MATRIX\_TECHNOLOGICAL\_PROFILE

Eng. Elda Alejandra Torres Reyes & Daniel Jose Carreno Quintero

2024-08-21

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

# Load Necessary Packages

* Description: This code block installs and loads the libraries needed to perform data manipulation and analysis operations in R.
* Libraries Used:
  + readxl: Allows you to read Excel files.
  + writexl: Facilitates writing data to Excel files.
  + tidyverse: A suite of packages for data science that includes tidyr and dplyr.
  + stringr: Provides functions for manipulating text strings.
  + openxlsx: Provides tools to read, write and modify Excel files.

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")

These lines install packages into R that are not already installed in your environment. Packages are collections of functions and data that expand the capabilities of R.

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

Here the installed libraries are loaded so that they are available in the current R session. Each library has specific functions:

* readxl and openxlsx handle Excel files.
* writexl allows you to write data frames to Excel files.
* tidyverse is a collection of packages (such as dplyr and tidyr) for manipulating
* stringr makes it easy to manipulate text strings.
* openxlsx: Package to read and write Excel files without depending on Java.

# What is a DataFrame?

A DataFrame is a data structure in R similar to a table in a database or an Excel spreadsheet. It consists of rows and columns, where each column can contain different types of data (numbers, texts, dates, etc.). It is the most common structure for storing and manipulating data in R.

## Specifies the path of the file

file.choose ()

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

• file.choose(): Opens a dialog box to select the Excel file manually. (Note: This line is optional if the file path is already known.)

## Specifies the path of the file

excel\_path <- "C: \\ Users \\ danie \\ Downloads \\ AGRO INNOVATION INTERNATIONAL.xlsx"

excel\_path: Variable that stores the path of the Excel file to be read.

## Read the Excel file

data <- read\_excel (excel\_path)

Here the Excel file is read and stored in the data object, which is a DataFrame. The DataFrame contains the data from the Excel file organized in rows and columns.

## Verify that data is a data frame

if ( ! is.data.frame (data)) {  
 stop ( "The object data is not a data frame" )   
}

This code checks if data is a DataFrame. If it is not, the code stops with an error message.

## Show the first records to verify the structure

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>

The head() function displays the first rows of the DataFrame data, allowing you to verify its structure.

## Verify column names

print ( colnames ( data ))

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

colnames(data) returns the names of the columns in the DataFrame data. This is useful for confirming that the expected columns are present.

## Make sure the Application Date column exists

if ( ! "Application Date" %in% colnames (data)) {  
 stop ( ""The 'Application Date' column does not exist in the data frame")   
}

This code verifies that the DataFrame contains a column named “Application Date”. If not, the code stops.

## Convert Application Date to date format

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

This line converts the Application Date column to a date format in R, creating a new Application\_Date column in the DataFrame.

# What is a Function?

A function is a block of code that performs a specific task and can be reused. Functions in R can take inputs (arguments), perform calculations or manipulations, and return a result.

## Function to filter by year range and count patents

filter\_and\_count <- function(data, start\_year, end\_year) {

filtered\_data <- data %>%

filter(format(Application\_Date, "%Y") >= start\_year & format(Application\_Date, "%Y") <= end\_year)  
 filter(format(Application\_Date, "%Y") >= start\_year & format(Application\_Date, "%Y") <= end\_year)

count\_per\_year <- filtered\_data %>%  
 group\_by(year = format(Application\_Date, "%Y")) %>%  
 summarize(count = n())  
   
 total\_patents <- nrow(filtered\_data)  
 range <- paste(start\_year, end\_year, sep = "-")  
   
 list(data = filtered\_data, count\_per\_year = count\_per\_year, total\_patents = total\_patents, range = range)  
}

This function takes a DataFrame, filters the patents within a range of years (start\_year to end\_year), and then counts the number of patents per year. It returns a list containing:

* data\_filtered: The filtered DataFrame.
* count\_per\_year: A DataFrame with the patent count per year.
* total\_patents: The total number of patents in the year range.
* range: The range of years considered.

## Filter by specified ranges

range\_1 <- filter\_and\_count (data, 2012 , 2016 )   
range\_2 <- filter\_and\_count (data, 2013 , 2017 )   
range\_3 <- filter\_and\_count (data, 2014 , 2018 )

Here the filter\_and\_count function is applied for three different year ranges, saving the results in the variables range\_1, range\_2, and range\_3.

# HOW TO FIND THE TECHNOLOGICAL DISTANCE

## Create a new data frame with the summarized results

df\_range\_1 <- range\_1 $ data   
df\_range\_2 <- range\_2 $ data   
df\_range\_3 <- range\_3 $ data

These lines extract the filtered DataFrame (filtered\_data) from each range list and store it in new variables.

## Check if the “Application Id” column exists

if(!"Application Id" %in% names(df\_range\_1)) {  
 stop("The 'Application Id' column does not exist in df\_range\_1")  
}  
if(!"Application Id" %in% names(df\_range\_2)) {  
 stop("The 'Application Id' column does not exist in df\_range\_2")  
}  
if(!"Application Id" %in% names(df\_range\_3)) {  
 stop("The 'Application Id' column does not exist in df\_range\_3")  
}

This code block ensures that the “Application Id” column exists in each of the DataFrames (df\_range\_1, df\_range\_2, df\_range\_3).

## Function to separate IPC codes into columns and keep the other columns

separate\_codes <- function (df) {  
 # Separate IPC codes into columns   
df\_codes <- df %>%  
 separate\_rows ( ` I P C ` , sep = ";" ) %>%  
 mutate ( ` I P C ` = str\_trim ( ` I P C ` ))  
   
 # Extract the first three digits of each IPC code   
df\_codes <- df\_codes %>%  
 mutate ( Code = substr ( ` I P C ` , 1 , 3 ))  
   
 # Combine separate IPC codes into a single column for each "Application Id"   
df\_codes\_combined <- df\_codes %>%  
 group\_by ( ` Application Id ` ) %>%  
 summarise ( Unique\_Codes = paste ( unique (Code), collapse = ";" )) %>%  
 ungroup ()  
   
 # Combine the original DataFrame with the separated codes   
df\_final <- df %>%  
 select ( - ` I P C ` ) %>%  
 left\_join (df\_codes\_combined, by = "Application Id" )  
   
 return (df\_final)   
}

**IPC codes** : An international patent classification system that identifies the technological area to which a patent refers. Each IPC code is composed of a series of digits that specify technological categories.

This function:

* Split CPI Codes: Breaks up the “CPI” column into multiple rows based on a delimiter (;).
* Extract the first three digits: Only the first three characters of each IPC code are kept.
* Regroup the codes: Join the unique codes by “Application Id”.
* Merge with original DataFrame: Reintegrates this information with the other columns of the original DataFrame.

## Apply the function to the DataFrames

df\_range\_1\_levels <- separate\_codes (df\_range\_1)   
df\_range\_2\_levels <- separate\_codes (df\_range\_2)   
df\_range\_3\_levels <- separate\_codes (df\_range\_3)

Finally, the separate\_codes function is applied to each DataFrame generated in the year ranges, obtaining new DataFrames with the IPC codes separated and organized.

# What is a Matrix?

A matrix is a data structure in R that organizes elements in two dimensions: rows and columns. Unlike a DataFrame, a matrix can only contain one type of data (for example, only numbers or only characters). Although this code does not directly use a matrix, it is important to understand the difference with a DataFrame, which is more flexible in allowing multiple data types.

# Creating the Comparison Matrix with Summation Column

## Function to create the comparison matrix with summation column

# Function to create the comparison matrix with summation column   
create\_comparison\_matrix <- function (df\_levels, target\_codes) {  
 # Get the names of the columns that containing the IPC codes   
codes\_columns <- grep ( "Unique\_Codes" , names (df\_levels), value = TRUE )  
   
 # Get the Application Ids   
application\_ids <- df\_levels $ ` Application Id `  
   
 # Create an empty matrix with the appropriate dimensions   
matrix <- matrix ( 0 , nrow = length (target\_codes), ncol = nrow (df\_levels))  
   
 # Fill the matrix with 1 if there is a match of IPC codes  
 for (i in 1 : nrow (df\_levels)) {   
present\_codes <- str\_split (df\_levels[i, codes\_columns], ";" ) %>% unlist () %>% na.omit ()  
 for (j in 1 : length (target\_codes)) {  
 if (target\_codes[j] %in% present\_codes) {   
matrix[j, i] <- 1   
}   
}   
}  
   
 # Convert the matrix to a data frame   
df\_matrix <- as.data.frame (matrix)  
   
 # Add the column names (Application Ids)  
 colnames (df\_matrix) <- application\_ids  
   
 # Add the codes as a column to the data frame   
df\_matriz <- cbind ( Code = target\_codes, matrix\_df)  
   
 # Check if df\_matrix has more than one column after excluding the "Code" column  
 if ( ncol (df\_matriz) > 2 ) { # If you have more than two columns (including "Code" and at least one data column)  
 # Add a column with the summation per row   
df\_matrix $ Total <- rowSums (df\_matriz[, - 1 ]) # Exclude the "Code" column in the calculation of sums   
} else {  
 # If there is only one data column, the sum of that row is simply the value of that column   
df\_matrix $ Total <- df\_matrix[, 2 ]   
}  
   
 return (df\_matriz)   
}

• create\_comparison\_matrix function: This function takes two arguments: df\_levels, which is a DataFrame containing patent information, and target\_codes, which is a vector containing the IPC (International Patent Classification) codes that will be used for the comparison.

## Get column names with IPC codes

The grep function is used to find columns that contain unique IPC codes. grep looks for patterns in column names and returns names that match.

## Get Application Ids:

Extracts the patent application identifiers (Application Ids) from the DataFrame df\_niveles.

## Create an empty array:

A zero matrix is created with rows corresponding to the target codes and columns corresponding to the patents.

* Matrix: A two-dimensional data structure with rows and columns, where each element is numeric in this case.

## Fill the matrix with matches:

* Outer Loop: Loop through each row of the DataFrame, corresponding to each patent.
* Inner loop: Compares each target code with the codes present in the patent.
* str\_split: Used to split IPC codes into a list of individual codes.
* Assignment: If a target code is present in the IPC codes of a patent, a 1 is placed in the matrix in the corresponding position.

## Converts the array to a DataFrame

Converts the array to a DataFrame for easy manipulation and display.

## Add column names

Assigns the patent application IDs as the column names of the DataFrame.

## Add codes as a column

Add a column with the target IPC codes to the DataFrame.

## Add the summation column

* rowSums: Calculates the sum of values in each row, excluding the first column containing the codes.
* Summation: Adds a Total column with the sum of matches per row, if there is more than one column of data.

## Define Target Codes:

Target\_codes<- 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")

Defines a vector containing the codes of the 132 classes to be compared with the patent CPIs. These codes represent specific classifications in the CPI system.

* **Three-Digit Level** : In this code, IPC codes are used at the three-digit level to represent the technological profile of each patent. This provides a sufficient level of detail to classify patents into specific technological areas without being too granular.

## Technological Profile

In the context of this analysis, a **technology profile** refers to a representation of the set of technological areas to which a patent is associated. Technology profiles are based on the **132 patent classes derived from the three-digit level of the IPC (International Patent Classification) codes .**

## Create the comparison matrices

matrix\_range\_1 <- create\_comparison\_matrix (df\_range\_1\_levels, target\_codes)   
matrix\_range\_2 <- create\_comparison\_matrix (df\_range\_2\_levels, target\_codes)   
matrix\_range\_3 <- create\_comparison\_matrix (df\_range\_3\_levels, target\_codes)

Applies the create\_comparison\_matrix function to each DataFrame (corresponding to different year ranges) to generate comparison matrices containing information about IPC code matches.

# Creating and Saving the Excel File with Results

## Create a new Excel file and add sheets with the results

wb <- createWorkbook ()

* createWorkbook(): Function of the openxlsx library that creates a new empty Excel workbook file.

## Define Function to Add Sheets with Results

add\_sheet <- function (wb, range, sheet\_name) {  
 addWorksheet (wb, sheet\_name)  
 if ( ! is.null (range $ data) && nrow (range $ data) > 0 ) {  
 writeData (wb, sheet\_name, range $ data, startCol = 1 , startRow = 1 )  
 writeData (wb, sheet\_name, paste ( "Total patents in range" , range $ range, ":" , range $ total\_patents), startCol = 1 , startRow = nrow (range $ data) + 2 )  
 if ( ! is.null (range $ count\_per\_year) && nrow (range $ count\_per\_year) > 0 ) {  
 writeData (wb, sheet\_name, range $ count\_per\_year, startCol = 1 , startRow = nrow (range $ data) + 4 )   
}   
} else {  
 writeData (wb, sheet\_name, "No data available" , startCol = 1 , startRow = 1 )   
}   
}

* add\_sheet: Function that adds a sheet to the workbook (wb) with a specific name (sheet\_name). Inside the sheet:
  + addWorksheet(wb, sheet\_name): Adds a new sheet to the workbook with the given name.
  + writeData(wb, sheet\_name, range Writes the data from the DataFrame range$data to the sheet, starting at column 1 and row 1.
  + paste(“Total patents in range”, range$range, ":", range$total\_patents): Creates a string that shows the total number of patents in the specified range.
  + writeData(wb, sheet\_name, …, startCol = 1, startRow = nrow(range $ Writes the total patents in a row below the data.
  + If count\_per\_year) &&nrow(range $ Check if there is count data by year and write it below the total patents.
  + Writedata count\_per\_year, startCol = 1, startRow = nrow(range$data) + 4): Writes the patent count per year to the sheet.

## Add sheets with filtered data

add\_sheet (wb, range\_1, "2012-2016" )   
add\_sheet (wb, range\_2, "2013-2017" )   
add\_sheet (wb, range\_3, "2014-2018" )

* add\_sheet(wb, range\_1, “2012-2016”): Calls the add\_sheet function to add a sheet with data from the year range 2012-2016 to the workbook.
* add\_sheet(wb, range\_2, “2013-2017”): Adds a sheet with data from the range 2013-2017.
* add\_sheet(wb, range\_3, “2014-2018”): Adds a sheet with data from the range 2014-2018.

## Add each matrix as a separate sheet

addWorksheet (wb, "Matrix\_Range\_1" )   
writeData ( wb , "Matrix\_Range\_1 " , matrix\_range\_1 )   
  
addWorksheet (wb, " Matrix\_Range\_2" )   
writeData (wb, "Matrix\_Range\_2" , matrix\_range\_2)   
  
addWorksheet (wb, "Matrix\_Range\_3" )   
writeData (wb, "Matrix\_Range\_3" , matrix\_range\_3)

* addWorksheet(wb, “Matrix\_Range\_1”): Adds a sheet named “Matrix\_Range\_1” to the workbook.
* writeData(wb, “Matrix\_Range\_1”, matrix\_range\_1): Writes the matrix matrix\_range\_1 to the sheet “Matrix\_Range\_1”.
* addWorksheet(wb, “Matriz\_Rango\_2”) and writeData(wb, “Matriz\_Rango\_2”, matrices\_rango\_2): Similar to the above but for matrices\_rango\_2.
* addWorksheet(wb, “Matriz\_Rango\_3”) and writeData(wb, “Matriz\_Rango\_3”, matrices\_rango\_3): Similar to the above but for matrices\_rango\_3.

## Save the excel file

saveWorkbook (wb, "C: \ Users \ danie \ Documents \ DT COMPANIES \ EXAMPLE\_COMPANY \ New.xlsx" , overwrite = TRUE )

saveWorkbook(wb, “file\_path”, overwrite = TRUE): Saves the workbook (wb) to the specified path. overwrite = TRUE allows you to overwrite the file if it already exists.

# Processing and Storing Data from Excel Files

## Function to process each Excel file for storage

process\_and\_store\_excel <- function (excel\_path, storage\_path) {  
 # Read the generated Excel file   
data\_matrix\_1 <- read\_excel (excel\_path, sheet = "Matrix\_Range\_1" )   
data\_matrix\_2 <- read\_excel (excel\_path, sheet = "Matrix\_Range\_2" )   
data\_matrix\_3 <- read\_excel (excel\_path, sheet = "Matrix\_Range\_3" )  
   
 # Check that data\_matrix are data frames  
 if ( ! is.data.frame (data\_matrix\_1) | ! is.data.frame (data\_matrix\_2) | ! is.data.frame (data\_matrix\_3)) {  
 stop ( " One of the data objects is not a data frame" )   
}  
   
 # Extract the IPC codes from the matrix sheets   
ipc\_codes\_1 <- data\_matrix\_1 $ Code   
ipc\_codes\_2 <- data\_matrix\_2 $ Code   
ipc\_doces\_3 <- data\_matrix\_3 $ Code  
   
 # Extract the "Total" column from the matrix sheets   
total\_1 <- data\_matrix\_1 $ Total   
total\_2 <- data\_matrix\_2 $ Total   
total\_3 <- data\_matrix\_3 $ Total  
   
 # Create a new data frame with the IPC codes as columns   
df\_result\_1 <- tibble (  
 Company = tools :: file\_path\_sans\_ext ( basename (excel\_path)),  
 !!! setNames ( as.list (total\_1), ipc\_codes\_1)   
)  
   
df\_result\_2 <- tibble (  
 Company = tools :: file\_path\_sans\_ext ( basename (excel\_path)),  
 !!! setNames ( as.list (total\_2), ipc\_codes\_2)   
)  
   
df\_result\_3 <- tibble (  
 Company = tools :: file\_path\_sans\_ext ( basename (excel\_path)),  
 !!! setNames ( as.list (total\_3), ipc\_codes\_3)   
)  
   
 # Read the current contents of the storage file  
 if ( file.exists (storage\_path)) {   
wb\_storage <- loadWorkbook (storage\_path)   
} else {  
 # If the file does not exist, create it and add the necessary sheets   
wb\_storage <- createWorkbook ()  
 addWorksheet (wb\_storage, "Storage\_Range\_1" )  
 addWorksheet (wb\_storage, "Storage\_Range\_2" )  
 addWorksheet (wb\_storage, "Storage\_Range\_3" )   
}  
   
 # Write the data in each sheet (checking if it already exists)  
 if ( "Storage\_Range\_1" %in% names (wb\_storage)) {   
current\_data\_1 <- read.xlsx (wb\_storage, sheet = "Storage\_Range\_1" )  
 writeData (wb\_storage, "Storage\_Range\_1" ,  
 rbind (current\_data\_1, df\_result\_1), colNames = TRUE )   
}  
   
 if ( "Storage\_Range\_2" %in% names (wb\_storage)) {   
current\_data\_2 <- read.xlsx (wb\_storage, sheet = "Storage\_Range\_2" )  
 writeData (wb\_storage, "Storage\_Range\_2" ,  
 rbind (current\_data\_2, df\_result\_2), colNames = TRUE )   
}  
   
 if ( "Storage\_Range\_3" %in% names (wb\_storage)) {   
current\_data\_3 <- read.xlsx (wb\_storage, sheet = "Storage\_Range\_3" )  
 writeData (wb\_storage, "Storage\_Range\_3" ,  
 rbind (current\_data\_3, df\_result\_3), colNames = TRUE )   
}  
   
 # Save the storage Excel file  
 saveWorkbook (wb\_storage, storage\_path, overwrite = TRUE )   
}

## Definition of the Function to Process and Store Excel

* process\_and\_store\_excel: Function that processes an input Excel file and stores the results in another Excel file.
* read\_excel(excel\_path, sheet = “Matriz\_Rango\_1”): Reads the sheet “Matriz\_Rango\_1” from the Excel file at the specified path and stores the data in data\_matriz\_1.

## Data Type Checking

* is.data.frame: Function that checks if an object is a DataFrame. Here it ensures that each object read is a DataFrame.
* stop(“Message”): Stops the execution of the function and displays an error message if any of the conditions are not met.

## Extracting Data from Sheets

* $Code and $Total: Access the specific columns of the DataFrames to extract the IPC codes and patent totals.

## Creating New DataFrames for Results

* tibble: Data structure similar to a DataFrame but with better data handling and visualization in R.
* tools::file\_path\_sans\_ext(basename(excel\_path)): Extracts the file name without the extension to use as the company name.
* setNames(as.list(total\_1), IPC\_codes\_1): Converts totals to a list and assigns them names based on IPC codes.

## Reading and Creating the Storage File

* file.exists(storage\_path): Checks if the storage file already exists.
* loadWorkbook(storage\_path): Loads an existing workbook file.
* createWorkbook(): Creates a new workbook if the archive file does not exist.
* addWorksheet: Adds new sheets to the workbook for each range.

## Writing Data to the Storage File

* rbind(current\_data\_1, df\_resultado\_1): Combines the current data in the sheet with the new data.
* writeData(wb\_storage, “Sheet\_Name”, …): Writes the combined data to the corresponding sheet.

## Save Excel File from Storage

* saveWorkbook(wb\_storage, storage\_path, overwrite = TRUE): Saves the workbook with the updated data to the specified path, allowing the file to be overwritten if it already exists.

# Run Function and Display Working Directory

## Excel file storage path

storage\_path <- "C: \ Users \ danie \ Documents \ DT\_COMPANIES \ EXAMPLE\_COMPANY \ DT\_STORAGE.xlsx"   
  
# Process and store the information in the generated file   
process\_and\_store\_excel ( "C: \ Users \ danie \ Documents \ DT\_COMPANIES \ EXAMPLE\_COMPANY \ new.xlsx" , storage\_path )   
  
# Get the current working directory   
print ( getwd ())

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

* storage\_path: Defines the path of the storage file.
* process\_and\_store\_excel: Calls the function to process the input Excel file and store the results in the storage file.
* getwd(): Gets and displays the current working directory.