```r
### SCRIPT 8 - R ###
# Tratamiento y limpieza de datos inicial de los requisitos de licitacion empresariales
#
#########################################################
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(devtools)
library(zoo)
library(stringr)
#library(SpeedReader) #paquete de github de Standford, para analisis de texto de alto rendimiento

### Ruta de trabajo ----
setwd("C:/Users/guillermo.alonso/Desktop/Tesis Guillermo/_BD actualizada/Datos atom/PLACSP")

# ============================
# INTEGRACIÓN DE CRITERIOS DE CUALIFICACIÓN (CODICE)
# ============================

# 🔧 Cargar librerías necesarias
library(dplyr)

# 🧹 Inicializamos data frames vacíos donde iremos acumulando los datos
Proy_CCBD <- data.frame()   # Clasificación empresarial exigida (CodeValue)
Proy_RTCBD <- data.frame()  # Requisitos declarativos (RequirementTypeCode)
Proy_TQRBD <- data.frame()  # Nodo general con ID, fecha, contrato

# 🗂️ Rango de años disponibles
anios <- 2012:2025

# 🔁 Iteramos sobre cada año para cargar y acumular los datos
for (anio in anios) {
  cat("Procesando año:", anio, "\n")

  # 📁 Nombre del archivo .RData correspondiente al año
  archivo <- paste0("4.30_TendererQualificationRequestet_", anio, ".Rdata")

  # ⚠️ Carga las variables Proy_CC, Proy_RTC, Proy_TQR
  load(archivo)

  # 📎 Añadimos los datos de este año a los acumuladores
  Proy_CCBD <- bind_rows(Proy_CCBD, Proy_CC)
  Proy_RTCBD <- bind_rows(Proy_RTCBD, Proy_RTC)
  Proy_TQRBD <- bind_rows(Proy_TQRBD, Proy_TQR)
}

# ====================================
# ✨ ¿QUÉ CONTIENE CADA OBJETO?
# ====================================

# Proy_TQRBD → Cada fila es una solicitud de cualificación (`TendererQualificationRequest`)
#              Incluye:
#                - Identificador del expediente
#                - Fecha de actualización (`updated`)
#                - Código del contrato (`ContractID`)
#                - Descripción o nota legal

# Proy_CCBD → Clasificación empresarial exigida (por ejemplo, K-6, O-2)
#             Se puede usar para evaluar barreras de entrada (clasificación necesaria por volumen o especialidad)

# Proy_RTCBD → Tipos de requisitos administrativos estandarizados.
```

```r
#              Codificados según el fichero `DeclarationTypeCode-2.0.gc.xml`, ejemplos:
#               - 1: Capacidad de obrar
#               - 2: No prohibición para contratar
#               - 4: Obligaciones con Hacienda

# ==================================
# 💾 Guardamos las estructuras unificadas
# ==================================

save(Proy_CCBD, file = "TendererQualificationRequest.RData")          # Clasificación
save(Proy_RTCBD, file = "ClassificationCategory.RData")               # Requisitos declarativos
save(Proy_TQRBD, file = "SpecificTendererRequirement.RData")          # Nodo raíz completo

cat("✅ Integración completada. Archivos guardados.\n")

# ============================
# CONSTRUCCIÓN DE BD_placsp_quali - SIN FILTRAR LONGITUD
# ============================

# 📂 Cargar objetos base
load("TendererQualificationRequest.RData")        # Proy_TQRBD
load("ClassificationCategory.RData")              # Proy_CCBD
load("SpecificTendererRequirement.RData")         # Proy_RTCBD

# 🔧 Cargar librerías
library(dplyr)
library(stringr)

# ============================
# 🧹 1. Preparación y renombrado
# ============================

Proy_TQRBD <- Proy_TQRBD |>
  rename(
    TQR_ID = ID,
    Identificador = entryID
  ) |>
  mutate(
    updated = as.Date(updated)
  )

Proy_CCBD <- Proy_CCBD |>
  select(TQR_ID, CodeValue) |>
  filter(!is.na(CodeValue))

Proy_RTCBD <- Proy_RTCBD |>
  select(TQR_ID, RequirementTypeCode) |>
  filter(!is.na(RequirementTypeCode))

# ============================
# 🔗 2. Uniones con identificadores y fechas
# ============================

# Clasificación empresarial
df_cc <- left_join(Proy_TQRBD, Proy_CCBD, by = "TQR_ID") |>
  select(Identificador, CodeValue, updated) |>
  distinct()

# Requisitos tipo declaración
df_rtc <- left_join(Proy_TQRBD, Proy_RTCBD, by = "TQR_ID") |>
  select(Identificador, RequirementTypeCode, updated) |>
  distinct()

# ============================
# 🧠 3. Tratamiento de CodeValue (actualización formato)
# ============================

# a) Transformar códigos antiguos (ej. O2d → O2-4)
df_transformados <- df_cc %>%
  filter(str_detect(CodeValue, "^[A-Z]\\d[a-i]$")) %>%
```

```r
  mutate(
    CodeValue = str_replace(CodeValue, "^([A-Z])(\\d)([a-i])$", "\\1\\2-\\3"),
    CodeValue = chartr("abcdefghi", "123456789", CodeValue)
  )

# b) Conservar los códigos ya en formato correcto
df_resto <- df_cc %>%
  filter(!str_detect(CodeValue, "^[A-Z]\\d[a-i]$"))

# c) Reunir ambos
df_cc <- bind_rows(df_transformados, df_resto)
rm(df_transformados, df_resto)

# ❗ NO se filtra por longitud de CodeValue — se conservan también códigos tipo "K1", "O3-", "A*-3", etc.

# ============================
# 📊 4. Agrupación final por Identificador
# ============================

# A. Clasificación empresarial agrupada
df_cc_grouped <- df_cc |>
  group_by(Identificador) |>
  summarise(
    Clasi_empresa = paste(sort(unique(CodeValue)), collapse = ":"),
    Fecha_actualizacion = max(updated, na.rm = TRUE),
    .groups = "drop"
  )

# B. Requisitos administrativos agrupados
df_rtc_grouped <- df_rtc |>
  group_by(Identificador) |>
  summarise(
    Requi_adm = paste(sort(unique(RequirementTypeCode)), collapse = ":"),
    .groups = "drop"
  )

# ============================
# 🔗 5. Unión final
# ============================

BD_placsp_quali <- left_join(df_cc_grouped, df_rtc_grouped, by = "Identificador")

# ✅ Vista previa
print(head(BD_placsp_quali, 10))


# 🧠 Resultado:
# - Un identificador por fila
# - Clasificación empresarial en formato normalizado y agrupado
# - Requisitos administrativos agrupados
# - Fecha de actualización más reciente por expediente

# ============================
# 🔢 6. Extraer sufijos de CodeValue y asignar Clasi_max_req
# ============================

# 1. Creamos vector de equivalencias sufijo → capacidad económica en €
nivel_economico <- c(
  "1" = 150000,
  "2" = 300000,
  "3" = 450000,
  "4" = 600000,
  "5" = 750000,
  "6" = 1200000,
  "7" = 2400000,
  "8" = 5000000,
  "9" = 5000001
)

# 2. Extraemos todos los sufijos como string desde Clasi_empresa
# Esto genera una lista de strings con todos los "-[n]" encontrados
sufijos_raw <- str_extract_all(BD_placsp_quali$Clasi_empresa, "-[1-9]")
```

```r

# 3. Limpiamos los sufijos para dejar solo los dígitos
# Convertimos la lista a vector numérico del máximo por fila
max_sufijo <- sapply(sufijos_raw, function(x) {
  if (length(x) == 0) return(NA_real_)
  nums <- as.numeric(gsub("-", "", x))
  max(nums, na.rm = TRUE)
})

# 4. Añadimos la nueva variable Clasi_max_req al data frame
BD_placsp_quali <- BD_placsp_quali |>
  mutate(
    Clasi_max_req = nivel_economico[as.character(max_sufijo)]
  )

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


```