```r
### SCRIPT 9 - R ###
# Tratamiento y limpieza de datos global de la base de datos
# hasta obtener la base de datos principal (3900 casos)
#########################################################

### Lectura de Librerias -----
library(lubridate)
library(XML)
library(corrplot)
library(dplyr)
library(foreach)
library(stringr)
library(stringi)
library(doParallel)
library(DBI)
library(ggplot2)
library(dplyr)
library(openxlsx)
library(RODBC)
library(quanteda) #paquete recomendado en todos laos
library(quanteda.textmodels) #aux
library(quanteda.textstats) # aux
library(readtext) # sencilla manera de leer data de texto en R
library(spacyr) # NLP usando la libreria spaCy, incluyendo etiquetado part-of-speech, entity recognition y dependency parsing.
library(zoo)
library(DBI)
library(tidyr)
library(readr)
#setup parallel backend to use many processors
cores=detectCores()
cl <- makeCluster(cores[1]-1, type = "PSOCK", outfile = "log.txt") #not to overload your computer #,outfile="log.txt"
registerDoParallel(cl)

f_ini <- Sys.time()


## Directorio ORIGEN----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP")


### Lectura de datos ----

#Awarding
load("DF_Awarding_Final.RData")

#Modificaciones
load("DF_Modis_FinalPrice.RData")

#Base de datos completa
load("BD_PLACSP.RData")


#Base de datos final de AEIPRO 2025, casos mas limpios

load("DF_Modis_Award_clean.RData")

##Limpieza y transformacion de DF_Modis_Award_clean ----
DF_Modis_Award_clean <- DF_Modis_Award_clean |>
  # Transformar y seleccionar columnas con renombramiento
  transmute(
    Identificador,
    Identificador_lici,
    Identificador_lote,

    CPV_lote = CPV_division,
    Ultima_actualizacion=Last_updated,
    Modificacion_coste=Modification_cost,
    Modificacion_p=Modification_p,
    Baja_porcentaje=Bid_reduction_p,
```

```r
    Intervalo_hasta_modi_m=Interval_time_m,
    C_precio_p=Price_p,
    C_otros_objetivos_p=Other_objective_p,
    C_juicios_valor_p=Value_judgement_p,

    Precio_final = Final_Price
  ) |>
  filter(!is.na(Precio_final)) #Quitamos casos con NA en precio final

DF_Modis_Award_clean <- DF_Modis_Award_clean |>
  mutate(
    Ultima_actualizacion = as.Date(paste0(Ultima_actualizacion, "-01"))
  )


### Limpieza y transformacion DF de PLACSP ------

BD_PLACSP <- BD_PLACSP |> #Seteamos el formato de fecha de nuevo
  mutate(
    Primera.publicacion = as.Date(as.numeric(Primera.publicacion), origin =
    "1899-12-30")
  )


BD_PLACSP_clean <- BD_PLACSP |> # Quitamos error de cargar XML
  mutate(across(
    where(is.character),
    ~ str_replace_all(.x, 'xml:space="preserve">', "")
  ))


BD_PLACSP_clean <- BD_PLACSP_clean |>
  # Crear columnas Identificador
  mutate(
    Identificador_lici = str_c(Identificador, coalesce(as.character(
    Numero.de.expediente), "NA"), sep = "//"),
    Identificador_lote = str_c(Identificador, coalesce(as.character(
    Numero.del.contrato.licitacion_lote), "NA"), sep = "//")
  ) |>

  # Transformar y seleccionar columnas con renombramiento
  transmute(
    Identificador,
    Identificador_lici,
    Identificador_lote,
    Primera_publicacion=Primera.publicacion,
    Objeto_del_Contrato=Objeto.del.Contrato,

    Presupuesto_licitacion = parse_number(Presupuesto.base.sin.impuestos),
    CPV = CPV,

    Tipo_de_contrato = Tipo.de.contrato,
    Tipo_de_Administracion = Tipo.de.Administracion,
    Tipo_de_procedimiento = Tipo.de.procedimiento,
    Tramitacion = Tramitacion,
    Lote = Lote,
    Codigo_Postal = Codigo.Postal,
    Tipo_ganador_lote=Tipo.de.identificador.de.adjudicatario.por.licitacion_lote,
    Ganador_NIF = Identificador.Adjudicatario.de.la.licitacion_lote,

    Presupuesto_licitacion_lote = parse_number(
    Presupuesto.base.sin.impuestos.licitacion_lote),
    CPV_lote = CPV.licitacion_lote,

    N_ofertantes = parse_number(Numero.de.ofertas.recibidas.por.licitacion_lote),
    Precio_adjudicacion = parse_number(
    Importe.adjudicacion.sin.impuestos.licitacion_lote)
  ) |>
  filter(!is.na(Precio_adjudicacion)) #Quitamos casos con NA en precio final

## Eliminamos casos duplicados
BD_PLACSP_clean<-unique(BD_PLACSP_clean)

```

```r
## Eliminamos casos duplicados menos en la fecha, que escogemos el más reciente

BD_PLACSP_clean <- BD_PLACSP_clean |>
  group_by(across(-Primera_publicacion)) |>
  slice_max(order_by = Primera_publicacion, n = 1, with_ties = FALSE) |>
  ungroup()

# --- Función auxiliar: divide CPV por ":" y chequea si alguno cumple un patrón ---
tiene_algun_cpv_que_cumple <- function(cpv_string, patron_regex) {
  cpvs <- str_split(cpv_string, ":", simplify = TRUE)
  any(str_detect(cpvs, patron_regex))
}

# --- 1. Casos con al menos un CPV de construcción (empieza por 45) ---
bd_placsp_constru <- BD_PLACSP_clean |>
  filter(
    !is.na(CPV) &
      sapply(CPV, tiene_algun_cpv_que_cumple, patron_regex = "^45")
  )

# --- 2. Casos con al menos un CPV de ingeniería civil (71311) ---
bd_placsp_ingecivil <- BD_PLACSP_clean |>
  filter(
    !is.na(CPV) &
      sapply(CPV, tiene_algun_cpv_que_cumple, patron_regex = "^71311")
  )

# --- 3. Filtro extra: quitar casos donde todos los CPVs son de arquitectura ---
# Arquitectura: 4521 (excepto 133), 452625, 4526261, ..., 452629
# 1. Lista explícita de CPVs de arquitectura (puede ampliarse según necesidad)
cpvs_arquitectura <- c(
  "45210000", "45262500", "45262610", "45262630", "45262650", "45262660",
  "45262670", "45262680", "45262690", "45262700", "45262800", "45262900"
)

# 2. Función: TRUE si todos los CPVs están en la lista de arquitectura


solo_arquitectura <- function(cpv_string) {
  cpvs <- str_split(cpv_string, ":", simplify = TRUE)
  cpvs <- str_trim(cpvs)  # limpiar espacios
  cpvs <- cpvs[cpvs != ""]  # quitar vacíos
  all(cpvs %in% cpvs_arquitectura)
}

# 3. Aplicamos filtro final: nos quedamos con casos que NO son solo arquitectura
bd_placsp_civil <- bd_placsp_constru |>
  filter(!sapply(CPV, solo_arquitectura))

# 4. Contamos casos de tipo Obra
n_casos_obra_civil <- bd_placsp_civil |>
  distinct(Identificador) |>
  filter("Obras" %in% bd_placsp_civil$Tipo_de_contrato) |>
  nrow()


## Hacemos un unique de bd_placsp_civil
bd_placsp_civil<-unique(bd_placsp_civil)

# Cruce por entryID (modificaciones) y Identificador (PLACSP civil)
n_casos_modi_civil <- DF_Modis_FinalPrice |>
  distinct(entryID) |>
  filter(entryID %in% bd_placsp_civil$Identificador) |>
  nrow()


#### CRUZAMOS BASES DE DATOS RESULTADO ------

# La idea general es buscar los casos que DF_Modis_Award_clean tiene dentro de bd_placsp_civil
# Luego creamos un data frame nuevo llamado bd_civil_modis_award, con los datos de ambos
# Buscamos Identificador_lote de bd_plascp_civil dentro de DF_Modis_Award_clean
```

```r
# Si existen más de una coincidencia, quedarnos con los casos (cuantos haya) de DF_Modis_Award_clean,
# y rellenar con los datos de bd_plascp_civil del caso con mayor valor en la variable Primera_publicacion
# Añade una columna Fuente_info_civil como primera columna indicando si el dato fue "Match 1 a 1" o "Match múltiple"

# Lista para guardar resultados finales del cruce
lista_resultados <- list()

# Iterar por cada Identificador_lote único
lotes_unicos <- unique(DF_Modis_Award_clean$Identificador_lote)

for (lote_id in lotes_unicos) {

  # Casos correspondientes en ambas tablas
  modi_casos <- DF_Modis_Award_clean |> filter(Identificador_lote == lote_id)
  civil_casos <- bd_placsp_civil |> filter(Identificador_lote == lote_id)

  # Si no hay match en civil, saltar al siguiente
  if (nrow(civil_casos) == 0) next

  # === CASO 1: 1x1 ===
  if (nrow(modi_casos) == 1 && nrow(civil_casos) == 1) {
    tipo_match <- "Match 1 a 1"
    civil_sel <- civil_casos
  } else {
    # === CASO 2: múltiples coincidencias ===
    tipo_match <- "Match múltiple (más reciente)"
    civil_sel <- civil_casos |>
      arrange(desc(Primera_publicacion)) |>
      slice(1)
  }

  # Repetir civil_sel tantas veces como haya filas en modi_casos
  civil_expandido <- civil_sel[rep(1, nrow(modi_casos)), ]

  # Detectar columnas de civil que NO están ya en modi
  columnas_nuevas <- setdiff(names(civil_expandido), names(modi_casos))

  # Unión: primero el tipo de match, luego columnas de modi, luego solo las nuevas de civil
  combinado <- bind_cols(
    tibble(Fuente_info_civil = tipo_match),
    modi_casos,
    civil_expandido[, columnas_nuevas, drop = FALSE]
  )

  # Agregar a la lista de resultados
  lista_resultados[[length(lista_resultados) + 1]] <- combinado
}

# Combinar todo en un único data frame final
bd_civil_modis_award <- bind_rows(lista_resultados)


# 4. Contamos casos de tipo Obra
n_casos_final_civil <- bd_civil_modis_award |>
  distinct(Identificador) |>
  filter("Obras" %in% bd_civil_modis_award$Tipo_de_contrato) |>
  nrow()

## Directorio GUARDAR ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/_final_conjunto_v4")

#### Guardamos bases de datos resultado 1 -----

save(bd_civil_modis_award, file="bd_civil_modis_award.RData")
save(bd_placsp_civil, file="bd_placsp_civil.RData")
save(bd_placsp_ingecivil, file="bd_placsp_ingecivil.RData")

```

```r
##Exporto csv Inge Civil
write.xlsx(bd_civil_modis_award,"bd_civil_modis_award.xlsx", colNames = T, rowNames=F)
write.xlsx(bd_placsp_civil,"bd_placsp_civil.xlsx", colNames = T, rowNames=F)
write.xlsx(bd_placsp_ingecivil,"bd_placsp_ingecivil.xlsx", colNames = T, rowNames=F)

# Borramos variables auxiliares temporales
rm(civil_casos)
rm(civil_expandido)
rm(civil_sel)
rm(combinado)
rm(columnas_nuevas)
rm(lote_id)
rm(tipo_match)
rm(f_ini)
rm(lotes_unicos)
rm(modi_casos)
rm(lista_resultados)



### Lectura de Librerias -----
library(readxl)
library(lubridate)
library(purrr)
library(XML)
library(corrplot)
library(dplyr)
library(foreach)
library(stringr)
library(stringi)
library(doParallel)
library(DBI)
library(ggplot2)
library(dplyr)
library(openxlsx)
library(RODBC)
library(quanteda) #paquete recomendado en todos laos
library(quanteda.textmodels) #aux
library(quanteda.textstats) # aux
library(readtext) # sencilla manera de leer data de texto en R
library(spacyr) # NLP usando la libreria spaCy, incluyendo etiquetado part-of-speech, entity recognition y dependency parsing.
library(zoo)
library(DBI)
library(tidyr)
library(readr)
#setup parallel backend to use many processors
cores=detectCores()
cl <- makeCluster(cores[1]-1, type = "PSOCK", outfile = "log.txt") #not to overload your computer #,outfile="log.txt"
registerDoParallel(cl)

f_ini <- Sys.time()


## Directorio lectura ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP")


### Lectura de datos ----

#Awarding
load("DF_Awarding_Final.RData")

#Modificaciones
load("DF_Modis_FinalPrice.RData")

#Base de datos completa
load("BD_PLACSP.RData")

#Base de datos plazo
```

```r
load("DF_plazo_u.RData")

#Base de datos PLACSP de requerimientos
load("BD_placsp_quali.RData")

## Directorio guardar ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/_final_conjunto_v4")



### Limpieza y transformacion DF de PLACSP ------

BD_PLACSP <- BD_PLACSP |> #Seteamos el formato de fecha de nuevo
  mutate(
    Primera.publicacion = as.Date(as.numeric(Primera.publicacion), origin = "1899-12-30")
  )


BD_PLACSP_clean <- BD_PLACSP |> # Quitamos error de cargar XML
  mutate(across(
    where(is.character),
    ~ str_replace_all(.x, 'xml:space="preserve">', "")
  ))


BD_PLACSP_clean <- BD_PLACSP_clean |>
  # Crear columnas Identificador
  mutate(
    Identificador_lici = str_c(Identificador, coalesce(as.character(Numero.de.expediente), "NA"), sep = "//"),
    Identificador_lote = str_c(Identificador, coalesce(as.character(Numero.del.contrato.licitacion_lote), "NA"), sep = "//")
  ) |>

  # Transformar y seleccionar columnas con renombramiento
  transmute(
    Identificador,
    Identificador_lici,
    Identificador_lote,
    Primera_publicacion=Primera.publicacion,
    Fecha_actualizacion=Fecha.actualizacion,
    Objeto_del_Contrato=Objeto.del.Contrato,

    Presupuesto_licitacion = parse_number(Presupuesto.base.sin.impuestos),
    CPV = CPV,

    Tipo_de_contrato = Tipo.de.contrato,
    Tipo_de_Administracion = Tipo.de.Administracion,
    Tipo_de_procedimiento = Tipo.de.procedimiento,
    Tramitacion = Tramitacion,
    Lote = Lote,
    Codigo_Postal = Codigo.Postal,
    Tipo_ganador_lote=Tipo.de.identificador.de.adjudicatario.por.licitacion_lote,
    Ganador_NIF = Identificador.Adjudicatario.de.la.licitacion_lote,

    Presupuesto_licitacion_lote = parse_number(Presupuesto.base.sin.impuestos.licitacion_lote),
    CPV_lote = CPV.licitacion_lote,

    N_ofertantes = parse_number(Numero.de.ofertas.recibidas.por.licitacion_lote),
    Precio_adjudicacion = parse_number(Importe.adjudicacion.sin.impuestos.licitacion_lote)
  ) |>
  filter(!is.na(Precio_adjudicacion)) #Quitamos casos con NA en precio final

## Eliminamos casos duplicados
BD_PLACSP_clean<-unique(BD_PLACSP_clean)

## Eliminamos casos duplicados menos en la fecha, que escogemos el más reciente

```

```r
BD_PLACSP_clean <- BD_PLACSP_clean |>
  group_by(across(-Fecha_actualizacion)) |>
  slice_max(order_by = Fecha_actualizacion, n = 1, with_ties = FALSE) |>
  ungroup()

# --- Función auxiliar: divide CPV por ":" y chequea si alguno cumple un patrón ---
tiene_algun_cpv_que_cumple <- function(cpv_string, patron_regex) {
  cpvs <- str_split(cpv_string, ":", simplify = TRUE)
  any(str_detect(cpvs, patron_regex))
}

# --- 1. Casos con al menos un CPV de construcción (empieza por 45) ---
bd_placsp_constru <- BD_PLACSP_clean |>
  filter(
    !is.na(CPV) &
      sapply(CPV, tiene_algun_cpv_que_cumple, patron_regex = "^45")
  )

# --- 2. Casos con al menos un CPV de ingeniería civil (71311) ---
# --- Nuevos CPV a incluir explícitamente ---
cpv_extras_civil <- c(
  "71322000", "71322100", "71322200", "71322300",
  "71322400", "71322500", "71325000"

)

## Para el analisis de buscar pares obra-servicio, en los que las obras ya limitan a Inge Civil
cpv_extras_pares <-c(
  "71322000", "71322100", "71322200", "71322300",
  "71322400", "71322500", "71325000",
  "71000000",#Servicios de ingenieria, arquitectura, construcción en general, para recoger casos que han puesto el general
  "71240000", #Servicios de arquitectura, ingeniería y planificación
  "71241000", #Estudio de viabilidad, servicios de asesoramiento, análisis
  "71242000",#Elaboración de proyectos y diseños, presupuestos
  "71243000",# Anteproyectos (sistemas e integración)
  "71250000", #Servicios de arquitectura, ingeniería y agrimensura
  "71300000", #Servicios de ingenieria
  "71310000", #Servicios de consultoría en ingeniería y construcción
  "71312000", #Servicios de consultoría en ingeniería de estructuras
  "71320000", #Servicios de diseño técnico
  "71335000", #Estudios de ingeniería
  "71336000" #Servicios complementarios de ingeniería
)

# --- Función que detecta coincidencias múltiples Inge Civil ---
tiene_cpv_inge_ampliado <- function(cpv_string) {
  cpvs <- str_split(cpv_string, ":", simplify = TRUE)
  cpvs <- str_trim(cpvs)

  any(str_detect(cpvs, "^71311")) ||      # ingeniería civil clásica
    any(str_detect(cpvs, "^71313")) ||   # ingeniería ambiental
    any(substr(cpvs, 1, 8) %in% cpv_extras_civil)  # códigos exactos extra inge civil
}

# --- Aplicamos el nuevo filtro ---
bd_placsp_ingecivil <- BD_PLACSP_clean |>
  filter(!is.na(CPV) & sapply(CPV, tiene_cpv_inge_ampliado))

# --- Función que detecta coincidencias múltiples Inge pares---
tiene_cpv_inge_pares <- function(cpv_string) {
  cpvs <- str_split(cpv_string, ":", simplify = TRUE)
  cpvs <- str_trim(cpvs)

  any(str_detect(cpvs, "^71311")) ||      # ingeniería civil clásica
    any(str_detect(cpvs, "^71313")) ||   # ingeniería ambiental
    any(substr(cpvs, 1, 8) %in% cpv_extras_pares)  # códigos exactos extra inge civil para pares
}

# --- Aplicamos el nuevo filtro ---
bd_placsp_ingepares <- BD_PLACSP_clean |>
```

```r
  filter(!is.na(CPV) & sapply(CPV, tiene_cpv_inge_pares))




# --- 3. Filtro extra: quitar casos donde todos los CPVs son de arquitectura ---
# Arquitectura: 4521 (excepto 133), 452625, 4526261, ..., 452629
# 1. Lista explícita de CPVs de arquitectura (puede ampliarse según necesidad)
cpvs_arquitectura <- c(
  "45210000", "45262500", "45262610", "45262630", "45262650", "45262660",
  "45262670", "45262680", "45262690", "45262700", "45262800", "45262900"
)

# 2. Función: TRUE si todos los CPVs están en la lista de arquitectura


solo_arquitectura <- function(cpv_string) {
  cpvs <- str_split(cpv_string, ":", simplify = TRUE)
  cpvs <- str_trim(cpvs)  # limpiar espacios
  cpvs <- cpvs[cpvs != ""]  # quitar vacíos
  all(cpvs %in% cpvs_arquitectura)
}

# 3. Aplicamos filtro final: nos quedamos con casos que NO son solo arquitectura
bd_placsp_civil <- bd_placsp_constru |>
  filter(!sapply(CPV, solo_arquitectura))

# 4. Contamos casos de tipo Obra
n_casos_obra_civil <- bd_placsp_civil |>
  distinct(Identificador) |>
  filter("Obras" %in% bd_placsp_civil$Tipo_de_contrato) |>
  nrow()


## Hacemos un unique de bd_placsp_civil
bd_placsp_civil<-unique(bd_placsp_civil)

##### TRABAJAMOS LOS LOTES EN bd_placsp_civil -----
# Casos sin lotes
bd_placsp_civil_u <- bd_placsp_civil |>
  filter(Lote == "Sin lotes")

## Enlos casos con Identificador repetidos, nos quedamos con el de mas reciente fecha de actualizacion
bd_placsp_civil_u <- bd_placsp_civil_u |>
  group_by(Identificador) |>
  slice_max(order_by = Fecha_actualizacion, n = 1, with_ties = FALSE) |>
  ungroup()

# Casos con lotes
bd_placsp_civil_lotes <- bd_placsp_civil |>
  filter(Lote != "Sin lotes")

# 1. Convertimos Lote a numérico solo para uso interno
bd_placsp_civil_lotes <- bd_placsp_civil_lotes |>
  filter(!is.na(as.integer(Lote))) |>
  mutate(Lote = as.integer(Lote)) |>

  # 2. Eliminamos duplicados: nos quedamos con el lote más reciente
  group_by(Identificador, Lote) |>
  slice_max(order_by = Fecha_actualizacion, n = 1, with_ties = FALSE) |>
  ungroup() |>

  # 3. Filtramos los proyectos donde Lote cubre exactamente 1:N sin huecos
  group_by(Identificador) |>
  filter(n() == max(Lote)) |>
  ungroup()

## Para cada Identificador, generemos un solo caso, sumando las caracteristicas de los lotes como te digo a continuación en cada variable:
#- Ganador NIF: Concatenar los casos divididos por ":". Luego, hacer limpieza para que no haya valores repetidos (entendiendo por valor los characteres entre cada ":")
```

```r
#- Presupuesto_licitacion_lote: Sumar los valores de todos
#- Precio_adjudicacion: Sumar los valores de todos
#- N_ofertantes: Escoger el valor maximo
#- CPV_lote: Concatenar los casos divididos por ":". Luego, hacer limpieza para que no haya valores repetidos (entendiendo por valor los characteres entre cada ":").
#- Lote: Se escoge el valor máximo
#- Identificador_lote: Concatenar los casos divididos por ":". Luego, hacer limpieza para que no haya valores repetidos (entendiendo por valor los characteres entre cada ":")
#- El resto de variables, nos quedamos con los valores del caso con Lote = 1

# --- Función auxiliar para limpiar concatenaciones ---
limpiar_concatenado <- function(x) {
  x |>
    str_split(":", simplify = TRUE) |>
    as.vector() |>
    str_trim() |>
    discard(~ .x == "" || is.na(.x)) |>
    unique() |>
    paste(collapse = ":")
}

# --- Consolidación de los casos con lotes ---
bd_placsp_civil_lotes_resumido <- bd_placsp_civil_lotes |>
  group_by(Identificador) |>
  summarise(
    Ganador_NIF = limpiar_concatenado(Ganador_NIF),
    CPV_lote = limpiar_concatenado(CPV_lote),
    Identificador_lote = limpiar_concatenado(Identificador_lote),

    Presupuesto_licitacion_lote = sum(Presupuesto_licitacion_lote, na.rm = TRUE),
    Precio_adjudicacion = sum(Precio_adjudicacion, na.rm = TRUE),

    N_ofertantes = if (all(is.na(N_ofertantes))) NA_integer_ else max(N_ofertantes, na.rm = TRUE),
    Lote = max(as.integer(Lote), na.rm = TRUE)
  ) |>
  # --- Añadir las variables del lote 1 ---
  left_join(
    bd_placsp_civil_lotes |>
      filter(Lote == 1) |>
      dplyr::select(
        Identificador,
        Identificador_lici,
        Primera_publicacion,
        Fecha_actualizacion,
        Objeto_del_Contrato,
        Presupuesto_licitacion,
        CPV,
        Tipo_de_contrato,
        Tipo_de_Administracion,
        Tipo_de_procedimiento,
        Tramitacion,
        Codigo_Postal,
        Tipo_ganador_lote
      ),
    by = "Identificador"
  )

## Juntamos bases de datos tratadas de bd_placsp_civil

# Asegurar que ambos 'N_lotes' son character
bd_placsp_civil_lotes_resumido <- bd_placsp_civil_lotes_resumido |>
  rename(N_lotes = Lote) |>
  mutate(N_lotes = as.character(N_lotes))

bd_placsp_civil_u <- bd_placsp_civil_u |>
  rename(N_lotes = Lote) |>
  mutate(N_lotes = as.character(N_lotes))

# Unir los data frames
bd_placsp_civil_globales <- bind_rows(
  bd_placsp_civil_lotes_resumido,
```

```r
  bd_placsp_civil_u
)

#### TRABAJAMOS E INTRODUCIMOS EL PLAZO CON DF_plazo_u ------

#Para cada Identificador de DF_plazo_u, escojas el que tiene fecha de actualizacion mas tardia
DF_plazo_u <- DF_plazo_u |>
  group_by(Identificador) |>
  slice_max(order_by = Fecha_actualizacion, n = 1, with_ties = FALSE) |>
  ungroup()

#Para cada Identificador de bd_placsp_civil_globales,
#busques el caso con Identificador correspondiente en DF_plazo_u,
#y le añadas los datos correspondientes de las columnas Plazo_m y Prorroga.
#Sino tienen, deja NA. Igual tienes que crear antes las columnas en bd_placsp_civil_globales
#No quiero que hagas un joint del tiron

bd_placsp_civil_globales <- bd_placsp_civil_globales |>
  mutate(
    Plazo_m = NA_real_,
    Prorroga = NA_character_
  )

# Crear índice de matching
idx_plazo <- match(bd_placsp_civil_globales$Identificador, DF_plazo_u$Identificador)

# Asignar solo donde hay match válido
bd_placsp_civil_globales$Plazo_m[!is.na(idx_plazo)] <- DF_plazo_u$Plazo_m[idx_plazo[!is.na(idx_plazo)]]
bd_placsp_civil_globales$Prorroga[!is.na(idx_plazo)] <- DF_plazo_u$Prorroga[idx_plazo[!is.na(idx_plazo)]]

#### TRABAJAMOS E INTRODUCIMOS LOS CRITERIOS DE ADJUDICACION CON DF_Awarding_Final ------

## Cambiamos nombre de variables
DF_Awarding_Final <- DF_Awarding_Final |>
  # Crear columnas Identificador
  mutate(
    Identificador = entryID,
    Identificador_lici = str_c(entryID, coalesce(as.character(ContractID), "NA"), sep = "//")
  )

# 1. Quedarse con el primer caso por Identificador (no hay Fecha_actualizacion)
DF_Awarding_Final <- DF_Awarding_Final |>
  group_by(Identificador) |>
  slice(1) |>
  ungroup()

# 2. Crear columnas vacías en bd_placsp_civil_globales
bd_placsp_civil_globales <- bd_placsp_civil_globales |>
  mutate(
    C_precio_p = NA_real_,
    C_resto_objetivos_p = NA_real_,
    C_juicio_valor_p = NA_real_
  )

# 3. Match por Identificador
idx_award <- match(bd_placsp_civil_globales$Identificador, DF_Awarding_Final$Identificador)

# 4. Asignar valores si hay match
bd_placsp_civil_globales$C_precio_p[!is.na(idx_award)] <-
  DF_Awarding_Final$`%_precio`[idx_award[!is.na(idx_award)]]

bd_placsp_civil_globales$C_resto_objetivos_p[!is.na(idx_award)] <-
  DF_Awarding_Final$`%_resto_objetivos`[idx_award[!is.na(idx_award)]]

bd_placsp_civil_globales$C_juicio_valor_p[!is.na(idx_award)] <-
  DF_Awarding_Final$`%_juicios_valor`[idx_award[!is.na(idx_award)]]
```

```r


##### TRABAJAMOS SOBRE EL DATA FRAME DE MODIFICACIONES ORIGINAL ------
## Directorio carga ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP")
# Cargamos el data frame si no está aún en el entorno
load("DF_Modificaciones.RData")  # Asegurate de tenerlo cargado con este nombre

## Directorio GUARDAR ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/_final_conjunto_v4")

# Cambiamos nombres y generamos el nuevo identificador
DF_mods <- DF_all_u |>
  transmute(
    Identificador = entryID,
    ContractID = ContractID,  # Necesaria para construir Identificador_lici
    Identificador_lici = str_c(entryID, coalesce(as.character(ContractID), "NA"), sep = "//"),
    Fecha_actualizacion = updated,
    Modi_temp = ContractModificationDurationMeasure,
    Modi_temp_Uc = ContractModificationDurationMeasure_Uc,
    Plazo_final = FinalDurationMeasure,
    Plazo_final_Uc = FinalDurationMeasure_Uc,
    Modi_eco_e = ContractModificationLegalMonetaryTotal,
    Precio_final_e = FinalLegalMonetaryTotal
  )

DF_mods <- DF_mods |> #Pasamos todos los plazos a meses
  mutate(
    Modi_temp_m = case_when(
      Modi_temp_Uc == "ANN" ~ floor(Modi_temp * 12),
      Modi_temp_Uc == "MON" ~ floor(Modi_temp),
      Modi_temp_Uc == "DAY" ~ floor(Modi_temp / 30),
      TRUE ~ NA_integer_
    ),

    Plazo_final_m = case_when(
      Plazo_final_Uc == "ANN" ~ floor(Plazo_final * 12),
      Plazo_final_Uc == "MON" ~ floor(Plazo_final),
      Plazo_final_Uc == "DAY" ~ floor(Plazo_final / 30),
      TRUE ~ NA_integer_
    )
  )

DF_mods <- DF_mods |> #Nos quedamos con las variables que queremos
  dplyr::select(
    Identificador,
    Identificador_lici,
    Fecha_actualizacion,
    Modi_temp_m,
    Plazo_final_m,
    Modi_eco_e,
    Precio_final_e
  )

### CRUZAMOS DF_mods con bd_placsp_civil_glboales -------
#Primero vamos a por los casos en que en la variable bd_placsp_civil_globales$N_lotes corresponde a "Sin lotes".
#En estos casos, quiero que para el Identificador de bd_placsp_civil_globales busques los casos con ese
#Identificador en DF_mods y escojas el caso con mayor valor en Precio_final_e.

# 1. Filtrar casos sin lotes de la base principal
bd_globales_u <- bd_placsp_civil_globales |>
  filter(N_lotes == "Sin lotes") |>
  mutate(
    Modi_temp_m = NA_integer_,
    Plazo_final_m = NA_integer_,
```

```r
    Modi_eco_e = NA_real_,
    Precio_final_e = NA_real_
  )

# 2. Obtener de DF_mods el caso con mayor Precio_final_e por Identificador
mods_u_filtrado <- DF_mods |>
  group_by(Identificador) |>
  slice_max(order_by = Precio_final_e, n = 1, with_ties = FALSE) |>
  ungroup()

# 3. Hacer match y asignar de forma vectorizada
idx_mods <- match(bd_globales_u$Identificador, mods_u_filtrado$Identificador)

bd_globales_u$Modi_temp_m[!is.na(idx_mods)] <- mods_u_filtrado$Modi_temp_m[idx_mods[!is.na(idx_mods)]]
bd_globales_u$Plazo_final_m[!is.na(idx_mods)] <- mods_u_filtrado$Plazo_final_m[idx_mods[!is.na(idx_mods)]]
bd_globales_u$Modi_eco_e[!is.na(idx_mods)] <- mods_u_filtrado$Modi_eco_e[idx_mods[!is.na(idx_mods)]]
bd_globales_u$Precio_final_e[!is.na(idx_mods)] <- mods_u_filtrado$Precio_final_e[idx_mods[!is.na(idx_mods)]]

# Verificamos la condición de igualdad con tolerancia de Modificaciones en precio
coincide <- abs(bd_globales_u$Precio_final_e - bd_globales_u$Precio_adjudicacion - bd_globales_u$Modi_eco_e) < 1e-6

# Filtrar índices válidos (no NA)
idx_corregir <- which(!coincide & !is.na(coincide))

# Mostrar porcentaje de coincidencia
porcentaje_ok <- mean(coincide, na.rm = TRUE) * 100
cat(sprintf("Coincidencia exacta en %.2f%% de los casos con 'Sin lotes'\n", porcentaje_ok))

# Corregir únicamente los que no coinciden y tienen datos completos
bd_globales_u$Modi_eco_e[idx_corregir] <- bd_globales_u$Precio_final_e[idx_corregir] -
 bd_globales_u$Precio_adjudicacion[idx_corregir]



# Verificar igualdad con tolerancia para plazos
coincide_plazo <- (bd_globales_u$Plazo_final_m - bd_globales_u$Plazo_m) == bd_globales_u$Modi_temp_m

# Filtrar índices válidos sin NA
idx_corregir_plazo <- which(!coincide_plazo & !is.na(coincide_plazo))

# Porcentaje de coincidencia
porcentaje_ok_plazo <- mean(coincide_plazo, na.rm = TRUE) * 100
cat(sprintf("Coincidencia exacta en %.2f%% de los casos con 'Sin lotes' (plazos)\n", porcentaje_ok_plazo))

# Corregir solo donde no coincide y no hay NA
bd_globales_u$Modi_temp_m[idx_corregir_plazo] <- bd_globales_u$Plazo_final_m[idx_corregir_plazo] - bd_globales_u$Plazo_m[idx_corregir_plazo]


## Ahora vamos con los casos que N_lotes != "Sin Lotes", para los que la operativa es diferente:
# Para cada Identificador de casos de bd_placsp_civil_globales con lotes, buscamos los casos con mismo Identificador en DF_mods, y hacemos las siguientes transformaciones:
#- La variable "Modi_eco_e" del caso en bd_placsp_civil_globales, será la suma de los valores Modi_eco_e de los casos en DF_Modis
#- La variable Modi_temp_m del caso en bd_placsp_civil_globales, será el máximo de los valores Modi_temp_m de los casos en DF_Modis
#- Las variables Plazo_final_m y Precio_final_e, se calcularán posteriormente en bd_placsp_civil_globales como la suma de Modi_temp_m y Modi_eco_e con Plazo_m y Precio_adjudicacion respectivamente

# Paso 1: Filtrar los casos con lotes desde la base principal
bd_globales_lotes <- bd_placsp_civil_globales |>
  filter(N_lotes != "Sin lotes") |>
```

```r
    mutate(
      Modi_temp_m = NA_integer_,
      Modi_eco_e = NA_real_,
      Plazo_final_m = NA_integer_,
      Precio_final_e = NA_real_
    )

  # Paso 2: Eliminar duplicados por Identificador + importe, conservar el más reciente
  DF_mods_filtrado <- DF_mods |>
    group_by(Identificador, Modi_eco_e, Precio_final_e) |>
    slice_max(order_by = Fecha_actualizacion, n = 1, with_ties = FALSE) |>
    ungroup()

  # Paso 3: Agrupar por Identificador y calcular agregados únicos
  mods_lotes_agg <- DF_mods_filtrado |>
    group_by(Identificador) |>
    summarise(
      Plazo_final_m = if (all(is.na(Plazo_final_m))) NA_integer_ else max(Plazo_final_m, na.rm = TRUE),
      Modi_eco_e = sum(Modi_eco_e, na.rm = TRUE),
      .groups = "drop"
    )

  # Paso 4: Match por Identificador
  idx_lotes <- match(bd_globales_lotes$Identificador, mods_lotes_agg$Identificador)

  # Paso 5: Asignar valores económicos y de plazo final
  bd_globales_lotes$Plazo_final_m[!is.na(idx_lotes)] <- mods_lotes_agg$Plazo_final_m[idx_lotes[!is.na(idx_lotes)]]
  bd_globales_lotes$Modi_eco_e[!is.na(idx_lotes)] <- mods_lotes_agg$Modi_eco_e[idx_lotes[!is.na(idx_lotes)]]

  # Paso 6: Calcular plazo modificado (como diferencia entre final y original)
  bd_globales_lotes <- bd_globales_lotes |>
    mutate(
      Modi_temp_m = ifelse(
        is.na(Plazo_final_m) | is.na(Plazo_m),
        NA_integer_,
        Plazo_final_m - Plazo_m
      ),

      # Precio final = adjudicación + modificación económica
      Precio_final_e = ifelse(
        is.na(Precio_adjudicacion) & is.na(Modi_eco_e),
        NA_real_,
        coalesce(Precio_adjudicacion, 0) + coalesce(Modi_eco_e, 0)
      )
    )

  # --- Verificar y corregir el componente económico ---
  coincide_econ <- abs(bd_globales_lotes$Precio_final_e - bd_globales_lotes$Precio_adjudicacion - bd_globales_lotes$Modi_eco_e) < 1e-6
  idx_corregir_econ <- which(!coincide_econ & !is.na(coincide_econ))

  porcentaje_ok_econ <- mean(coincide_econ, na.rm = TRUE) * 100
  cat(sprintf("Coincidencia exacta en %.2f%% de los casos con lotes (económico)\n", porcentaje_ok_econ))

  bd_globales_lotes$Modi_eco_e[idx_corregir_econ] <- bd_globales_lotes$Precio_final_e[idx_corregir_econ] - bd_globales_lotes$Precio_adjudicacion[idx_corregir_econ]

  # --- Verificar y corregir el componente de plazos ---
  coincide_plazo <- (bd_globales_lotes$Plazo_final_m - bd_globales_lotes$Plazo_m) == bd_globales_lotes$Modi_temp_m
  idx_corregir_plazo <- which(!coincide_plazo & !is.na(coincide_plazo))

  porcentaje_ok_plazo <- mean(coincide_plazo, na.rm = TRUE) * 100
  cat(sprintf("Coincidencia exacta en %.2f%% de los casos con lotes (plazo)\n", porcentaje_ok_plazo))

  bd_globales_lotes$Modi_temp_m[idx_corregir_plazo] <- bd_globales_lotes$Plazo_final_m[idx_corregir_plazo] - bd_globales_lotes$Plazo_m[idx_corregir_plazo]

```

```r
# --- Unión final: todos los casos con y sin lotes ---
bd_civil_todo <- bind_rows(
  bd_globales_u,
  bd_globales_lotes
)

# --- Calculo de nuevas variables ---
bd_civil_todo <- bd_civil_todo |>
  mutate(
    # 1. Baja porcentual (% de descuento sobre presupuesto)
    Baja_p = ifelse(
      !is.na(Presupuesto_licitacion) & Presupuesto_licitacion != 0,
      round(((Presupuesto_licitacion - Precio_adjudicacion) / Presupuesto_licitacion)
      * 100,2),
      NA_real_
    ),

    # 2. Modificación económica porcentual
    Modi_eco_p = ifelse(
      !is.na(Precio_adjudicacion) & Precio_adjudicacion != 0,
      round(((Precio_final_e - Precio_adjudicacion) / Precio_adjudicacion) * 100, 2),
      NA_real_
    ),

    # 3. Modificación temporal porcentual
    Modi_temp_p = ifelse(
      !is.na(Plazo_final_m) & Plazo_final_m != 0,
      round(((Plazo_final_m - Plazo_m) / Plazo_final_m) * 100, 2),
      NA_real_
    ),

    # 4. Intervalo entre publicación y última actualización (en meses, redondeado)
    Intervalo_modi = ifelse(
      !is.na(Fecha_actualizacion) & !is.na(Primera_publicacion),
      round(as.numeric(difftime(Fecha_actualizacion, Primera_publicacion, units =
      "days")) / 30),
      NA_real_
    )
  )
# Si se han registrado modificaciones de ese proyecto

bd_civil_todo <- bd_civil_todo |>
  mutate(
    Modis_s_n = factor(
      if_else(
        Identificador %in% DF_mods$Identificador,
        "Sí", "No"
      ),
      levels = c("Sí", "No")
    )
  )

# --- Vamos a filtrar los casos de bd_civil_todo con precios de adjudicacion menores
a 10.000 --
bd_civil_todo <- bd_civil_todo |>
  filter(Precio_adjudicacion >= 10000 | is.na(Precio_adjudicacion))

# --- Subconjunto: solo los casos con modificaciones económicas registradas ---
bd_civil_modis <- bd_civil_todo |>
  filter(!is.na(Modi_eco_e))

# --- Subconjunto: solo los casos con modificaciones económicas registradas y sin NA
en criterio economico ---
bd_civil_sin_NA <- bd_civil_modis |>
  filter(!is.na(C_precio_p))


#### VAMOS A METER LOS DATOS SOBRE REQUISITOS DE LAS LICITACIONES ##### ------

### VAMOS A METER LOS DATOS SOBRE REQUISITOS DE LAS LICITACIONES ##### --


# Crear columnas vacías en bd_civil_todo
```

```r
bd_civil_todo <- bd_civil_todo |>
  mutate(
    Clasi_empresa = NA_character_,
    Requi_adm = NA_character_,
    Clasi_max_req = NA_real_
  )

# Realizar el matching vectorizado por Identificador
idx_quali <- match(bd_civil_todo$Identificador, BD_placsp_quali$Identificador)

# Asignar valores desde BD_placsp_quali
bd_civil_todo$Clasi_empresa[!is.na(idx_quali)] <- BD_placsp_quali$Clasi_empresa[
idx_quali[!is.na(idx_quali)]]
bd_civil_todo$Requi_adm[!is.na(idx_quali)] <- BD_placsp_quali$Requi_adm[idx_quali[!
is.na(idx_quali)]]
bd_civil_todo$Clasi_max_req[!is.na(idx_quali)] <- BD_placsp_quali$Clasi_max_req[
idx_quali[!is.na(idx_quali)]]

## Actualizamos subconjuntos
# --- Subconjunto: solo los casos con modificaciones económicas registradas ---
bd_civil_modis <- bd_civil_todo |>
  filter(!is.na(Modi_eco_e))

# --- Subconjunto: solo los casos con modificaciones económicas registradas y sin NA
en criterio economico ---
bd_civil_sin_NA <- bd_civil_modis |>
  filter(!is.na(C_precio_p))

### VAMOS A METER LOS DATOS SOBRE FECHAS DEL PROYECTO##### ----

# Paso 0: Crear vector de identificadores únicos a usar
ids_utiles <- unique(c(bd_civil_modis$Identificador, bd_civil_sin_NA$Identificador))

# Paso 1: Filtrar BD_PLACSP por identificadores relevantes y columnas necesarias
BD_PLACSP_filtrado <- BD_PLACSP |>
  filter(Identificador %in% ids_utiles) |>
  dplyr::select(
    Identificador,
    Fecha.actualizacion,
    Fecha.entrada.en.vigor.del.contrato.de.licitacion_lote,
    Fecha.formalizacion.del.contrato.licitacion_lote,
    Fecha.de.presentacion.de.ofertas
  )

# Paso 2: Calcular fechas brutas
BD_PLACSP_fechas <- BD_PLACSP_filtrado |>
  mutate(
    Fecha_entrada_vigor_raw = as.Date(
      coalesce(
        Fecha.entrada.en.vigor.del.contrato.de.licitacion_lote,
        Fecha.formalizacion.del.contrato.licitacion_lote
      ),
      origin = "1899-12-30"
    ),
    Fecha_presentacion_raw = as.Date(
      Fecha.de.presentacion.de.ofertas,
      origin = "1899-12-30"
    )
  )

# Paso 3: Mantener solo la fila más reciente por Identificador
BD_PLACSP_max_fechas <- BD_PLACSP_fechas |>
  group_by(Identificador) |>
  slice_max(order_by = Fecha.actualizacion, n = 1, with_ties = FALSE) |>
  ungroup()

# Paso 4: Formatear las fechas como "YYYY/MM/DD"
BD_PLACSP_max_fechas <- BD_PLACSP_max_fechas |>
  mutate(
    Fecha_entrada_vigor = format(Fecha_entrada_vigor_raw, "%Y/%m/%d"),
    Fecha_presentacion = format(Fecha_presentacion_raw, "%Y/%m/%d")
  ) |>
  dplyr::select(Identificador, Fecha_entrada_vigor, Fecha_presentacion)
```

```r

# Paso 5: Añadir columnas vacías en los data frames destino
bd_civil_modis <- bd_civil_modis |>
  mutate(
    Fecha_entrada_vigor = NA_character_,
    Fecha_presentacion = NA_character_
  )

bd_civil_sin_NA <- bd_civil_sin_NA |>
  mutate(
    Fecha_entrada_vigor = NA_character_,
    Fecha_presentacion = NA_character_
  )

# Paso 6: Match por Identificador y asignar valores si hay
idx_modis <- match(bd_civil_modis$Identificador, BD_PLACSP_max_fechas$Identificador)
idx_sin_na <- match(bd_civil_sin_NA$Identificador, BD_PLACSP_max_fechas$Identificador)

bd_civil_modis$Fecha_entrada_vigor[!is.na(idx_modis)] <-
  BD_PLACSP_max_fechas$Fecha_entrada_vigor[idx_modis[!is.na(idx_modis)]]

bd_civil_modis$Fecha_presentacion[!is.na(idx_modis)] <-
  BD_PLACSP_max_fechas$Fecha_presentacion[idx_modis[!is.na(idx_modis)]]

bd_civil_sin_NA$Fecha_entrada_vigor[!is.na(idx_sin_na)] <-
  BD_PLACSP_max_fechas$Fecha_entrada_vigor[idx_sin_na[!is.na(idx_sin_na)]]

bd_civil_sin_NA$Fecha_presentacion[!is.na(idx_sin_na)] <-
  BD_PLACSP_max_fechas$Fecha_presentacion[idx_sin_na[!is.na(idx_sin_na)]]

# --- Rellenar NA con valores auxiliares (nuevas reglas) ---

bd_civil_modis <- bd_civil_modis |>
  mutate(
    Fecha_presentacion = if_else(
      is.na(Fecha_presentacion) & !is.na(Primera_publicacion),
      as.character(Primera_publicacion),
      Fecha_presentacion
    ),
    Fecha_entrada_vigor = if_else(
      is.na(Fecha_entrada_vigor) & !is.na(Fecha_presentacion),
      Fecha_presentacion,
      Fecha_entrada_vigor
    )
  )


# En bd_civil_sin_NA
bd_civil_sin_NA <- bd_civil_sin_NA |>
  mutate(
    Fecha_presentacion = if_else(
      is.na(Fecha_presentacion) & !is.na(Primera_publicacion),
      as.character(Primera_publicacion),
      Fecha_presentacion
    ),
    Fecha_entrada_vigor = if_else(
      is.na(Fecha_entrada_vigor) & !is.na(Fecha_presentacion),
      Fecha_presentacion,
      Fecha_entrada_vigor
    )
  ) |>
  # Eliminar casos con ambas fechas aún en NA
  filter(!(is.na(Fecha_entrada_vigor) & is.na(Fecha_presentacion)))


# --- Conversión de fechas a clase Date (YYYY-MM-DD) ---
# Estándar: convertir todo a "YYYY-MM-DD" (ISO 8601)
bd_civil_modis <- bd_civil_modis |>
  mutate(
    Fecha_presentacion = as.character(Fecha_presentacion),
    Fecha_entrada_vigor = as.character(Fecha_entrada_vigor),
    Primera_publicacion = as.character(Primera_publicacion)
  ) |>
```

```r
  mutate(
    Fecha_presentacion = str_replace_all(Fecha_presentacion, "/", "-"),
    Fecha_entrada_vigor = str_replace_all(Fecha_entrada_vigor, "/", "-"),
    Primera_publicacion = str_replace_all(Primera_publicacion, "/", "-")
  )
bd_civil_sin_NA <- bd_civil_sin_NA |>
  mutate(
    Fecha_presentacion = as.character(Fecha_presentacion),
    Fecha_entrada_vigor = as.character(Fecha_entrada_vigor),
    Primera_publicacion = as.character(Primera_publicacion)
  ) |>
  mutate(
    Fecha_presentacion = str_replace_all(Fecha_presentacion, "/", "-"),
    Fecha_entrada_vigor = str_replace_all(Fecha_entrada_vigor, "/", "-"),
    Primera_publicacion = str_replace_all(Primera_publicacion, "/", "-")
  )

convertir_fechas <- function(df) {
  df |>
    mutate(
      Fecha_entrada_vigor = as.Date(Fecha_entrada_vigor),
      Fecha_presentacion = as.Date(Fecha_presentacion),
      Primera_publicacion = as.Date(Primera_publicacion),
      Fecha_actualizacion = as.Date(Fecha_actualizacion)
    )
}


bd_civil_modis <- convertir_fechas(bd_civil_modis)
bd_civil_sin_NA <- convertir_fechas(bd_civil_sin_NA)

## Ultima comprobacion, forzamos a Primer_publicacion a ser la mas temprana

# Aseguramos que Primera_publicacion sea la más temprana de las tres fechas relevantes
bd_civil_modis <- bd_civil_modis |>
  mutate(
    Primera_publicacion = pmin(
      Primera_publicacion,
      Fecha_presentacion,
      Fecha_entrada_vigor,
      na.rm = TRUE
    )
  )

bd_civil_sin_NA <- bd_civil_sin_NA |>
  mutate(
    Primera_publicacion = pmin(
      Primera_publicacion,
      Fecha_presentacion,
      Fecha_entrada_vigor,
      na.rm = TRUE
    )
  )



# --- Outliers manuales (reasignar entrada_vigor = presentacion si en lista) ---
ids_outliers_f <- c("1676574", "4051825", "3418862", "2819257", "9856662", "4614642",
"6715534", "5446016")

bd_civil_modis <- bd_civil_modis |>
  mutate(
    Fecha_entrada_vigor = if_else(
      Identificador %in% ids_outliers_f,
      Fecha_presentacion,
      Fecha_entrada_vigor
    )
  )

bd_civil_sin_NA <- bd_civil_sin_NA |>
  mutate(
    Fecha_entrada_vigor = if_else(
      Identificador %in% ids_outliers_f,
```

```r
        Fecha_presentacion,
        Fecha_entrada_vigor
      )
    )

  # --- Cálculo de intervalos temporales ---
  bd_civil_modis <- bd_civil_modis |>
    mutate(
      Intervalo_lici_d = as.integer(Fecha_presentacion - Primera_publicacion),
      Intervalo_vigor_d = as.integer(Fecha_entrada_vigor - Fecha_presentacion),
      Intervalo_urge_d = as.integer(Fecha_entrada_vigor - Primera_publicacion),
      Intervalo_modi_m = round(as.numeric(difftime(Fecha_actualizacion,
      Fecha_entrada_vigor, units = "days")) / 30)
    )

  bd_civil_sin_NA <- bd_civil_sin_NA |>
    mutate(
      Intervalo_lici_d = as.integer(Fecha_presentacion - Primera_publicacion),
      Intervalo_vigor_d = as.integer(Fecha_entrada_vigor - Fecha_presentacion),
      Intervalo_urge_d = as.integer(Fecha_entrada_vigor - Primera_publicacion),
      Intervalo_modi_m = round(as.numeric(difftime(Fecha_actualizacion,
      Fecha_entrada_vigor, units = "days")) / 30)
    )



  #### GENERAMOS DATA FRAME DE RELACION OBRA-REDACCION ##### ------

  #### ANÁLISIS SEMÁNTICO DE PARES OBRA-REDACCIÓN (QUANTEDA) #### ------
  library(quanteda)
  library(readxl)
  library(dplyr)
  library(stringr)
  library(stringi)

  # 1. Leer archivo de coincidencias TF-IDF
  coincidencias_tfidf <- read_excel("C:/Users/guillermo.alonso/Desktop/Tesis
  Guillermo/_BD actualizada/Datos atom/PLACSP/coincidencias_modis_tfidf_ampliado.xlsx")

  # 2. Filtrar coincidencias con score alto (> 0.75)
  coincidencias_filtradas <- coincidencias_tfidf |>
    filter(precision_score > 0.75)

  # 3. Seleccionar mejor par por Identificador_modis
  coincidencias_filtradas <- coincidencias_filtradas |>
    group_by(Identificador_modis) |>
    slice_max(order_by = precision_score, n = 1, with_ties = FALSE) |>
    ungroup()

  # 4. Seleccionar mejor par por Identificador_redac (resultado: pares únicos)
  coincidencias_filtradas <- coincidencias_filtradas |>
    group_by(Identificador_redac) |>
    slice_max(order_by = precision_score, n = 1, with_ties = FALSE) |>
    ungroup()

  # 5. Guardamos el resultado inicial
  bd_inge_obra_ampliado <- coincidencias_filtradas

  # --- NORMALIZACIÓN DE TEXTO Y TOKENIZACIÓN ROBUSTA ---
  # Limpieza: eliminar tildes y pasar a minúsculas
  bd_inge_obra_ampliado <- bd_inge_obra_ampliado |>
    mutate(Objeto_redac_clean = stri_trans_general(Objeto_redac, "Latin-ASCII"))

  # Crear tokens limpios con stemming
  tokens_redac <- tokens(
    bd_inge_obra_ampliado$Objeto_redac_clean,
    remove_punct = TRUE,
    remove_symbols = TRUE,
    remove_numbers = TRUE
  ) |>
    tokens_tolower() |>
    tokens_wordstem(language = "spanish")
```

```r
# --- 1. Expresiones clave nuevas (frases comunes en redacción de proyecto) ---
multi_expresiones <- list(
  c("direccion", "facultativa"),
  c("direccion", "obra"),
  c("proyecto", "constructivo"),
  c("proyecto", "ejecutivo"),
  c("proyecto", "detalle"),
  c("plan", "seguridad"),
  c("coordinacion", "seguridad"),
  c("estudio", "soluciones"),
  c("ingenieria", "detalle"),
  c("elaboracion", "proyecto"),
  c("asistencia", "tecnica"),
  c("servicio", "elaboracion"),
  c("elaboracion", "integra")
)

tokens_redac <- tokens_compound(tokens_redac, phrase(multi_expresiones))

# --- FILTRADO DE CASOS CON CONTENIDO DE REDACCIÓN ---
# Definir términos básicos de redacción
terminos_redaccion <- c(
  "redaccion", "diseno", "elaboracion", "anteproyecto", "proyecto",
  "tecnico", "estudio", "ingenieria", "planificacion",
  "memoria", "constructiva", "basica", "ejecutiva"
)

# Crear matriz documento-característica
dfm_redac <- dfm(tokens_redac)

# Detectar si hay contenido clave de redacción
dfm_filtro_redac <- dfm_select(dfm_redac, pattern = terminos_redaccion)
filtro_redaccion <- rowSums(dfm_filtro_redac) > 0

# Separar casos
inge_obra_redaccion <- bd_inge_obra_ampliado[filtro_redaccion, ]
inge_obra_no_redaccion <- bd_inge_obra_ampliado[!filtro_redaccion, ]

# --- ANÁLISIS SEMÁNTICO: CATEGORIZACIÓN DE TEMAS ---
# --- . Diccionario mejorado con términos expandidos y compuestos
diccionario_temas <- dictionary(list(
  anteproyecto = c("anteproyecto", "preliminar", "viabilidad", "alternativas",
  "solucion"),

  redaccion_proyecto = c(
    "redaccion", "memoria", "documentacion", "constructivo", "detalle", "ejecutivo",
    "ingenieria", "proyecto", "elaboracion", "asistencia tecnica", "elaboracion
    proyecto",
    "elaboracion integra", "servicio elaboracion", "proyecto tecnico"
  ),

  direccion_facultativa = c("direccion facultativa", "direccion obra", "supervision",
  "control", "coordinacion"),

  seguridad_salud = c("seguridad", "salud", "riesgo", "plan seguridad", "prevencion"),

  construccion = c("ejecucion", "realizacion", "construccion", "obra civil")
))
# Aplicar diccionario temático
dfm_temas <- dfm_lookup(dfm_redac[filtro_redaccion, ], dictionary = diccionario_temas)

# Convertir a binario: 1 si contiene el tema, 0 si no
temas_binarias <- convert(dfm_temas, to = "data.frame") |>
  dplyr::select(-doc_id) |>
  mutate(across(everything(), ~ ifelse(. > 0, 1, 0)))

# Añadir columnas temáticas al data frame
inge_obra_redaccion <- bind_cols(inge_obra_redaccion, temas_binarias)

inge_obra_redaccion <- inge_obra_redaccion |>
  dplyr::select(-Objeto_redac_clean)

## JUNTAMOS CON LOS DATOS QUE TENEMOS -----
```

```r

# --- 1. Seleccionar columnas clave de temas y scores ---
base_pares <- inge_obra_redaccion |>
  dplyr::select(
    precision_score,
    anteproyecto,
    redaccion_proyecto,
    direccion_facultativa,
    seguridad_salud,
    construccion,
    Identificador_redac,
    Identificador_modis
  ) |>
  mutate(
    Identificador_redac = as.character(Identificador_redac),
    Identificador_modis = as.character(Identificador_modis)
  )

# --- 2. Traer info de Identificador_redac desde bd_placsp_ingepares ---
info_redac <- bd_placsp_ingepares |>
  filter(Identificador %in% base_pares$Identificador_redac) |>
  group_by(Identificador) |>
  slice_max(order_by = Fecha_actualizacion, n = 1, with_ties = FALSE) |>
  ungroup()

info_redac <- info_redac |>
  rename_with(~ paste0("redac_", .x), .cols = -Identificador)

# --- 3. Traer info de Identificador_modis desde bd_civil_modis ---
info_modis <- bd_civil_modis |>
  filter(Identificador %in% base_pares$Identificador_modis) |>
  group_by(Identificador) |>
  slice_max(order_by = Fecha_actualizacion, n = 1, with_ties = FALSE) |>
  ungroup()

info_modis <- info_modis |>
  rename_with(~ paste0("modis_", .x), .cols = -Identificador)

# --- 4. Join completo con claves convertidas ---
bd_inge_obra_pares <- base_pares |>
  left_join(info_redac, by = c("Identificador_redac" = "Identificador")) |>
  left_join(info_modis, by = c("Identificador_modis" = "Identificador"))



#### DETECCION DE OUTLIERS #####

## Por percentiles extremos
vars_percentiles <- c("Presupuesto_licitacion", "Modi_temp_m", "Modi_eco_e",
                      "Intervalo_vigor_d", "Baja_p", "Precio_final_e")

for (var in vars_percentiles) {
  q_low <- quantile(bd_civil_sin_NA[[var]], 0.005, na.rm = TRUE)
  q_high <- quantile(bd_civil_sin_NA[[var]], 0.995, na.rm = TRUE)

  bd_civil_sin_NA <- bd_civil_sin_NA |>
    filter(.data[[var]] >= q_low & .data[[var]] <= q_high | is.na(.data[[var]]))
}

## Casos Manuales
bd_civil_sin_NA <- bd_civil_sin_NA |>
  filter(
    Presupuesto_licitacion != 0 | is.na(Presupuesto_licitacion),
    Modi_temp_m >= -12 | is.na(Modi_temp_m),
    Modi_eco_p >= -80 | is.na(Modi_eco_p),
    Intervalo_vigor_d >= 0 | is.na(Intervalo_vigor_d),
    Intervalo_modi_m >= 0 | is.na(Intervalo_modi_m)
  )

for (var in vars_percentiles) {
  q_low <- quantile(bd_civil_modis[[var]], 0.005, na.rm = TRUE)
  q_high <- quantile(bd_civil_modis[[var]], 0.995, na.rm = TRUE)

```

```r
  bd_civil_modis <- bd_civil_modis |>
    filter(.data[[var]] >= q_low & .data[[var]] <= q_high | is.na(.data[[var]]))
}

## Casos Manuales
bd_civil_modis <- bd_civil_modis |>
  filter(
    Presupuesto_licitacion != 0 | is.na(Presupuesto_licitacion),
    Modi_temp_m >= -12 | is.na(Modi_temp_m),
    Modi_eco_p >= -80 | is.na(Modi_eco_p),
    Intervalo_vigor_d >= 0 | is.na(Intervalo_vigor_d),
    Intervalo_modi_m >= 0 | is.na(Intervalo_modi_m)
  )

## Eliminación manual de casos concretos:

# --- Eliminación por Identificador manual ---
ids_excluir <- c("4051825", "1676574", "3418862", "2819257", "6135224",
                 "4111947", "11544946", "6508881", "10373168", "10614482")

bd_civil_sin_NA <- bd_civil_sin_NA[!bd_civil_sin_NA$Identificador %in% ids_excluir, ]

# --- Forzar a NA los casos con Modi_temp_m > 60 ---
bd_civil_sin_NA <- bd_civil_sin_NA |>
  mutate(
    Modi_temp_m = ifelse(Modi_temp_m > 60, NA_integer_, Modi_temp_m),
    Plazo_final_m = ifelse(is.na(Modi_temp_m), NA_integer_, Plazo_final_m),
    Modi_temp_p = ifelse(is.na(Modi_temp_m), NA_real_, Modi_temp_p)
  )

bd_civil_modis <- bd_civil_modis[!bd_civil_modis$Identificador %in% ids_excluir, ]

# --- Forzar a NA los casos con Modi_temp_m > 60 ---
bd_civil_modis <- bd_civil_modis |>
  mutate(
    Modi_temp_m = ifelse(Modi_temp_m > 60, NA_integer_, Modi_temp_m),
    Plazo_final_m = ifelse(is.na(Modi_temp_m), NA_integer_, Plazo_final_m),
    Modi_temp_p = ifelse(is.na(Modi_temp_m), NA_real_, Modi_temp_p)
  )


#### GUARDAMOS RESULTADOS ------


## Directorio GUARDAR ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/_final_conjunto_v4")

# Exportar bd_civil_todo
save(bd_civil_todo, file = "bd_civil_todo_v2.RData")
write.xlsx(bd_civil_todo, "bd_civil_todo_v2.xlsx", colNames = TRUE, rowNames = FALSE)

# Exportar bd_civil_modis
save(bd_civil_modis, file = "bd_civil_modis_v2.RData")
write.xlsx(bd_civil_modis, "bd_civil_modis_v2.xlsx", colNames = TRUE, rowNames = FALSE)

# Exportar bd_placsp_ingecivil
save(bd_placsp_ingecivil, file = "bd_placsp_ingecivil_v2.RData")
write.xlsx(bd_placsp_ingecivil, "bd_placsp_ingecivil_v2.xlsx", colNames = TRUE, rowNames = FALSE)

# Exportar bd_placsp_ingepares
save(bd_placsp_ingepares, file = "bd_placsp_ingepares_v2.RData")
write.xlsx(bd_placsp_ingepares, "bd_placsp_ingepares_v2.xlsx", colNames = TRUE, rowNames = FALSE)

# Exportar bd_civil_sin_NA
save(bd_civil_sin_NA, file = "bd_civil_sin_NA_v2.RData")
write.xlsx(bd_civil_sin_NA, "bd_civil_sin_NA_v2.xlsx", colNames = TRUE, rowNames = FALSE)

```

```r
# Guardar en Excel
save(bd_inge_obra_pares, file = "bd_inge_obra_pares_v2.RData")
write.xlsx(bd_inge_obra_pares, "bd_inge_obra_pares_v2.xlsx", colNames = TRUE, rowNames = FALSE)



# Exportar todo el entorno global
save.image(file = "_Final_v2_entorno_global_completo_v2.RData")

# Limpiamos objetos

rm(list=ls())


### SCRIPT DE SOM_v1 ---------------
# En los scripts de regresion y PCA v1 se han hecho los mismos o menos tratamientos de datos que aqui
# Se empieza de cero cargando el data frame base
# Se introduce al guardar las variables Identificador y Identificado_lote
### --------------------------------

## Directorio LEER v1 ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/_final_conjunto_v4")

## Cargamos datos ----------

load("bd_civil_sin_NA_v2.RData")

## Directorio GUARDAR ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/_final_conjunto_v4")


## PREPROCESAMIENTO de VARIABLES ###### -------

# Aseguramos que estamos trabajando sobre la base correcta
df <- bd_civil_sin_NA

# Convertir factores de interés
df <- df |>
  mutate(
    Tipo_de_Administracion = as.factor(Tipo_de_Administracion),
    Tipo_de_procedimiento = as.factor(Tipo_de_procedimiento),
    Tipo_de_contrato = as.factor(Tipo_de_contrato),
    Tramitacion = as.factor(Tramitacion),
    Tipo_ganador_lote = as.factor(Tipo_ganador_lote),
    Modis_s_n = as.factor(Modis_s_n)
  )


# --- Tratamiento de la variable CPV en df ---
df <- df |>
  mutate(
    CPV_truncs = purrr::map(CPV, function(x) {
      if (is.na(x) || stringr::str_trim(x) == "") return(character(0))

      x_split <- stringr::str_split(x, ":", simplify = FALSE)[[1]]
      x_trimmed <- stringr::str_trim(x_split)
      x_valid <- x_trimmed[x_trimmed != ""]
      stringr::str_sub(x_valid, 1, 5)
    }),

    N_CPV = purrr::map_int(CPV_truncs, length),

    CPV_main = purrr::map_chr(CPV_truncs, function(cpvs) {
      if (length(cpvs) == 0) return(NA_character_)

      cpvs_45 <- cpvs[stringr::str_starts(cpvs, "45")]
      if (length(cpvs_45) == 0) return(NA_character_)
```

```
1511
1512        ceros_derecha <- stringr::str_length(cpvs_45) -
1513          stringr::str_length(stringr::str_replace(cpvs_45, "0*$", ""))
1514
1515        min_ceros <- min(ceros_derecha)
1516        seleccionados <- cpvs_45[ceros_derecha == min_ceros]
1517
1518        return(seleccionados[1])
1519      })
1520    )
1521
1522
1523  # --- Tratamiento de la variable Clasi_empresa sobre el data frame df ---
1524
1525  # 1. Calcular N_clasi_empresa: número de clasificaciones registradas (separadas por
      ":")
1526  df <- df |>
1527    mutate(
1528      N_clasi_empresa = if_else(
1529        is.na(Clasi_empresa) | str_trim(Clasi_empresa) == "",
1530        0L,
1531        str_count(Clasi_empresa, ":") + 1L
1532      )
1533    )
1534
1535  # 2. Función para extraer la letra con el número más alto (o la primera si hay empate)
1536  extraer_letra_dominante <- function(clasi_string) {
1537    if (is.na(clasi_string) || str_trim(clasi_string) == "") {
1538      return("0")  # Regla: si no hay clasificaciones, devolver "0"
1539    }
1540
1541    # Separar clasificaciones por ":"
1542    clasificaciones <- unlist(str_split(clasi_string, ":", simplify = FALSE))
1543
1544    # Extraer letra y número si está, o solo letra si número no es válido
1545    letras_valores <- str_match(clasificaciones, "^([A-Z])[^0-9]*([0-9]*)")
1546
1547    # Filtrar filas válidas
1548    letras <- letras_valores[, 2]
1549    numeros <- suppressWarnings(as.numeric(letras_valores[, 3]))
1550
1551    # Si no hay letras válidas, devolver "0"
1552    if (all(is.na(letras))) {
1553      return("0")
1554    }
1555
1556    # En caso de números faltantes (NA), asumir valor 0
1557    numeros[is.na(numeros)] <- 0
1558
1559    # Agrupar por letra y conservar el máximo valor
1560    resumen <- aggregate(numeros, by = list(letras), FUN = max)
1561    colnames(resumen) <- c("letra", "max_valor")
1562
1563    # Escoger la letra con mayor valor, priorizando la primera en caso de empate
1564    maximo <- max(resumen$max_valor, na.rm = TRUE)
1565    letras_max <- resumen$letra[resumen$max_valor == maximo]
1566
1567    return(letras_max[1])
1568  }
1569
1570  # 3. Aplicar función y ajustar según la regla de N_clasi_empresa
1571  df <- df |>
1572    mutate(
1573      Clasi_empresa_main = vapply(Clasi_empresa, extraer_letra_dominante, character(1)),
1574      Clasi_empresa_main = if_else(N_clasi_empresa == 0, "0", Clasi_empresa_main)
1575    )
1576
1577  # --- Tratamiento de la variable Requi_adm: contar requisitos administrativos ---
1578
1579  df <- df |>
1580    mutate(
1581      # Si Requi_adm es NA o está vacía, asignar 0; si no, contar los elementos
        separados por ":"
```

```r
      N_requi = if_else(
        is.na(Requi_adm) | str_trim(Requi_adm) == "",
        0L,
        str_count(Requi_adm, ":") + 1L
      )
    )

# --- Tratamiento de la variable Prorroga ---


df <- df |>
  mutate(
    Prorroga = case_when(
      is.na(Prorroga) ~ "No",  # Si está vacía → No
      str_detect(tolower(Prorroga), "no|sin|ninguna|ningun|ningún|n\\.a") ~ "No",  # Si contiene negaciones comunes → No
      TRUE ~ "Sí"  # En cualquier otro caso → Sí
    ),
    Prorroga = factor(Prorroga, levels = c("No", "Sí"))
  )

# --- Tratamiento de la variable N_lotes ---

df <- df |>
  mutate(
    # Sustituimos "Sin lotes" por 1
    N_lotes = if_else(N_lotes == "Sin lotes", "1", N_lotes),

    # Convertimos a entero
    N_lotes = as.integer(N_lotes)
  )

# --- Tratamiento de la variable Clasi_max_req ---
df$Clasi_max_req <- ifelse(is.na(df$Clasi_max_req), 0, df$Clasi_max_req)


# --- Selección de variables para modelado predictivo ---

# Lista de variables seleccionadas para el modelo
vars_modelo <- c(
  "Identificador",
  "Identificador_lote",
  "Precio_adjudicacion",
  "N_ofertantes",
  "N_lotes",
  "Presupuesto_licitacion",
  "Tipo_de_contrato",
  "Tipo_de_Administracion",
  "Tipo_de_procedimiento",
  "Tramitacion",
  "Tipo_ganador_lote",
  "Plazo_m",
  #"Prorroga",
  "C_precio_p",
  "C_resto_objetivos_p",
  "C_juicio_valor_p",
  "Baja_p",
  "Modi_eco_p",
  "Clasi_max_req",
  "Intervalo_lici_d",
  "Intervalo_vigor_d",
  "Intervalo_urge_d",
  "N_CPV",
  "N_clasi_empresa",
  "Clasi_empresa_main",
  "Codigo_Postal", #-- Lo sacamos
  "CPV_main",
  "N_requi"
)

# Filtramos el data frame conservando solo esas variables
df <- df |>
  dplyr::select(all_of(vars_modelo))
```

```r

## Generamos nuevas variables categoricas a booleanas
df <- df |>
  mutate(
    Tipo_de_contrato_Obras = if_else(Tipo_de_contrato == "Obras", 1L, 0L),
    Tipo_de_Administracion_Autoridad_local = if_else(Tipo_de_Administracion ==
    "Autoridad local", 1L, 0L),
    Tipo_de_procedimiento_Abierto = if_else(Tipo_de_procedimiento == "Abierto", 1L, 0L
    ),
    Tramitacion_Ordinaria = if_else(Tramitacion == "Ordinaria", 1L, 0L),
    Tipo_ganador_lote_NIF = if_else(Tipo_ganador_lote == "NIF", 1L, 0L)
  )
##Variables categóricas: aseguramos que estén como factor
df$CPV_main <- factor(df$CPV_main)
df$Clasi_empresa_main <- factor(df$Clasi_empresa_main)
df$Codigo_Postal <- factor(df$Codigo_Postal)

# Eliminar variables con muchos NAs o crear versión limpia si es necesario
df_model <- df |>
  dplyr::select(
    "Identificador",
    "Identificador_lote",
    "Precio_adjudicacion",
    "N_ofertantes",
    "N_lotes",
    "Presupuesto_licitacion",
    #"Tipo_de_contrato",
    "Tipo_de_Administracion",
    "Tipo_de_procedimiento",
    #"Tramitacion",
    #"Tipo_ganador_lote",
    "Tipo_de_contrato_Obras",
    #"Tipo_de_Administracion_Autoridad_local",
    #"Tipo_de_procedimiento_Abierto",
    "Tramitacion_Ordinaria",
    "Tipo_ganador_lote_NIF",
    "Plazo_m",
    #"Prorroga",
    "C_precio_p",
    "C_resto_objetivos_p",
    "C_juicio_valor_p",
    "Baja_p",
    "Modi_eco_p",
    "Clasi_max_req",
    "Intervalo_lici_d",
    "Intervalo_vigor_d",
    "Intervalo_urge_d",
    "N_CPV",
    "N_clasi_empresa",
    "Clasi_empresa_main",
    "Codigo_Postal", #-- Lo sacamos
    "CPV_main",
    "N_requi"

  ) |>
  na.omit()  # Quita NAs para el modelo base


### DIVISIÓN DE DATOS 80/20 ALEATORIA ----
set.seed(321)
n <- nrow(df_model)
train_idx <- sample(seq_len(n), size = 0.8 * n)
df_train <- df_model[train_idx, ]
df_test  <- df_model[-train_idx, ]
save(df_train, df_test, file="train_test_v1_g.RData")
write.xlsx(df_train, "df_train_v1_g.xlsx", colNames = TRUE, rowNames = FALSE)
write.xlsx(df_test, "df_test_v1_g.xlsx", colNames = TRUE, rowNames = FALSE)

# Borramos

rm(list=ls())

```

```r
## ---------------------------
## SCRIPT 3 PCA_v2 ## --------
## ---------------------------

## Directorio LEER v2 ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/_final_conjunto_v4")

## Cargamos datos ----------

load("bd_civil_sin_NA_v2.RData")

## Directorio GUARDAR ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/_final_conjunto_v4")

## PREPROCESAMIENTO de VARIABLES ###### -------

# Aseguramos que estamos trabajando sobre la base correcta
df <- bd_civil_sin_NA

# Convertir factores de interés
df <- df |>
  mutate(
    Tipo_de_Administracion = as.factor(Tipo_de_Administracion),
    Tipo_de_procedimiento = as.factor(Tipo_de_procedimiento),
    Tipo_de_contrato = as.factor(Tipo_de_contrato),
    Tramitacion = as.factor(Tramitacion),
    Tipo_ganador_lote = as.factor(Tipo_ganador_lote),
    Modis_s_n = as.factor(Modis_s_n)
  )


# --- Tratamiento de la variable CPV en df ---
df <- df |>
  mutate(
    CPV_truncs = purrr::map(CPV, function(x) {
      if (is.na(x) || stringr::str_trim(x) == "") return(character(0))

      x_split <- stringr::str_split(x, ":", simplify = FALSE)[[1]]
      x_trimmed <- stringr::str_trim(x_split)
      x_valid <- x_trimmed[x_trimmed != ""]
      stringr::str_sub(x_valid, 1, 5)
    }),

    N_CPV = purrr::map_int(CPV_truncs, length),

    CPV_main = purrr::map_chr(CPV_truncs, function(cpvs) {
      if (length(cpvs) == 0) return(NA_character_)

      cpvs_45 <- cpvs[stringr::str_starts(cpvs, "45")]
      if (length(cpvs_45) == 0) return(NA_character_)

      ceros_derecha <- stringr::str_length(cpvs_45) -
        stringr::str_length(stringr::str_replace(cpvs_45, "0*$", ""))

      min_ceros <- min(ceros_derecha)
      seleccionados <- cpvs_45[ceros_derecha == min_ceros]

      return(seleccionados[1])
    })
  )


# --- Tratamiento de la variable Clasi_empresa sobre el data frame df ---

# 1. Calcular N_clasi_empresa: número de clasificaciones registradas (separadas por ":")
df <- df |>
  mutate(
    N_clasi_empresa = if_else(
```

```r
      is.na(Clasi_empresa) | str_trim(Clasi_empresa) == "",
      0L,
      str_count(Clasi_empresa, ":") + 1L
    )
  )

# 2. Función para extraer la letra con el número más alto (o la primera si hay empate)
extraer_letra_dominante <- function(clasi_string) {
  if (is.na(clasi_string) || str_trim(clasi_string) == "") {
    return("0")  # Regla: si no hay clasificaciones, devolver "0"
  }

  # Separar clasificaciones por ":"
  clasificaciones <- unlist(str_split(clasi_string, ":", simplify = FALSE))

  # Extraer letra y número si está, o solo letra si número no es válido
  letras_valores <- str_match(clasificaciones, "^([A-Z])[^0-9]*([0-9]*)")

  # Filtrar filas válidas
  letras <- letras_valores[, 2]
  numeros <- suppressWarnings(as.numeric(letras_valores[, 3]))

  # Si no hay letras válidas, devolver "0"
  if (all(is.na(letras))) {
    return("0")
  }

  # En caso de números faltantes (NA), asumir valor 0
  numeros[is.na(numeros)] <- 0

  # Agrupar por letra y conservar el máximo valor
  resumen <- aggregate(numeros, by = list(letras), FUN = max)
  colnames(resumen) <- c("letra", "max_valor")

  # Escoger la letra con mayor valor, priorizando la primera en caso de empate
  maximo <- max(resumen$max_valor, na.rm = TRUE)
  letras_max <- resumen$letra[resumen$max_valor == maximo]

  return(letras_max[1])
}

# 3. Aplicar función y ajustar según la regla de N_clasi_empresa
df <- df |>
  mutate(
    Clasi_empresa_main = vapply(Clasi_empresa, extraer_letra_dominante, character(1)),
    Clasi_empresa_main = if_else(N_clasi_empresa == 0, "0", Clasi_empresa_main)
  )

# --- Tratamiento de la variable Requi_adm: contar requisitos administrativos ---

df <- df |>
  mutate(
    # Si Requi_adm es NA o está vacía, asignar 0; si no, contar los elementos separados por ":"
    N_requi = if_else(
      is.na(Requi_adm) | str_trim(Requi_adm) == "",
      0L,
      str_count(Requi_adm, ":") + 1L
    )
  )

# --- Tratamiento de la variable Prorroga ---


df <- df |>
  mutate(
    Prorroga = case_when(
      is.na(Prorroga) ~ "No",  # Si está vacía → No
      str_detect(tolower(Prorroga), "no|sin|ninguna|ningun|ningún|n\\.a") ~ "No",  # Si contiene negaciones comunes → No
      TRUE ~ "Sí"  # En cualquier otro caso → Sí
    ),
    Prorroga = factor(Prorroga, levels = c("No", "Sí"))
```

```r
  )

# --- Tratamiento de la variable N_lotes ---

df <- df |>
  mutate(
    # Sustituimos "Sin lotes" por 1
    N_lotes = if_else(N_lotes == "Sin lotes", "1", N_lotes),

    # Convertimos a entero
    N_lotes = as.integer(N_lotes)
  )

# --- Tratamiento de la variable Clasi_max_req ---
df$Clasi_max_req <- ifelse(is.na(df$Clasi_max_req), 0, df$Clasi_max_req)


# --- Selección inicial de variables para modelado predictivo ---

# Lista de variables seleccionadas para el modelo
vars_modelo <- c(
  "Identificador",
  "Identificador_lote",
  "Precio_adjudicacion",
  "N_ofertantes",
  "N_lotes",
  "Presupuesto_licitacion",
  "Tipo_de_contrato",
  "Tipo_de_Administracion",
  "Tipo_de_procedimiento",
  "Tramitacion",
  "Tipo_ganador_lote",
  "Plazo_m",
  #"Prorroga",
  "C_precio_p",
  "C_resto_objetivos_p",
  "C_juicio_valor_p",
  "Baja_p",
  "Modi_eco_e", ## Para poder hacer limpieza de casos
  "Modi_eco_p",
  "Modi_temp_p", ## Para poder hacer limpieza de casos
  "Clasi_max_req",
  "Intervalo_lici_d",
  "Intervalo_vigor_d",
  "Intervalo_urge_d",
  "N_CPV",
  "N_clasi_empresa",
  "Clasi_empresa_main",
  "Codigo_Postal", #-- Lo sacamos
  "CPV_main",
  "N_requi"
)

# Filtramos el data frame conservando solo esas variables
df <- df |>
  dplyr::select(all_of(vars_modelo))

# --- Transformacion de variables para modelado predictivo ---
## Generamos nuevas variables categoricas a booleanas
df <- df |>
  mutate(
    Tipo_de_contrato_Obras = if_else(Tipo_de_contrato == "Obras", 1L, 0L),
    Tipo_de_Administracion_Autoridad_local = if_else(Tipo_de_Administracion ==
    "Autoridad local", 1L, 0L),
    Tipo_de_procedimiento_Abierto = if_else(Tipo_de_procedimiento == "Abierto", 1L, 0L
    ),
    Tipo_de_procedimiento_Simplificado = if_else(Tipo_de_procedimiento == "Abierto
    simplificado", 1L, 0L),
    Tramitacion_Ordinaria = if_else(Tramitacion == "Ordinaria", 1L, 0L),
    Tipo_ganador_lote_NIF = if_else(Tipo_ganador_lote == "NIF", 1L, 0L)
  )
##Variables categóricas: aseguramos que estén como factor
df$CPV_main <- factor(df$CPV_main)
```

```r
df$Clasi_empresa_main <- factor(df$Clasi_empresa_main)
df$Codigo_Postal <- factor(df$Codigo_Postal)


# Seleccion para transformacion de variables ---
df_model <- df |>
  dplyr::select(
    "Identificador",
    "Identificador_lote",
    "Precio_adjudicacion",
    "N_ofertantes",
    "N_lotes",
    #"Presupuesto_licitacion",
    #"Tipo_de_contrato",
    #"Tipo_de_Administracion",
    #"Tipo_de_procedimiento",
    #"Tramitacion",
    #"Tipo_ganador_lote",
    "Tipo_de_contrato_Obras",
    "Tipo_de_Administracion_Autoridad_local",
    "Tipo_de_procedimiento_Abierto",
    "Tipo_de_procedimiento_Simplificado",
    "Tramitacion_Ordinaria",
    "Tipo_ganador_lote_NIF",
    "Plazo_m",
    #"Prorroga",
    "C_precio_p",
    #"C_resto_objetivos_p",
    #"C_juicio_valor_p",
    "Baja_p",
    "Modi_eco_e", ## Para poder hacer limpieza de casos
    "Modi_eco_p",
    "Modi_temp_p",## Para poder hacer limpieza de casos
    "Clasi_max_req",
    #"Intervalo_lici_d",
    #"Intervalo_vigor_d",
    "Intervalo_urge_d",
    "N_CPV",
    "N_clasi_empresa",
    "Clasi_empresa_main",
    #"Codigo_Postal", #-- Lo sacamos
    #"CPV_main",
    "N_requi"

  )


### NUEVO FILTRO DE MODIS ECO 0 ####

# Contar casos que cumplen la condición de eliminación (Modi eco y temp 0)
casos_eliminar <- df_model |>
  filter(
    Modi_eco_p == 0 &
      (is.na(Modi_temp_p) | Modi_temp_p == 0)
  )

n_eliminados <- nrow(casos_eliminar)

# Crear nuevo data frame excluyendo esos casos
df_model_v2 <- df_model |>
  filter(
    !(Modi_eco_p == 0 &
        (is.na(Modi_temp_p) | Modi_temp_p == 0))
  )

# Mostrar el número de casos eliminados
cat("Número de casos eliminados:", n_eliminados, "\n")

# Crear variable categórica Modi_cat a partir de Modi_eco_p según rangos definidos
df_model_v2 <- df_model_v2 |>
  mutate(
    Modi_cat = cut(
      Modi_eco_p,
```

```
      breaks = c(-Inf, 0, 5, 9, 10, 14, 15, 19, 20, 35, 49, 50, Inf),
      labels = 0:11,
      right = TRUE,
      include.lowest = FALSE
    ),
    Modi_cat = as.integer(as.character(Modi_cat))  # Convertir a entero
  )

# Etiquetas legibles para cada categoría de Modi_cat
modi_cat_labels <- c(
  "(-Inf, 0]", "(0,5]", "(5,9]", "(9,10]", "(10,14]", "(14,15]",
  "(15,19]", "(19,20]", "(20,35]", "(35,49]", "(49,50]", "(50, Inf]"
)




# Seleccion final de variables para entrenamiento ---
df_model_v2 <- df_model_v2 |>
  dplyr::select(
    "Identificador",
    "Identificador_lote",
    "Precio_adjudicacion",
    "N_ofertantes",
    "N_lotes",
    #"Presupuesto_licitacion",
    #"Tipo_de_contrato",
    #"Tipo_de_Administracion",
    #"Tipo_de_procedimiento",
    #"Tramitacion",
    #"Tipo_ganador_lote",
    "Tipo_de_contrato_Obras",
    "Tipo_de_Administracion_Autoridad_local",
    "Tipo_de_procedimiento_Abierto",
    "Tipo_de_procedimiento_Simplificado",
    "Tramitacion_Ordinaria",
    "Tipo_ganador_lote_NIF",
    "Plazo_m",
    #"Prorroga",
    "C_precio_p",
    #"C_resto_objetivos_p",
    #"C_juicio_valor_p",
    "Baja_p",
    #"Modi_eco_e", ## Para poder hacer limpieza de casos
    #"Modi_eco_p",
    #"Modi_temp_p",## Para poder hacer limpieza de casos
    #"Clasi_max_req",
    #"Intervalo_lici_d",
    #"Intervalo_vigor_d",
    "Intervalo_urge_d",
    "N_CPV",
    "N_clasi_empresa",
    #"Clasi_empresa_main",
    #"Codigo_Postal", #-- Lo sacamos
    #"CPV_main",
    "N_requi",
    "Modi_cat"

  )


save(df_model_v2, file="df_model_v2_g.RData")
write.xlsx(df_model_v2, "df_model_v2_g.xlsx", colNames = TRUE, rowNames = FALSE)

#Limpiamos

rm(list=ls())

# ------------------------
# SCRIPT 4 SOM_v2 -------
# ------------------------


```

```r
## Directorio LEER v2 ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/_final_conjunto_v4")

## Cargamos datos ----------

load("bd_civil_sin_NA_v2.RData")

## Directorio GUARDAR ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/_final_conjunto_v4")



## PREPROCESAMIENTO de VARIABLES ###### -------

# Aseguramos que estamos trabajando sobre la base correcta
df <- bd_civil_sin_NA

# Convertir factores de interés
df <- df |>
  mutate(
    Tipo_de_Administracion = as.factor(Tipo_de_Administracion),
    Tipo_de_procedimiento = as.factor(Tipo_de_procedimiento),
    Tipo_de_contrato = as.factor(Tipo_de_contrato),
    Tramitacion = as.factor(Tramitacion),
    Tipo_ganador_lote = as.factor(Tipo_ganador_lote),
    Modis_s_n = as.factor(Modis_s_n)
  )


# --- Tratamiento de la variable CPV en df ---
df <- df |>
  mutate(
    CPV_truncs = purrr::map(CPV, function(x) {
      if (is.na(x) || stringr::str_trim(x) == "") return(character(0))

      x_split <- stringr::str_split(x, ":", simplify = FALSE)[[1]]
      x_trimmed <- stringr::str_trim(x_split)
      x_valid <- x_trimmed[x_trimmed != ""]
      stringr::str_sub(x_valid, 1, 5)
    }),

    N_CPV = purrr::map_int(CPV_truncs, length),

    CPV_main = purrr::map_chr(CPV_truncs, function(cpvs) {
      if (length(cpvs) == 0) return(NA_character_)

      cpvs_45 <- cpvs[stringr::str_starts(cpvs, "45")]
      if (length(cpvs_45) == 0) return(NA_character_)

      ceros_derecha <- stringr::str_length(cpvs_45) -
        stringr::str_length(stringr::str_replace(cpvs_45, "0*$", ""))

      min_ceros <- min(ceros_derecha)
      seleccionados <- cpvs_45[ceros_derecha == min_ceros]

      return(seleccionados[1])
    })
  )


# --- Tratamiento de la variable Clasi_empresa sobre el data frame df ---

# 1. Calcular N_clasi_empresa: número de clasificaciones registradas (separadas por ":")
df <- df |>
  mutate(
    N_clasi_empresa = if_else(
      is.na(Clasi_empresa) | str_trim(Clasi_empresa) == "",
      0L,
```

```r
      str_count(Clasi_empresa, ":") + 1L
    )
  )

# 2. Función para extraer la letra con el número más alto (o la primera si hay empate)
extraer_letra_dominante <- function(clasi_string) {
  if (is.na(clasi_string) || str_trim(clasi_string) == "") {
    return("0")  # Regla: si no hay clasificaciones, devolver "0"
  }

  # Separar clasificaciones por ":"
  clasificaciones <- unlist(str_split(clasi_string, ":", simplify = FALSE))

  # Extraer letra y número si está, o solo letra si número no es válido
  letras_valores <- str_match(clasificaciones, "^([A-Z])[^0-9]*([0-9]*)")

  # Filtrar filas válidas
  letras <- letras_valores[, 2]
  numeros <- suppressWarnings(as.numeric(letras_valores[, 3]))

  # Si no hay letras válidas, devolver "0"
  if (all(is.na(letras))) {
    return("0")
  }

  # En caso de números faltantes (NA), asumir valor 0
  numeros[is.na(numeros)] <- 0

  # Agrupar por letra y conservar el máximo valor
  resumen <- aggregate(numeros, by = list(letras), FUN = max)
  colnames(resumen) <- c("letra", "max_valor")

  # Escoger la letra con mayor valor, priorizando la primera en caso de empate
  maximo <- max(resumen$max_valor, na.rm = TRUE)
  letras_max <- resumen$letra[resumen$max_valor == maximo]

  return(letras_max[1])
}

# 3. Aplicar función y ajustar según la regla de N_clasi_empresa
df <- df |>
  mutate(
    Clasi_empresa_main = vapply(Clasi_empresa, extraer_letra_dominante, character(1)),
    Clasi_empresa_main = if_else(N_clasi_empresa == 0, "0", Clasi_empresa_main)
  )

# --- Tratamiento de la variable Requi_adm: contar requisitos administrativos ---

df <- df |>
  mutate(
    # Si Requi_adm es NA o está vacía, asignar 0; si no, contar los elementos
    separados por ":"
    N_requi = if_else(
      is.na(Requi_adm) | str_trim(Requi_adm) == "",
      0L,
      str_count(Requi_adm, ":") + 1L
    )
  )

# --- Tratamiento de la variable Prorroga ---


df <- df |>
  mutate(
    Prorroga = case_when(
      is.na(Prorroga) ~ "No",  # Si está vacía → No
      str_detect(tolower(Prorroga), "no|sin|ninguna|ningun|ningún|n\\.a") ~ "No",  #
      Si contiene negaciones comunes → No
      TRUE ~ "Sí"  # En cualquier otro caso → Sí
    ),
    Prorroga = factor(Prorroga, levels = c("No", "Sí"))
  )
```

```r
# --- Tratamiento de la variable N_lotes ---

df <- df |>
  mutate(
    # Sustituimos "Sin lotes" por 1
    N_lotes = if_else(N_lotes == "Sin lotes", "1", N_lotes),

    # Convertimos a entero
    N_lotes = as.integer(N_lotes)
  )

# --- Tratamiento de la variable Clasi_max_req ---
df$Clasi_max_req <- ifelse(is.na(df$Clasi_max_req), 0, df$Clasi_max_req)


# --- Selección inicial de variables para modelado predictivo ---

# Lista de variables seleccionadas para el modelo
vars_modelo <- c(
  "Identificador",
  "Identificador_lote",
  "Precio_adjudicacion",
  "N_ofertantes",
  "N_lotes",
  "Presupuesto_licitacion",
  "Tipo_de_contrato",
  "Tipo_de_Administracion",
  "Tipo_de_procedimiento",
  "Tramitacion",
  "Tipo_ganador_lote",
  "Plazo_m",
  #"Prorroga",
  "C_precio_p",
  "C_resto_objetivos_p",
  "C_juicio_valor_p",
  "Baja_p",
  "Modi_eco_e", ## Para poder hacer limpieza de casos
  "Modi_eco_p",
  "Modi_temp_p", ## Para poder hacer limpieza de casos
  "Clasi_max_req",
  "Intervalo_lici_d",
  "Intervalo_vigor_d",
  "Intervalo_urge_d",
  "N_CPV",
  "N_clasi_empresa",
  "Clasi_empresa_main",
  "Codigo_Postal", #-- Lo sacamos
  "CPV_main",
  "N_requi"
)

# Filtramos el data frame conservando solo esas variables
df <- df |>
  dplyr::select(all_of(vars_modelo))

# --- Transformacion de variables para modelado predictivo ---
## Generamos nuevas variables categoricas a booleanas
df <- df |>
  mutate(
    Tipo_de_contrato_Obras = if_else(Tipo_de_contrato == "Obras", 1L, 0L),
    Tipo_de_Administracion_Autoridad_local = if_else(Tipo_de_Administracion ==
    "Autoridad local", 1L, 0L),
    Tipo_de_procedimiento_Abierto = if_else(Tipo_de_procedimiento == "Abierto", 1L, 0L
    ),
    Tipo_de_procedimiento_Simplificado = if_else(Tipo_de_procedimiento == "Abierto
    simplificado", 1L, 0L),
    Tramitacion_Ordinaria = if_else(Tramitacion == "Ordinaria", 1L, 0L),
    Tipo_ganador_lote_NIF = if_else(Tipo_ganador_lote == "NIF", 1L, 0L)
  )
##Variables categóricas: aseguramos que estén como factor
df$CPV_main <- factor(df$CPV_main)
df$Clasi_empresa_main <- factor(df$Clasi_empresa_main)
df$Codigo_Postal <- factor(df$Codigo_Postal)
```

```r

# Seleccion para transformacion de variables ---
df_model <- df |>
  dplyr::select(
    "Identificador",
    "Identificador_lote",
    "Precio_adjudicacion",
    "N_ofertantes",
    "N_lotes",
    #"Presupuesto_licitacion",
    #"Tipo_de_contrato",
    #"Tipo_de_Administracion",
    #"Tipo_de_procedimiento",
    #"Tramitacion",
    #"Tipo_ganador_lote",
    "Tipo_de_contrato_Obras",
    "Tipo_de_Administracion_Autoridad_local",
    "Tipo_de_procedimiento_Abierto",
    "Tipo_de_procedimiento_Simplificado",
    "Tramitacion_Ordinaria",
    "Tipo_ganador_lote_NIF",
    "Plazo_m",
    #"Prorroga",
    "C_precio_p",
    #"C_resto_objetivos_p",
    #"C_juicio_valor_p",
    "Baja_p",
    "Modi_eco_e", ## Para poder hacer limpieza de casos
    "Modi_eco_p",
    "Modi_temp_p",## Para poder hacer limpieza de casos
    "Clasi_max_req",
    #"Intervalo_lici_d",
    #"Intervalo_vigor_d",
    "Intervalo_urge_d",
    "N_CPV",
    "N_clasi_empresa",
    "Clasi_empresa_main",
    #"Codigo_Postal", #-- Lo sacamos
    #"CPV_main",
    "N_requi"

  )

#### HISTOGRAMA PREVIO #### -----

# Crear cortes de 1% entre -5 y 55, incluyendo extremos
intervalos <- c(-Inf, seq(-5, 55, by = 1), Inf)

# Crear factor por rangos tipo (a, b]
df_model <- df_model |>
  mutate(
    modi_interval = cut(
      Modi_eco_p,
      breaks = intervalos,
      include.lowest = FALSE,
      right = TRUE,
      dig.lab = 5
    )
  )

# Guardar histograma en PDF
pdf("Histograma_previo_v2.pdf", width = 10, height = 6)
ggplot(df_model, aes(x = modi_interval)) +
  geom_bar(fill = "steelblue", color = "black") +
  labs(
    title = "Distribución de Modi_eco_p por Intervalos de 1%",
    x = "Intervalo Modi_eco_p (%)",
    y = "Frecuencia"
  ) +
  theme_minimal(base_size = 11) +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1, size = 7),
```

```r
    plot.title = element_text(hjust = 0.5)
  )
dev.off()

### NUEVO FILTRO DE MODIS ECO 0 ####

# Contar casos que cumplen la condición de eliminación (Modi eco y temp 0)
casos_eliminar <- df_model |>
  filter(
    Modi_eco_p == 0 &
      (is.na(Modi_temp_p) | Modi_temp_p == 0)
  )

n_eliminados <- nrow(casos_eliminar)

# Crear nuevo data frame excluyendo esos casos
df_model_v2 <- df_model |>
  filter(
    !(Modi_eco_p == 0 &
        (is.na(Modi_temp_p) | Modi_temp_p == 0))
  ) |>
  mutate(
    modi_interval = cut(
      Modi_eco_p,
      breaks = intervalos,
      include.lowest = FALSE,
      right = TRUE,
      dig.lab = 5
    )
  )

# Mostrar el número de casos eliminados
cat("Número de casos eliminados:", n_eliminados, "\n")

pdf("Histograma_previo_sin_ceros_v2.pdf", width = 10, height = 6)
ggplot(df_model_v2, aes(x = modi_interval)) +
  geom_bar(fill = "steelblue", color = "black") +
  labs(
    title = "Distribución de Modi_eco_p por Intervalos de 1%_Filtrados 0s",
    x = "Intervalo Modi_eco_p (%)",
    y = "Frecuencia"
  ) +
  theme_minimal(base_size = 11) +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1, size = 7),
    plot.title = element_text(hjust = 0.5)
  )

dev.off()

df_model_v2 <-df_model_v2 |>
  dplyr::select( - modi_interval)



# Crear variable categórica Modi_cat a partir de Modi_eco_p según rangos definidos
df_model_v2 <- df_model_v2 |>
  mutate(
    Modi_cat = cut(
      Modi_eco_p,
      breaks = c(-Inf, 0, 5, 9, 10, 14, 15, 19, 20, 35, 49, 50, Inf),
      labels = 0:11,
      right = TRUE,
      include.lowest = FALSE
    ),
    Modi_cat = as.integer(as.character(Modi_cat))  # Convertir a entero
  )

# Etiquetas legibles para cada categoría de Modi_cat
modi_cat_labels <- c(
  "(-Inf, 0]", "(0,5]", "(5,9]", "(9,10]", "(10,14]", "(14,15]",
  "(15,19]", "(19,20]", "(20,35]", "(35,49]", "(49,50]", "(50, Inf]"
)
```

```r

# Asignar factor con etiquetas para graficar
df_model_v2 <- df_model_v2 |>
  mutate(Modi_cat_factor = factor(Modi_cat, levels = 0:11, labels = modi_cat_labels))

# Guardar histograma actualizado en PDF
pdf("Histograma_post_v2.pdf", width = 10, height = 6)
ggplot(df_model_v2, aes(x = Modi_cat_factor)) +
  geom_bar(fill = "darkcyan", color = "black") +
  labs(
    title = "Distribución de Modi_eco_p por Categorías (Modi_cat)",
    x = "Categorías de Modi_eco_p (%)",
    y = "Frecuencia"
  ) +
  theme_minimal(base_size = 11) +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1, size = 7),
    plot.title = element_text(hjust = 0.5)
  )
dev.off()



# Seleccion final de variables para entrenamiento ---
df_model_v2 <- df_model_v2 |>
  dplyr::select(
    "Identificador",
    "Identificador_lote",
    "Precio_adjudicacion",
    "N_ofertantes",
    "N_lotes",
    #"Presupuesto_licitacion",
    #"Tipo_de_contrato",
    #"Tipo_de_Administracion",
    #"Tipo_de_procedimiento",
    #"Tramitacion",
    #"Tipo_ganador_lote",
    "Tipo_de_contrato_Obras",
    "Tipo_de_Administracion_Autoridad_local",
    "Tipo_de_procedimiento_Abierto",
    "Tipo_de_procedimiento_Simplificado",
    "Tramitacion_Ordinaria",
    "Tipo_ganador_lote_NIF",
    "Plazo_m",
    #"Prorroga",
    "C_precio_p",
    #"C_resto_objetivos_p",
    #"C_juicio_valor_p",
    "Baja_p",
    #"Modi_eco_e", ## Para poder hacer limpieza de casos
    #"Modi_eco_p",
    #"Modi_temp_p",## Para poder hacer limpieza de casos
    #"Clasi_max_req",
    #"Intervalo_lici_d",
    #"Intervalo_vigor_d",
    "Intervalo_urge_d",
    "N_CPV",
    "N_clasi_empresa",
    #"Clasi_empresa_main",
    #"Codigo_Postal", #-- Lo sacamos
    #"CPV_main",
    "N_requi",
    "Modi_cat"

  )

#### ENTRENAMIENTO SOM ### -----
### LIBRERÍAS NECESARIAS ----

# ---- PARTE 1: Preprocesamiento y División Train/Test ----

# División reproducible de los datos en train y test -----------------

```

```r
set.seed(321)

n <- nrow(df_model_v2)
train_idx <- sample(seq_len(n), size = 0.8 * n)

df_train_v2 <- df_model_v2[train_idx, ]
df_test_v2  <- df_model_v2[-train_idx, ]

# Guardamos conjuntos base
saveRDS(list(
  df_train = df_train_v2,
  df_test = df_test_v2,
  vars_predictoras = setdiff(colnames(df_model_v2), "Modi_cat")
), "00_datos_base_SOM_g.rds")

save(df_train_v2, file="df_train_SOM_v2_g.RData")
write.xlsx(df_train_v2, "df_train_SOM_v2_g.xlsx", colNames = TRUE, rowNames = FALSE)
save(df_test_v2, file="df_test_SOM_v2_g.RData")
write.xlsx(df_test_v2, "df_test_SOM_v2_g.xlsx", colNames = TRUE, rowNames = FALSE)


## Limpiamos
rm(list=ls())



# ------------------------
# SCRIPT 5 SOM_v3 -------
# ------------------------


## Directorio LEER v3 ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/_final_conjunto_v4")

## Cargamos datos ----------

load("bd_civil_sin_NA_v2.RData")

## Directorio GUARDAR ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/_final_conjunto_v4")


## PREPROCESAMIENTO de VARIABLES ###### -------

# Aseguramos que estamos trabajando sobre la base correcta
df <- bd_civil_sin_NA

# Convertir factores de interés
df <- df |>
  mutate(
    Tipo_de_Administracion = as.factor(Tipo_de_Administracion),
    Tipo_de_procedimiento = as.factor(Tipo_de_procedimiento),
    Tipo_de_contrato = as.factor(Tipo_de_contrato),
    Tramitacion = as.factor(Tramitacion),
    Tipo_ganador_lote = as.factor(Tipo_ganador_lote),
    Modis_s_n = as.factor(Modis_s_n)
  )


# --- Tratamiento de la variable CPV en df ---
df <- df |>
  mutate(
    CPV_truncs = purrr::map(CPV, function(x) {
      if (is.na(x) || stringr::str_trim(x) == "") return(character(0))

      x_split <- stringr::str_split(x, ":", simplify = FALSE)[[1]]
      x_trimmed <- stringr::str_trim(x_split)
      x_valid <- x_trimmed[x_trimmed != ""]
      stringr::str_sub(x_valid, 1, 5)
```

```
      }),

      N_CPV = purrr::map_int(CPV_truncs, length),

      CPV_main = purrr::map_chr(CPV_truncs, function(cpvs) {
        if (length(cpvs) == 0) return(NA_character_)

        cpvs_45 <- cpvs[stringr::str_starts(cpvs, "45")]
        if (length(cpvs_45) == 0) return(NA_character_)

        ceros_derecha <- stringr::str_length(cpvs_45) -
          stringr::str_length(stringr::str_replace(cpvs_45, "0*$", ""))

        min_ceros <- min(ceros_derecha)
        seleccionados <- cpvs_45[ceros_derecha == min_ceros]

        return(seleccionados[1])
      })
    )


  # --- Tratamiento de la variable Clasi_empresa sobre el data frame df ---

  # 1. Calcular N_clasi_empresa: número de clasificaciones registradas (separadas por
  ":")
  df <- df |>
    mutate(
      N_clasi_empresa = if_else(
        is.na(Clasi_empresa) | str_trim(Clasi_empresa) == "",
        0L,
        str_count(Clasi_empresa, ":") + 1L
      )
    )

  # 2. Función para extraer la letra con el número más alto (o la primera si hay empate)
  extraer_letra_dominante <- function(clasi_string) {
    if (is.na(clasi_string) || str_trim(clasi_string) == "") {
      return("0")  # Regla: si no hay clasificaciones, devolver "0"
    }

    # Separar clasificaciones por ":"
    clasificaciones <- unlist(str_split(clasi_string, ":", simplify = FALSE))

    # Extraer letra y número si está, o solo letra si número no es válido
    letras_valores <- str_match(clasificaciones, "^([A-Z])[^0-9]*([0-9]*)")

    # Filtrar filas válidas
    letras <- letras_valores[, 2]
    numeros <- suppressWarnings(as.numeric(letras_valores[, 3]))

    # Si no hay letras válidas, devolver "0"
    if (all(is.na(letras))) {
      return("0")
    }

    # En caso de números faltantes (NA), asumir valor 0
    numeros[is.na(numeros)] <- 0

    # Agrupar por letra y conservar el máximo valor
    resumen <- aggregate(numeros, by = list(letras), FUN = max)
    colnames(resumen) <- c("letra", "max_valor")

    # Escoger la letra con mayor valor, priorizando la primera en caso de empate
    maximo <- max(resumen$max_valor, na.rm = TRUE)
    letras_max <- resumen$letra[resumen$max_valor == maximo]

    return(letras_max[1])
  }

  # 3. Aplicar función y ajustar según la regla de N_clasi_empresa
  df <- df |>
    mutate(
      Clasi_empresa_main = vapply(Clasi_empresa, extraer_letra_dominante, character(1)),
```

```r
      Clasi_empresa_main = if_else(N_clasi_empresa == 0, "0", Clasi_empresa_main)
    )

  # --- Tratamiento de la variable Requi_adm: contar requisitos administrativos ---

  df <- df |>
    mutate(
      # Si Requi_adm es NA o está vacía, asignar 0; si no, contar los elementos separados por ":"
      N_requi = if_else(
        is.na(Requi_adm) | str_trim(Requi_adm) == "",
        0L,
        str_count(Requi_adm, ":") + 1L
      )
    )

  # --- Tratamiento de la variable Prorroga ---


  df <- df |>
    mutate(
      Prorroga = case_when(
        is.na(Prorroga) ~ "No",  # Si está vacía → No
        str_detect(tolower(Prorroga), "no|sin|ninguna|ningun|ningún|n\\.a") ~ "No",  # Si contiene negaciones comunes → No
        TRUE ~ "Sí"  # En cualquier otro caso → Sí
      ),
      Prorroga = factor(Prorroga, levels = c("No", "Sí"))
    )

  # --- Tratamiento de la variable N_lotes ---

  df <- df |>
    mutate(
      # Sustituimos "Sin lotes" por 1
      N_lotes = if_else(N_lotes == "Sin lotes", "1", N_lotes),

      # Convertimos a entero
      N_lotes = as.integer(N_lotes)
    )

  # --- Tratamiento de la variable Clasi_max_req ---
  df$Clasi_max_req <- ifelse(is.na(df$Clasi_max_req), 0, df$Clasi_max_req)


  # --- Selección inicial de variables para modelado predictivo ---

  # Lista de variables seleccionadas para el modelo
  vars_modelo <- c(
    "Identificador",
    "Identificador_lote",
    "Precio_adjudicacion",
    "N_ofertantes",
    "N_lotes",
    "Presupuesto_licitacion",
    "Tipo_de_contrato",
    "Tipo_de_Administracion",
    "Tipo_de_procedimiento",
    "Tramitacion",
    "Tipo_ganador_lote",
    "Plazo_m",
    #"Prorroga",
    "C_precio_p",
    "C_resto_objetivos_p",
    "C_juicio_valor_p",
    "Baja_p",
    "Modi_eco_e", ## Para poder hacer limpieza de casos
    "Modi_eco_p",
    "Modi_temp_p", ## Para poder hacer limpieza de casos
    "Clasi_max_req",
    "Intervalo_lici_d",
    "Intervalo_vigor_d",
    "Intervalo_urge_d",
```

```
    "N_CPV",
    "N_clasi_empresa",
    "Clasi_empresa_main",
    "Codigo_Postal", #-- Lo sacamos
    "CPV_main",
    "N_requi"
  )

  # Filtramos el data frame conservando solo esas variables
  df <- df |>
    dplyr::select(all_of(vars_modelo))

  # --- Transformacion de variables para modelado predictivo ---
  ## Generamos nuevas variables categoricas a booleanas
  df <- df |>
    mutate(
      Tipo_de_contrato_Obras = if_else(Tipo_de_contrato == "Obras", 1L, 0L),
      Tipo_de_Administracion_Autoridad_local = if_else(Tipo_de_Administracion ==
      "Autoridad local", 1L, 0L),
      Tipo_de_procedimiento_Abierto = if_else(Tipo_de_procedimiento == "Abierto", 1L, 0L
      ),
      Tipo_de_procedimiento_Simplificado = if_else(Tipo_de_procedimiento == "Abierto
      simplificado", 1L, 0L),
      Tramitacion_Ordinaria = if_else(Tramitacion == "Ordinaria", 1L, 0L),
      Tipo_ganador_lote_NIF = if_else(Tipo_ganador_lote == "NIF", 1L, 0L)
    )
  ##Variables categóricas: aseguramos que estén como factor
  df$CPV_main <- factor(df$CPV_main)
  df$Clasi_empresa_main <- factor(df$Clasi_empresa_main)
  df$Codigo_Postal <- factor(df$Codigo_Postal)


  # Seleccion para transformacion de variables ---
  df_model <- df |>
    dplyr::select(
      "Identificador",
      "Identificador_lote",
      "Precio_adjudicacion",
      "N_ofertantes",
      "N_lotes",
      #"Presupuesto_licitacion",
      #"Tipo_de_contrato",
      #"Tipo_de_Administracion",
      #"Tipo_de_procedimiento",
      #"Tramitacion",
      #"Tipo_ganador_lote",
      "Tipo_de_contrato_Obras",
      "Tipo_de_Administracion_Autoridad_local",
      "Tipo_de_procedimiento_Abierto",
      "Tipo_de_procedimiento_Simplificado",
      "Tramitacion_Ordinaria",
      "Tipo_ganador_lote_NIF",
      "Plazo_m",
      #"Prorroga",
      "C_precio_p",
      #"C_resto_objetivos_p",
      #"C_juicio_valor_p",
      "Baja_p",
      "Modi_eco_e", ## Para poder hacer limpieza de casos
      "Modi_eco_p",
      "Modi_temp_p",## Para poder hacer limpieza de casos
      "Clasi_max_req",
      #"Intervalo_lici_d",
      #"Intervalo_vigor_d",
      "Intervalo_urge_d",
      "N_CPV",
      "N_clasi_empresa",
      "Clasi_empresa_main",
      #"Codigo_Postal", #-- Lo sacamos
      #"CPV_main",
      "N_requi"

    )
```

```r

#### HISTOGRAMA PREVIO #### -----

# Crear cortes de 1% entre -5 y 55, incluyendo extremos
intervalos <- c(-Inf, seq(-5, 55, by = 1), Inf)

# Crear factor por rangos tipo (a, b]
df_model <- df_model |>
  mutate(
    modi_interval = cut(
      Modi_eco_p,
      breaks = intervalos,
      include.lowest = FALSE,
      right = TRUE,
      dig.lab = 5
    )
  )

# Guardar histograma en PDF
pdf("Histograma_previo_v3_g.pdf", width = 10, height = 6)
ggplot(df_model, aes(x = modi_interval)) +
  geom_bar(fill = "steelblue", color = "black") +
  labs(
    title = "Distribución de Modi_eco_p por Intervalos de 1%",
    x = "Intervalo Modi_eco_p (%)",
    y = "Frecuencia"
  ) +
  theme_minimal(base_size = 11) +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1, size = 7),
    plot.title = element_text(hjust = 0.5)
  )
dev.off()

### NUEVO FILTRO DE MODIS ECO 0 ####

# Contar casos que cumplen la condición de eliminación (Modi eco y temp 0)
casos_eliminar <- df_model |>
  filter(
    Modi_eco_p == 0 &
      (is.na(Modi_temp_p) | Modi_temp_p == 0)
  )

n_eliminados <- nrow(casos_eliminar)

# Crear nuevo data frame excluyendo esos casos
df_model_v3 <- df_model |>
  filter(
    !(Modi_eco_p == 0 &
        (is.na(Modi_temp_p) | Modi_temp_p == 0))
  ) |>
  mutate(
    modi_interval = cut(
      Modi_eco_p,
      breaks = intervalos,
      include.lowest = FALSE,
      right = TRUE,
      dig.lab = 5
    )
  )

# Mostrar el número de casos eliminados
cat("Número de casos eliminados con Modi_eco=0 y Modi_temp=NA/0:", n_eliminados, "\n")

pdf("Histograma_previo_sin_ceros_v3_g.pdf", width = 10, height = 6)
ggplot(df_model_v3, aes(x = modi_interval)) +
  geom_bar(fill = "steelblue", color = "black") +
  labs(
    title = "Distribución de Modi_eco_p por Intervalos de 1%_Filtrados 0s",
    x = "Intervalo Modi_eco_p (%)",
    y = "Frecuencia"
  ) +
  theme_minimal(base_size = 11) +
```

```r
    theme(
      axis.text.x = element_text(angle = 45, hjust = 1, size = 7),
      plot.title = element_text(hjust = 0.5)
    )

  dev.off()

  ## Forzamos a todos los valores del intervalo "(-1,0]" a ser 0
  df_model_v3 <- df_model_v3 |>
    mutate(
      Modi_eco_p = if_else(
        modi_interval == "(-1,0]",
        0,
        Modi_eco_p
      )
    )


  # === Guardamos el número de casos actuales antes del filtrado ===
  n_antes <- nrow(df_model_v3)

  # === Eliminamos los casos donde Modi_eco_p sea menor o igual a 0 o mayor a 54.99 ===
  df_model_v3 <- df_model_v3 |>
    filter(Modi_eco_p > 0, Modi_eco_p <= 54.99)

  # === Guardamos cuántos casos se han eliminado ===
  n_despues <- nrow(df_model_v3)
  n_casos_colas <- n_antes - n_despues

  # === Mostramos resultados por consola ===
  cat("✅ Número de casos eliminados (Modi_eco_p fuera de (0, 54.99]):", n_casos_colas,
  "\n")
  cat("✅ Número de casos restantes en df_model_v3:", n_despues, "\n")


  pdf("Histograma_post_v3_sin_colas_v3_g.pdf", width = 10, height = 6)
  ggplot(df_model_v3, aes(x = modi_interval)) +
    geom_bar(fill = "coral", color = "black") +
    labs(
      title = "Distribución de Modi_eco_p por Intervalos sin colas",
      x = "Intervalo Modi_eco_p (%)",
      y = "Frecuencia"
    ) +
    theme_minimal(base_size = 11) +
    theme(
      axis.text.x = element_text(angle = 45, hjust = 1, size = 7),
      plot.title = element_text(hjust = 0.5)
    )

  dev.off()

  ## Eliminamos variable categorica de intervalos
  df_model_v3 <-df_model_v3 |>
    dplyr::select( - modi_interval)


  ### Seleccion final de variables para entrenamiento ### ----
  df_model_v3 <- df_model_v3 |>
    dplyr::select(
      "Identificador",
      "Identificador_lote",
      "Precio_adjudicacion",
      "N_ofertantes",
      "N_lotes",
      #"Presupuesto_licitacion",
      #"Tipo_de_contrato",
      #"Tipo_de_Administracion",
      #"Tipo_de_procedimiento",
      #"Tramitacion",
      #"Tipo_ganador_lote",
      "Tipo_de_contrato_Obras",
      "Tipo_de_Administracion_Autoridad_local",
      "Tipo_de_procedimiento_Abierto",
```

```
      #"Tipo_de_procedimiento_Simplificado",
      "Tramitacion_Ordinaria",
      #"Tipo_ganador_lote_NIF",
      "Plazo_m",
      #"Prorroga",
      "C_precio_p",
      #"C_resto_objetivos_p",
      #"C_juicio_valor_p",
      "Baja_p",
      #"Modi_eco_e", ## Para poder hacer limpieza de casos
      "Modi_eco_p",
      #"Modi_temp_p",## Para poder hacer limpieza de casos
      #"Clasi_max_req",
      #"Intervalo_lici_d",
      #"Intervalo_vigor_d",
      "Intervalo_urge_d",
      "N_CPV",
      "N_clasi_empresa",
      #"Clasi_empresa_main",
      #"Codigo_Postal", #-- Lo sacamos
      #"CPV_main",
      #"N_requi",

    )



# === División reproducible de los datos en entrenamiento y test === -----

set.seed(321)

n <- nrow(df_model_v3)
train_idx <- sample.int(n, size = floor(0.8 * n))  # Selección aleatoria reproducible

# Crear conjuntos de entrenamiento y test
df_train_v3 <- df_model_v3[train_idx, ]
df_test_v3  <- df_model_v3[-train_idx, ]

# Comprobamos que la variable objetivo está presente
stopifnot("Modi_eco_p" %in% names(df_model_v3))

# Guardamos los conjuntos y nombres de variables predictoras
saveRDS(list(
  df_train = df_train_v3,
  df_test = df_test_v3,
  vars_predictoras = setdiff(colnames(df_model_v3), "Modi_eco_p")
), "datos_base_SOM_v3_g.rds")


save(df_train_v3, file="df_train_SOM_v3_g.RData")
write.xlsx(df_train_v3, "df_train_SOM_v3_g.xlsx", colNames = TRUE, rowNames = FALSE)
save(df_test_v3, file="df_test_SOM_v3_g.RData")
write.xlsx(df_test_v3, "df_test_SOM_v3_g.xlsx", colNames = TRUE, rowNames = FALSE)


## Limpiamos
rm(list=ls())


# ------------------------
# SCRIPT 6 y 7 SOM_v4 y Script_Distribucion_v3 -------
# ------------------------



## Directorio LEER v4 ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos
atom/PLACSP/_final_conjunto_v4")

## Cargamos datos ----------

load("bd_civil_sin_NA_v2.RData")
```

```r

## Directorio GUARDAR ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/_final_conjunto_v4")



## PREPROCESAMIENTO de VARIABLES ###### -------

# Aseguramos que estamos trabajando sobre la base correcta
df <- bd_civil_sin_NA

# Convertir factores de interés
df <- df |>
  mutate(
    Tipo_de_Administracion = as.factor(Tipo_de_Administracion),
    Tipo_de_procedimiento = as.factor(Tipo_de_procedimiento),
    Tipo_de_contrato = as.factor(Tipo_de_contrato),
    Tramitacion = as.factor(Tramitacion),
    Tipo_ganador_lote = as.factor(Tipo_ganador_lote),
    Modis_s_n = as.factor(Modis_s_n)
  )


# --- Tratamiento de la variable CPV en df ---
df <- df |>
  mutate(
    CPV_truncs = purrr::map(CPV, function(x) {
      if (is.na(x) || stringr::str_trim(x) == "") return(character(0))

      x_split <- stringr::str_split(x, ":", simplify = FALSE)[[1]]
      x_trimmed <- stringr::str_trim(x_split)
      x_valid <- x_trimmed[x_trimmed != ""]
      stringr::str_sub(x_valid, 1, 5)
    }),

    N_CPV = purrr::map_int(CPV_truncs, length),

    CPV_main = purrr::map_chr(CPV_truncs, function(cpvs) {
      if (length(cpvs) == 0) return(NA_character_)

      cpvs_45 <- cpvs[stringr::str_starts(cpvs, "45")]
      if (length(cpvs_45) == 0) return(NA_character_)

      ceros_derecha <- stringr::str_length(cpvs_45) -
        stringr::str_length(stringr::str_replace(cpvs_45, "0*$", ""))

      min_ceros <- min(ceros_derecha)
      seleccionados <- cpvs_45[ceros_derecha == min_ceros]

      return(seleccionados[1])
    })
  )


# --- Tratamiento de la variable Clasi_empresa sobre el data frame df ---

# 1. Calcular N_clasi_empresa: número de clasificaciones registradas (separadas por ":")
df <- df |>
  mutate(
    N_clasi_empresa = if_else(
      is.na(Clasi_empresa) | str_trim(Clasi_empresa) == "",
      0L,
      str_count(Clasi_empresa, ":") + 1L
    )
  )

# 2. Función para extraer la letra con el número más alto (o la primera si hay empate)
extraer_letra_dominante <- function(clasi_string) {
  if (is.na(clasi_string) || str_trim(clasi_string) == "") {
    return("0")  # Regla: si no hay clasificaciones, devolver "0"
```

```r
  }

  # Separar clasificaciones por ":"
  clasificaciones <- unlist(str_split(clasi_string, ":", simplify = FALSE))

  # Extraer letra y número si está, o solo letra si número no es válido
  letras_valores <- str_match(clasificaciones, "^([A-Z])[^0-9]*([0-9]*)")

  # Filtrar filas válidas
  letras <- letras_valores[, 2]
  numeros <- suppressWarnings(as.numeric(letras_valores[, 3]))

  # Si no hay letras válidas, devolver "0"
  if (all(is.na(letras))) {
    return("0")
  }

  # En caso de números faltantes (NA), asumir valor 0
  numeros[is.na(numeros)] <- 0

  # Agrupar por letra y conservar el máximo valor
  resumen <- aggregate(numeros, by = list(letras), FUN = max)
  colnames(resumen) <- c("letra", "max_valor")

  # Escoger la letra con mayor valor, priorizando la primera en caso de empate
  maximo <- max(resumen$max_valor, na.rm = TRUE)
  letras_max <- resumen$letra[resumen$max_valor == maximo]

  return(letras_max[1])
}

# 3. Aplicar función y ajustar según la regla de N_clasi_empresa
df <- df |>
  mutate(
    Clasi_empresa_main = vapply(Clasi_empresa, extraer_letra_dominante, character(1)),
    Clasi_empresa_main = if_else(N_clasi_empresa == 0, "0", Clasi_empresa_main)
  )

# --- Tratamiento de la variable Requi_adm: contar requisitos administrativos ---

df <- df |>
  mutate(
    # Si Requi_adm es NA o está vacía, asignar 0; si no, contar los elementos separados por ":"
    N_requi = if_else(
      is.na(Requi_adm) | str_trim(Requi_adm) == "",
      0L,
      str_count(Requi_adm, ":") + 1L
    )
  )

# --- Tratamiento de la variable Prorroga ---


df <- df |>
  mutate(
    Prorroga = case_when(
      is.na(Prorroga) ~ "No",  # Si está vacía → No
      str_detect(tolower(Prorroga), "no|sin|ninguna|ningun|ningún|n\\.a") ~ "No",  # Si contiene negaciones comunes → No
      TRUE ~ "Sí"  # En cualquier otro caso → Sí
    ),
    Prorroga = factor(Prorroga, levels = c("No", "Sí"))
  )

# --- Tratamiento de la variable N_lotes ---

df <- df |>
  mutate(
    # Sustituimos "Sin lotes" por 1
    N_lotes = if_else(N_lotes == "Sin lotes", "1", N_lotes),

    # Convertimos a entero
```

```r
    N_lotes = as.integer(N_lotes)
  )

# --- Tratamiento de la variable Clasi_max_req ---
df$Clasi_max_req <- ifelse(is.na(df$Clasi_max_req), 0, df$Clasi_max_req)


# --- Selección inicial de variables para modelado predictivo ---

# Lista de variables seleccionadas para el modelo
vars_modelo <- c(
  "Identificador",
  "Identificador_lote",
  "Precio_adjudicacion",
  "N_ofertantes",
  "N_lotes",
  "Presupuesto_licitacion",
  "Tipo_de_contrato",
  "Tipo_de_Administracion",
  "Tipo_de_procedimiento",
  "Tramitacion",
  "Tipo_ganador_lote",
  "Plazo_m",
  #"Prorroga",
  "C_precio_p",
  "C_resto_objetivos_p",
  "C_juicio_valor_p",
  "Baja_p",
  "Modi_eco_e", ## Para poder hacer limpieza de casos
  "Modi_eco_p",
  "Modi_temp_p", ## Para poder hacer limpieza de casos
  "Clasi_max_req",
  "Intervalo_lici_d",
  "Intervalo_vigor_d",
  "Intervalo_urge_d",
  "N_CPV",
  "N_clasi_empresa",
  "Clasi_empresa_main",
  "Codigo_Postal", #-- Lo sacamos
  "CPV_main",
  "N_requi"
)

# Filtramos el data frame conservando solo esas variables
df <- df |>
  dplyr::select(all_of(vars_modelo))

# --- Transformacion de variables para modelado predictivo ---
## Generamos nuevas variables categoricas a booleanas
df <- df |>
  mutate(
    Tipo_de_contrato_Obras = if_else(Tipo_de_contrato == "Obras", 1L, 0L),
    Tipo_de_Administracion_Autoridad_local = if_else(Tipo_de_Administracion ==
    "Autoridad local", 1L, 0L),
    Tipo_de_procedimiento_Abierto = if_else(Tipo_de_procedimiento == "Abierto", 1L, 0L
    ),
    Tipo_de_procedimiento_Simplificado = if_else(Tipo_de_procedimiento == "Abierto
    simplificado", 1L, 0L),
    Tramitacion_Ordinaria = if_else(Tramitacion == "Ordinaria", 1L, 0L),
    Tipo_ganador_lote_NIF = if_else(Tipo_ganador_lote == "NIF", 1L, 0L)
  )
##Variables categóricas: aseguramos que estén como factor
df$CPV_main <- factor(df$CPV_main)
df$Clasi_empresa_main <- factor(df$Clasi_empresa_main)
df$Codigo_Postal <- factor(df$Codigo_Postal)


# Seleccion para transformacion de variables ---
df_model <- df |>
  dplyr::select(
    "Identificador",
    "Identificador_lote",
    "Precio_adjudicacion",
```

```
      "N_ofertantes",
      "N_lotes",
      #"Presupuesto_licitacion",
      #"Tipo_de_contrato",
      #"Tipo_de_Administracion",
      #"Tipo_de_procedimiento",
      #"Tramitacion",
      #"Tipo_ganador_lote",
      "Tipo_de_contrato_Obras",
      "Tipo_de_Administracion_Autoridad_local",
      "Tipo_de_procedimiento_Abierto",
      "Tipo_de_procedimiento_Simplificado",
      "Tramitacion_Ordinaria",
      "Tipo_ganador_lote_NIF",
      "Plazo_m",
      #"Prorroga",
      "C_precio_p",
      #"C_resto_objetivos_p",
      #"C_juicio_valor_p",
      "Baja_p",
      "Modi_eco_e", ## Para poder hacer limpieza de casos
      "Modi_eco_p",
      "Modi_temp_p",## Para poder hacer limpieza de casos
      "Clasi_max_req",
      #"Intervalo_lici_d",
      #"Intervalo_vigor_d",
      "Intervalo_urge_d",
      "N_CPV",
      "N_clasi_empresa",
      "Clasi_empresa_main",
      #"Codigo_Postal", #-- Lo sacamos
      #"CPV_main",
      "N_requi"

    )

  #### HISTOGRAMA PREVIO #### -----

  # Crear cortes de 1% entre -5 y 55, incluyendo extremos
  intervalos <- c(-Inf, seq(-5, 55, by = 1), Inf)

  # Crear factor por rangos tipo (a, b]
  df_model <- df_model |>
    mutate(
      modi_interval = cut(
        Modi_eco_p,
        breaks = intervalos,
        include.lowest = FALSE,
        right = TRUE,
        dig.lab = 5
      )
    )

  # Guardar histograma en PDF
  pdf("Histograma_previo_v4_g.pdf", width = 10, height = 6)
  ggplot(df_model, aes(x = modi_interval)) +
    geom_bar(fill = "steelblue", color = "black") +
    labs(
      title = "Distribución de Modi_eco_p por Intervalos de 1%",
      x = "Intervalo Modi_eco_p (%)",
      y = "Frecuencia"
    ) +
    theme_minimal(base_size = 11) +
    theme(
      axis.text.x = element_text(angle = 45, hjust = 1, size = 7),
      plot.title = element_text(hjust = 0.5)
    )
  dev.off()

  ### NUEVO FILTRO DE MODIS ECO 0 ####

  # Contar casos que cumplen la condición de eliminación (Modi eco y temp 0)
  casos_eliminar <- df_model |>
```

```r
  filter(
    Modi_eco_p == 0 &
      (is.na(Modi_temp_p) | Modi_temp_p == 0)
  )

n_eliminados <- nrow(casos_eliminar)

# Crear nuevo data frame excluyendo esos casos
df_model_v3 <- df_model |>
  filter(
    !(Modi_eco_p == 0 &
        (is.na(Modi_temp_p) | Modi_temp_p == 0))
  ) |>
  mutate(
    modi_interval = cut(
      Modi_eco_p,
      breaks = intervalos,
      include.lowest = FALSE,
      right = TRUE,
      dig.lab = 5
    )
  )

# Mostrar el número de casos eliminados
cat("Número de casos eliminados con Modi_eco=0 y Modi_temp=NA/0:", n_eliminados, "\n")

pdf("Histograma_previo_sin_ceros_v4_g.pdf", width = 10, height = 6)
ggplot(df_model_v3, aes(x = modi_interval)) +
  geom_bar(fill = "steelblue", color = "black") +
  labs(
    title = "Distribución de Modi_eco_p por Intervalos de 1%_Filtrados 0s",
    x = "Intervalo Modi_eco_p (%)",
    y = "Frecuencia"
  ) +
  theme_minimal(base_size = 11) +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1, size = 7),
    plot.title = element_text(hjust = 0.5)
  )

dev.off()

## Forzamos a todos los valores del intervalo "(-1,0]" a ser 0
df_model_v3 <- df_model_v3 |>
  mutate(
    Modi_eco_p = if_else(
      modi_interval == "(-1,0]",
      0,
      Modi_eco_p
    )
  )


# === Guardamos el número de casos actuales antes del filtrado ===
n_antes <- nrow(df_model_v3)

# === Eliminamos los casos donde Modi_eco_p sea menor o igual a 0 o mayor a 54.99 ===
df_model_v3 <- df_model_v3 |>
  filter(Modi_eco_p > 0, Modi_eco_p <= 54.99)

# === Guardamos cuántos casos se han eliminado ===
n_despues <- nrow(df_model_v3)
n_casos_colas <- n_antes - n_despues

# === Mostramos resultados por consola ===
cat("✅ Número de casos eliminados (Modi_eco_p fuera de (0, 54.99]):", n_casos_colas,
"\n")
cat("✅ Número de casos restantes en df_model_v3:", n_despues, "\n")


pdf("Histograma_post_v3_sin_colas.pdf", width = 10, height = 6)
ggplot(df_model_v3, aes(x = modi_interval)) +
  geom_bar(fill = "coral", color = "black") +
```

```
    labs(
      title = "Distribución de Modi_eco_p por Intervalos sin colas",
      x = "Intervalo Modi_eco_p (%)",
      y = "Frecuencia"
    ) +
    theme_minimal(base_size = 11) +
    theme(
      axis.text.x = element_text(angle = 45, hjust = 1, size = 7),
      plot.title = element_text(hjust = 0.5)
    )

  dev.off()

  ## Eliminamos variable categorica de intervalos
  df_model_v3 <-df_model_v3 |>
    dplyr::select( - modi_interval)


  ### Seleccion final de variables para entrenamiento ### ----
  df_model_v3 <- df_model_v3 |>
    dplyr::select(
      "Identificador",
      "Identificador_lote",
      "Precio_adjudicacion",
      "N_ofertantes",
      "N_lotes",
      #"Presupuesto_licitacion",
      #"Tipo_de_contrato",
      #"Tipo_de_Administracion",
      #"Tipo_de_procedimiento",
      #"Tramitacion",
      #"Tipo_ganador_lote",
      "Tipo_de_contrato_Obras",
      "Tipo_de_Administracion_Autoridad_local",
      "Tipo_de_procedimiento_Abierto",
      #"Tipo_de_procedimiento_Simplificado",
      "Tramitacion_Ordinaria",
      #"Tipo_ganador_lote_NIF",
      "Plazo_m",
      #"Prorroga",
      "C_precio_p",
      #"C_resto_objetivos_p",
      #"C_juicio_valor_p",
      "Baja_p",
      #"Modi_eco_e", ## Para poder hacer limpieza de casos
      "Modi_eco_p",
      #"Modi_temp_p",## Para poder hacer limpieza de casos
      #"Clasi_max_req",
      #"Intervalo_lici_d",
      #"Intervalo_vigor_d",
      "Intervalo_urge_d",
      "N_CPV",
      "N_clasi_empresa",
      #"Clasi_empresa_main",
      #"Codigo_Postal", #-- Lo sacamos
      #"CPV_main",
      #"N_requi",

    )


  ### 📦 Cargar librerías necesarias ----
  library(dplyr)
  library(ggplot2)
  library(patchwork)

  ### 📦 Suponemos que ya existe df_model_v3 con los datos filtrados ----

  ### ☑ 1. Nube de puntos completa (orden menor a mayor) ----

  df_model_v3_ordenado <- df_model_v3 |>
    arrange(Modi_eco_p) |>
    mutate(caso_ordenado = row_number())
```

```r

pdf("v3_Casos_ordinal_vs_Modi_p_g.pdf", width = 10, height = 6)
ggplot(df_model_v3_ordenado, aes(x = caso_ordenado, y = Modi_eco_p)) +
  geom_point(color = "steelblue", size = 0.3, alpha = 0.6) +
  scale_x_continuous(breaks = seq(0, max(df_model_v3_ordenado$caso_ordenado), by = 250)) +
  scale_y_continuous(breaks = seq(0, ceiling(max(df_model_v3_ordenado$Modi_eco_p)), by = 5)) +
  labs(
    title = "Distribución ordinal de los casos según Modi_eco_p",
    x = "Casos ordenados (menor a mayor Modi_eco_p)",
    y = "Modi_eco_p (%)"
  ) +
  theme_minimal(base_size = 12) +
  theme(
    plot.title = element_text(hjust = 0.5),
    axis.text.x = element_text(angle = 45, size = 8)
  )
dev.off()


### 🕰️ 2. Gráfico de densidad con líneas de referencia ----

pdf("v3_Densidad_ModiEco_con_lineas_g.pdf", width = 10, height = 6)
ggplot(df_model_v3, aes(x = Modi_eco_p)) +
  geom_density(fill = "skyblue", alpha = 0.5) +
  geom_vline(xintercept = c(10, 15, 20, 50),
             color = "red", linetype = "dashed") +
  labs(
    title = "Densidad de Modi_eco_p con líneas guía",
    x = "Modi_eco_p (%)",
    y = "Densidad"
  ) +
  theme_minimal(base_size = 12)
dev.off()


### 🔍 3. Zooms localizados en torno a 10, 15, 20 y 50 ----

# Función para graficar un zoom individual
graficar_zoom <- function(df, valor_central) {
  rango <- c(valor_central - 1.5, valor_central + 1)

  df_zoom <- df |>
    filter(Modi_eco_p >= rango[1], Modi_eco_p <= rango[2])

  ggplot(df_zoom, aes(x = caso_ordenado, y = Modi_eco_p)) +
    geom_point(size = 0.2, alpha = 0.6, color = "steelblue") +
    scale_y_continuous(breaks = seq(rango[1], rango[2], by = 0.1),
                       limits = rango) +
    scale_x_continuous(
      breaks = scales::pretty_breaks(n = 6)
    ) +
    labs(
      title = paste("Zoom entorno al", valor_central, "%"),
      x = "Casos ordenados",
      y = "Modi_eco_p"
    ) +
    theme_minimal(base_size = 10) +
    theme(plot.title = element_text(hjust = 0.5))
}

# Crear PDF, una página por gráfico
pdf("v3_Zooms_ModiEco_detalle_g.pdf", width = 9, height = 6)
print(graficar_zoom(df_model_v3_ordenado, 10))
print(graficar_zoom(df_model_v3_ordenado, 15))
print(graficar_zoom(df_model_v3_ordenado, 20))
print(graficar_zoom(df_model_v3_ordenado, 50))
dev.off()



### 🔁 4. Derivada discreta (ΔModi_eco_p) ----
```

```r

df_model_v3_ordenado <- df_model_v3_ordenado |>
  mutate(delta_modi = c(NA, diff(Modi_eco_p)))

pdf("v3_Delta_ModiEco_ordenado_g.pdf", width = 10, height = 6)
ggplot(df_model_v3_ordenado, aes(x = caso_ordenado, y = delta_modi)) +
  geom_line(color = "darkorange", linewidth = 0.4) +
  geom_hline(yintercept = 0, linetype = "dotted") +
  labs(
    title = "Variación local (Δ Modi_eco_p)",
    y = "Cambio respecto al anterior",
    x = "Casos ordenados"
  ) +
  theme_minimal(base_size = 12)
dev.off()


## Directorio guardar ----

### CREAMOS DATA V4 ELIMINANDO PICOS ---------
# Crear los tramos a eliminar (intervalos cerrados)
tramos_eliminar <- list(
  c(9.8, 10.1),
  c(14.8, 15.1),
  c(19.8, 20.1),
  c(49.8, 50.1)
)

# Función para verificar si está dentro de alguno de los intervalos
esta_en_tramos <- function(valor) {
  any(sapply(tramos_eliminar, function(rango) valor >= rango[1] && valor <= rango[2]))
}

# Aplicar filtro al dataset original ordenado
df_model_v4 <- df_model_v3_ordenado |>
  filter(!sapply(Modi_eco_p, esta_en_tramos))

# Confirmación por consola
cat("Número de casos eliminados:", nrow(df_model_v3_ordenado) - nrow(df_model_v4),
"\n")
cat("Número de casos restantes en df_model_v4:", nrow(df_model_v4), "\n")

## VISUALIZACION DF_MODEL_V4

### 📦 Suponemos que ya existe df_model_v3 con los datos filtrados ----

### ☑ 1. Nube de puntos completa (orden menor a mayor) ----


pdf("v4_Casos_ordinal_vs_Modi_p_g.pdf", width = 10, height = 6)
ggplot(df_model_v4, aes(x = caso_ordenado, y = Modi_eco_p)) +
  geom_point(color = "steelblue", size = 0.3, alpha = 0.6) +
  scale_x_continuous(breaks = seq(0, max(df_model_v4$caso_ordenado), by = 250)) +
  scale_y_continuous(breaks = seq(0, ceiling(max(df_model_v4$Modi_eco_p)), by = 5)) +
  labs(
    title = "v4_Distribución ordinal de los casos según Modi_eco_p",
    x = "Casos ordenados (menor a mayor Modi_eco_p)",
    y = "Modi_eco_p (%)"
  ) +
  theme_minimal(base_size = 12) +
  theme(
    plot.title = element_text(hjust = 0.5),
    axis.text.x = element_text(angle = 45, size = 8)
  )
dev.off()


### 🕰 2. Gráfico de densidad con líneas de referencia ----

pdf("v4_Densidad_ModiEco_con_lineas_g.pdf", width = 10, height = 6)
ggplot(df_model_v4, aes(x = Modi_eco_p)) +
  geom_density(fill = "skyblue", alpha = 0.5) +
  geom_vline(xintercept = c(10, 15, 20, 50),
```

```r
               color = "red", linetype = "dashed") +
    labs(
      title = "v4_Densidad de Modi_eco_p con líneas guía",
      x = "Modi_eco_p (%)",
      y = "Densidad"
    ) +
    theme_minimal(base_size = 12)
  dev.off()


  ### 🔍 3. Zooms localizados en torno a 10, 15, 20 y 50 ----

  # Función para graficar un zoom individual
  graficar_zoom <- function(df, valor_central) {
    rango <- c(valor_central - 1.5, valor_central + 1)

    df_zoom <- df |>
      filter(Modi_eco_p >= rango[1], Modi_eco_p <= rango[2])

    ggplot(df_zoom, aes(x = caso_ordenado, y = Modi_eco_p)) +
      geom_point(size = 0.2, alpha = 0.6, color = "steelblue") +
      scale_y_continuous(breaks = seq(rango[1], rango[2], by = 0.1),
                         limits = rango) +
      scale_x_continuous(
        breaks = scales::pretty_breaks(n = 6)
      ) +
      labs(
        title = paste("v4_Zoom entorno al", valor_central, "%"),
        x = "Casos ordenados",
        y = "Modi_eco_p"
      ) +
      theme_minimal(base_size = 10) +
      theme(plot.title = element_text(hjust = 0.5))
  }

  # Crear PDF, una página por gráfico
  pdf("v4_Zooms_ModiEco_detalle_g.pdf", width = 9, height = 6)
  print(graficar_zoom(df_model_v4, 10))
  print(graficar_zoom(df_model_v4, 15))
  print(graficar_zoom(df_model_v4, 20))
  print(graficar_zoom(df_model_v4, 50))
  dev.off()



  ### 🔁 4. Derivada discreta (ΔModi_eco_p) ----


  pdf("v4_Delta_ModiEco_ordenado_g.pdf", width = 10, height = 6)
  ggplot(df_model_v4, aes(x = caso_ordenado, y = delta_modi)) +
    geom_line(color = "darkorange", linewidth = 0.4) +
    geom_hline(yintercept = 0, linetype = "dotted") +
    labs(
      title = "v4_Variación local (Δ Modi_eco_p)",
      y = "Cambio respecto al anterior",
      x = "Casos ordenados"
    ) +
    theme_minimal(base_size = 12)
  dev.off()

  ## Eliminamos variables auxiniales
  df_model_v4 <-df_model_v4 |>
    dplyr::select( - c(caso_ordenado,delta_modi))



  # === División reproducible de los datos en entrenamiento y test === -----

  set.seed(321)

  n <- nrow(df_model_v4)
  train_idx <- sample.int(n, size = floor(0.8 * n))  # Selección aleatoria reproducible
```

```r
# Crear conjuntos de entrenamiento y test
df_train_v4 <- df_model_v4[train_idx, ]
df_test_v4  <- df_model_v4[-train_idx, ]

# Comprobamos que la variable objetivo está presente
stopifnot("Modi_eco_p" %in% names(df_model_v4))

# Guardamos los conjuntos y nombres de variables predictoras
saveRDS(list(
  df_train = df_train_v4,
  df_test = df_test_v4,
  vars_predictoras = setdiff(colnames(df_model_v4), "Modi_eco_p")
), "datos_base_SOM_v4_g.rds")



save(df_train_v4, file="df_train_SOM_v4_g.RData")
write.xlsx(df_train_v4, "df_train_SOM_v4_g.xlsx", colNames = TRUE, rowNames = FALSE)
save(df_test_v4, file="df_test_SOM_v4_g.RData")
write.xlsx(df_test_v4, "df_test_SOM_v4_g.xlsx", colNames = TRUE, rowNames = FALSE)




#### TRABAJAMOS SOBRE LA CURVA DE DENSIDAD --------
# Crear subconjunto del conjunto de entrenamiento con Modi_eco_p ≤ 40
distrib_train_40 <- df_train_v4 |>
  filter(Modi_eco_p <= 40)

# Crear subconjunto del conjunto de test con Modi_eco_p ≤ 40
distrib_test_40 <- df_test_v4 |>
  filter(Modi_eco_p <= 40)


# Confirmar cuántos casos hay
cat("Número de casos en distrib_train_40:", nrow(distrib_train_40), "\n")

# Creamos data frame de casos eliminados en los picos, en el train, menores de 40

# Recuperar los casos que fueron eliminados al pasar de v3 a v4
distrib_elimi <- df_model_v3_ordenado |>
  filter(sapply(Modi_eco_p, esta_en_tramos)) |>
  dplyr::select(-caso_ordenado,-delta_modi)

# Confirmación
cat("Casos eliminados (picos 9.8–10.1, 14.8–15.1, etc.):", nrow(distrib_elimi), "\n")

# Crear subconjunto del conjunto de eliminados con Modi_eco_p ≤ 40
distrib_elimi_40 <- distrib_elimi |>
  filter(Modi_eco_p <= 40)


save(distrib_train_40, file="distrib_train_40_v4_g.RData")
write.xlsx(distrib_train_40, "distrib_train_40_v4_g.xlsx", colNames = TRUE, rowNames =
 FALSE)
save(distrib_test_40, file="distrib_test_40_v4_g.RData")
write.xlsx(distrib_test_40, "distrib_test_40_v4_g.xlsx", colNames = TRUE, rowNames =
FALSE)
save(distrib_elimi, file="distrib_elimi40_v4_g.RData")
write.xlsx(distrib_elimi, "distrib_elimi40_v4_g.xlsx", colNames = TRUE, rowNames =
FALSE)



#### FIT de la DISTRIBUCION A OTRAS #### --------
# Librerías necesarias
library(fitdistrplus)
library(ggplot2)
library(gridExtra)

# === 1. Datos a ajustar ===
x <- distrib_test_40$Modi_eco_p

```

```r
# === 2. Ajuste de distribuciones ===
fit_weibull <- fitdist(x, "weibull")
fit_gamma   <- fitdist(x, "gamma")
fit_lnorm   <- fitdist(x, "lnorm")
fit_norm    <- fitdist(x, "norm")

# Rayleigh como caso particular de Weibull con shape = 2
fit_rayleigh <- fitdist(
  data = x,
  distr = "weibull",
  fix.arg = list(shape = 2),
  start = list(scale = 10)
)

# Agrupamos todos
ajustes <- list(
  Rayleigh = fit_rayleigh,
  Weibull  = fit_weibull,
  Gamma    = fit_gamma,
  Lognorm  = fit_lnorm,
  Normal   = fit_norm
)

# === 3. Comparativa de ajustes ===
gof <- gofstat(ajustes)
print(gof)

# === 4. Comparativa de densidad con curva suavizada ===

# Secuencia de valores para evaluar densidades teóricas
x_vals <- seq(min(x), max(x), length.out = 500)

# Crear data frame con densidades teóricas
densidades <- data.frame(
  x = x_vals,
  Rayleigh = VGAM::drayleigh(x_vals, scale = ajustes$Rayleigh$estimate["scale"]),
  Weibull  = dweibull(x_vals, shape = ajustes$Weibull$estimate["shape"], scale = ajustes$Weibull$estimate["scale"]),
  Gamma    = dgamma(x_vals, shape = ajustes$Gamma$estimate["shape"], rate = ajustes$Gamma$estimate["rate"]),
  Lognorm  = dlnorm(x_vals, meanlog = ajustes$Lognorm$estimate["meanlog"], sdlog = ajustes$Lognorm$estimate["sdlog"]),
  Normal   = dnorm(x_vals, mean = ajustes$Normal$estimate["mean"], sd = ajustes$Normal$estimate["sd"])
)

# Convertimos a formato largo
library(tidyr)
densidades_long <- densidades |>
  pivot_longer(-x, names_to = "Distribucion", values_to = "Densidad")

# Gráfico final
pdf("v4_Distribuciones_ajustadas_con_curva_g.pdf", width = 10, height = 6)

ggplot() +
  geom_density(data = distrib_test_40, aes(x = Modi_eco_p),
               fill = "gray80", alpha = 0.5, linewidth = 0.3) +
  geom_line(data = densidades_long,
            aes(x = x, y = Densidad, color = Distribucion),
            linewidth = 1.1) +
  scale_color_brewer(palette = "Dark2") +
  labs(
    title = "Comparación: curva de datos vs ajustes teóricos",
    x = "Modi_eco_p (%)",
    y = "Densidad"
  ) +
  theme_minimal(base_size = 12) +
  theme(
    legend.title = element_blank(),
    plot.title = element_text(hjust = 0.5)
  )

dev.off()
```

```r


# === 5. Q-Q plots seguros ===
pdf("v4_QQplots_Distribuciones_ajustadas_g.pdf", width = 10, height = 8)

par(mfrow = c(2, 3))  # 2 filas, 3 columnas

for (i in seq_along(ajustes)) {
  qqcomp(
    list(ajustes[[i]]),  # lista con un solo ajuste
    main = paste("Q-Q plot:", names(ajustes)[i]),
    line01 = TRUE
  )
}

par(mfrow = c(1, 1))  # restaurar parámetros gráficos
dev.off()




par(mfrow = c(1, 1))  # restaurar
dev.off()

# === 1. Extraer métricas del ajuste ===
aic_vals <- gof$aic
bic_vals <- gof$bic
logliks  <- sapply(ajustes, function(f) f$loglik)
ks_vals  <- gof$ks
cvm_vals <- gof$cvm
ad_vals  <- gof$ad

# === 2. Construir tabla resumen ===
tabla_resultados <- data.frame(
  Distribucion = names(ajustes),
  AIC      = round(aic_vals, 2),
  BIC      = round(bic_vals, 2),
  LogLik   = round(logliks, 2),
  KS       = round(ks_vals, 4),
  CvM      = round(cvm_vals, 4),
  AD       = round(ad_vals, 4)
)

# === 3. Ordenadas por AIC y BIC ===
tabla_por_aic <- tabla_resultados[order(tabla_resultados$AIC), ]
tabla_por_bic <- tabla_resultados[order(tabla_resultados$BIC), ]

# === 4. Guardar a archivo TXT ===
sink("v4_Resumen_Ajustes_Distribuciones_g.txt")

cat("=== MÉTRICAS DE AJUSTE DE DISTRIBUCIONES ===\n\n")

cat(">> Tabla ordenada por AIC (menor es mejor):\n")
print(tabla_por_aic, row.names = FALSE)

cat("\n>> Tabla ordenada por BIC (menor es mejor):\n")
print(tabla_por_bic, row.names = FALSE)

cat("\n>> Tabla completa sin ordenar:\n")
print(tabla_resultados, row.names = FALSE)

sink()



# === Crear archivo de salida ===
sink("v4_Formulas_Distribuciones_ajustadas_g.txt")

cat("=== FORMULAS DE DISTRIBUCIONES AJUSTADAS ===\n\n")

# === 1. Weibull ===
cat("◇ Weibull Distribution\n")
cat("f(x) = (shape/scale) * (x/scale)^(shape - 1) * exp(-(x/scale)^shape)\n")
```

```r
params <- ajustes$Weibull$estimate
cat(sprintf("→ shape = %.4f\n→ scale = %.4f\n\n", params["shape"], params["scale"]))

# === 2. Gamma ===
cat("◈ Gamma Distribution\n")
cat("f(x) = (1 / (Γ(shape) * scale^shape)) * x^(shape - 1) * exp(-x / scale)\n")
params <- ajustes$Gamma$estimate
cat(sprintf("→ shape = %.4f\n→ scale = %.4f\n\n", params["shape"], params["scale"]))

# === 3. Lognormal ===
cat("◈ Lognormal Distribution\n")
cat("f(x) = (1 / (x * sdlog * sqrt(2π))) * exp(-(log(x) - meanlog)^2 / (2 *
sdlog^2))\n")
params <- ajustes$Lognorm$estimate
cat(sprintf("→ meanlog = %.4f\n→ sdlog   = %.4f\n\n", params["meanlog"], params[
"sdlog"]))

# === 4. Normal ===
cat("◈ Normal Distribution\n")
cat("f(x) = (1 / (sd * sqrt(2π))) * exp(-(x - mean)^2 / (2 * sd^2))\n")
params <- ajustes$Normal$estimate
cat(sprintf("→ mean = %.4f\n→ sd   = %.4f\n\n", params["mean"], params["sd"]))

# === 5. Rayleigh ===
cat("◈ Rayleigh Distribution (Weibull con shape = 2)\n")
cat("f(x) = (x / scale^2) * exp(-(x^2) / (2 * scale^2))\n")
params <- ajustes$Rayleigh$estimate
cat(sprintf("→ shape (fijo) = 2.0000\n→ scale        = %.4f\n\n", params["scale"]))

sink()

### AJUSTE FINO WEIBULL ### -------------------------

# a) Ajuste vía optimización directa personalizada (con optim())
# 1. Función de log-verosimilitud negativa para Weibull
loglik_weibull <- function(params, data) {
  shape <- params[1]
  scale <- params[2]

  if (shape <= 0 || scale <= 0) return(Inf)

  -sum(dweibull(data, shape = shape, scale = scale, log = TRUE))
}

# 2. Ejecutar optimización
resultado_opt <- optim(
  par = c(shape = 1, scale = 10),
  fn = loglik_weibull,
  data = x,
  method = "L-BFGS-B",
  lower = c(0.01, 0.01)
)

# 3. Extraer parámetros óptimos
parametros_optimos <- resultado_opt$par
parametros_optimos

# b) Ajuste por minimización de la distancia de Kolmogorov-Smirnov (KS)

ks_distancia <- function(params, data) {
  shape <- params[1]
  scale <- params[2]

  if (shape <= 0 || scale <= 0) return(Inf)

  ks <- suppressWarnings(ks.test(data, "pweibull", shape = shape, scale = scale))
  ks$statistic
}

res_ks <- optim(
  par = c(1, 10),
  fn = ks_distancia,
  data = x,
```

```r
  method = "L-BFGS-B",
  lower = c(0.01, 0.01)
)

res_ks$par

# c) Ajuste bayesiano de la Weibull con rstan o brms

# Usando brms (más sencillo que rstan)
library(brms)

# Modelar Weibull bayesiano
modelo_bayes <- brm(
  bf(Modi_eco_p ~ 1),
  data = distrib_test_40,
  family = weibull(),
  chains = 4, iter = 2000, cores = 4
)

summary(modelo_bayes)


### 📊 COMPARACIÓN FINAL DE LOS AJUSTES FINOS DE WEIBULL ###


# 1. Recolectar los ajustes
weibull_mle   <- ajustes$Weibull$estimate
weibull_optim <- parametros_optimos
weibull_ks    <- res_ks$par
names(weibull_ks)<-c("shape","scale")

# 2. Obtener shape y scale del ajuste bayesiano
shape_bayes <- posterior_summary(modelo_bayes, variable = "shape")["shape", "Estimate"
]
scale_bayes <- exp(fixef(modelo_bayes)["Intercept", "Estimate"])
weibull_bayes <- c(shape = shape_bayes, scale = scale_bayes)

# 3. Tabla resumen con verificación
df_compara <- data.frame(
  Metodo = c("Inicial (MLE)", "Optimización LogLik", "Minimización KS", "Bayesiano"),
  Shape  = c(
    weibull_mle["shape"],
    weibull_optim["shape"],
    weibull_ks["shape"],
    weibull_bayes["shape"]
  ),
  Scale = c(
    weibull_mle["scale"],
    weibull_optim["scale"],
    weibull_ks["scale"],
    weibull_bayes["scale"]
  )
)

# 4. Guardar la tabla como .TXT
write.table(
  df_compara,
  "v4_Ajustes_Weibull_Finos_g.txt",
  sep = "\t", row.names = FALSE, quote = FALSE
)

# 5. Filtrar métodos con parámetros válidos
metodos_validos <- complete.cases(df_compara[, c("Shape", "Scale")])
df_compara_valid <- df_compara[metodos_validos, ]

# 6. Calcular las curvas solo para métodos válidos
x_vals <- seq(min(x), max(x), length.out = 500)

densidades_df <- data.frame(x = x_vals)

for (i in seq_len(nrow(df_compara_valid))) {
  metodo <- df_compara_valid$Metodo[i]
  shape  <- df_compara_valid$Shape[i]
```

```r
  scale  <- df_compara_valid$Scale[i]

  densidades_df[[metodo]] <- dweibull(x_vals, shape = shape, scale = scale)
}

# 7. Convertir a formato largo
densidades_long <- densidades_df |>
  pivot_longer(-x, names_to = "Metodo", values_to = "Densidad")

# 8. Crear gráfico PDF
pdf("v4_Comparacion_Ajustes_Finos_Weibull_g.pdf", width = 10, height = 6)
ggplot() +
  geom_density(
    data = distrib_test_40,
    aes(x = Modi_eco_p),
    fill = "gray80", alpha = 0.5, linewidth = 0.3
  ) +
  geom_line(
    data = densidades_long,
    aes(x = x, y = Densidad, color = Metodo),
    linewidth = 1.1
  ) +
  scale_color_brewer(palette = "Set1") +
  labs(
    title = "Comparación de ajustes finos de Weibull",
    x = "Modi_eco_p (%)",
    y = "Densidad"
  ) +
  theme_minimal(base_size = 12) +
  theme(
    legend.title = element_blank(),
    plot.title = element_text(hjust = 0.5)
  )
dev.off()


### 📊 TABLA FINAL DE COMPARACIÓN DE AJUSTES FINOS WEIBULL ###

# --- 1. Cálculo de métricas para cada ajuste ---

# Datos de entrada
x_vals_eval <- distrib_test_40$Modi_eco_p

# MLE (ya calculado)
loglik_mle <- ajustes$Weibull$loglik
ks_mle <- ks.test(x_vals_eval, "pweibull",
                  shape = ajustes$Weibull$estimate["shape"],
                  scale = ajustes$Weibull$estimate["scale"])$statistic
aic_mle <- AIC(ajustes$Weibull)
bic_mle <- BIC(ajustes$Weibull)

# Optimización loglik
loglik_optim <- -resultado_opt$value
ks_optim <- ks.test(x_vals_eval, "pweibull",
                    shape = resultado_opt$par["shape"],
                    scale = resultado_opt$par["scale"])$statistic
aic_optim <- -2 * loglik_optim + 2 * 2
bic_optim <- -2 * loglik_optim + log(length(x_vals_eval)) * 2

# Minimización KS
ks_ks <- ks.test(x_vals_eval, "pweibull",
                 shape = weibull_ks["shape"],
                 scale = weibull_ks["scale"])$statistic
loglik_ks <- sum(dweibull(x_vals_eval, shape = weibull_ks["shape"], scale = weibull_ks["scale"], log = TRUE))
aic_ks <- -2 * loglik_ks + 2 * 2
bic_ks <- -2 * loglik_ks + log(length(x_vals_eval)) * 2

# Bayesiano
loglik_bayes <- log_lik(modelo_bayes)
mean_loglik_bayes <- sum(colMeans(loglik_bayes))
ks_bayes <- ks.test(
  x_vals_eval,
```

```r
  "pweibull",
  shape = posterior_summary(modelo_bayes, variable = "shape")[1],
  scale = exp(fixef(modelo_bayes)["Intercept", "Estimate"])
)$statistic
# AIC/BIC bayesianos no aplicables directamente

# --- 2. Construcción de tabla resumen ---

tabla_final <- data.frame(
  Metodo = c("Inicial (MLE)", "Optimización LogLik", "Minimización KS", "Bayesiano"),
  LogLik = round(c(loglik_mle, loglik_optim, loglik_ks, mean_loglik_bayes), 2),
  KS     = round(c(ks_mle, ks_optim, ks_ks, ks_bayes), 4),
  AIC    = round(c(aic_mle, aic_optim, aic_ks, NA), 2),
  BIC    = round(c(bic_mle, bic_optim, bic_ks, NA), 2)
)

# Nota: El warning de 'ties should not be present' en ks.test()
# es esperado ya que los datos tienen valores repetidos.
# Como sólo usamos el estadístico KS (D) para comparar ajustes sobre los mismos datos,
# la comparación sigue siendo válida.
# pero la p-valor (si la calcularas) no sería confiable
# porque el procedimiento asume una distribución continua estricta.

# --- 3. Guardar como .txt ---
sink("v4_Comparativa_Metricas_Ajustes_Weibull_g.txt")
cat("=== COMPARACIÓN DE MÉTRICAS ENTRE AJUSTES FINOS DE WEIBULL ===\n\n")
print(tabla_final, row.names = FALSE)
sink()


## Limpiamos
rm(list=ls())


# ------------------------
# SCRIPT 8 SOM_v5 -------
# ------------------------



## Directorio GUARDAR ----

setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/_final_conjunto_v4")



# === 1.1 Cargar datos base ===
datos_base <- readRDS("datos_base_SOM_v4_g.rds")

df_train <- datos_base$df_train
df_test  <- datos_base$df_test
vars     <- datos_base$vars_predictoras


# === 1.2. Filtro extra datos base modelos ===

### CREAMOS DATA V5 ELIMINANDO PICOS (Aumentamos el filtro de 50)
# Crear los tramos a eliminar (intervalos cerrados)
tramos_eliminar <- list(
  c(9.8, 10.1),
  c(14.8, 15.1),
  c(19.8, 20.1),
  c(47.5, 50.5)
)

# Función para verificar si está dentro de alguno de los intervalos
esta_en_tramos <- function(valor) {
  any(sapply(tramos_eliminar, function(rango) valor >= rango[1] && valor <= rango[2]))
}

# Aplicar filtro al dataset original train
df_train_v5 <- df_train |>
```

```r
  filter(!sapply(Modi_eco_p, esta_en_tramos))

# Aplicar filtro al dataset original test
df_test_v5 <- df_test |>
  filter(!sapply(Modi_eco_p, esta_en_tramos))

#Seleccionamos casos eliminados

# Aplicar filtro al dataset original train
df_elim_train_v5 <- df_train |>
  filter(sapply(Modi_eco_p, esta_en_tramos))
df_elim_test_v5 <- df_test |>
  filter(sapply(Modi_eco_p, esta_en_tramos))
# Unimos
df_elim_v5<-rbind(df_elim_train_v5,df_elim_test_v5)


# Confirmación por consola
cat("Número de casos eliminados entrenamiento (extra pico 50):", nrow(df_train) - nrow(df_train_v5), "\n")
cat("Número de casos restantes en df_train_v5:", nrow(df_train_v5), "\n")
cat("Número de casos restantes en df_test_v5:", nrow(df_test_v5), "\n")

# === 1.3. Guardamos datos base de entrenamiento y test v5 === #

saveRDS(list(
  df_train = df_train_v5,
  df_test = df_test_v5,
  vars_predictoras = setdiff(colnames(df_train_v5), "Modi_eco_p")
), "datos_base_SOM_v5_g.rds")


save(df_train_v5, file="df_train_SOM_v5_g.RData")
write.xlsx(df_train_v5, "df_train_SOM_v5_g.xlsx", colNames = TRUE, rowNames = FALSE)
save(df_test_v5, file="df_test_SOM_v5_g.RData")
write.xlsx(df_test_v5, "df_test_SOM_v5_g.xlsx", colNames = TRUE, rowNames = FALSE)
save(df_elim_v5, file="df_elim_v5_g.RData")
write.xlsx(df_elim_v5, "df_elim_v5_g.xlsx", colNames = TRUE, rowNames = FALSE)



# ==== Unir archivos de eliminados por filas y guardar ====

suppressPackageStartupMessages({ library(readxl); library(dplyr); library(stringr) })


# Rutas reales (busca primero en _final_conjunto_v2)
f1 <- ("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/_final_conjunto_v4/distrib_elimi40_v4_g.xlsx")
f2 <- ("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP/_final_conjunto_v4/df_elim_v5_g.xlsx")

# Lectura (primera hoja por defecto). Añadimos columna de procedencia para trazabilidad
x1 <- readxl::read_excel(f1, guess_max = 100000) |> mutate(Fuente = basename(f1))
x2 <- readxl::read_excel(f2, guess_max = 100000) |> mutate(Fuente = basename(f2))

# Unir por filas; dplyr alineará columnas distintas rellenando con NA donde falten
casos_eliminados_picos <- dplyr::bind_rows(x1, x2)
casos_eliminados_picos <- casos_eliminados_picos [,-17] #Quitamos el df de fuente

# Recolocar IDs si existen (Identificador / Identificador_lote) al principio
if (exists("relocate_id_cols_first")) {
  casos_eliminados_picos <- relocate_id_cols_first(casos_eliminados_picos)
}

#Ordenamos por Modi_eco_p
# Ascendente (NAs al final)
casos_eliminados_picos <- casos_eliminados_picos %>%
  dplyr::arrange(is.na(Modi_eco_p), Modi_eco_p)

# Guardado casos eliminados simple:
write.xlsx(casos_eliminados_picos, "casos_eliminados_picos_g.xlsx", colNames = TRUE,
```

```
    rowNames = FALSE)
4227 save(casos_eliminados_picos, file="casos_eliminados_picos_g.RData")
4228
4229
4230
4231 # ==== Cruce: casos_eliminado_picos □ bd_civil_sin_NA (por Identificador ) ====
4232
4233 suppressPackageStartupMessages({ library(dplyr); library(stringr) })
4234
4235 ## Cargamos datos ----------
4236
4237 load("bd_civil_sin_NA_v2.RData")
4238
4239 # ==== Cruce: casos_eliminado(s)_picos □ bd_civil_sin_NA (por Identificador +
    Identificador_lote) ====
4240
4241 suppressPackageStartupMessages({ library(dplyr); library(stringr) })
4242
4243
4244 # 1) Definir claves y validar presencia
4245 req_keys <- c("Identificador")
4246
4247 if (!all(req_keys %in% names(casos_eliminados_picos))) {
4248   stop("Faltan claves en el df de picos: ", paste(setdiff(req_keys, names(
      casos_eliminados_picos)), collapse = ", "))
4249 }
4250 if (!exists("bd_civil_sin_NA")) stop("'bd_civil_sin_NA' no está cargado en el
    entorno.")
4251
4252 if (!all(req_keys %in% names(bd_civil_sin_NA))) {
4253   stop("Faltan claves en 'bd_civil_sin_NA': ", paste(setdiff(req_keys, names(
      bd_civil_sin_NA)), collapse = ", "))
4254 }
4255
4256
4257 # 2) Renombrar TODAS las columnas de bd_civil_sin_NA salvo las claves con sufijo _c
4258 to_suffix <- setdiff(names(bd_civil_sin_NA), req_keys)
4259 bd_civil_sin_NA_c <- bd_civil_sin_NA %>%
4260   dplyr::rename_with(~ paste0(.x, "_c"), .cols = dplyr::all_of(to_suffix))
4261
4262 # 3) Forzar tipo character en claves para un matching robusto
4263 casos_df2 <- casos_eliminados_picos %>%
4264   mutate(
4265     Identificador      = as.character(Identificador),
4266   )
4267 bd_civil_sin_NA_c2 <- bd_civil_sin_NA_c %>%
4268   mutate(
4269     Identificador      = as.character(Identificador),
4270   )
4271
4272 # 4) LEFT JOIN: primero columnas de casos_df, después las *_c
4273 casos_eliminado_picos_completo <- casos_df2 %>%
4274   dplyr::left_join(bd_civil_sin_NA_c2, by = req_keys)
4275
4276 # 5) Recolocar IDs al principio si tienes el helper
4277 if (exists("relocate_id_cols_first")) {
4278   casos_eliminado_picos_completo <- relocate_id_cols_first(
      casos_eliminado_picos_completo)
4279 }
4280
4281 # 6) Guardar (usa tu register_output si está disponible)
4282 if (exists("register_output")) {
4283   register_output("casos_eliminado_picos_completo", casos_eliminado_picos_completo,
4284                   save = TRUE, formats = c("rdata","xlsx","csv"))
4285 } else {
4286   suppressPackageStartupMessages(library(openxlsx))
4287   out_dir <- if (exists("paths")) paths$processed_dir else getwd()
4288   openxlsx::write.xlsx(casos_eliminado_picos_completo,
4289                        file.path(out_dir, "casos_eliminado_picos_completo.xlsx"),
4290                        colNames = TRUE, rowNames = FALSE)
4291   message("[saved] ", file.path(out_dir, "casos_eliminado_picos_completo.xlsx"))
4292 }
4293
```

```r

# ==== Cruce: df_train_v5 / df_test_v5 □ bd_civil_sin_NA _v2(por Identificador) ====

suppressPackageStartupMessages({ library(dplyr); library(stringr) })

# 0) Cargar base de referencia si no estuviera en memoria
if (!exists("bd_civil_sin_NA")) {
  load("bd_civil_sin_NA_v2.RData")  # debe contener un objeto llamado bd_civil_sin_NA
}
if (!exists("bd_civil_sin_NA")) stop("'bd_civil_sin_NA' no está disponible tras el load().")

# 1) Definir clave y validar
req_keys <- c("Identificador")

# 2) Renombrar TODAS las columnas de bd_civil_sin_NA salvo la clave con sufijo _c (para evitar colisiones)
to_suffix <- setdiff(names(bd_civil_sin_NA), req_keys)
bd_civil_sin_NA_c <- bd_civil_sin_NA %>%
  dplyr::rename_with(~ paste0(.x, "_c"), .cols = dplyr::all_of(to_suffix))

# 3) Helper para cruzar de forma idéntica train/test
.cruzar_por_identificador <- function(df_in, name_out) {
  if (!all(req_keys %in% names(df_in))) {
    stop("Faltan claves en '", deparse(substitute(df_in)), "': ",
         paste(setdiff(req_keys, names(df_in)), collapse = ", "))
  }

  # Forzar tipo character en la clave para evitar mismatches
  df_left <- df_in %>%
    dplyr::mutate(Identificador = as.character(Identificador))
  df_right <- bd_civil_sin_NA_c %>%
    dplyr::mutate(Identificador = as.character(Identificador))

  res <- df_left %>%
    dplyr::left_join(df_right, by = req_keys)

  # Recolocar IDs al principio si tienes el helper
  if (exists("relocate_id_cols_first")) {
    res <- relocate_id_cols_first(res)
  }

  # Guardar
  if (exists("register_output")) {
    register_output(name_out, res, save = TRUE, formats = c("rdata","xlsx","csv"))
  } else {
    # Fallback simple
    suppressPackageStartupMessages(library(openxlsx))
    out_dir <- if (exists("paths")) paths$processed_dir else getwd()
    openxlsx::write.xlsx(res, file.path(out_dir, paste0(name_out, ".xlsx")),
                         colNames = TRUE, rowNames = FALSE)
    message("[saved] ", file.path(out_dir, paste0(name_out, ".xlsx")))
  }

  invisible(res)
}

# 4) Ejecutar para TRAIN y TEST (exige que existan df_train_v5 y df_test_v5)
if (!exists("df_train_v5")) stop("'df_train_v5' no existe en el entorno.")
if (!exists("df_test_v5"))  stop("'df_test_v5' no existe en el entorno.")

df_train_v5_completo <- .cruzar_por_identificador(df_train_v5, "df_train_v5_completo")
df_test_v5_completo  <- .cruzar_por_identificador(df_test_v5,  "df_test_v5_completo")
```