neurona_bmu_xyf = integer(0),
                  cluster_xyf = factor(character(0), levels = levels(y_train_xyf))))
  }
  new_mm_scaled <- as.matrix(new_mm_scaled)
  n_rows <- nrow(new_mm_scaled)
  if (is.null(dim(new_mm_scaled)) || n_rows == 0 || ncol(new_mm_scaled) == 0) {
    return(tibble(neurona_bmu_xyf = integer(0),
                  cluster_xyf = factor(character(0), levels = levels(y_train_xyf))))
  }
  storage.mode(new_mm_scaled) <- "double"
  pr <- tryCatch(
    predict(xyf_model, newdata = list(X = new_mm_scaled), whatmap = 1, maxNA.fraction = 1),
    error = function(e) { message("Aviso: predict(XYF) omitido: ", e$message); NULL }
  )
  if (is.null(pr)) {
    return(tibble(
      neurona_bmu_xyf = rep(NA_integer_, n_rows),
      cluster_xyf = factor(rep(NA_character_, n_rows), levels = levels(y_train_xyf))
    ))
  }
  bmu <- as.integer(pr$unit.classif)
  lab <- unit_labels$label[match(bmu, unit_labels$unit)]
  tibble(neurona_bmu_xyf = bmu, cluster_xyf = factor(lab, levels = levels(y_train_xyf)))
}

# ---- Predicciones en TEST -------------------------------------

if (sum(keep_test) > 0) {
  som_test_rows <- tibble(.row_id = which(keep_test)) |>
    bind_cols(predict_som_cluster(x_test)) |>
    add_expected_obj()
} else {
  som_test_rows <- tibble(.row_id = integer(0), neurona_bmu = integer(0),
                          cluster_som = factor(character(0), levels = sort(unique(unit_to_cluster))),
                          esperado_cluster = numeric(0))
}

keep_xyf_test <- stats::complete.cases(df_test[, vars_licit])
message(sprintf("XYF | Test completas (licitación): %d/%d", sum(keep_xyf_test), nrow(df_test)))
if (sum(keep_xyf_test) > 0 && !is.null(xyf_model)) {
  x_licit_test_mm_full <- make_mm_licit(df_test[keep_xyf_test, vars_licit, drop = FALSE])
  # Alinear columnas del test a las del train (mm_cols)
  x_licit_test_mm <- align_cols(x_licit_test_mm_full, mm_cols)

  # Debug opcional: categorías faltantes o extra
```

```r
    miss_cols <- setdiff(mm_cols, colnames(x_licit_test_mm_full))
    extra_cols <- setdiff(colnames(x_licit_test_mm_full), mm_cols)
    if (length(miss_cols) > 0) message("XYF | Dummies faltantes en TEST añadidas como 0: ", paste(miss_cols, collapse = ", "))
    if (length(extra_cols) > 0) message("XYF | Dummies extra en TEST ignoradas: ", paste(extra_cols, collapse = ", "))

    if (nrow(x_licit_test_mm) > 0 && ncol(x_licit_test_mm) > 0) {
      x_licit_test <- scale_apply(as.matrix(x_licit_test_mm), sc_xyf$params)

      xyf_preds <- predict_xyf_cluster(x_licit_test)
      xyf_test_rows <- tibble(.row_id = which(keep_xyf_test)) |>
        bind_cols(xyf_preds)
    } else {
      message("Aviso: matriz XYF de TEST sin dimensión válida; se omiten predicciones XYF.")
      xyf_test_rows <- tibble(.row_id = integer(0), neurona_bmu_xyf = integer(0),
                              cluster_xyf = factor(character(0), levels = sort(unique(cl_train_som))))
    }
  } else {
    if (sum(keep_xyf_test) == 0) message("Aviso: ninguna fila de TEST completa para licitación; se omiten predicciones XYF.")
    xyf_test_rows <- tibble(.row_id = integer(0), neurona_bmu_xyf = integer(0),
                            cluster_xyf = factor(character(0), levels = sort(unique(cl_train_som))))
  }

  pred_test <- tibble(.row_id = seq_len(nrow(df_test))) |>
    left_join(som_test_rows, by = ".row_id") |>
    left_join(xyf_test_rows, by = ".row_id")

  # ---- Evaluación en TEST: Confusión y Scatter -------------------
  # Ahorro real
  ahorro_real_test <- with(df_test, (Presupuesto_licitacion_lote_c - Precio_final_e_c) / Presupuesto_licitacion_lote_c)
  ahorro_real_test[!is.finite(ahorro_real_test)] <- NA_real_

  # Añadir real y renombrar predicción de SOM
  pred_test <- pred_test |>
    mutate(
      ahorro_real     = ahorro_real_test,
      ahorro_pred_som = esperado_cluster
    )

  # Predicción de ahorro por XYF (media del cluster XYF)
  pred_test <- pred_test |>
    left_join(cluster_mean_obj |>
                rename(cluster_xyf = cluster, ahorro_pred_xyf = obj_medio),
              by = "cluster_xyf")

  # Matriz de confusión SOM vs XYF (solo filas con ambos clusters)
  idx_cm <- stats::complete.cases(pred_test$cluster_som, pred_test$cluster_xyf)
  if (any(idx_cm)) {
    cm <- table(SOM = pred_test$cluster_som[idx_cm], XYF = pred_test$cluster_xyf[idx_cm])
    acc <- mean(pred_test$cluster_som[idx_cm] == pred_test$cluster_xyf[idx_cm])
    cm_df <- as.data.frame.matrix(cm)
    readr::write_csv(tibble::rownames_to_column(as.data.frame(cm_df), var = "SOM"),
                     file.path(root_dir, "SOM_K8_Simple_confusion_SOM_vs_XYF.csv"))
  } else {
    acc <- NA_real_
  }

  # Métricas de ajuste (real vs predicho)
  mae  <- function(a, p) { idx <- stats::complete.cases(a, p); if (!any(idx)) return(NA_real_); mean(abs(a[idx] - p[idx])) }
  rmse <- function(a, p) { idx <- stats::complete.cases(a, p); if (!any(idx)) return(NA_real_); sqrt(mean((a[idx] - p[idx])^2)) }
  correl <- function(a, p) { idx <- stats::complete.cases(a, p); if (sum(idx) < 2) return(NA_real_); suppressWarnings(cor(a[idx], p[idx])) }

  metrics_test <- dplyr::bind_rows(
```

```r
  tibble(modelo = "SOM",
         MAE = mae(pred_test$ahorro_real, pred_test$ahorro_pred_som),
         RMSE = rmse(pred_test$ahorro_real, pred_test$ahorro_pred_som),
         COR = correl(pred_test$ahorro_real, pred_test$ahorro_pred_som)),
  tibble(modelo = "XYF",
         MAE = mae(pred_test$ahorro_real, pred_test$ahorro_pred_xyf),
         RMSE = rmse(pred_test$ahorro_real, pred_test$ahorro_pred_xyf),
         COR = correl(pred_test$ahorro_real, pred_test$ahorro_pred_xyf))
) |>
  mutate(ACC_clusters_SOM_vs_XYF = acc)

readr::write_csv(metrics_test, file.path(root_dir, "SOM_K8_Simple_test_metrics.csv"))

# Log dimensiones para depuración
message(sprintf("XYF dims | train: %d x %d | test: %d x %d | preds: %d filas",
                ncol(x_licit_train), nrow(x_licit_train),
                if (exists("x_licit_test")) ncol(x_licit_test) else -1,
                if (exists("x_licit_test")) nrow(x_licit_test) else -1,
                nrow(xyf_test_rows)))

# Scatter real vs predicho
plot_df <- tidyr::pivot_longer(
  pred_test,
  cols = c(ahorro_pred_som, ahorro_pred_xyf),
  names_to = "metodo",
  values_to = "ahorro_pred"
)

p_sc <- ggplot(plot_df, aes(x = ahorro_real, y = ahorro_pred, color = metodo)) +
  geom_point(alpha = 0.6) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "grey40") +
  scale_color_manual(values = c(ahorro_pred_som = "steelblue", ahorro_pred_xyf =
  "darkorange"),
                     labels = c(ahorro_pred_som = "SOM", ahorro_pred_xyf = "XYF")) +
  labs(title = "TEST | Ahorro real vs predicho", x = "Ahorro real", y = "Ahorro
  predicho", color = "Modelo") +
  theme_minimal()

ggsave(file.path(root_dir, "SOM_K8_Simple_Test_Scatter_Ahorro.pdf"), plot = p_sc,
width = 8, height = 6)

# ---- Guardado ----------------------------------------------------

saveRDS(list(
  som_model        = som_model,
  som_grid         = som_grid,
  kmeans_codes     = km_codes,
  unit_to_cluster  = unit_to_cluster,
  scaler_params_som= sc_som$params,
  vars_som         = vars_som,
  cluster_mean_obj = cluster_mean_obj,
  objetivo_label   = obj_label
), file.path(root_dir, "SOM_K8_Simple_model.rds"))

saveRDS(list(
  xyf_model        = xyf_model,
  xyf_grid         = xyf_grid,
  scaler_params_xyf= sc_xyf$params,
  mm_cols_xyf      = mm_cols,
  unit_labels_xyf  = unit_labels,
  vars_licit       = vars_licit
), file.path(root_dir, "SOM_K8_Simple_xyf_model.rds"))

readr::write_csv(resumen_clusters, file.path(root_dir,
"SOM_K8_Simple_resumen_clusters.csv"))
readr::write_csv(pred_test,        file.path(root_dir, "SOM_K8_Simple_pred_TEST.csv"))

cat("\n=== Resumen por cluster (SOM + K=8) ===\n"); print(resumen_clusters)
cat("\nArchivos guardados en: ", root_dir, "\n")

# -------------------------- FIN --------------------------------
```