MATRIZ\_PERFIL\_TECNOLÓGICO

Ing. Elda Alejandra Torres Reyes & Daniel José Carreño Quintero

2024-08-21

options(repos = c(CRAN = "https://cloud.r-project.org/"))

# Cargar Paquetes Necesarios

* Descripción: Este bloque de código instala y carga las librerías necesarias para realizar operaciones de manipulación y análisis de datos en R.
* Librerías Utilizadas:
  + readxl: Permite leer archivos Excel.
  + writexl: Facilita la escritura de datos en archivos Excel.
  + tidyverse: Conjunto de paquetes para la ciencia de datos que incluye tidyr y dplyr.
  + stringr: Proporciona funciones para manipulación de cadenas de texto.
  + openxlsx: Ofrece herramientas para leer, escribir y modificar archivos Excel.

if (!require(readxl)) install.packages("readxl")  
if (!require(writexl)) install.packages("writexl")  
if (!require(tidyverse)) install.packages("tidyverse")  
if (!require(stringr)) install.packages("stringr")  
if (!require(openxlsx)) install.packages("openxlsx")

Estas líneas instalan paquetes en R que no estén previamente instalados en tu entorno. Los paquetes son colecciones de funciones y datos que expanden las capacidades de R.

library(readxl)  
library(writexl)  
library(tidyr)  
library(dplyr)  
library(stringr)   
library(openxlsx)

Aquí se cargan las librerías instaladas para que estén disponibles en la sesión actual de R. Cada librería tiene funciones específicas:

* readxl y openxlsx manejan archivos Excel.
* writexl permite escribir data frames en archivos Excel.
* tidyverse es una colección de paquetes (como dplyr y tidyr) para manipular
* stringr facilita la manipulación de cadenas de texto.
* openxlsx: Paquete para leer y escribir archivos Excel sin depender de Java.

# ¿Qué es un DataFrame?

Un DataFrame es una estructura de datos en R similar a una tabla en una base de datos o una hoja de cálculo de Excel. Consiste en filas y columnas, donde cada columna puede contener diferentes tipos de datos (números, textos, fechas, etc.). Es la estructura más común para almacenar y manipular datos en R.

## Especifica la ruta del archivo

file.choose()

## [1] "C:\\Users\\danie\\Downloads\\DOCUMENTO\_MATRIZ\_PERFIL\_TECNOLOGICO.docx"

• file.choose(): Abre un cuadro de diálogo para seleccionar el archivo Excel manualmente. (Nota: Esta línea es opcional si ya se conoce la ruta del archivo).

## Especifica la ruta del archivo

ruta\_excel <-"C:\\Users\\danie\\Downloads\\AGRO INNOVATION INTERNATIONAL.xlsx"

ruta\_excel: Variable que almacena la ruta del archivo Excel que se va a leer.

## Leer el archivo Excel

data <- read\_excel(ruta\_excel)

Aquí se lee el archivo Excel y se almacena en el objeto data, que es un DataFrame. El DataFrame data contiene los datos del archivo Excel organizados en filas y columnas.

## Verificar que data es un data frame

if (!is.data.frame(data)) {  
 stop("El objeto data no es un data frame")  
}

Este código verifica si data es un DataFrame. Si no lo es, el código se detiene con un mensaje de error.

## Mostrar los primeros registros para verificar la estructura

print(head(data))

## # A tibble: 6 × 11  
## `Application Id` `Application Number` `Application Date` `Publication Number`  
## <chr> <chr> <chr> <chr>   
## 1 ES153371902 05746681 30.03.2005 2550098   
## 2 BR153881849 PI0919612 15.10.2009 PI0919612   
## 3 PT232814093 117042507 04.01.2011 2521703   
## 4 WO2015150645 PCT/FR2014/050780 01.04.2014 WO/2015/150645   
## 5 RS294599817 20200120 06.03.2015 59953   
## 6 WO2015155476 PCT/FR2015/050917 08.04.2015 WO/2015/155476   
## # ℹ 7 more variables: `Publication Date` <chr>, Country <chr>, Title <chr>,  
## # `I P C` <chr>, Applicants <chr>, Inventors <chr>, `Priorities Data` <chr>

La función head() muestra las primeras filas del DataFrame data, lo que permite verificar su estructura.

## Verificar los nombres de las columnas

print(colnames(data))

## [1] "Application Id" "Application Number" "Application Date"   
## [4] "Publication Number" "Publication Date" "Country"   
## [7] "Title" "I P C" "Applicants"   
## [10] "Inventors" "Priorities Data"

colnames(data) devuelve los nombres de las columnas en el DataFrame data. Esto es útil para confirmar que las columnas esperadas están presentes.

## Asegurarte de que la columna Application Date existe

if (!"Application Date" %in% colnames(data)) {  
 stop("La columna 'Application Date' no existe en el data frame")  
}

Este código verifica que el DataFrame data contiene una columna llamada “Application Date”. Si no es así, se detiene el código.

## Convertir Application Date a fecha

data <- data %>%  
 mutate(Application\_Date = as.Date(`Application Date`, format = "%d.%m.%Y"))

Esta línea convierte la columna Application Date en un formato de fecha (Date) en R, creando una nueva columna Application\_Date en el DataFrame.

# ¿Qué es una Función?

Una función es un bloque de código que realiza una tarea específica y puede ser reutilizado. Las funciones en R pueden tomar entradas (argumentos), realizar cálculos o manipulaciones, y devolver un resultado.

## Función para filtrar por rango de años y contar patentes

filtrar\_y\_contar <- function(data, start\_year, end\_year) {  
 data\_filtrada <- data %>%  
 filter(format(Application\_Date, "%Y") >= start\_year & format(Application\_Date, "%Y") <= end\_year)  
   
 conteo\_por\_ano <- data\_filtrada %>%  
 group\_by(ano = format(Application\_Date, "%Y")) %>%  
 summarize(conteo = n())  
   
 total\_patentes <- nrow(data\_filtrada)  
 rango <- paste(start\_year, end\_year, sep = "-")  
   
 list(data = data\_filtrada, conteo\_por\_ano = conteo\_por\_ano, total\_patentes = total\_patentes, rango = rango)  
}

Esta función toma un DataFrame data, filtra las patentes dentro de un rango de años (start\_year a end\_year), y luego cuenta el número de patentes por año. Devuelve una lista que contiene:

* data\_filtrada: El DataFrame filtrado.
* conteo\_por\_ano: Un DataFrame con el conteo de patentes por año.
* total\_patentes: El número total de patentes en el rango de años.
* rango: El rango de años considerado.

## Filtrar por los rangos especificados

rango\_1 <- filtrar\_y\_contar(data, 2012, 2016)  
rango\_2 <- filtrar\_y\_contar(data, 2013, 2017)  
rango\_3 <- filtrar\_y\_contar(data, 2014, 2018)

Aquí se aplica la función filtrar\_y\_contar para tres rangos de años distintos, guardando los resultados en las variables rango\_1, rango\_2, y rango\_3.

# CÓMO HALLAR LA DISTANCIA TECNOLÓGICA

## Crear un nuevo data frame con los resultados resumidos

df\_rango\_1 <- rango\_1$data  
df\_rango\_2 <- rango\_2$data  
df\_rango\_3 <- rango\_3$data

Estas líneas extraen el DataFrame filtrado (data\_filtrada) de cada lista rango y lo almacenan en nuevas variables.

## Comprobar si la columna “Application Id” existe

if(!"Application Id" %in% names(df\_rango\_1)) {  
 stop("La columna 'Application Id' no existe en df\_rango\_1")  
}  
if(!"Application Id" %in% names(df\_rango\_2)) {  
 stop("La columna 'Application Id' no existe en df\_rango\_2")  
}  
if(!"Application Id" %in% names(df\_rango\_3)) {  
 stop("La columna 'Application Id' no existe en df\_rango\_3")  
}

Este bloque de código asegura que la columna “Application Id” existe en cada uno de los DataFrames (df\_rango\_1, df\_rango\_2, df\_rango\_3).

## Función para separar los códigos IPC en columnas y mantener las otras columnas

separar\_codigos <- function(df) {  
 # Separar los códigos IPC en filas  
 df\_codigos <- df %>%  
 separate\_rows(`I P C`, sep = ";") %>%  
 mutate(`I P C` = str\_trim(`I P C`))  
   
 # Extraer los primeros tres dígitos de cada código IPC  
 df\_codigos <- df\_codigos %>%  
 mutate(Codigo = substr(`I P C`, 1, 3))  
   
 # Combinar los códigos IPC separados en una única columna para cada "Application Id"  
 df\_codigos\_unidos <- df\_codigos %>%  
 group\_by(`Application Id`) %>%  
 summarise(Codigos\_Unicos = paste(unique(Codigo), collapse = ";")) %>%  
 ungroup()  
   
 # Combinar el DataFrame original con los códigos separados  
 df\_final <- df %>%  
 select(-`I P C`) %>%  
 left\_join(df\_codigos\_unidos, by = "Application Id")  
   
 return(df\_final)  
}

**Códigos IPC**: Son un sistema de clasificación internacional para patentes que permite identificar el área tecnológica a la que se refiere una patente. Cada código IPC está compuesto por una serie de dígitos que especifican categorías tecnológicas.

Esta función:

* Separa los códigos IPC: Descompone la columna “I P C” en varias filas según un delimitador (;).
* Extrae los primeros tres dígitos: Solo se mantienen los primeros tres caracteres de cada código IPC.
* Reagrupa los códigos: Une los códigos únicos por “Application Id”.
* Combina con el DataFrame original: Reintegra esta información con las demás columnas del DataFrame original.

## Aplicar la función a los DataFrames

df\_rango\_1\_niveles <- separar\_codigos(df\_rango\_1)  
df\_rango\_2\_niveles <- separar\_codigos(df\_rango\_2)  
df\_rango\_3\_niveles <- separar\_codigos(df\_rango\_3)

Finalmente, se aplica la función separar\_codigos a cada DataFrame generado en los rangos de años, obteniendo nuevos DataFrames con los códigos IPC separados y organizados.

# ¿Qué es una Matriz?

Una matriz es una estructura de datos en R que organiza elementos en dos dimensiones: filas y columnas. A diferencia de un DataFrame, una matriz solo puede contener un tipo de datos (por ejemplo, solo números o solo caracteres). Aunque en este código no se utiliza directamente una matriz, es importante entender la diferencia con un DataFrame, que es más flexible al permitir múltiples tipos de datos.

# Creación de la Matriz de Comparación con Columna de Sumatoria

## Función para crear la matriz de comparación con columna de sumatoria

# Función para crear la matriz de comparación con columna de sumatoria  
crear\_matriz\_comparacion <- function(df\_niveles, codigos\_objetivo) {  
 # Obtener los nombres de las columnas que contienen los códigos IPC  
 columnas\_codigos <- grep("Codigos\_Unicos", names(df\_niveles), value = TRUE)  
   
 # Obtener los Application Ids  
 application\_ids <- df\_niveles$`Application Id`  
   
 # Crear una matriz vacía con las dimensiones adecuadas  
 matriz <- matrix(0, nrow = length(codigos\_objetivo), ncol = nrow(df\_niveles))  
   
 # Rellenar la matriz con 1 si hay coincidencia de códigos IPC  
 for (i in 1:nrow(df\_niveles)) {  
 codigos\_presentes <- str\_split(df\_niveles[i, columnas\_codigos], ";") %>% unlist() %>% na.omit()  
 for (j in 1:length(codigos\_objetivo)) {  
 if (codigos\_objetivo[j] %in% codigos\_presentes) {  
 matriz[j, i] <- 1  
 }  
 }  
 }  
   
 # Convertir la matriz en un data frame  
 df\_matriz <- as.data.frame(matriz)  
   
 # Añadir los nombres de columna (Application Ids)  
 colnames(df\_matriz) <- application\_ids  
   
 # Añadir los códigos como una columna al data frame  
 df\_matriz <- cbind(Codigo = codigos\_objetivo, df\_matriz)  
   
 # Verificar si df\_matriz tiene más de una columna después de excluir la columna "Codigo"  
 if (ncol(df\_matriz) > 2) { # Si tiene más de dos columnas (incluyendo "Codigo" y al menos una columna de datos)  
 # Añadir una columna con la sumatoria por fila  
 df\_matriz$Total <- rowSums(df\_matriz[, -1]) # Excluir la columna "Codigo" en el cálculo de sumas  
 } else {  
 # Si solo hay una columna de datos, la suma de esa fila es simplemente el valor de esa columna  
 df\_matriz$Total <- df\_matriz[, 2]  
 }  
   
 return(df\_matriz)  
}

• Función crear\_matriz\_comparacion: Esta función toma dos argumentos: df\_niveles, que es un DataFrame que contiene información sobre patentes, y codigos\_objetivo, que es un vector con los códigos IPC (International Patent Classification) que se usarán para la comparación.

## Obtener nombres de columnas con códigos IPC

Se usa la función grep para encontrar las columnas que contienen los códigos IPC únicos. grep busca patrones en los nombres de columnas y devuelve los nombres que coinciden.

## Obtener Application Ids:

Extrae los identificadores de las solicitudes de patente (Application Ids) del DataFrame df\_niveles.

## Crear una matriz vacía:

Se crea una matriz de ceros con filas correspondientes a los códigos objetivo y columnas correspondientes a las patentes.

* Matriz: Una estructura de datos bidimensional con filas y columnas, donde cada elemento es numérico en este caso.

## Rellenar la matriz con coincidencias:

* Bucle exterior: Recorre cada fila del DataFrame, correspondiente a cada patente.
* Bucle interior: Compara cada código objetivo con los códigos presentes en la patente.
* str\_split: Se usa para dividir los códigos IPC en una lista de códigos individuales.
* Asignación: Si un código objetivo está presente en los códigos IPC de una patente, se coloca un 1 en la matriz en la posición correspondiente.

## Convierte la matriz en un DataFrame

Convierte la matriz a un DataFrame para facilitar la manipulación y visualización.

## Añadir nombres de columnas

Asigna los identificadores de solicitud de patente como nombres de las columnas del DataFrame.

## Añadir codigos como columna

Añade una columna con los códigos IPC objetivo al DataFrame.

## Añadir la columna sumatoria

* rowSums: Calcula la suma de valores en cada fila, excluyendo la primera columna que contiene los códigos.
* Sumatoria: Añade una columna Total con la suma de coincidencias por fila, si hay más de una columna de datos.

## Definir los Códigos Objetivo:

codigos\_objetivo <- c(  
 "A01", "A21", "A22", "A23", "A24", "A41", "A42", "A43", "A44",   
 "A45", "A46", "A47", "A61", "A62", "A63", "A99", "B01", "B02", "B03", "B04",   
 "B05", "B06", "B07", "B08", "B09", "B21", "B22", "B23", "B24", "B25", "B26",   
 "B27", "B28", "B29", "B30", "B31", "B32", "B33", "B41", "B42", "B43",   
 "B44", "B60", "B61", "B62", "B63", "B64", "B65", "B66", "B67", "B68", "B81",   
 "B82", "B99", "C01", "C02", "C03", "C04", "C05", "C06", "C07", "C08", "C09",   
 "C10", "C11", "C12", "C13", "C14", "C21", "C22", "C23", "C25", "C30", "C40",   
 "C99", "D01", "D02", "D03", "D04", "D05", "D06", "D07", "D21", "D99", "E01",   
 "E02", "E03", "E04", "E05", "E06", "E21", "E99", "F01", "F02", "F03", "F04",   
 "F15", "F16", "F17", "F21", "F22", "F23", "F24", "F25", "F26", "F27", "F28",   
 "F41", "F42", "F99", "G01", "G02", "G03", "G04", "G05", "G06", "G07", "G08",   
 "G09", "G10", "G11", "G12", "G16", "G21", "G99", "H01", "H02", "H03", "H04",   
 "H05", "H10", "H99")

Define un vector con los códigos de las 132 clases que se van a comparar con los IPC de las patentes. Estos códigos representan clasificaciones específicas en el sistema IPC.

* **Nivel de Tres Dígitos**: En este código, se utilizan los códigos IPC a nivel de tres dígitos para representar el perfil tecnológico de cada patente. Esto proporciona un nivel de detalle suficiente para clasificar patentes en áreas tecnológicas específicas sin ser demasiado granular.

## Perfil Tecnológico

En el contexto de este análisis, un **perfil tecnológico** se refiere a una representación del conjunto de áreas tecnológicas a las que se asocia una patente. Los perfiles tecnológicos están basados en las **132 clases de patentes derivadas del nivel de tres dígitos de los códigos IPC (International Patent Classification).**

## Crear las matrices de comparación

matriz\_rango\_1 <- crear\_matriz\_comparacion(df\_rango\_1\_niveles, codigos\_objetivo)  
matriz\_rango\_2 <- crear\_matriz\_comparacion(df\_rango\_2\_niveles, codigos\_objetivo)  
matriz\_rango\_3 <- crear\_matriz\_comparacion(df\_rango\_3\_niveles, codigos\_objetivo)

Aplica la función crear\_matriz\_comparacion a cada DataFrame (correspondiente a diferentes rangos de años) para generar matrices de comparación que contienen la información sobre las coincidencias de códigos IPC.

# Creación y Guardado del Archivo Excel con Resultados

## Crear un nuevo archivo Excel y agregar hojas con los resultados

wb <- createWorkbook()

* createWorkbook(): Función de la librería openxlsx que crea un nuevo archivo de libro de trabajo de Excel (Workbook) vacío.

## Definir Función para Agregar Hojas con Resultados

agregar\_hoja <- function(wb, rango, nombre\_hoja) {  
 addWorksheet(wb, nombre\_hoja)  
 if (!is.null(rango$data) && nrow(rango$data) > 0) {  
 writeData(wb, nombre\_hoja, rango$data, startCol = 1, startRow = 1)  
 writeData(wb, nombre\_hoja, paste("Total patentes en el rango", rango$rango, ":", rango$total\_patentes), startCol = 1, startRow = nrow(rango$data) + 2)  
 if (!is.null(rango$conteo\_por\_ano) && nrow(rango$conteo\_por\_ano) > 0) {  
 writeData(wb, nombre\_hoja, rango$conteo\_por\_ano, startCol = 1, startRow = nrow(rango$data) + 4)  
 }  
 } else {  
 writeData(wb, nombre\_hoja, "No hay datos disponibles", startCol = 1, startRow = 1)  
 }  
}

* agregar\_hoja: Función que agrega una hoja al libro de trabajo (wb) con un nombre específico (nombre\_hoja). Dentro de la hoja:
  + addWorksheet(wb, nombre\_hoja): Añade una nueva hoja al libro de trabajo con el nombre dado.
  + writeData(wb, nombre\_hoja, rango Escribe los datos del DataFrame rango$data en la hoja, comenzando en la columna 1 y fila 1.
  + paste(“Total patentes en el rango”, rango$rango, ":", rango$total\_patentes): Crea una cadena que muestra el total de patentes en el rango especificado.
  + writeData(wb, nombre\_hoja, …, startCol = 1, startRow = nrow(rango$ Escribe el total de patentes en una fila debajo de los datos.
  + If conteo\_por\_ano) &&nrow(rango$ Verifica si hay datos de conteo por año y los escribe debajo del total de patentes.
  + Writedataconteo\_por\_ano, startCol = 1, startRow = nrow(rango$data) + 4): Escribe el conteo de patentes por año en la hoja.

## Agregar hojas con los datos filtrados

agregar\_hoja(wb, rango\_1, "2012-2016")  
agregar\_hoja(wb, rango\_2, "2013-2017")  
agregar\_hoja(wb, rango\_3, "2014-2018")

* agregar\_hoja(wb, rango\_1, “2012-2016”): Llama a la función agregar\_hoja para agregar una hoja con datos del rango de años 2012-2016 al libro de trabajo.
* agregar\_hoja(wb, rango\_2, “2013-2017”): Agrega una hoja con datos del rango 2013-2017.
* agregar\_hoja(wb, rango\_3, “2014-2018”): Agrega una hoja con datos del rango 2014-2018.

## Añadir cada matriz como una hoja separada

addWorksheet(wb, "Matriz\_Rango\_1")  
writeData(wb, "Matriz\_Rango\_1", matriz\_rango\_1)  
  
addWorksheet(wb, "Matriz\_Rango\_2")  
writeData(wb, "Matriz\_Rango\_2", matriz\_rango\_2)  
  
addWorksheet(wb, "Matriz\_Rango\_3")  
writeData(wb, "Matriz\_Rango\_3", matriz\_rango\_3)

* addWorksheet(wb, “Matriz\_Rango\_1”): Agrega una hoja llamada “Matriz\_Rango\_1” al libro de trabajo.
* writeData(wb, “Matriz\_Rango\_1”, matriz\_rango\_1): Escribe la matriz matriz\_rango\_1 en la hoja “Matriz\_Rango\_1”.
* addWorksheet(wb, “Matriz\_Rango\_2”) y writeData(wb, “Matriz\_Rango\_2”, matriz\_rango\_2): Similar a los anteriores pero para matriz\_rango\_2.
* addWorksheet(wb, “Matriz\_Rango\_3”) y writeData(wb, “Matriz\_Rango\_3”, matriz\_rango\_3): Similar a los anteriores pero para matriz\_rango\_3.

## Guardar el archivo excel

saveWorkbook(wb, "C:\\Users\\danie\\Documents\\EMPRESAS DT\\EMPRESA\_EJEMPLO\\Nueva.xlsx", overwrite = TRUE)

saveWorkbook(wb, “ruta\_del\_archivo”, overwrite = TRUE): Guarda el libro de trabajo (wb) en la ruta especificada. overwrite = TRUE permite sobrescribir el archivo si ya existe.

# Procesamiento y Almacenamiento de Datos de Archivos Excel

## Función para procesar cada archivo Excel para el almacenamiento

procesar\_y\_almacenar\_excel <- function(ruta\_excel, ruta\_almacenamiento) {  
 # Leer el archivo Excel generado  
 data\_matriz\_1 <- read\_excel(ruta\_excel, sheet = "Matriz\_Rango\_1")  
 data\_matriz\_2 <- read\_excel(ruta\_excel, sheet = "Matriz\_Rango\_2")  
 data\_matriz\_3 <- read\_excel(ruta\_excel, sheet = "Matriz\_Rango\_3")  
   
 # Verificar que data\_matriz son data frames  
 if (!is.data.frame(data\_matriz\_1) | !is.data.frame(data\_matriz\_2) | !is.data.frame(data\_matriz\_3)) {  
 stop("Alguno de los objetos data no es un data frame")  
 }  
   
 # Extraer los códigos IPC de las hojas matriz  
 codigos\_ipc\_1 <- data\_matriz\_1$Codigo  
 codigos\_ipc\_2 <- data\_matriz\_2$Codigo  
 codigos\_ipc\_3 <- data\_matriz\_3$Codigo  
   
 # Extraer la columna "Total" de las hojas matriz  
 total\_1 <- data\_matriz\_1$Total  
 total\_2 <- data\_matriz\_2$Total  
 total\_3 <- data\_matriz\_3$Total  
   
 # Crear un nuevo data frame con los códigos IPC como columnas  
 df\_resultado\_1 <- tibble(  
 Empresa = tools::file\_path\_sans\_ext(basename(ruta\_excel)),  
 !!!setNames(as.list(total\_1), codigos\_ipc\_1)  
 )  
   
 df\_resultado\_2 <- tibble(  
 Empresa = tools::file\_path\_sans\_ext(basename(ruta\_excel)),  
 !!!setNames(as.list(total\_2), codigos\_ipc\_2)  
 )  
   
 df\_resultado\_3 <- tibble(  
 Empresa = tools::file\_path\_sans\_ext(basename(ruta\_excel)),  
 !!!setNames(as.list(total\_3), codigos\_ipc\_3)  
 )  
   
 # Leer el contenido actual del archivo de almacenamiento  
 if (file.exists(ruta\_almacenamiento)) {  
 wb\_almacenamiento <- loadWorkbook(ruta\_almacenamiento)  
 } else {  
 # Si el archivo no existe, crearlo y agregar las hojas necesarias  
 wb\_almacenamiento <- createWorkbook()  
 addWorksheet(wb\_almacenamiento, "Almacenamiento\_Rango\_1")  
 addWorksheet(wb\_almacenamiento, "Almacenamiento\_Rango\_2")  
 addWorksheet(wb\_almacenamiento, "Almacenamiento\_Rango\_3")  
 }  
   
 # Escribir los datos en cada hoja (verificando si ya existen)  
 if ("Almacenamiento\_Rango\_1" %in% names(wb\_almacenamiento)) {  
 current\_data\_1 <- read.xlsx(wb\_almacenamiento, sheet = "Almacenamiento\_Rango\_1")  
 writeData(wb\_almacenamiento, "Almacenamiento\_Rango\_1",   
 rbind(current\_data\_1, df\_resultado\_1), colNames = TRUE)  
 }  
   
 if ("Almacenamiento\_Rango\_2" %in% names(wb\_almacenamiento)) {  
 current\_data\_2 <- read.xlsx(wb\_almacenamiento, sheet = "Almacenamiento\_Rango\_2")  
 writeData(wb\_almacenamiento, "Almacenamiento\_Rango\_2",   
 rbind(current\_data\_2, df\_resultado\_2), colNames = TRUE)  
 }  
   
 if ("Almacenamiento\_Rango\_3" %in% names(wb\_almacenamiento)) {  
 current\_data\_3 <- read.xlsx(wb\_almacenamiento, sheet = "Almacenamiento\_Rango\_3")  
 writeData(wb\_almacenamiento, "Almacenamiento\_Rango\_3",   
 rbind(current\_data\_3, df\_resultado\_3), colNames = TRUE)  
 }  
   
 # Guardar el archivo Excel de almacenamiento  
 saveWorkbook(wb\_almacenamiento, ruta\_almacenamiento, overwrite = TRUE)  
}

## Definición de la Función para Procesar y Almacenar Excel

* procesar\_y\_almacenar\_excel: Función que procesa un archivo Excel de entrada y almacena los resultados en otro archivo Excel.
* read\_excel(ruta\_excel, sheet = “Matriz\_Rango\_1”): Lee la hoja “Matriz\_Rango\_1” del archivo Excel en la ruta especificada y almacena los datos en data\_matriz\_1.

## Verificación de Tipos de Datos

* is.data.frame: Función que verifica si un objeto es un DataFrame. Aquí se asegura de que cada objeto leído sea un DataFrame.
* stop(“Mensaje”): Detiene la ejecución de la función y muestra un mensaje de error si alguna de las condiciones no se cumple.

## Extracción de Datos de las Hojas

* $Codigo y $Total: Acceden a las columnas específicas de los DataFrames para extraer los códigos IPC y los totales de patentes.

## Creación de Nuevos DataFrames para Resultados

* tibble: Estructura de datos similar a un DataFrame pero con mejor manejo de datos y visualización en R.
* tools::file\_path\_sans\_ext(basename(ruta\_excel)): Extrae el nombre del archivo sin la extensión para usarlo como nombre de la empresa.
* setNames(as.list(total\_1), codigos\_ipc\_1): Convierte los totales en una lista y les asigna nombres basados en los códigos IPC.

## Lectura y Creación del Archivo de Almacenamiento

* file.exists(ruta\_almacenamiento): Verifica si el archivo de almacenamiento ya existe.
* loadWorkbook(ruta\_almacenamiento): Carga un archivo de libro de trabajo existente.
* createWorkbook(): Crea un nuevo libro de trabajo si el archivo de almacenamiento no existe.
* addWorksheet: Añade nuevas hojas al libro de trabajo para cada rango.

## Escribir Datos en el Archivo de Almacenamiento

* rbind(current\_data\_1, df\_resultado\_1): Combina los datos actuales en la hoja con los nuevos datos.
* writeData(wb\_almacenamiento, “Nombre\_Hoja”, …): Escribe los datos combinados en la hoja correspondiente.

## Guardar el Archivo Excel de Almacenamiento

* saveWorkbook(wb\_almacenamiento, ruta\_almacenamiento, overwrite = TRUE): Guarda el libro de trabajo con los datos actualizados en la ruta especificada, permitiendo sobrescribir el archivo si ya existe.

# Ejecutar la Función y Mostrar Directorio de Trabajo

## Ruta del archivo Excel de almacenamiento

ruta\_almacenamiento <- "C:\\Users\\danie\\Documents\\EMPRESAS DT\\EMPRESA\_EJEMPLO\\ALMACENAMIENTO\_DT.xlsx"  
  
# Procesar y almacenar la información del archivo generado  
procesar\_y\_almacenar\_excel("C:\\Users\\danie\\Documents\\EMPRESAS DT\\EMPRESA\_EJEMPLO\\nueva.xlsx", ruta\_almacenamiento)  
  
# Obtener el directorio de trabajo actual  
print(getwd())

## [1] "C:/Users/danie/Downloads"

* ruta\_almacenamiento: Define la ruta del archivo de almacenamiento.
* procesar\_y\_almacenar\_excel: Llama a la función para procesar el archivo Excel de entrada y almacenar los resultados en el archivo de almacenamiento.
* getwd(): Obtiene y muestra el directorio de trabajo actual.