Trabajo de Fin de Grado

Análisis y agrupamiento óptimo de tarjetas gráficas

Autor: Maik Mende

Tutora: Marina Leal Palazón

Grado en Estadística Empresarial

Facultad de Ciencias Sociales y Jurídicas

Curso 2024-2025

Índice general

1 Resumen

En este Trabajo de Fin de Grado se analiza y agrupa una base de datos de tarjetas gráficas de los últimos 20 años utilizando técnicas de análisis de datos y aprendizaje no supervisado, principalmente el algoritmo K-means. El proceso incluyó la limpieza y transformación de los datos, el análisis exploratorio de variables numéricas y categóricas, y la implementación de diferentes modelos de agrupamiento: uno solo con variables numéricas, otro combinado con variables categóricas y numéricas, y una versión manual de K-means que permitió comparar distintas métricas de distancia (euclidiana y Minkowski).

Se trabaja sobre una base de datos que contiene información sobre tarjetas gráficas de diferentes fabricantes y generaciones.

2 Introducción

2.1 Contexto histórico y motivación

Históricamente, las tarjetas gráficas han tenido un papel fundamental en la evolución de los videojuegos y la computación gráfica. Desde sus inicios, estas unidades de procesamiento gráfico (GPUs) han evolucionado de manera significativa, permitiendo la creación de gráficos cada vez más complejos y realistas. En sus primeras etapas, las GPUs estaban diseñadas exclusivamente para acelerar el renderizado de gráficos en 2D y 3D, pero con el tiempo, su arquitectura se ha adaptado para soportar cálculos más generales.

Un ejemplo destacado de esta evolución es su aplicación en el análisis de imágenes de resonancia magnética funcional (fMRI). Según el trabajo de (Eklund et al.), las GPUs han demostrado ser herramientas altamente eficientes para acelerar el procesamiento de datos de fMRI, que requiere manejar grandes volúmenes de información y realizar cálculos intensivos. Este enfoque ha permitido reducir significativamente los tiempos de procesamiento en comparación con las CPUs tradicionales, haciendo posible realizar análisis más rápidos y detallados de la actividad cerebral. Sin embargo, los autores también destacan los desafíos asociados, como la necesidad de adaptar algoritmos existentes para aprovechar al máximo la arquitectura de las GPUs.

De manera complementaria, un estudio más reciente de (Kalaiselvi et al.) también explora el uso de GPUs en aplicaciones biomédicas, destacando su capacidad para manejar tareas de procesamiento intensivo en áreas como la neurociencia y la biomedicina. Ambos estudios coinciden en que las GPUs no solo ofrecen ventajas en términos de velocidad, sino que también permiten realizar análisis más complejos que antes eran inviables debido a las limitaciones de las CPUs. Además, el estudio de 2017 amplía el enfoque al considerar no solo el rendimiento, sino también los desafíos prácticos, como la optimización de algoritmos y la gestión de recursos computacionales en entornos biomédicos.

Más recientemente, (Zhang et al.) ha desarrollado un marco computacional basado en GPUs que conecta la simulación neuronal con la inteligencia artificial. Este trabajo destaca cómo las GPUs pueden ser utilizadas para integrar modelos de simulación neuronal con algoritmos de aprendizaje profundo, permitiendo avances significativos en la comprensión de la actividad cerebral y en el desarrollo de sistemas de inteligencia artificial inspirados en el cerebro. Este enfoque no solo mejora la eficiencia computacional, sino que también abre nuevas posibilidades para explorar la interacción entre la neurociencia y la inteligencia artificial, un campo emergente con un enorme potencial.

Estos hallazgos refuerzan la idea de que las GPUs han trascendido su propósito original, convirtiéndose en herramientas clave en campos como la investigación médica, la neurociencia y la inteligencia artificial. Su capacidad para realizar cálculos paralelos de manera eficiente las convierte en un recurso indispensable para tareas de computación intensiva, como el análisis de datos de fMRI, la simulación neuronal y otras aplicaciones biomédicas. Además, con la evolución de la inteligencia artificial y el aprendizaje profundo, las GPUs continúan ampliando su impacto en áreas como la simulación científica, la minería de datos y el desarrollo de tecnologías innovadoras.

2.2 Objetivos

El objetivo principal de este trabajo es analizar y agrupar tarjetas gráficas en función de sus especificaciones técnicas y características relevantes, utilizando técnicas de análisis de datos y aprendizaje no supervisado. Para ello, se plantean los siguientes objetivos específicos:

1. Describir y preparar la base de datos: Realizar una limpieza y transformación exhaustiva de los datos, identificando y tratando valores atípicos, variables irrelevantes y datos faltantes, con el fin de obtener un conjunto de datos adecuado para el análisis.

2. Explorar y analizar las variables clave: Llevar a cabo un análisis exploratorio de las variables numéricas y categóricas, identificando patrones, tendencias y relaciones relevantes entre las diferentes características de las tarjetas gráficas.

3. Implementar y comparar métodos de agrupamiento: Aplicar el algoritmo K-means, tanto con variables numéricas como combinando variables numéricas y categóricas, y comparar los resultados obtenidos. Además, implementar el algoritmo K-means de forma manual para explorar el impacto de diferentes métricas de distancia, como la distancia euclidiana y la distancia de Minkowski.

4. Interpretar y validar los grupos obtenidos: Analizar las características de los clústeres resultantes, identificando patrones comunes y diferencias entre los grupos, y evaluar la robustez y coherencia de los agrupamientos en función de las especificaciones técnicas y la evolución tecnológica de las tarjetas gráficas.

5. Extraer conclusiones relevantes: Sintetizar los hallazgos obtenidos a lo largo del análisis, destacando las implicaciones prácticas y teóricas del agrupamiento de tarjetas gráficas y proponiendo posibles líneas de investigación futura.

2.3 Herramientas

En este trabajo se utilizará el lenguaje de programación R como herramienta principal para el análisis de datos. A lo largo del documento, se combinarán explicaciones teóricas con fragmentos de código en R, lo que permitirá ilustrar de manera práctica los conceptos y métodos empleados. Este enfoque busca facilitar la comprensión de los análisis realizados y fomentar la reproducibilidad de los resultados.

2.4 Fundamentos académicos y aportación novedosa

A lo largo de la carrera, he adquirido una sólida base en técnicas estadísticas y de análisis de datos, incluyendo el análisis exploratorio de datos, el estudio de correlaciones entre variables, el análisis de componentes principales (PCA) y la aplicación de métodos de agrupamiento como el algoritmo k-means utilizando el software R. Estas herramientas me han permitido abordar problemas complejos de clasificación y segmentación de datos, así como interpretar y visualizar patrones relevantes en conjuntos de datos multidimensionales.

Sin embargo, este trabajo incorpora también una vertiente novedosa respecto a lo aprendido en la carrera. En particular, se ha desarrollado una implementación manual del algoritmo k-means, lo que ha permitido un mayor control sobre el proceso de agrupamiento y la posibilidad de experimentar con diferentes métricas de distancia, como la distancia de Minkowski. Esta extensión no solo enriquece el análisis, sino que también permite evaluar la robustez de los resultados frente a la presencia de valores atípicos y explorar alternativas más flexibles a la distancia euclidiana tradicionalmente utilizada. De este modo, el trabajo combina los conocimientos adquiridos durante la formación académica con una aproximación más avanzada y personalizada al problema de agrupamiento de tarjetas gráficas.

3 Base de datos: descripción, preparación y análisis.

3.1 Descripción de los datos

Vamos a usar una base de datos que contiene información sobre tarjetas gráficas de diferentes fabricantes de los últimos 20 años. La base de datos incluye diferentes especificaciones técnicas que nos ayudarán a analizar y agrupar las tarjetas gráficas.

A continuación se procederá a cargar la base de datos con ayuda de la librería **readr**.

```
set.seed(1234)
library(readr)
setwd("C:/Users/maikm/Desktop/Estadística/TFG_git")
datos <- read_csv("data/tpu_gpus.csv")
```

El conjunto de datos contiene tanto variables categóricas como numéricas, aunque la mayoría de las variables son categóricas en su estado original. A continuación, se describen las principales variables y las transformaciones que se realizarán para adaptarlas al análisis:

- **Product_Name**: Identificador único de cada tarjeta gráfica. Por lo tanto no la trataremos como una variable, ya que su único propósito es identificar cada tarjeta gráfica.
- **GPU_Chip**: Es una variable categórica que nos indica el tipo de chip que utiliza la tarjeta gráfica. A continuación mostraremos cuántos tipos de chips hay en la base de datos.

Tenemos 450 tipos de chips distintos. De los 450 tipos de chips diferentes, solamente tenemos 106 tarjetas gráficas con al menos 10 tarjetas por tipo de chip y tenemos alrededor de 161 tipos de chip con solamente 2 o menos tarjetas gráficas.

- **Released**: Fecha de lanzamiento de la tarjeta gráfica. Originalmente, esta variable incluye valores como “Unknown” y “Never Released”, que serán tratados de la siguiente manera:
  - “Unknown”: Se convertirá a NA.
  - “Never Released”: Se eliminará la fila correspondiente.
  - La fecha será transformada a un formato de fecha estándar y categorizada en cinco intervalos: “Antes de 2000”, “2000-2009”, “2010-2015”, “2016-2020” y “Después de 2020”.
- **Bus**: Es una variable categórica que nos indica el tipo de bus que utiliza la tarjeta gráfica. A continuación mostraremos cuántos tipos de bus hay en la base de datos.

Tenemos 30 tipos de bus distintos.

- **Memory**: Variable categórica que combina información sobre el tamaño de memoria, el tipo de memoria y el ancho del bus de memoria. Para facilitar el análisis, esta variable será descompuesta en tres nuevas variables:

  - **Memory_Size (GB)**: Tamaño de la memoria, que será transformado a un formato numérico y convertido a gigabytes (GB), eliminando filas con valores como “System Shared”.
  - **Memory_Type**: Tipo de memoria utilizada (por ejemplo, GDDR6, HBM2).
  - **Memory_Bus (bits)**: Ancho del bus de memoria, que será transformado a un formato numérico.

- **GPU_clock**: Es una variable categórica que nos indica la velocidad del reloj de la GPU. Convertiremos esta variable a numérica y la llamaremos GPU_clock (MHz).

- **Memory_clock**: Es una variable categórica que nos indica la velocidad del reloj de la memoria. Al igual que con GPU_clock, convertiremos esta variable a numérica y la llamaremos Memory_clock (MHz).

- **Shaders_TMUs_ROPs**: Variable categórica que combina información sobre el número de unidades de sombreado (Shaders), unidades de mapeo de texturas (TMUs) y tuberías de operaciones de rasterización (ROPs). Esta variable será descompuesta en tres nuevas variables:

  - **Shaders**: Número de unidades de sombreado, que será transformado a un formato numérico.
  - **TMUs**: Número de unidades de mapeo de texturas, que será transformado a un formato numérico.
  - **ROPs**: Número de tuberías de operaciones de rasterización, que será transformado a un formato numérico.

- **..1**: Es una variable numérica que sólo indica la fila en la que estamos. No aporta información relevante para el análisis y por tanto la eliminamos.

Resumen de tratamiento de las variables

El tratamiento de las variables incluye:

- **Limpieza de datos**: Eliminación de valores irrelevantes como “Never Released” y conversión de “Unknown” a NA.
- **Transformación de formatos**: Conversión de variables categóricas a numéricas donde sea necesario.
- **Separación de variables compuestas**: Descomposición de variables como “Memory” y “Shaders_TMUs_ROPs” en variables individuales para facilitar el análisis.
- **Eliminación de variables irrelevantes**: Eliminación de la variable “..1”.

3.2 Preparación y procesamiento de los datos

En esta sección, se realizarán las transformaciones necesarias para limpiar y preparar el conjunto de datos. Esto incluye la eliminación de valores irrelevantes, la transformación de variables categóricas y numéricas, y la creación de nuevas variables derivadas para facilitar el análisis posterior.

Cabe destacar que algunas de las filas de la variable Shaders_TMUs_ROPs tienen 4 valores en lugar de 3. Esto se debe a que las tarjetas gráficas más antiguas no tenían shaders unificados, sino que tenían shaders separados para píxeles y vértices. Por este motivo, primero tendremos que separar la variable en 4 columnas: “Shaders_1”, “Shaders_2”, “TMUs” y “ROPs”. Luego trataremos 2 casos:

- Si hay 4 valores, los mantendremos como están y elegiremos el valor más alto entre Shaders_1 y Shaders_2 como el número de shaders.

- Si hay 3 valores, nos llenará la columna ROPs con NAs y tendremos que mover el valor de TMUs a ROPs y el valor de Shaders_2 a TMUs y rellenar Shaders_2 con NAs.

Una vez hayamos tratado ambos casos, crearemos una nueva variable llamada Shaders, que será el máximo entre Shaders_1 y Shaders_2, y como en el caso de 3 valores, Shaders_2 será NA, entonces Shaders será igual a Shaders_1.

Elegimos el máximo entre Shaders_1 y Shaders_2 porque el máximo entre los shaders de píxeles y de vértices en las tarjetas antiguas es más comparable con los shaders unificados que tenemos hoy en día.

3.2.1 Limpieza y transformación de variables

En esta sección, se realizarán las transformaciones de datos anteriormente descritas. Esto incluye:

- Eliminación de filas irrelevantes.
- Separación de variables compuestas.
- Conversión de variables categóricas a numéricas.
- Reorganización y eliminación de columnas innecesarias.

Para ello, utilizaremos las siguientes librerías:

- **dplyr**: Para la manipulación de datos.
- **tidyr**: Para la separación de columnas y transformación de datos.
- **stringr**: Para la manipulación de cadenas de texto.
- **lubridate**: Para el manejo de fechas.

A continuación se detallan las transformaciones y limpiezas realizadas en el conjunto de datos:

1. **Eliminación de filas irrelevantes**: Se eliminaron las filas donde la columna “Memory” contenía “System Shared” y aquellas donde “Released” era “Never Released”.
2. **Separación de variables compuestas**:

- La columna “Memory” se separó en tres nuevas columnas:
  - **“Memory_Size (GB)”**: Tamaño de la memoria, convertido a un formato numérico y expresado en gigabytes (GB).
  - **“Memory_Type”**: Tipo de memoria utilizada.
  - **“Memory_Bus (bits)”**: Ancho del bus de memoria, convertido a un formato numérico.

3. **Conversión de formatos**:

- La columna “Released” se transformó a un formato de fecha estándar y se categorizó en cinco intervalos: “Antes de 2000”, “2000-2009”, “2010-2015”, “2016-2020” y “Después de 2020”.
- Las columnas “Memory_clock” y “GPU_clock” se convirtieron a un formato numérico y se renombraron como “Memory_clock (MHz)” y “GPU_clock (MHz)”, respectivamente.

```r
library(dplyr)
library(tidyr)
library(stringr)
library(lubridate)

datos <- datos %>%
  # Si la columna Memory tiene "System Shared" Y Released tiene "Never Released",
  #eliminar esas filas
  filter(
         !str_detect(Memory, "System Shared"),
         !str_detect(Released, "Never Released")) %>%

  # Separar la columna Memory
  separate(Memory, into = c("Memory_Size (GB)", "Memory_Type",
                            "Memory_Bus (bits)"),
           sep = ", ", remove = TRUE, fill = "right") %>%

  # Convertir Memory_Bus (bits) a numérico Y Released a date
  mutate(
    `Memory_Bus (bits)` = as.numeric(str_extract(`Memory_Bus (bits)`, "\\d+")),
    # Primero convertimos "Unknown" a NA
    Released = ifelse(Released == "Unknown", NA, Released),
    Released = as.Date(parse_date_time(Released, orders = c("b d, Y", "Y")))
  ) %>%

  # Convertir Memory_Size (GB) a numérico y convertir a GB
  mutate(`Memory_Size (GB)` = case_when(
    str_detect(`Memory_Size (GB)`, "GB") ~ as.numeric(str_extract(`Memory_Size (GB)`,
    "\\d+")),
    str_detect(`Memory_Size (GB)`, "MB") ~ as.numeric(str_extract(`Memory_Size (GB)`,
    "\\d+")) / 1024,
    str_detect(`Memory_Size (GB)`, "KB") ~ as.numeric(str_extract(`Memory_Size (GB)`,
    "\\d+")) / (1024 * 1024),
    TRUE ~ NA_real_  # En cualquier otro caso, NA
  )) %>%

  # Separar Shaders_TMUs_ROPs en 4 columnas (Shaders_1, Shaders_2, TMUs, ROPs)
  separate(Shaders_TMUs_ROPs, into = c("Shaders_1", "Shaders_2", "TMUs", "ROPs"),
           sep = " / ", remove = TRUE, fill = "right") %>%

# Manejar casos donde hay solo 3 valores (x1 / x2 / x3)
mutate(
  # Si ROPs es NA, significa que solo hay 3 valores (x1 / x2 / x3)
```

```
  ROPs = ifelse(is.na(ROPs), TMUs, ROPs),  # Mover TMUs a ROPs
  TMUs = ifelse(TMUs == ROPs, Shaders_2, TMUs),  # Mover Shaders_2 a TMUs
  Shaders_2 = ifelse(Shaders_2 == TMUs, NA, Shaders_2)  # Poner NA en Shaders_2
) %>%

  # Convertir a numérico Memory_clock y GPU_clock
  mutate(
    `Memory_clock (MHz)` = as.numeric(str_extract(`Memory_clock`, "\\d+")),
    `GPU_clock (MHz)` = as.numeric(str_extract(`GPU_clock`, "\\d+"))
  ) %>%

  # Crear la nueva variable Shaders: máximo entre Shaders_1 y Shaders_2.
  #Si Shaders_2 es NA, significa que estamos en el caso de 3 valores.
  mutate(
    Shaders = ifelse(!is.na(Shaders_2), pmax(Shaders_1, Shaders_2), Shaders_1)
  ) %>%

  # Mover la columna Shaders delante de TMUs y ROPs
  relocate(Shaders, .before = TMUs) %>%

  # Eliminar columnas innecesarias
  select(-Shaders_1, -Shaders_2, -Memory_clock, -GPU_clock, -`...1`) %>%

  # Convertir Shaders, TMUs y ROPs a numérico
  mutate(across(c(`Shaders`, `TMUs`, `ROPs`), ~ suppressWarnings(as.numeric(.)))) %>%

  # Convertir Released a categórico y dividir en 5 categorías
  mutate(
    Released = case_when(
      Released < as.Date("2000-01-01") ~ "Antes de 2000",
      Released < as.Date("2010-01-01") ~ "2000-2009",
      Released < as.Date("2016-01-01") ~ "2010-2015",
      Released < as.Date("2021-01-01") ~ "2016-2020",
      TRUE ~ "Después de 2020"
    )
  )

num_vars <- datos %>% select(where(is.numeric)) %>% names()
cat_vars <- datos %>% select(where(is.character)) %>% names()
```

3.2.2 Descripción de las variables transformadas

Tras el procesamiento, el conjunto de datos contiene las siguientes variables:

Variables categóricas:

```
cat_vars
```

```
## [1] "Product_Name" "GPU_Chip"     "Released"     "Bus"          "Memory_Type"
```

- **GPU_Chip**: Como hemos mencionado anteriormente, es el tipo de chip que utiliza la tarjeta gráfica.
- **Released**: Es una variable categórica nueva que hemos creado, y nos indica la fecha de lanzamiento de la tarjeta gráfica en 4 categorías: “Antes de 2000”, “2000-2009”, “2010-2015”, “2016-2020” y “Después de 2020”.
- **Bus**: Es el tipo de bus que utiliza la tarjeta gráfica.
- **Memory_Type**: Es una variable categórica nueva que hemos creado a partir de la variable “Memory”, y nos indica el tipo de memoria que utiliza la tarjeta gráfica.

Variables numéricas:

```
num_vars
```

```
## [1] "Memory_Size (GB)"   "Memory_Bus (bits)"  "Shaders"
## [4] "TMUs"               "ROPs"               "Memory_clock (MHz)"
## [7] "GPU_clock (MHz)"
```

- **Memory_Size (GB)**: Es una variable numérica nueva que hemos creado a partir de “Memory”, y nos indica la cantidad de memoria que tiene la tarjeta gráfica, medida en gigabytes (GB). Una mayor cantidad de memoria permite manejar texturas y datos gráficos más grandes, lo que es crucial para juegos y aplicaciones con gráficos de alta resolución o entornos complejos.
- **Memory_Bus (bits)**: Es una variable numérica nueva que hemos creado a partir de “Memory”, y nos indica el ancho del bus de memoria, medido en bits. Determina cuántos datos pueden transferirse entre la GPU y la memoria en un ciclo de reloj. Un bus más ancho permite un mayor ancho de banda, lo que mejora el rendimiento en tareas gráficas intensivas.
- **Shaders**: Es una variable numérica nueva que hemos extraído de la variable “Shaders_TMUs_ROPs”, y representa el número de unidades de sombreado (shaders) en la GPU. Los shaders son responsables de procesar píxeles, vértices y otros elementos gráficos. Un mayor número de shaders generalmente indica un mayor poder de procesamiento gráfico.
- **TMUs**: Es una variable numérica nueva que hemos extraído de la variable “Shaders_TMUs_ROPs”, y nos indica las unidades de mapeo de texturas (TMUs), encargadas de aplicar texturas a los objetos 3D. Un mayor número de TMUs permite manejar texturas más complejas y mejorar la calidad visual.
- **ROPs**: Es una variable numérica nueva que hemos extraído de la variable “Shaders_TMUs_ROPs”, y son las tuberías de operaciones de rasterización (ROPs), responsables de escribir los píxeles finales en la memoria de video. Un mayor número de ROPs mejora el rendimiento en tareas como el antialiasing y la renderización de píxeles.

- **Memory_clock (MHz)**: Es la velocidad del reloj de la memoria, medida en megahercios (MHz). Indica la frecuencia a la que opera la memoria. Una mayor velocidad de reloj de la memoria permite un mayor ancho de banda y un mejor rendimiento general.
- **GPU_clock (MHz)**: Es la velocidad del reloj de la GPU, medida en megahercios (MHz). Indica la frecuencia a la que opera el núcleo de la GPU. Una mayor velocidad de reloj de la GPU mejora el rendimiento en tareas gráficas y de procesamiento.

Tras las transformaciones, el conjunto de datos contiene un total de 7 variables numéricas y 5 variables categóricas.

3.3 Análisis exploratorio de datos

3.3.1 Estadísticas descriptivas

En esta sección, se llevará a cabo un análisis descriptivo de las variables numéricas y categóricas del conjunto de datos. Este análisis tiene como objetivo proporcionar una visión general de las principales características de los datos, identificar patrones relevantes y detectar posibles irregularidades.

Además, se realizará la transformación de la variable **Released** a un formato de factor ordenado, lo que permitirá organizar las fechas de lanzamiento de manera adecuada y facilitar su representación en los gráficos.

```
# Convertir Released a factor
datos$Released <- factor(datos$Released, levels = c("Antes de 2000", "2000-2009",
"2010-2015", "2016-2020", "Después de 2020"))
```

```
# Resumen de las variables
summary(datos)
```

```
##  Product_Name         GPU_Chip                    Released
##  Length:3047        Length:3047        Antes de 2000  : 106
##  Class :character   Class :character   2000-2009      : 972
##  Mode  :character   Mode  :character   2010-2015      :1367
##                                        2016-2020      : 402
##                                        Después de 2020: 200
##
##      Bus             Memory_Size (GB)    Memory_Type        Memory_Bus (bits)
##  Length:3047        Min.   :  0.00003   Length:3047        Min.   :  32.0
##  Class :character   1st Qu.:  0.50000   Class :character   1st Qu.: 128.0
##  Mode  :character   Median :  2.00000   Mode  :character   Median : 128.0
##                     Mean   :  3.46161                      Mean   : 289.6
##                     3rd Qu.:  4.00000                      3rd Qu.: 256.0
##                     Max.   :128.00000                      Max.   :8192.0
##     Shaders             TMUs             ROPs        Memory_clock (MHz)
##  Min.   :    0.0   Min.   :  0.00   Min.   :  0.00   Min.   :   5.0
```

```
##  1st Qu.:   40.0   1st Qu.:  8.00   1st Qu.:  8.00   1st Qu.: 600.0
##  Median :  384.0   Median : 32.00   Median : 16.00   Median : 902.0
##  Mean   :  984.8   Mean   : 60.25   Mean   : 23.36   Mean   : 957.9
##  3rd Qu.: 1152.0   3rd Qu.: 80.00   3rd Qu.: 32.00   3rd Qu.:1253.0
##  Max.   :21760.0   Max.   :880.00   Max.   :192.00   Max.   :3000.0
##  GPU_clock (MHz)
##  Min.   :  10.0
##  1st Qu.: 520.0
##  Median : 750.0
##  Mean   : 758.2
##  3rd Qu.: 954.0
##  Max.   :2505.0
```

A continuación, se presentan los principales hallazgos:

- **Released**: Vemos que antes de los 2000 se lanzaron muy pocas tarjetas gráficas, entre 2000-2009 aumentó considerablemente la frecuencia de lanzamiento y entre 2010-2015 hubo un "boom" de lanzamientos que disminuyó entre 2016-2020 y volvió a aumentar después de 2020.

Este comportamiento se visualiza mejor en el siguiente histograma:

```r
library(ggplot2)
# Crear el histograma usando la variable Released como categórica
datos %>%
  ggplot(aes(x = Released)) +
  geom_bar(fill = "skyblue", color = "black") +
  labs(title = "Distribución de Fechas de Lanzamiento",
       x = "Fecha de Lanzamiento",
       y = "Frecuencia") +
  theme_minimal()
```

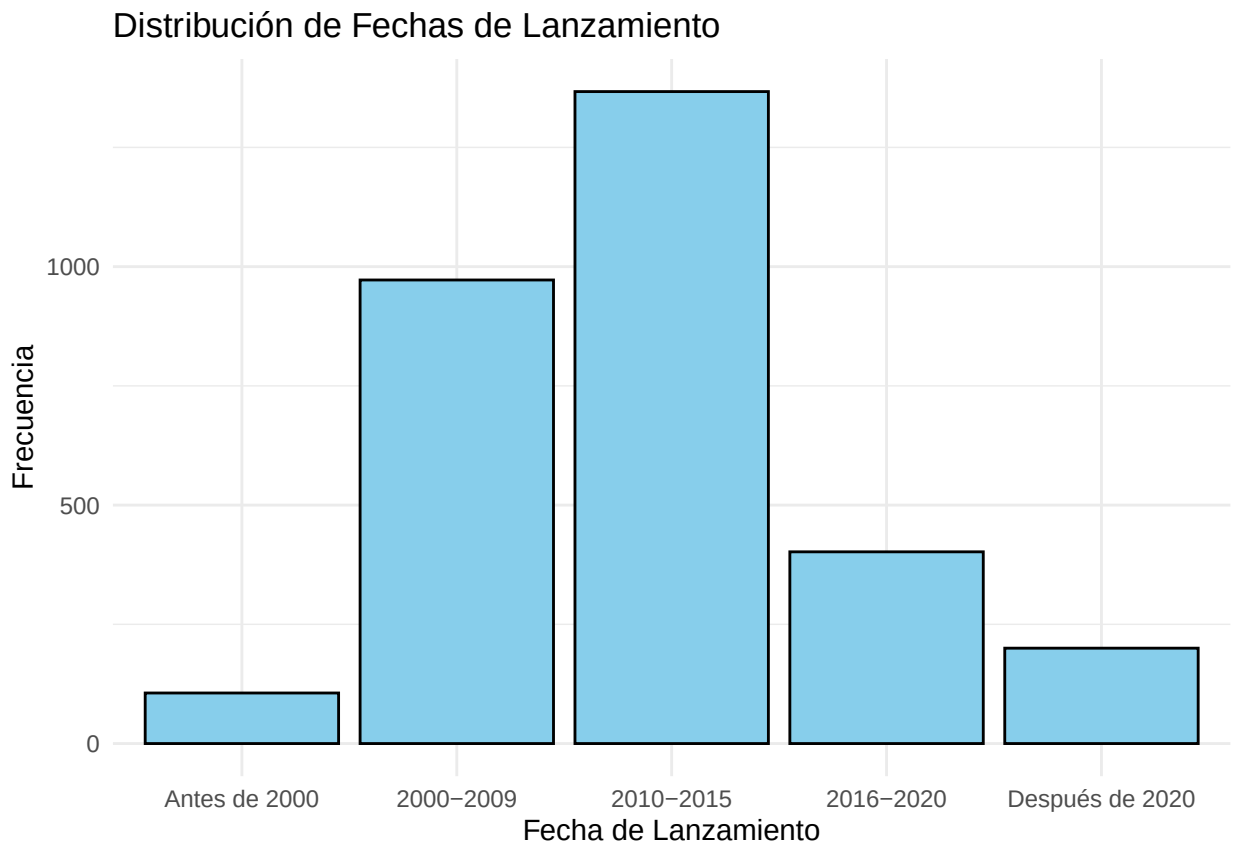

Figura 1: Histograma de Fechas de Lanzamiento

- **Memory_Size (GB)**: Vemos que la media es de 3.46 GB y el máximo es 128 GB, lo que puede indicar que podría haber valores atípicos.
- **Memory_Bus (bits)**: Vemos que la media es de 289.6 bits y el máximo es 8192 bits, lo que puede indicar que podría haber valores atípicos.

En general, vemos que la mayoría de las variables numéricas tienen una media bastante baja y un máximo muy alto, lo que puede indicar que hay valores atípicos.

3.3.2 Tratamiento de valores perdidos

Vamos a comprobar si hay valores perdidos en las variables. Para ello, vamos a contar los NAs por variable y mostrar todas las variables.

```
# Contar los NAs por variable y mostrar todas las variables
na_counts <- datos %>%
  summarise_all(~ sum(is.na(.))) %>%
  pivot_longer(everything(), names_to = "Variable", values_to = "NA_Count")

# Mostrar la tabla completa de cuentas de NAs
print(na_counts)
```

```
## # A tibble: 12 x 2
##    Variable           NA_Count
##    <chr>                 <int>
##  1 Product_Name              0
##  2 GPU_Chip                  0
##  3 Released                  0
##  4 Bus                       0
##  5 Memory_Size (GB)          0
##  6 Memory_Type               0
##  7 Memory_Bus (bits)         0
##  8 Shaders                   0
##  9 TMUs                      0
## 10 ROPs                      0
## 11 Memory_clock (MHz)        0
## 12 GPU_clock (MHz)           0
```

No tenemos valores perdidos.

3.3.3 Tratamiento de valores atípicos

En esta sección, se analizarán las variables numéricas para identificar posibles valores atípicos. Estos valores pueden representar errores en los datos o, por el contrario, ser tarjetas gráficas modernas con especificaciones muy altas. Para ello, se utilizará un gráfico de caja para visualizar la distribución de cada variable numérica y detectar valores atípicos.

```
# Vemos si hay valores atípicos en las variables numéricas
datos %>%
  select(where(is.numeric)) %>%
```

```
  gather() %>%
  ggplot(aes(value)) +
  geom_boxplot() +
  facet_wrap(~key, scales = "free")
```

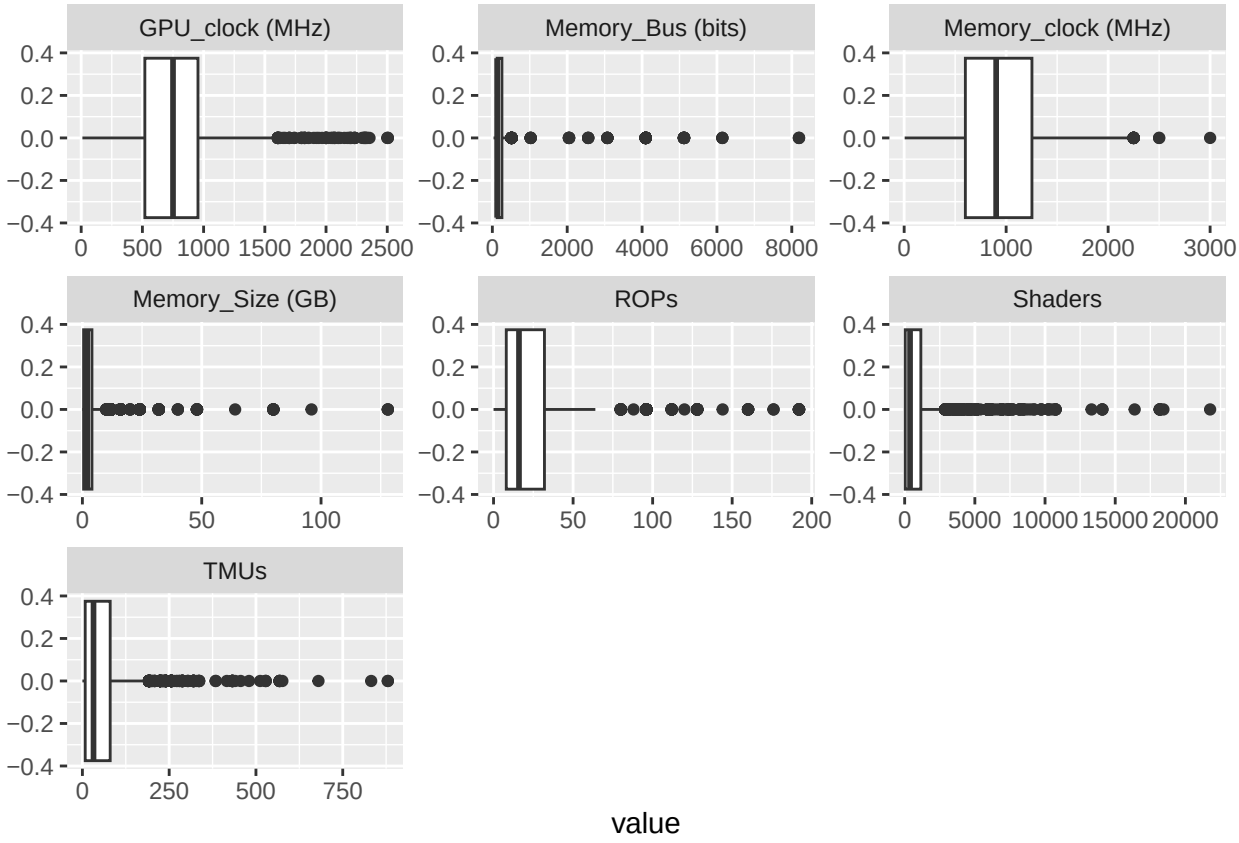

Figura 2: Boxplot de las variables numéricas

Como sospechábamos, hay valores atípicos en todas las variables numéricas. Vamos a ver si esos valores atípicos son errores o son tarjetas gráficas modernas con especificaciones muy altas. Para ello, vamos a comparar la media de los valores atípicos con la media de la variable completa. Esto nos permitirá ver si de verdad son errores o son tarjetas gráficas modernas con especificaciones muy altas.

Vamos a crear una función que nos permita identificar los valores atípicos de una variable numérica y ver su media, así como la media de la variable completa para poder comparar resultados. Además, nos mostrará un gráfico de puntos de la variable seleccionada con Released como categórica para ver su distribución en las diferentes categorías de fecha.

```
identificar_atipicos <- function(datos, variable) {
  # Calcular los cuartiles y el IQR
  Q1 <- quantile(datos[[variable]], 0.25, na.rm = TRUE)
  Q3 <- quantile(datos[[variable]], 0.75, na.rm = TRUE)
```

```r
IQR <- Q3 - Q1

# Definir los límites para valores atípicos
limite_inferior <- Q1 - 1.5 * IQR
limite_superior <- Q3 + 1.5 * IQR

# Identificar los valores atípicos
valores_atipicos <- datos[[variable]][datos[[variable]] < limite_inferior
| datos[[variable]] > limite_superior]

# Filtrar las filas con valores atípicos
tarjetas_atipicas <- datos %>%
  filter(datos[[variable]] %in% valores_atipicos) %>%
  select(Product_Name, `Released`, all_of(variable))

# Contar el número de atípicos por fechas
num_atipicos <- tarjetas_atipicas %>%
  count(Released, sort = TRUE) %>%
  arrange(Released)

# Mostrar la media de la variable seleccionada con el nombre de la variable
media_variable <- mean(datos[[variable]], na.rm = TRUE)

# Crear un gráfico de puntos para la variable seleccionada con Released como categórica
grafico <- datos %>%
  ggplot(aes(x = Released, y = .data[[variable]])) +
  geom_jitter(width = 0.2, alpha = 0.6, color = "blue") +
  labs(
    title = paste("Distribución de", variable, "por Categorías de Fecha"),
    x = "Categoría de Fecha de Lanzamiento",
    y = variable
  ) +
  theme_minimal()

# Mostrar el gráfico
print(grafico)

# Devolver un resumen de los valores atípicos y el número de atípicos por fechas para
#compararlos con el resto de la variable y poner títulos a la lista
return(list(
  valores_atipicos = summary(valores_atipicos),
  media_variable = media_variable,
  num_atipicos = num_atipicos
```

```
  ))
}
```

A continuación se usará la función para las variables numéricas. Vamos a empezar por la variable **Memory_Size (GB)**.

1. **Memory_Size (GB)**:

```
identificar_atipicos(datos, "Memory_Size (GB)")
```

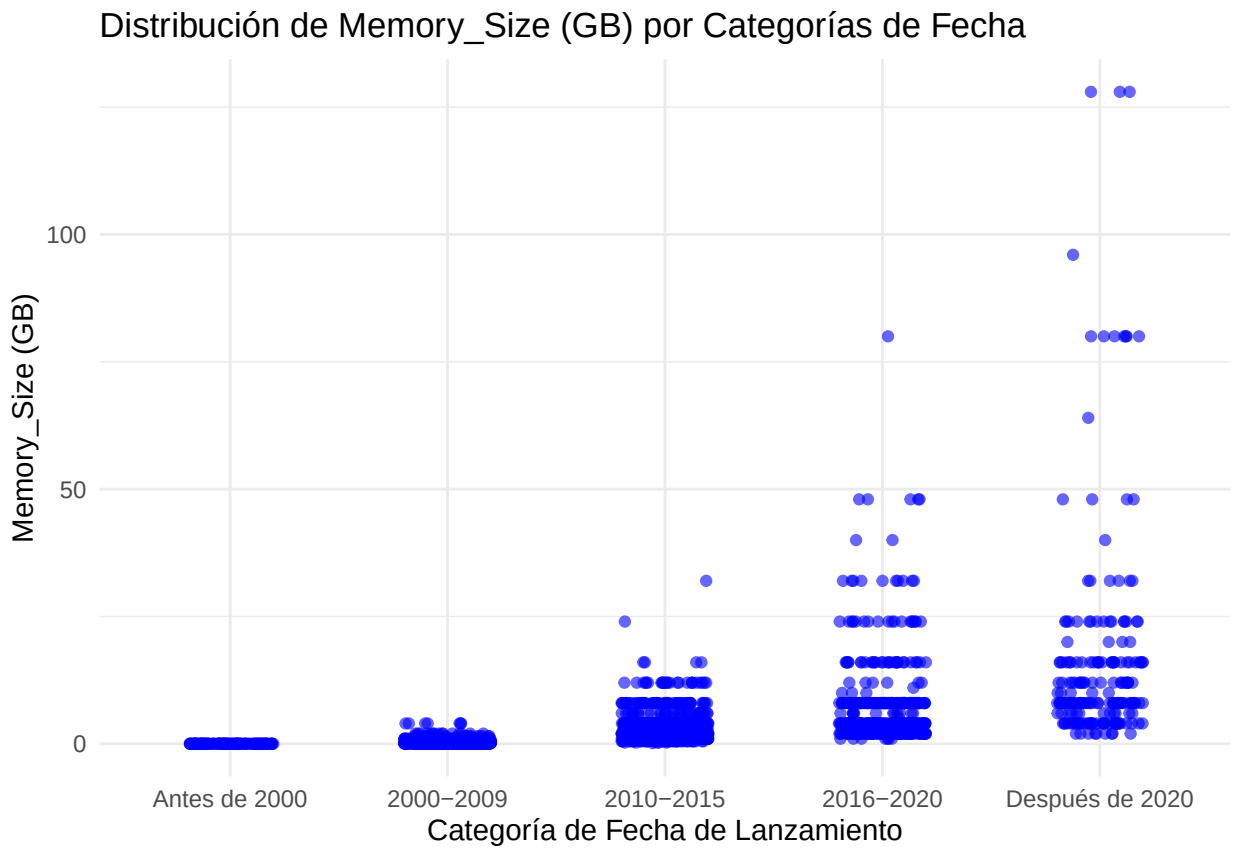

Figura 3: Valores atípicos de Memory_Size (GB)

```
## $valores_atipicos
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   10.00   12.00   16.00   24.13   24.00  128.00
##
## $media_variable
## [1] 3.461614
##
## $num_atipicos
```

```
## # A tibble: 3 x 2
##   Released            n
##   <fct>           <int>
## 1 2010-2015          31
## 2 2016-2020          75
## 3 Después de 2020    95
```

Resultados:

- La media de los valores atípicos es de 24.13 GB, considerablemente mayor que la media general de 3.46 GB.
- Los valores atípicos corresponden a tarjetas gráficas lanzadas principalmente después de 2020, con algunas entre 2016 y 2020 y muy pocas entre 2010 y 2015.

Vamos a ver con las demás variables:

2. **Memory_Bus (bits)**:

```
identificar_atipicos(datos, "Memory_Bus (bits)")
```

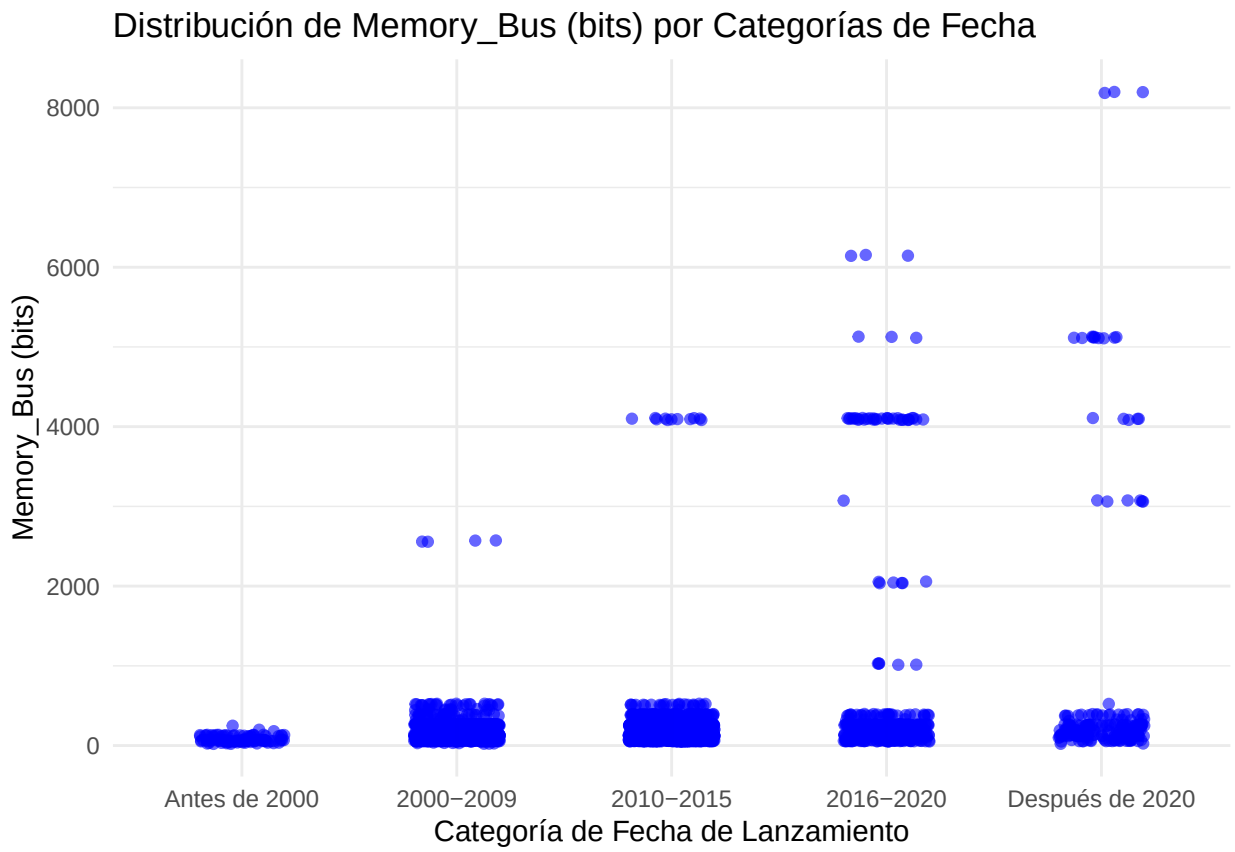

Figura 4: Valores atípicos de Memory_Bus (bits)

```
## $valores_atipicos
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     512     512    3072    2709    4096    8192
##
## $media_variable
## [1] 289.6278
##
## $num_atipicos
## # A tibble: 4 x 2
##   Released            n
##   <fct>           <int>
## 1 2000-2009          30
```

```
## 2 2010-2015          36
## 3 2016-2020          50
## 4 Después de 2020    25
```

Resultados:

- La media de los valores atípicos es de 2319 MHz, considerablemente mayor que la media general de 957.9 MHz.
- Los valores atípicos corresponden a tarjetas gráficas lanzadas principalmente después de 2020.

3. **Memory_clock (MHz)**:

```
identificar_atipicos(datos, "Memory_clock (MHz)")
```

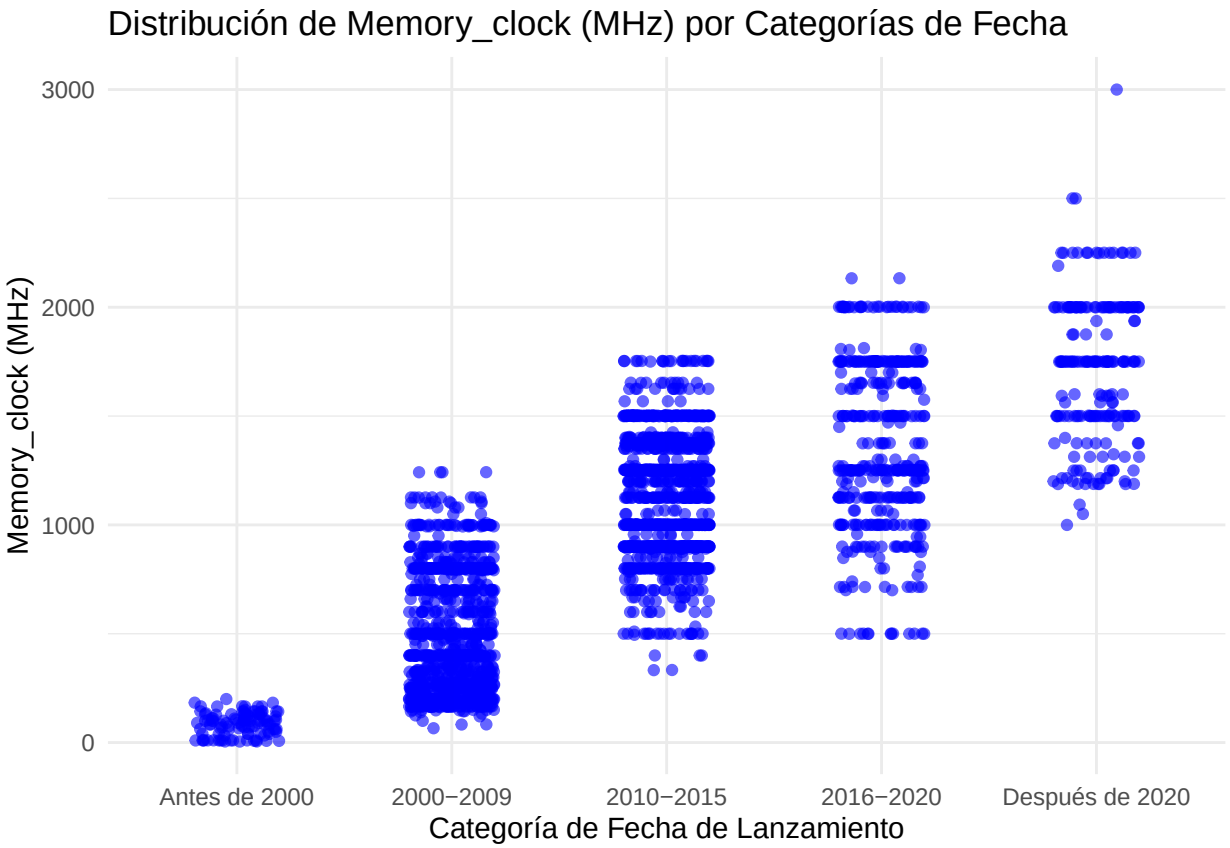

Figura 5: Valores atípicos de Memory_clock (MHz)

```
## $valores_atipicos
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2248    2250    2250    2319    2250    3000
##
```

```
## $media_variable
## [1] 957.9084
##
## $num_atipicos
## # A tibble: 1 x 2
##   Released            n
##   <fct>           <int>
## 1 Después de 2020    18
```

Para la velocidad del reloj podemos observar el mismo patrón donde la media de los valores atípicos es bastante mayor que la media de la variable, y tenemos valores atípicos en tarjetas gráficas modernas, la mayoría después de 2020 y algunas entre 2016-2020.

Resultados:

- Los valores atípicos representan un patrón similar, con una media significativamente mayor que la media general, con la mayoría de estos valores correspondientes a tarjetas gráficas lanzadas después de 2020.

4. **Shaders, TMUs y ROPs**:

```
identificar_atipicos(datos, "Shaders")
```

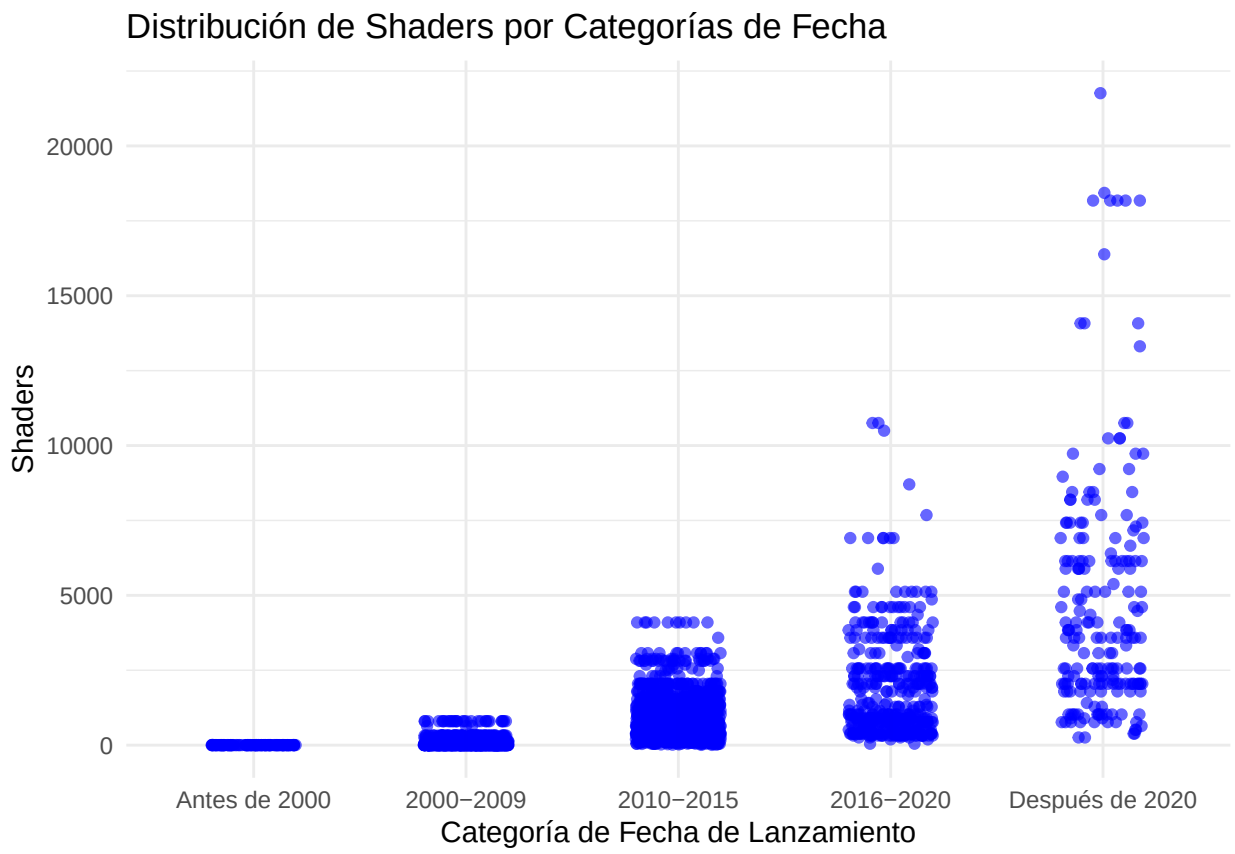

Figura 6: Valores atípicos de Shaders

```
## $valores_atipicos
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2880    3584    4224    5594    6208   21760
##
## $media_variable
## [1] 984.7594
##
## $num_atipicos
## # A tibble: 3 x 2
##   Released            n
##   <fct>           <int>
## 1 2010-2015          34
## 2 2016-2020          89
## 3 Después de 2020   113
```

```
identificar_atipicos(datos, "TMUs")
```

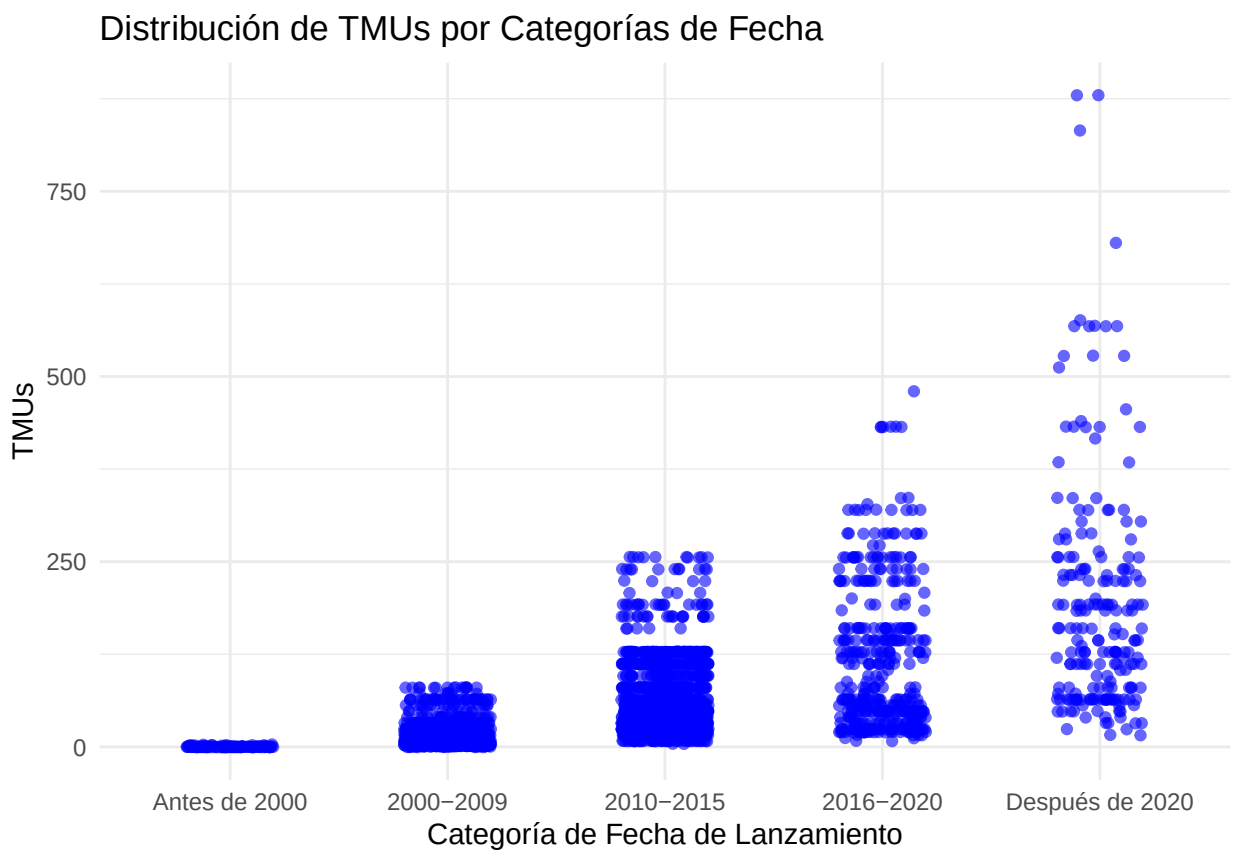

Figura 7: Valores atípicos de TMUs

```
## $valores_atipicos
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   192.0   224.0   256.0   283.3   288.0   880.0
##
## $media_variable
## [1] 60.25205
##
## $num_atipicos
## # A tibble: 3 x 2
##   Released            n
##   <fct>           <int>
## 1 2010-2015          48
## 2 2016-2020          86
## 3 Después de 2020    84
```

```r
identificar_atipicos(datos, "ROPs")
```

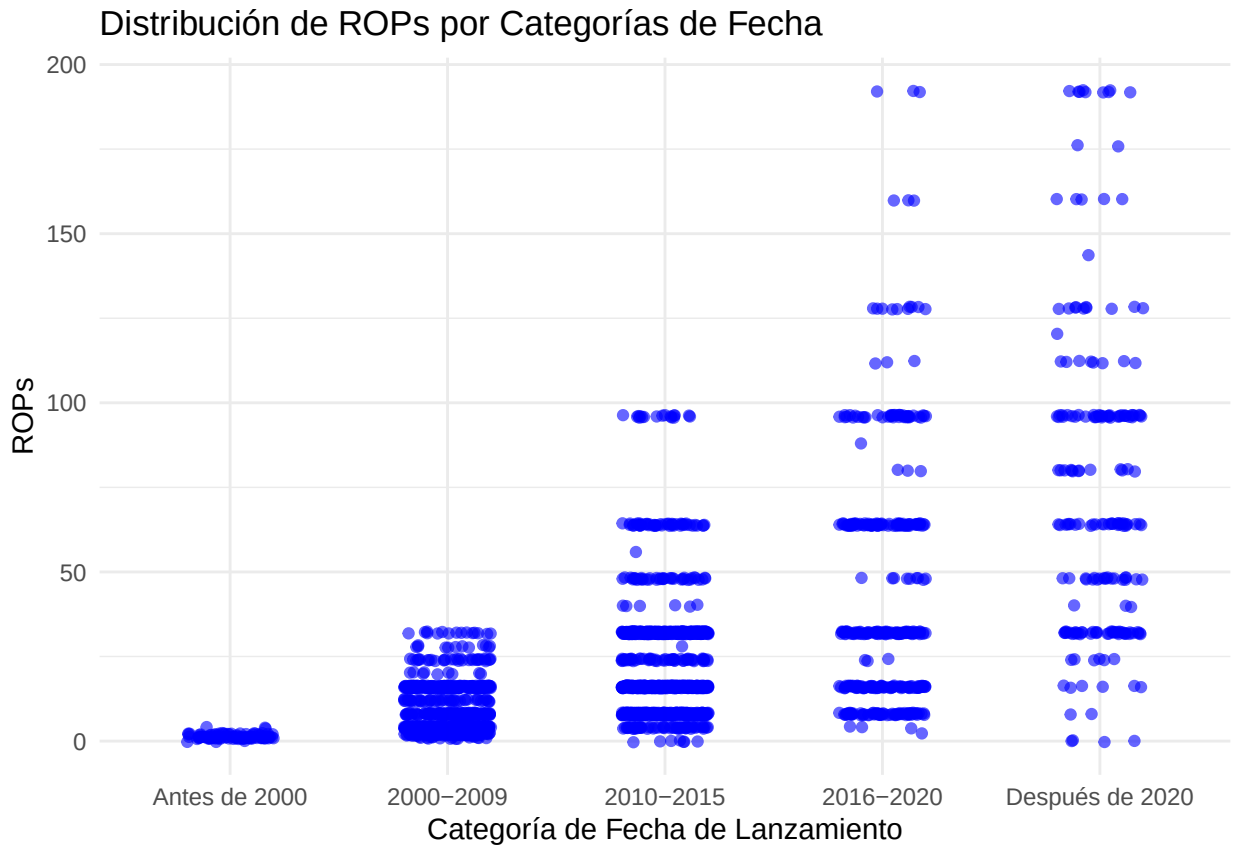

Figura 8: Valores atípicos de ROPs

```
## $valores_atipicos
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    80.0    96.0    96.0   110.6   128.0   192.0
##
## $media_variable
## [1] 23.3597
##
## $num_atipicos
## # A tibble: 3 x 2
##   Released            n
##   <fct>           <int>
## 1 2010-2015          17
## 2 2016-2020          63
## 3 Después de 2020    88
```

Para Shaders, TMUs y ROPs vemos el mismo patrón, donde la media de los valores atípicos es bastante mayor que

la media de la variable, y tenemos valores atípicos en las tarjetas más modernas, aunque aquí sí que tenemos valores atípicos en tarjetas gráficas más antiguas entre 2010 y 2015, aunque son muy pocos.

Resultados:

- Los valores atípicos en estas variables también corresponden principalmente a tarjetas gráficas modernas. Sin embargo, se identificaron algunos valores atípicos en tarjetas lanzadas entre 2010 y 2015, aunque son pocos.

5. **Memory_Bus (bits)**:

```
identificar_atipicos(datos, "Memory_Bus (bits)")
```

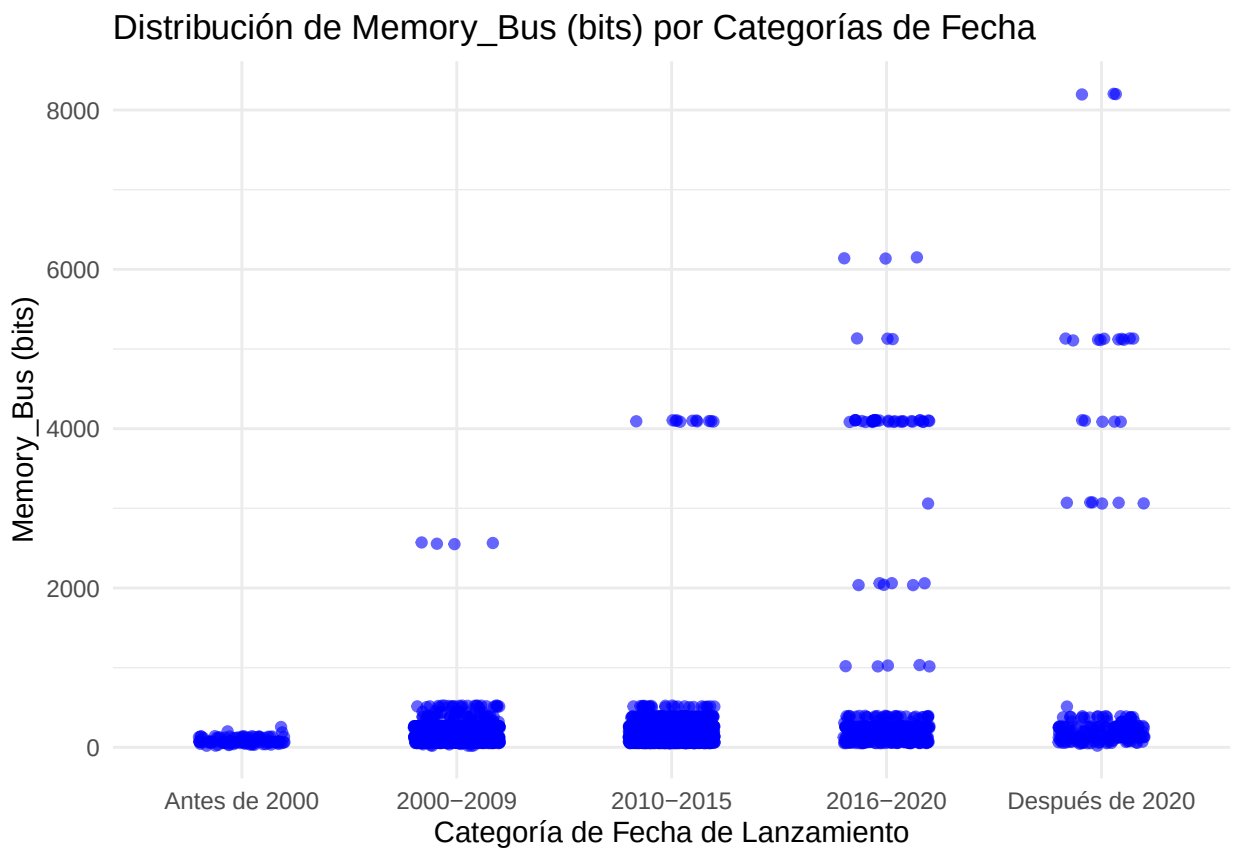

Figura 9: Valores atípicos de Memory_Bus (bits)

```
## $valores_atipicos
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     512     512    3072    2709    4096    8192
##
## $media_variable
## [1] 289.6278
```

```
##
## $num_atipicos
## # A tibble: 4 x 2
##   Released            n
##   <fct>           <int>
## 1 2000-2009          30
## 2 2010-2015          36
## 3 2016-2020          50
## 4 Después de 2020    25
```

Para esta variable observamos un patrón un poco diferente, ya que tenemos valores atípicos en casi todas las categorías de fechas. Habría que investigar más a fondo esta variable.

Para ello vamos a profundizar en lo que es el ancho de bus de memoria:

El ancho de bus de memoria se encarga de determinar cuántos bits de datos pueden ser transferidos entre la memoria y la GPU en un ciclo de reloj. Un bus de memoria alto puede ser más útil en tareas que requieren un alto ancho de banda, como la computación científica, la inteligencia artificial o el procesamiento de gráficos en tiempo real, donde es importante transferir grandes cantidades de datos rápidamente.

Teniendo en cuenta que las GPUs, en su mayoría se utilizan para juegos, podría indicar que al haber valores muy altos de ancho de bus de memoria para todas las categorías de fecha, hay ciertas tarjetas gráficas diseñadas para tareas específicas que requieren un alto ancho de banda, lo que podría explicar la presencia de valores atípicos en todas las categorías de fecha.

Vamos a visualizar un gráfico que muestre el reparto de Memory_Bus (bits) por tipo de memoria, para ver si hay algún patrón en los valores atípicos:

```r
# Gráfico de dispersión de Memory_Bus (bits) por tipo de memoria
ggplot(datos, aes(x = Memory_Type, y = `Memory_Bus (bits)`)) +
  geom_boxplot() +
  labs(
    title = "Distribución de Ancho de Bus de Memoria por Tipo de Memoria",
    x = "Tipo de Memoria",
    y = "Ancho de Bus de Memoria (bits)"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

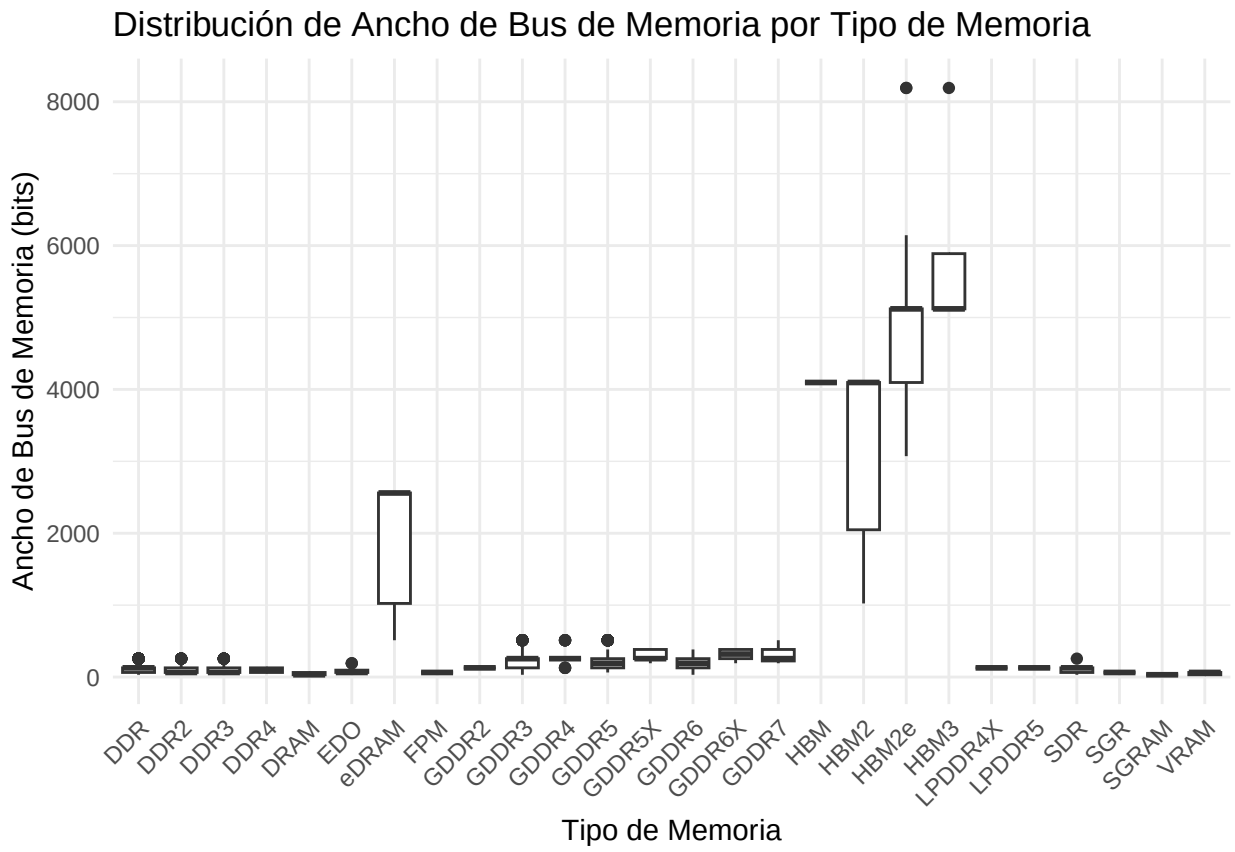

Figura 10: Distribución de Memory_Bus (bits) por Tipo de Memoria

Se ve claramente que las tecnologías HBM, HBM2 y HBM2e tienen un ancho de bus de memoria más alto que las demás tecnologías. Esto es coherente con el propósito de la memoria HBM (High Bandwidth Memory), diseñada específicamente para ofrecer un mayor ancho de banda. Como resultado, estas tarjetas son ideales para aplicaciones que requieren una alta capacidad de transferencia de datos, como la computación científica, la inteligencia artificial y el aprendizaje profundo.

Resultados:

- A diferencia de las demás variables, los valores atípicos en esta variable están presentes en casi todas las categorías de fechas. Esto sugiere que el ancho del bus de memoria puede variar significativamente según el diseño de la tarjeta gráfica, independientemente de su antigüedad y por lo tanto no son errores.

3.3.4 Conclusiones del análisis exploratorio

1. **Valores atípicos**:

- La mayoría de los valores atípicos en las variables numéricas corresponden a tarjetas gráficas modernas, lanzadas principalmente después de 2020. Esto es consistente con la evolución tecnológica y el aumento de especificaciones en las tarjetas gráficas más recientes. Por tanto, estos valores no se considerarán como atípicos en el sentido tradicional, ya que representan avances tecnológicos en lugar de errores o anomalías.
- En el caso de Memory_Bus (bits), se observaron valores atípicos en casi todas las categorías de fechas, pero esto puede explicarse por el diseño específico de algunas tarjetas gráficas que requieren un alto ancho de banda para aplicaciones especializadas. Por ejemplo, las tecnologías de memoria HBM, están diseñadas para maximizar el ancho de banda. Estas tarjetas gráficas suelen priorizar el rendimiento en la transferencia de datos, lo que explica los valores significativamente más altos en el ancho de bus de memoria.

2. **Tendencias en el lanzamiento de tarjetas gráficas**:

- En la **Figura 1** se observaba un aumento significativo en el número de lanzamientos de tarjetas gráficas entre 2000 y 2015, seguido de una disminución entre 2016 y 2020, y un nuevo aumento después de 2020.

3. **Calidad de datos**:

- No se encontraron valores perdidos en el conjunto de datos, lo que sugiere una buena calidad de datos en general.
- Los valores atípicos parecen ser representativos de la evolución tecnológica y diferencias en el diseño y aplicación de las diferentes tarjetas gráficas.

3.4 Relaciones entre variables clave

Parece interesante investigar si existen relaciones entre las diferentes variables para ver si podemos encontrar patrones o tendencias en los datos.

3.4.1 Correlación entre variables numéricas

Para investigar las relaciones entre las variables numéricas, se generó una matriz de correlación. Este análisis permite identificar las asociaciones más fuertes entre las variables clave:

```
# Crear la matriz de correlación
correlation_matrix <- cor(datos %>% select(where(is.numeric)),
                          use = "pairwise.complete.obs")
correlation_matrix
```

```
##                     Memory_Size (GB) Memory_Bus (bits)   Shaders      TMUs
## Memory_Size (GB)          1.0000000        0.63647622 0.7360566 0.8155536
## Memory_Bus (bits)         0.6364762        1.00000000 0.4721906 0.6242267
## Shaders                   0.7360566        0.47219056 1.0000000 0.9161432
## TMUs                      0.8155536        0.62422670 0.9161432 1.0000000
## ROPs                      0.6079784        0.45209917 0.8330208 0.8535990
## Memory_clock (MHz)        0.3828492        0.04784241 0.5043913 0.5532427
## GPU_clock (MHz)           0.4034626        0.15275376 0.5236997 0.5461143
##                            ROPs Memory_clock (MHz) GPU_clock (MHz)
## Memory_Size (GB)   0.6079784         0.38284920       0.4034626
## Memory_Bus (bits)  0.4520992         0.04784241       0.1527538
## Shaders            0.8330208         0.50439129       0.5236997
## TMUs               0.8535990         0.55324266       0.5461143
## ROPs               1.0000000         0.57651257       0.5919399
## Memory_clock (MHz) 0.5765126         1.00000000       0.8211419
## GPU_clock (MHz)    0.5919399         0.82114193       1.0000000
```

Las relaciones más fuertes las podemos encontrar obviamente entre Shaders, TMUs y ROPs, ya que son variables que están relacionadas entre sí por naturaleza. Otras relaciones interesantes son entre el tamaño de la memoria y las unidades de sombreado y las unidades de mapeo de texturas. También tenemos una relación fuerte de esta variable con el ancho del bus de memoria, lo que tiene sentido, ya que a mayor tamaño de memoria, mayor ancho de bus.

Otra relación muy fuerte es entre la velocidad del reloj de la GPU y la velocidad del reloj de la memoria, lo que también tiene sentido, ya que a mayor velocidad de reloj, mayor rendimiento.

Principales hallazgos:

- **Shaders, TMUs y ROPs**: Estas variables están fuertemente correlacionadas entre sí, lo que es esperado, ya que estas variables están relacionadas por diseño y representan diferentes aspectos del procesamiento gráfico.
- **Memory_Size (GB)**: Esta variable muestra una correlación positiva con Shaders, TMUs, ROPs y Memory_Bus (bits). Esto tiene sentido, ya que con mayor capacidad de memoria, se pueden manejar más unidades de sombreado y un mayor ancho de bus.
- **Memory_Bus (bits)**: Tiene una correlación positiva con Memory_Size (GB) y Shaders, lo que indica que un mayor ancho de bus está asociado con un mayor tamaño de memoria y un mayor número de unidades de sombreado.
- **GPU_clock (MHz)** y **Memory_clock (MHz)**: Estas variables están fuertemente correlacionadas entre sí, lo que sugiere que a medida que aumenta la velocidad del reloj de la GPU, también aumenta la velocidad del reloj de la memoria.

3.4.2 Visualización de relaciones clave

Relación entre el tamaño de memoria y el ancho de bus de memoria

Para explorar la relación entre el tamaño de memoria y el ancho de bus de memoria, se generó un gráfico de dispersión agrupado por el tipo de memoria:

```
# Relación entre Memory_Size (GB) y Memory_clock (MHz), coloreado por GPU_Chip
ggplot(datos, aes(x = `Memory_Size (GB)`, y = `Memory_Bus (bits)`,
                  color = Memory_Type )) +
  geom_point(alpha = 0.7, size = 3) +
  labs(
    title = "Relación entre Tamaño de Memoria y Ancho de Bus",
    subtitle = "Agrupado por tipo de Memoria",
    x = "Memory_Size (GB)",
    y = "Memory_Bus (bits)"
  ) +
  theme_minimal() +
  theme(legend.position = "bottom")  # Para evitar que la leyenda oculte datos
```

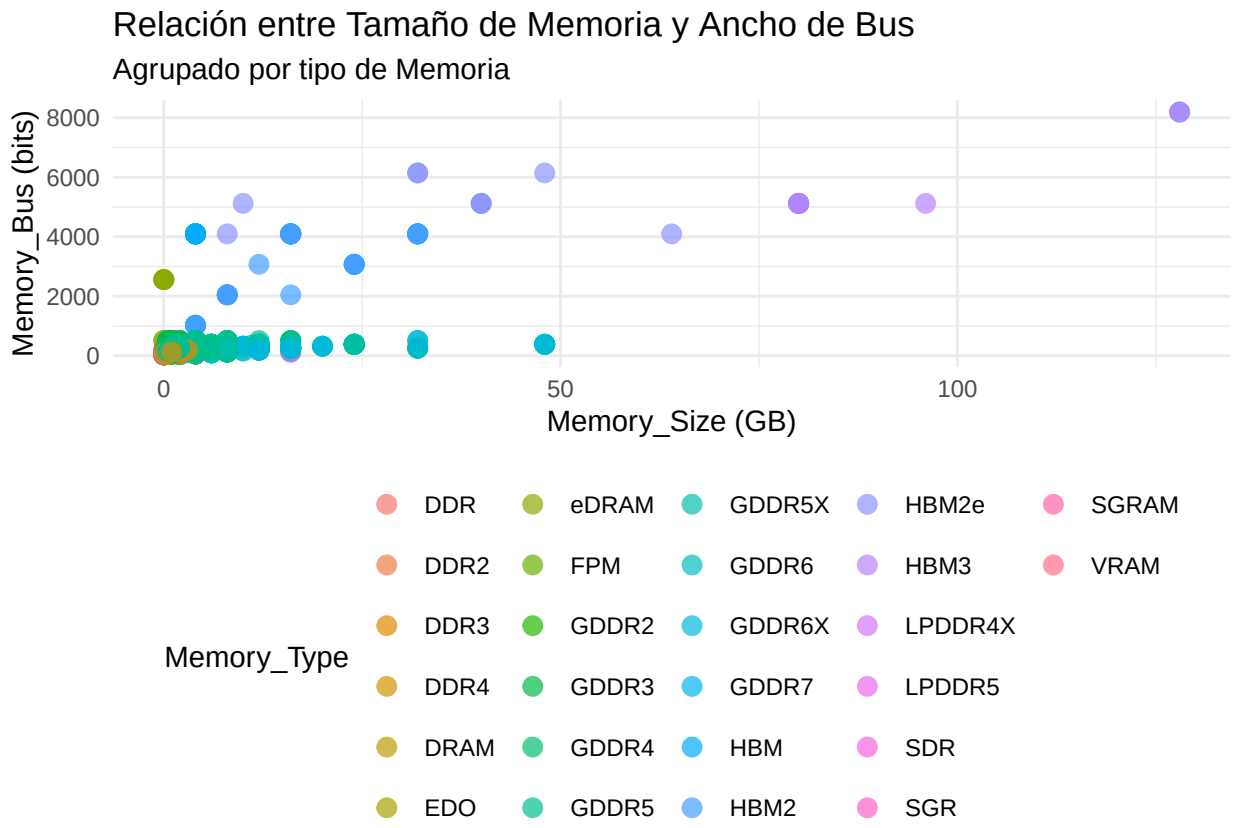

Figura 11: Relación entre Tamaño de Memoria y Ancho de Bus de Memoria

Observaciones iniciales: La **Figura 11** muestra una gran cantidad de puntos agrupados, lo que dificulta la interpretación debido a la diversidad de tipos de memoria. Para mejorar la visualización, se procederá a filtrar los tipos de memoria más comunes y se volverá a generar el gráfico:

```r
# Ver cuántos datos hay por tipo de memoria
datos %>%
  count(Memory_Type, sort = TRUE) %>%
  arrange(desc(n)) %>%
  print(n=nrow(.))
```

```
## # A tibble: 26 x 2
##    Memory_Type     n
##    <chr>       <int>
##  1 GDDR5        1145
##  2 GDDR3         508
##  3 DDR3          424
##  4 DDR           314
##  5 GDDR6         211
##  6 DDR2          140
##  7 SDR            87
##  8 HBM2           36
##  9 GDDR5X         30
## 10 GDDR4          25
## 11 HBM            23
## 12 HBM2e          22
## 13 GDDR6X         20
## 14 DRAM           17
## 15 EDO            11
## 16 VRAM            7
## 17 eDRAM           6
## 18 DDR4            4
## 19 HBM3            4
## 20 GDDR2           3
## 21 GDDR7           3
## 22 LPDDR4X         2
## 23 SGR             2
## 24 FPM             1
## 25 LPDDR5          1
## 26 SGRAM           1
```

Viendo el reparto de los tipos de memoria, vamos a quedarnos con aquellos tipos de memoria que tengan más de 15 tarjetas gráficas.

```r
# Filtrar los tipos de memoria más comunes
tipos_memoria_comunes <- datos %>%
  count(Memory_Type, sort = TRUE) %>%
  filter(n > 15) %>%
```

```
  pull(Memory_Type)
```

A continuación, vamos a filtrar los datos para quedarnos sólo con los tipos de memoria más comunes y eliminamos el valor más alto de Memory_Size (GB) para evitar que el gráfico esté muy sesgado por un solo punto. Esto nos permitirá ver mejor la relación entre el tamaño de memoria y el ancho de bus de memoria.

```
# Filtrar los datos para quedarnos sólo con los tipos de memoria más comunes
datos_filtrados <- datos %>%
  filter(Memory_Type %in% tipos_memoria_comunes) %>%
  filter(`Memory_Size (GB)` != max(`Memory_Size (GB)`, na.rm = TRUE))

# Hacemos el gráfico de nuevo
ggplot(datos_filtrados, aes(x = `Memory_Size (GB)`, y = `Memory_Bus (bits)`,
color = Memory_Type )) +
  geom_point(alpha = 0.7, size = 3) +
  labs(
    title = "Relación entre Tamaño de Memoria y Ancho de Bus",
    subtitle = "Agrupado por tipo de Memoria",
    x = "Memory_Size (GB)",
    y = "Memory_Bus (bits)"
  ) +
  theme_minimal() +
  theme(legend.position = "bottom")  # Para evitar que la leyenda oculte datos
```

```
  coord_cartesian(xlim = c(0, 32), ylim = c(0, 1024))  # Ajustar límites de los ejes
```

```
## <ggproto object: Class CoordCartesian, Coord, gg>
##     aspect: function
##     backtransform_range: function
##     clip: on
##     default: FALSE
##     distance: function
##     expand: TRUE
##     is_free: function
##     is_linear: function
##     labels: function
##     limits: list
##     modify_scales: function
##     range: function
##     render_axis_h: function
##     render_axis_v: function
##     render_bg: function
```

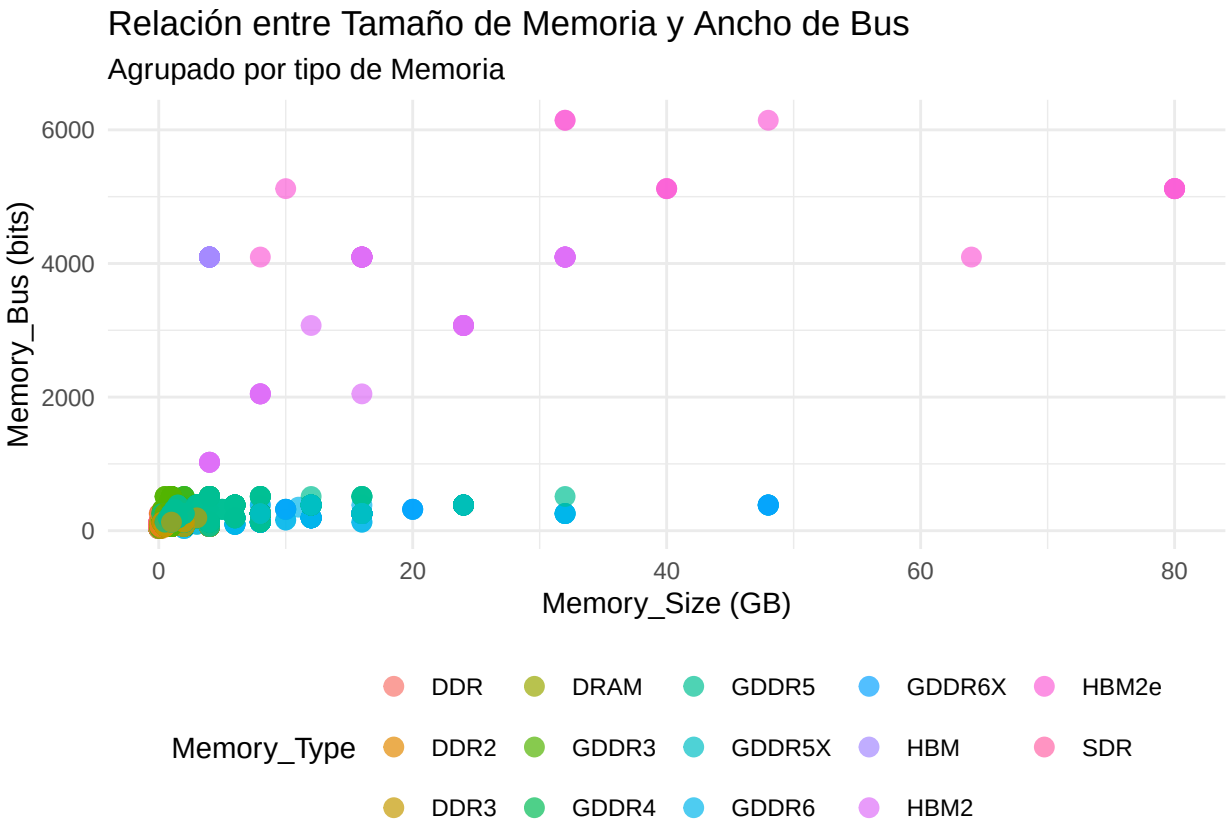

Figura 12: Relación entre Tamaño de Memoria y Ancho de Bus de Memoria (Filtrado)

```
##     render_fg: function
##     setup_data: function
##     setup_layout: function
##     setup_panel_guides: function
##     setup_panel_params: function
##     setup_params: function
##     train_panel_guides: function
##     transform: function
##     super:  <ggproto object: Class CoordCartesian, Coord, gg>
```

Este gráfico nos permite visualizar con mayor claridad las diferencias entre los distintos tipos de memoria en las tarjetas gráficas.

Interpretación:

- Las tecnologías más antiguas, como GDDR3 y DDR3, presentan un reloj de memoria y un reloj de GPU más bajos en comparación con las tecnologías más recientes.
- Las tarjetas gráficas con memoria HBM, HBM2 y HBM2e destacan por su ancho de bus significativamente más alto, aunque su tamaño de memoria es similar al de tecnologías como GDDR5X, GDDR6 y GDDR6X. Como habíamos mencionado anteriormente, esto se debe al propósito de las memorias de tipo HBM, que están diseñadas para ofrecer un mayor ancho de banda, lo que las hace ideales para aplicaciones que requieren una alta capacidad de transferencia de datos.
- Por otro lado, las tarjetas gráficas con memoria GDDR5X, GDDR6 y GDDR6X parecen priorizar un mayor tamaño de memoria, lo que resulta beneficioso para juegos y aplicaciones gráficas avanzadas que necesitan almacenar y procesar texturas y datos gráficos complejos. Esto es consistente con el enfoque de NVIDIA, la marca líder en el mercado de videojuegos, que utiliza estas tecnologías en sus tarjetas gráficas más modernas.

Relación entre el reloj de memoria y el reloj de GPU

Otra relación interesante es la existente entre el reloj de memoria y el reloj de GPU, agrupada por el tipo de chip. Dado que hay muchos tipos de chips diferentes, vamos a filtrar los tipos de chip más comunes para facilitar la visualización.

```r
# Ver cuántos datos hay por tipo de chip
datos %>%
  count(GPU_Chip, sort = TRUE) %>%
  arrange(desc(n)) %>%
  print(n = 25)
```

```
## # A tibble: 335 x 2
##    GPU_Chip       n
##    <chr>      <int>
##  1 GK104        151
##  2 GM107         81
##  3 GK107         79
```

```
##  4 Tahiti        64
##  5 G92           53
##  6 Oland         48
##  7 GM108         47
##  8 Pitcairn      47
##  9 Cape Verde    46
## 10 GK208         45
## 11 GK106         42
## 12 GM204         41
## 13 Amethyst      37
## 14 GP104         37
## 15 Tropo         35
## 16 Baffin        32
## 17 G71           32
## 18 G80           32
## 19 Meso          30
## 20 Venus         30
## 21 RV770         29
## 22 GT200B        28
## 23 Bonaire       27
## 24 Jet           27
## 25 RV670         27
## # i 310 more rows
```

```r
# Filtrar los tipos de chip más comunes
tipos_chip_comunes <- datos %>%
  count(GPU_Chip, sort = TRUE) %>%
  filter(n > 50) %>%
  pull(GPU_Chip)
```

Usaremos los tipos de chip que tengan más de 50 tarjetas gráficas.

```r
# Filtrar los datos para quedarnos sólo con los tipos de chip más comunes
datos_filtrados_chip <- datos %>%
  filter(GPU_Chip %in% tipos_chip_comunes)

# Hacemos el gráfico de nuevo
ggplot(datos_filtrados_chip, aes(x = `Memory_clock (MHz)`,
       y = `GPU_clock (MHz)`, color = GPU_Chip)) +
  geom_point(alpha = 0.7, size = 3) +
  labs(
    title = "Relación entre Reloj de Memoria y Reloj de GPU",
    subtitle = "Agrupado por tipo de Chip",
    x = "Memory_clock (MHz)",
```

```
    y = "GPU_clock (MHz)"
  ) +
  theme_minimal() +
  theme(legend.position = "bottom")  # Para evitar que la leyenda oculte datos
```

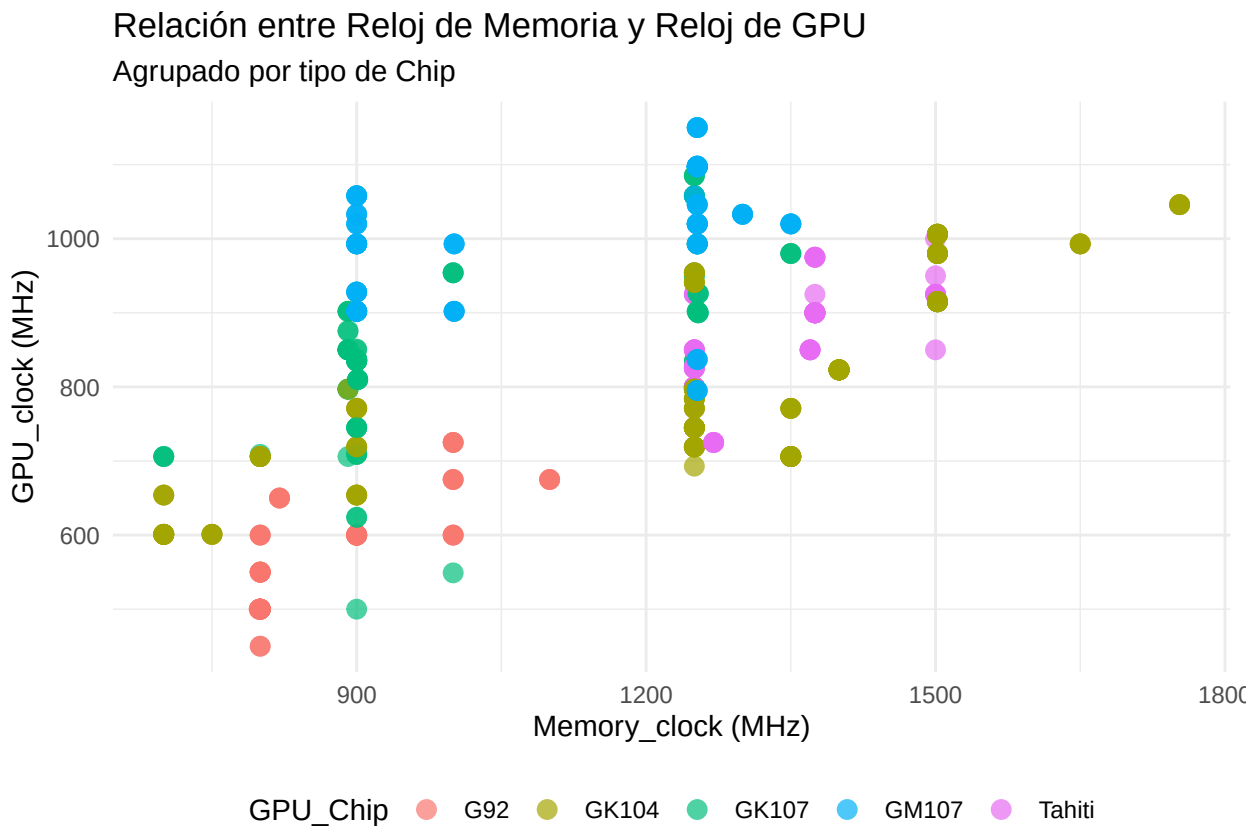

Figura 13: Relación entre Reloj de Memoria y Reloj de GPU (Filtrado)

Interpretación:

- Las tarjetas gráficas con chips GK104 y Tahiti parecen ofrecer un rendimiento superior, con relojes de memoria y GPU más altos.
- Los chips GM107 y GK107 presentan un reloj de GPU más alto, pero un reloj de memoria más bajo, lo que sugiere una arquitectura más antigua.
- Las tarjetas gráficas más modernas tienden a tener una mayor capacidad de memoria, lo que puede estar relacionado con un aumento en la frecuencia de la memoria.

3.4.3 Conclusiones del análisis de relaciones

En esta sección, se han explorado las relaciones entre las variables clave del conjunto de datos, centrándose en las variables numéricas más relevantes. A continuación, se presentan las principales conclusiones:

1. **Correlaciones**:
    - Se identificaron correlaciones significativas entre las variables numéricas, especialmente entre Shaders, TMUs y ROPs, así como entre Memory_Size (GB) y Memory_Bus (bits).
    - La velocidad del reloj de la GPU y la velocidad del reloj de la memoria también mostraron una fuerte correlación, lo que sugiere que a medida que aumenta la velocidad del reloj de la GPU, también lo hace la velocidad del reloj de la memoria.
2. **Visualización de relaciones clave**:
    - Relación entre el tamaño de memoria y el ancho de bus de memoria (agrupado por tipo de memoria): (**Figura 12**)
        - Se observó que las tecnologías de memoria HBM, HBM2 y HBM2e tienen un ancho de bus significativamente más alto en comparación con otras tecnologías, lo que es coherente con su propósito de ofrecer un mayor ancho de banda.
        - Las tarjetas gráficas con memoria GDDR5X, GDDR6 y GDDR6X tienden a priorizar un mayor tamaño de memoria, lo que es beneficioso para juegos y aplicaciones gráficas avanzadas.
    - Relación entre el reloj de memoria y el reloj de GPU (agrupado por tipo de chip): (**Figura 13**)
    - Las tarjetas gráficas con chips GK104 y Tahiti destacaron por ofrecer un rendimiento superior, con relojes de memoria y GPU más altos.
    - Los chips GM107 y GK107 mostraron un reloj de GPU más alto, pero un reloj de memoria más bajo, lo que sugiere una arquitectura más antigua.

4 Clustering

4.1 Fundamentos teóricos del método

Vamos a realizar un análisis clustering para ver si podemos agrupar las tarjetas gráficas según sus especificaciones técnicas. Para ello, vamos a usar el método de k-means.

A continuación, una breve descripción de qué es clustering y cómo funciona el método de k-means, extraído del manual: “An Introduction to Statistical Learning with aplications in R” (Manual).

El clustering (o agrupamiento) es una técnica de aprendizaje no supervisado que busca identificar subgrupos o “clústeres” dentro de un conjunto de datos. El objetivo es agrupar observaciones de manera que aquellas dentro de un mismo clúster sean similares entre sí, mientras que las observaciones de diferentes clústeres sean distintas. Para lograr esto, es necessario definir qué significa que dos observaciones sean similares o distintas, lo cual depende del contexto y del conocimiento del dominio.

Por ejemplo, en un estudio de muestras de tejido de pacientes con cáncer de mama, donde cada muestra tiene varias características (como medidas clínicas o expresiones génicas), el clustering podría ayudar a identificar subtipos desconocidos de la enfermedad. Este es un problema no supervisado, ya que no se cuenta con etiquetas predefinidas, a diferencia de los problemas supervisados, donde el objetivo es predecir un resultado específico.

En el artículo "A Review of Clustering Techniques and Developments" (Saxena et al.), se menciona que no existe una definición precisa de lo que constituye un "clúster", lo que ha llevado al desarrollo de diferentes enfoques y técnicas de clustering. Algunos sugieren que las técnicas de clustering pueden dividirse en dos grandes categorías: jerárquicas y de partición. Otros proponen categorías adicionales, como métodos basados en densidad, métodos basados en modelos y métodos basados en cuadrículas.

Cada enfoque utiliza un principio de inclusión diferente para definir los clústeres, y la elección del método adecuado depende del tipo de datos y del objetivo del análisis. Además, el número de clústeres en los que se divide un conjunto de datos suele ser decidido por el usuario, utilizando métodos heurísticos, de prueba y error, o enfoques evolutivos. La precisión del clustering depende en gran medida de esta decisión, ya que un número adecuado de clústeres maximiza la similitud intra-clúster y minimiza la similitud inter-clúster.

(Saxena et al.) menciona dos enfoques principales: jerárquico y particional.

1. **Clustering jerárquico**: Este enfoque puede ser aglomerativo (comenzando con cada punto como un clúster individual y fusionándolos) o divisivo (comenzando con un único clúster y dividiéndolo). Dentro de estas categorías, se utilizan métodos como el enlace simple, completo o promedio. Ejemplos de algoritmos incluyen BIRCH, CURE, ROCK y CHAMELEON.

Como se menciona en el artículo "Hierarchical Clustering Algorithms for Document Datasets" (Zhao et al.), el clustering jerárquico es especialmente útil en aplicaciones donde los datos tienen una estructura jerárquica natural, como taxonomías biológicas o árboles filogenéticos. Este enfoque genera dendrogramas, que son representaciones visuales que permiten explorar los datos en diferentes niveles de granularidad. Esto resulta ideal para tareas como la organización de grandes colecciones de documentos, donde los dendrogramas facilitan la navegación y exploración interactiva.

Además, los métodos jerárquicos son útiles en casos donde los clústeres tienen subclústeres, proporcionando una representación consistente y predecible de los datos. Por ejemplo, en el análisis de textos, los dendrogramas pueden organizar documentos en categorías generales y subcategorías más específicas, lo que mejora la comprensión y el análisis de grandes volúmenes de información.

2. **Clustering particional**: Este enfoque divide los datos en un número predefinido de clústeres. Puede basarse en distancias, modelos probabilísticos o densidad. Ejemplos de algoritmos incluyen K-means, PAM, CLARA, CLARANS, DBSCAN y CLIQUE.

Entre los métodos de clustering particional, k-means es uno de los más utilizados, y es el que utilizaremos en este análisis. Algunos ejemplos de su uso son por ejemplo en (Neshat et al.), donde se utiliza para identificar valores atípicos dentro de cada clúster o en (Oyelade et al.), donde se ha usado para agrupar diferentes estudiantes en función de sus resultados académicos.

Esta clasificación resalta la diversidad de técnicas disponibles, cada una adecuada para diferentes tipos de datos y objetivos analíticos.

De entre los diferentes métodos de clustering, nosotros usaremos el método k-means.

4.2 Clustering con K-means

El método K-means es una técnica de agrupamiento que busca dividir un conjunto de datos en un número predefinido de grupos o clústeres. Su objetivo principal es organizar las observaciones de manera que las que pertenecen a un mismo clúster sean lo más similares posible entre sí, mientras que las observaciones de diferentes clústeres sean lo más distintas posible. Para lograr esto, K-means utiliza un enfoque iterativo que ajusta continuamente los grupos hasta encontrar una solución óptima.

4.2.1 Algoritmo K-means

Como bien se explica en (Manual), el algoritmo K-means sigue los siguientes pasos:

1. **Inicialización**: Se asigna aleatoriamente un número del 1 al K a cada observación, lo que sirve como asignación inicial de clústeres.

$$C_1, C_2, \dots, C_k \in \mathbb{R}^p$$

   donde C_i es el i-ésimo centroide y p es el número de variables.

2. **Iteración**: Se repiten los siguientes pasos hasta que las asignaciones de clústeres no cambien:

(a) Para cada uno de los (k) clústeres, se calcula el centroide como el vector de medias de las observaciones asignadas:

$$C_j = \frac{1}{|S_j|} \sum_{x_i \in S_j} x_i$$

   donde S_j es el conjunto de observaciones asignadas al clúster (j) y $|S_j|$ su tamaño.

(b) Para cada observación x_i, se calcula la distancia a cada centroide y se asigna al clúster más cercano:

$$x_i \in S_{c^*} \quad \text{donde} \quad c^* = \arg\min_{j=1,\dots,k} d(x_i, C_j)$$

donde $d(x_i, C_j)$ es la distancia entre x_i y el centroide C_j.

El algoritmo K-means garantiza que el valor de la función objetivo, que mide la variación intra-clúster total, disminuya en cada iteración. Esto se debe a que cada paso del algoritmo está diseñado para mejorar la asignación de clústeres o los centroides:

1. **Cálculo de los centroides (Paso 2(a))**: En este paso, los centroides de los clústeres se calculan como las medias de las observaciones asignadas a cada clúster. Este cálculo minimiza la suma de las desviaciones cuadradas dentro de cada clúster, ya que las medias son los valores que minimizan esta suma.

2. **Reasignación de observaciones (Paso 2(b))**: Cada observación se reasigna al clúster cuyo centroide esté más cerca. Este proceso asegura que cada observación esté asignada al clúster que minimiza su contribución a la variación intra-clúster.

En cada iteración, estos dos pasos reducen la variación intra-clúster total, ya que:

- El cálculo de los centroides minimiza las desviaciones dentro de cada clúster.
- La reasignación de observaciones mejora la asignación de clústeres, reduciendo aún más la variación.

Por lo tanto, el algoritmo K-means siempre converge a un óptimo local, ya que la función objetivo no puede aumentar en ninguna iteración. Sin embargo, debido a que el algoritmo encuentra un óptimo local, los resultados pueden depender de la asignación inicial de los clústeres. Por esta razón, es recomendable ejecutar el algoritmo varias veces con diferentes inicializaciones y seleccionar la solución con el valor más bajo de la función objetivo.

En (Singh et al.) se presentan diferentes métricas para calcular la distancia entre observaciones y centroides, entre las que destacan la distancia euclidiana, la distancia de Manhattan, la distancia de Chebychev y la distancia de Minkowski.

- **Distancia euclidiana**: Se calcula como la raíz cuadrada de la suma de las diferencias al cuadrado entre las coordenadas de dos puntos. Es la métrica más comúnmente utilizada en el algoritmo K-means debido a su simplicidad y a que funciona bien en datos donde las relaciones entre las variables son lineales.
- **Distancia de Manhattan**: Se calcula como la suma de las diferencias absolutas entre las coordenadas de dos puntos. Es útil cuando los datos tienen una estructura en forma de cuadrícula o cuando las relaciones entre las variables no son lineales. Por ejemplo, se utiliza en análisis de redes o en datos geográficos.
- **Distancia de Chebychev**: Se calcula como la máxima diferencia absoluta entre las coordenadas de dos puntos. Es adecuada cuando se desea priorizar la dimensión con la mayor diferencia, como en problemas donde el peor caso es más relevante que la suma total de diferencias.
- **Distancia de Minkowski**: Es una generalización de las distancias euclidiana y de Manhattan, y se calcula como la raíz p-ésima de la suma de las diferencias absolutas elevadas a la p-ésima potencia. Es útil cuando se desea ajustar el parámetro p para controlar la sensibilidad a las diferencias en las dimensiones. Por ejemplo, con $p = 1$ se obtiene la distancia de Manhattan, y con $p = 2$ se obtiene la distancia euclidiana.

La elección de la métrica de distancia depende del contexto del problema y de las características de los datos. En este análisis, como habíamos mencionado en los objetivos, vamos a utilizar por una parte la distancia euclidiana, que es la más comúnmente utilizada en el algoritmo K-means, y por otra parte la distancia de Minkowski, que es útil cuando hay muchos valores atípicos.

Matemáticamente, la variación intra-clúster para un clúster C_k se calcula como:

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j}^2)$$

Donde:

- $|C_k|$ es el número de observaciones en el clúster C_k.
- x_{ij} es el valor de la j-ésima variable para la i-ésima observación.

El objetivo del algoritmo es minimizar la suma total de $W(C_k)$ para todos los clústeres K:

$$\text{Minimizar} \sum_{k=1}^{K} W(C_k)$$

En el contexto de este análisis, usaremos K-means para agrupar tarjetas gráficas según sus especificaciones técnicas (como memoria, velocidad de reloj, etc.) y así identificar patrones o categorías naturales.

4.3 Aplicación del método K-means

En este análisis, aplicaremos el método K-means para identificar patrones en las tarjetas gráficas basándonos en sus especificaciones técnicas. El objetivo es agrupar las tarjetas gráficas en clústeres que compartan características similares, lo que nos permitirá identificar categorías naturales dentro de los datos.

Primero, aplicaremos el método K-means utilizando únicamente las variables numéricas. Este análisis nos permitirá identificar patrones basados exclusivamente en las especificaciones técnicas cuantitativas de las tarjetas gráficas. A continuación, aplicaremos el método K-means utilizando tanto las variables numéricas como las categóricas. Este enfoque nos permitirá identificar si existen patrones adicionales al considerar las variables categóricas, como el tipo de memoria o el fabricante, junto con las especificaciones técnicas.

Finalmente, compararemos los resultados obtenidos de ambos enfoques para evaluar si la inclusión de variables categóricas mejora la identificación de patrones en las tarjetas gráficas.

4.3.1 Clustering con variables numéricas

En este apartado, aplicaremos el método K-means utilizando únicamente las variables numéricas seleccionadas. Este análisis nos permitirá identificar patrones basados exclusivamente en las especificaciones técnicas cuantitativas de las tarjetas gráficas.

4.3.1.1 Estandardización de los datos

Es importante estandarizar los datos antes de aplicar el método K-means, ya que este algoritmo es sensible a la escala de las variables. La estandarización transforma los datos para que tengan una media de 0 y una desviación estándar de 1, lo que permite que todas las variables contribuyan de manera equitativa al cálculo.

Para ello, primero vamos a seleccionar las variables numéricas relevantes para el clustering. En este caso, vamos a usar las siguientes variables:

- **Memory_Size (GB)**
- **Memory_Bus (bits)**
- **Shaders, TMUs y ROPs**
- **Memory_clock (MHz)**
- **GPU_clock (MHz)**

```r
# Seleccionar las variables numéricas relevantes para el clustering
variables_clustering <- datos %>%
  select(`Memory_Size (GB)`, `Memory_Bus (bits)`, Shaders, TMUs, ROPs,
         `Memory_clock (MHz)`, `GPU_clock (MHz)`)

# Escalar los datos para normalizar las variables
variables_clustering_scaled <- scale(variables_clustering)

# Verificar los datos escalados
summary(variables_clustering_scaled)
```

```
##  Memory_Size (GB)   Memory_Bus (bits)     Shaders              TMUs
##  Min.   :-0.43884   Min.   :-0.37302   Min.   :-0.56361   Min.   :-0.7510
##  1st Qu.:-0.37546   1st Qu.:-0.23403   1st Qu.:-0.54072   1st Qu.:-0.6513
##  Median :-0.18529   Median :-0.23403   Median :-0.34384   Median :-0.3522
##  Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.0000
##  3rd Qu.: 0.06825   3rd Qu.:-0.04869   3rd Qu.: 0.09572   3rd Qu.: 0.2462
##  Max.   :15.78824   Max.   :11.44203   Max.   :11.89044   Max.   :10.2180
##       ROPs         Memory_clock (MHz) GPU_clock (MHz)
##  Min.   :-0.8659   Min.   :-1.9464    Min.   :-2.02622
##  1st Qu.:-0.5693   1st Qu.:-0.7311    1st Qu.:-0.64506
##  Median :-0.2728   Median :-0.1142    Median :-0.02218
##  Mean   : 0.0000   Mean   : 0.0000    Mean   : 0.00000
##  3rd Qu.: 0.3203   3rd Qu.: 0.6028    3rd Qu.: 0.53028
##  Max.   : 6.2508   Max.   : 4.1712    Max.   : 4.73063
```

4.3.1.2 Determinación del número óptimo de clústeres (K)

Para determinar el número óptimo de clústeres (K) en el método K-means, utilizamos el **método del codo** según (Cui et al.), una técnica que evalúa la calidad del agrupamiento en función de la variación intra-clúster. Este método se basa en la métrica **WCSS (Within-Cluster Sum-of-Squares)**, que mide la suma de las distancias al cuadrado entre cada punto y el centroide de su clúster. Cuanto menor sea el valor de WCSS, mejor será la compactación de los clústeres.

Funcionamiento del método del codo:

1. **Cálculo de WCSS para diferentes valores de K**: Se ejecuta el algoritmo K-means para diferentes valores de K y se calcula el WCSS correspondiente. Inicialmente, con $K = 1$, el valor de WCSS es alto, ya que todos los puntos pertenecen a un único clúster. A medida que K aumenta, WCSS disminuye porque los puntos están más cerca de sus centroides.

2. **Gráfica de WCSS vs. K**: Se grafica el valor de WCSS en función de K. En el eje x se representan los valores de K, y en el eje y, los valores de WCSS.

3. **Identificación del "codo"**: El "codo" de la gráfica es el punto donde la disminución de WCSS comienza a estabilizarse. Este punto indica el número óptimo de clústeres, ya que agregar más clústeres después de este punto no mejora significativamente la compactación.

La métrica WCSS se calcula como:

$$WCSS = \sum_{k=1}^{K} \sum_{i \in C_k} \text{distancia}(P_i, C_k)^2$$

Donde:

- C_k es el centroide del clúster k.
- P_i es un punto de datos en el clúster k.
- La distancia se calcula típicamente utilizando la métrica euclidiana.

En nuestro análisis, utilizaremos el método del codo para determinar el número óptimo de clústeres antes de aplicar el algoritmo K-means.

Para graficar los resultados del método del codo y visualizar los clústeres obtenidos, utilizaremos la librería `factoextra`. Esta librería proporciona herramientas para visualizar e interpretar resultados de análisis multivariado, incluyendo métodos de clustering como K-means.

La función `fviz_nbclust` de `factoextra` permite calcular y graficar la métrica WCSS (Within-Cluster Sum-of-Squares) para diferentes valores de K, ayudándonos a identificar el número óptimo de clústeres.

```r
library(factoextra)


# Calcular la variación intra-clúster para diferentes valores de K
fviz_nbclust(variables_clustering_scaled, kmeans, method = "wss") +
  labs(title = "Método del Codo para Determinar K",
       x = "Número de Clústeres (K)",
       y = "Suma de Distancias al Cuadrado (WCSS)")
```

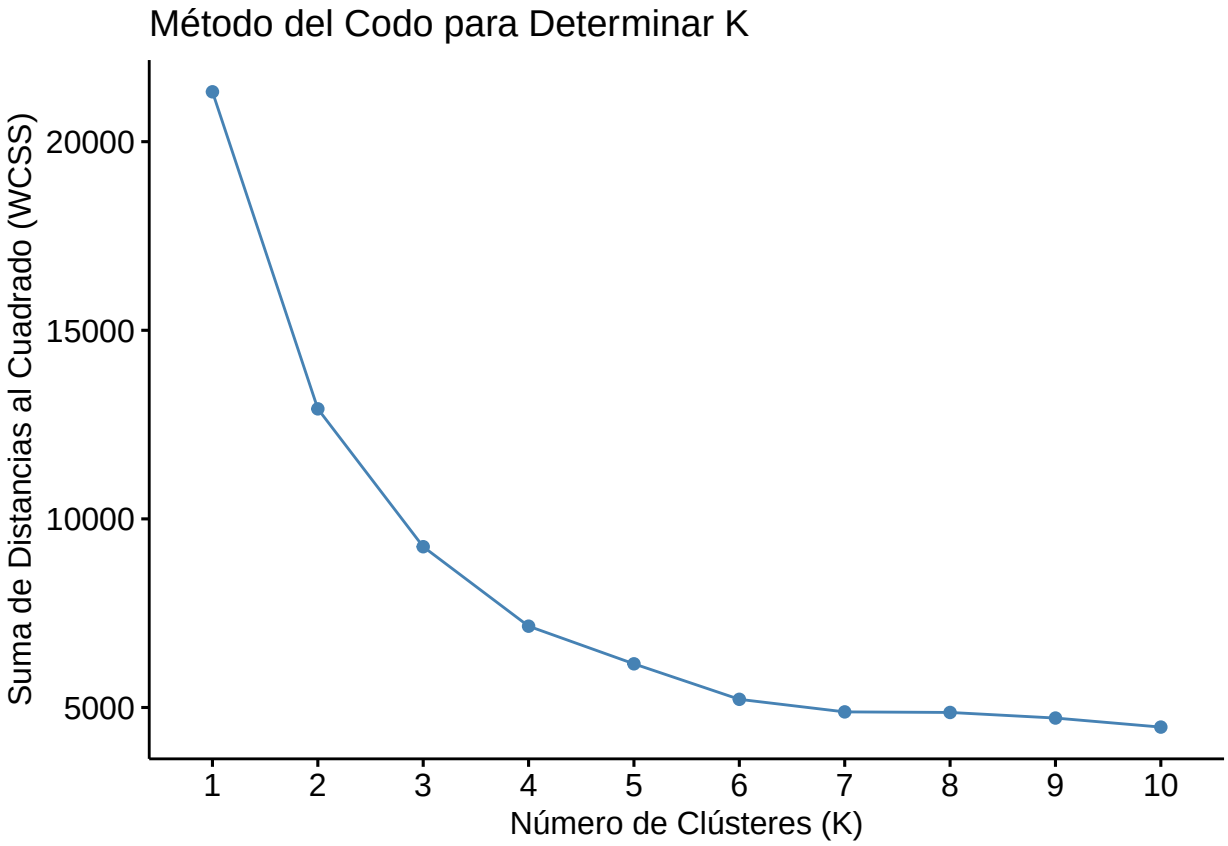

Figura 14: Método del Codo para Determinar K

Vemos que el codo está entre k=5 y k=6. Primero vamos a probar con k=5 y luego con k=6.

4.3.1.3 Aplicación del método K-means

Vamos a aplicar el método K-means para k=5 y k=6. Para ello, vamos a usar la función `kmeans` de R. La función `kmeans` toma como argumentos los datos, el número de clústeres (K) y el número de inicios aleatorios (nstart). El número de inicios aleatorios se usa para evitar que el algoritmo se quede atrapado en un mínimo local.

En el manual de (Manual) se recomienda usar un valor de nstart entre 20 y 50. En este caso, vamos a usar nstart=25.

Para graficar los clústeres obtenidos vamos a usar la función `fviz_cluster` de la librería `factoextra`, que permite visualizar los resultados del clustering de manera intuitiva.

Esta representación gráfica muestra los clústeres obtenidos al aplicar el método K-means con (K = 5). Cada punto en el gráfico representa una tarjeta gráfica, proyectada en un espacio bidimensional utilizando componentes principales (PCA) para reducir la dimensionalidad de los datos. Los puntos están coloreados según el clúster al que pertenecen, y los centroides de los clústeres están marcados con un símbolo distintivo.

El gráfico permite observar cómo se agrupan las tarjetas gráficas en función de sus especificaciones técnicas. La separación entre los clústeres indica qué tan distintos son entre sí, mientras que la compactación dentro de cada clúster refleja la similitud de las tarjetas gráficas agrupadas.

```r
# Aplicar K-means con K=5
kmeans_result_5 <- kmeans(variables_clustering_scaled, centers = 5, nstart = 25)

# Calcular la variación intra-clúster
intra_cluster_variation_5 <- sum(kmeans_result_5$withinss)

# Graficar los clústeres obtenidos con K=5
fviz_cluster(kmeans_result_5, data = variables_clustering_scaled) +
  labs(title = "Clústeres de Tarjetas Gráficas (K=5)",
       x = "Componente 1",
       y = "Componente 2")
```

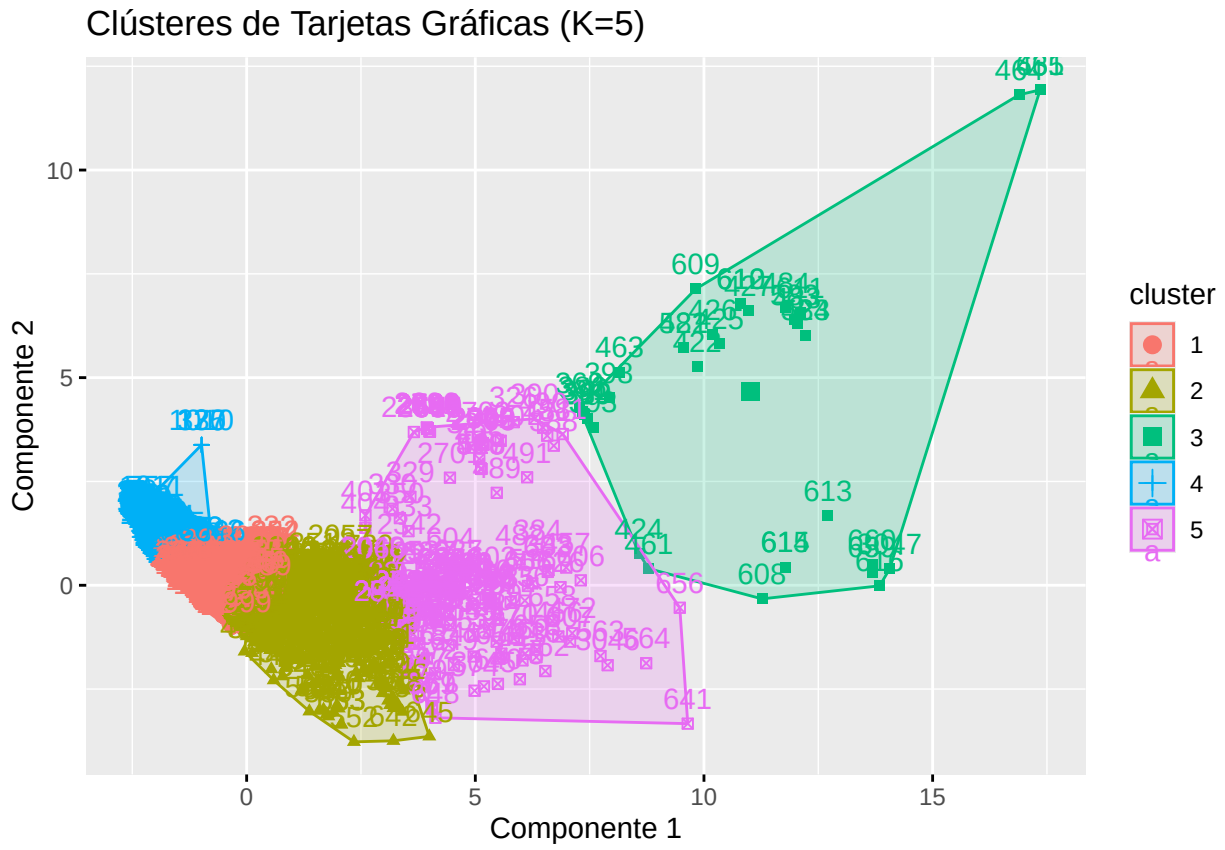

Figura 15: Clústeres de Tarjetas Gráficas (K=5)

```r
# Mostrar la variación intra-clúster
cat("Variación intra-clúster total:", intra_cluster_variation_5, "\n")
```

```
## Variación intra-clúster total: 6157.303
```

Resultados:

- La variación intra-clúster total es de 6157.302724.
- El gráfico de la **Figura 15** muestra la distribución de los clústeres obtenidos con K=5. Se observa que los clústeres están bien definidos, aunque en el grupo 3, las tarjetas gráficas están muy dispersas, lo que sugiere que este grupo puede contener tarjetas gráficas de diferentes generaciones o arquitecturas.

A continuación, aplicaremos el método K-means con K=6 para ver si obtenemos una mejor definición de los clústeres y si la variación intra-clúster disminuye.

Se seguirán los mismos pasos que antes, pero ahora con K=6. Esto nos permitirá comparar los resultados obtenidos con K=5 y K=6, y ver si la inclusión de un clúster adicional mejora la definición de los grupos.

```r
# Aplicar K-means con K=6
kmeans_result_6 <- kmeans(variables_clustering_scaled, centers = 6, nstart = 25)

# Calcular la variación intra-clúster
intra_cluster_variation_6 <- sum(kmeans_result_6$withinss)

# Mostrar la variación intra-clúster
cat("Variación intra-clúster total:", intra_cluster_variation_6, "\n")
```

```
## Variación intra-clúster total: 4842.295
```

```r
# Graficar los clústeres obtenidos con K=6
fviz_cluster(kmeans_result_6, data = variables_clustering_scaled) +
  labs(title = "Clústeres de Tarjetas Gráficas (K=6)",
       x = "Componente 1",
       y = "Componente 2")
```

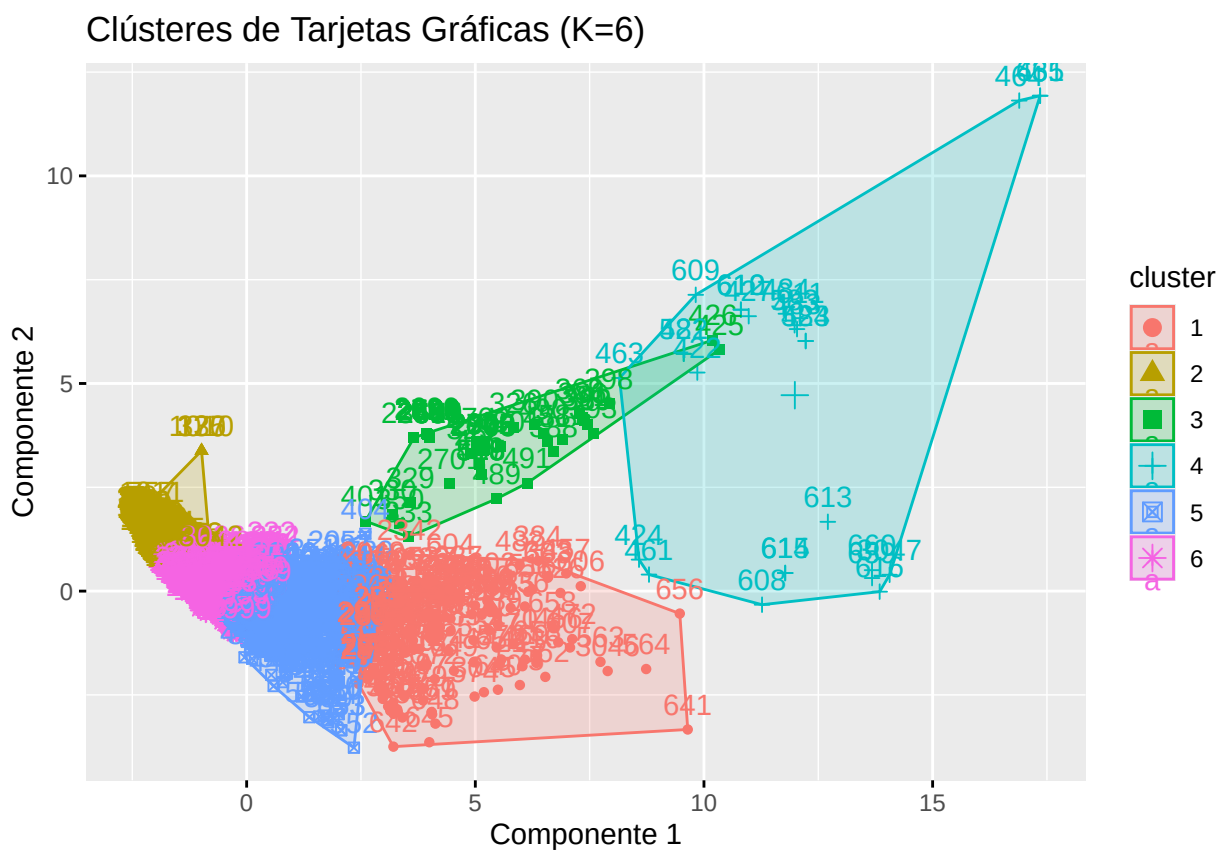

Figura 16: Clústeres de Tarjetas Gráficas (K=6)

```r
# Mostrar la variación intra-clúster
cat("Variación intra-clúster total:", intra_cluster_variation_6, "\n")
```

```
## Variación intra-clúster total: 4842.295
```

Resultados:

- La variación intra-clúster total es de 4842.2953772.
- El gráfico de la **Figura 16** muestra la distribución de los clústeres obtenidos con K=6. Se observa que los clústeres están bien definidos, aunque aquí también hay un grupo (grupo 4) que está muy disperso, y además hay cierta superposición entre los grupos 3 y 4.

Resultados globales

- La variación intra-clúster total es de 6157.302724 para k=5 y de 4842.2953772 para k=6, lo que indica que el modelo con k=6 tiene menos variación interna. Esto se debe a que tenemos un grupo más.
- Ambos modelos muestran clústeres bien definidos, aunque para el modelo con k=5, el grupo 3 está más disperso, lo mismo pasa en el grupo 4 para el modelo k=6
- El modelo con k=6 tiene una variación intra-clúster total menor, lo que indica que los clústeres son más compactos y están mejor definidos. Pese a haber superposición entre los grupos 3 y 4, sólamente son 2 tarjetas gráficas las que están en ambos grupos.

  En un análisis posterior, veremos con más claridad si es un problema para el modelo o no.

4.3.1.4 Análisis de los grupos obtenidos en el clustering

Vamos a analizar los grupos obtenidos en el clustering para ver si podemos identificar patrones o características comunes entre las tarjetas gráficas de cada grupo.

Para ello, tendremos que seguir los siguientes pasos:

1. Añadir la variable de clúster al conjunto de datos original.
2. Agrupar los datos por clúster y calcular las estadísticas descriptivas.
3. Visualizar las características de cada clúster.
4. Analizar los resultados y ver si podemos identificar patrones o características comunes entre las tarjetas gráficas de cada grupo.

Empezamos con el análisis de los grupo obtenidos con k=6. Para ello, vamos a seguir los pasos mencionados anteriormente, donde mostraremos por una parte una tabla Con el resumen de las medias de cada variable por clúster y por otra parte un gráfico de barras con las medias de cada variable por clúster para una mejor visualización.

Resumen por clúster con medias (K=6):

```r
library(kableExtra)
# Añadir la variable de clúster al conjunto de datos original
datos$cluster_6 <- kmeans_result_6$cluster

# Agrupar los datos por clúster y calcular la media
resumen_por_cluster_media <- datos %>%
  group_by(cluster_6) %>%
  summarise(across(c(`Memory_Size (GB)`, `Memory_Bus (bits)`, Shaders, TMUs, ROPs,
                     `Memory_clock (MHz)`, `GPU_clock (MHz)`), mean,
                     .names = "media_{.col}"))

# Generar una tabla con kable
kable(resumen_por_cluster_media,
      caption = "\\label{tab:resumen_por_cluster_media} Resumen de las medias por clúster",
      col.names = c("Clúster", "Memoria (GB)", "Bus (bits)",
      "Shaders", "TMUs", "ROPs", "Reloj Memoria (MHz)", "Reloj GPU (MHz)"),
      format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("!H", "striped", "scale_down", "hold_position"))
```

Tabla 1: Resumen de las medias por clúster

Clúster	Memoria (GB)	Bus (bits)	Shaders	TMUs	ROPs	Reloj Memoria (MHz)	Reloj GPU (MHz)
1	13.5392157	285.9608	4195.45098	202.117647	83.960784	1697.2549	1412.7451
2	0.1734311	146.1390	12.81016	5.308823	5.287433	305.7420	327.4746
3	14.0967742	3897.8065	4347.87097	260.645161	88.645161	805.3710	1081.8226
4	64.5925926	3712.0000	11553.18519	526.518519	122.074074	1602.5556	1276.1852
5	3.9983871	221.5226	1402.52903	89.641290	31.194839	1429.2568	970.5213
6	1.5751929	163.5093	379.07717	31.305443	13.912266	928.4614	750.0991

```r
# Gráfico de barras para las medias de cada variable por clúster
resumen_por_cluster_media %>%
  pivot_longer(cols = -cluster_6, names_to = "Variable", values_to = "Media") %>%
  ggplot(aes(x = factor(cluster_6), y = Media, fill = Variable)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Media de las Variables por Clúster", x = "Clúster", y = "Media") +
  theme_minimal()
```

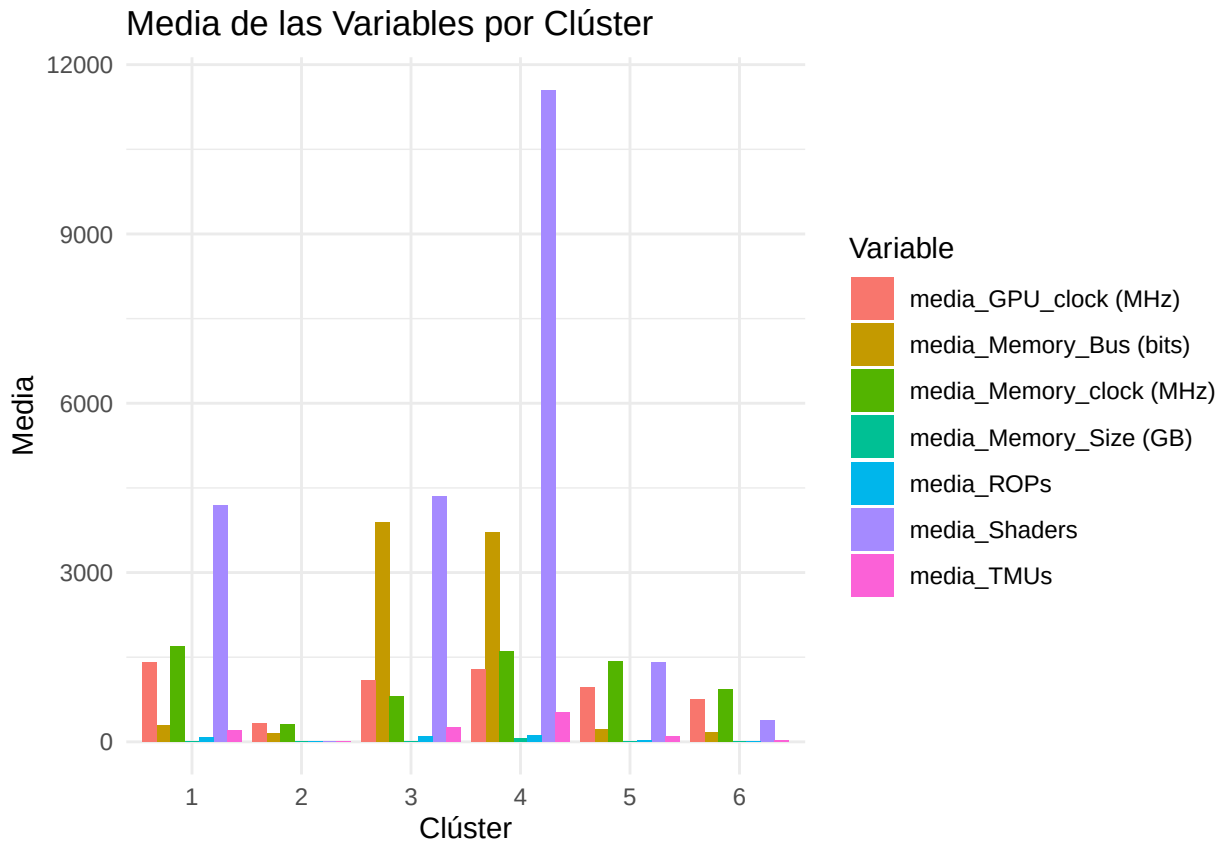

Figura 17: Media de las Variables por Clúster (K=6)

Observaciones:

- El grupo 4 destaca por tener una media de Shaders bastante más alta que el resto de grupos, que podría indicar que el número de Shaders es el principal factor que define este grupo.
- El grupo 2 es el que tiene los valores más bajos, lo que sugiere que este grupo está formado por tarjetas gráficas de gama baja.
- Los grupos 3 y 4 tienen valores de ancho de banda más altos, lo que sugiere que estos grupos están formados por tarjetas gráficas diseñadas para tareas que requieren un alto ancho de banda.

Ahora vamos a analizar los grupos obtenidos con k=5. Para ello, vamos a seguir los mismos pasos que hemos seguido para el análisis de los grupos obtenidos con k=6.

```
# Añadir la variable de clúster al conjunto de datos original
datos$cluster_5 <- kmeans_result_5$cluster

# Agrupar los datos por clúster y calcular la media

resumen_por_cluster_media5 <- datos %>%
  group_by(cluster_5) %>%
  summarise(across(c(`Memory_Size (GB)`, `Memory_Bus (bits)`, Shaders, TMUs, ROPs,
                     `Memory_clock (MHz)`, `GPU_clock (MHz)`), mean,
                     .names = "media_{.col}"))

# Generar una tabla con kable
kable(resumen_por_cluster_media5,
      caption = "\\label{tab:resumen_por_cluster_media5} Resumen de las medias por clúster",
      col.names = c("Clúster", "Memoria (GB)", "Bus (bits)",
      "Shaders", "TMUs", "ROPs", "Reloj Memoria (MHz)", "Reloj GPU (MHz)"),
      format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("!H", "striped", "scale_down", "hold_position"))
```

Tabla 2: Resumen de las medias por clúster

Clúster	Memoria (GB)	Bus (bits)	Shaders	TMUs	ROPs	Reloj Memoria (MHz)	Reloj GPU (MHz)
1	1.6396161	163.2756	394.39685	32.11417	14.102362	939.9283	758.9039
2	4.4273585	223.1950	1521.00629	96.32201	34.193711	1468.6025	1013.4201
3	56.4444444	3921.7778	10115.55556	485.55556	125.333333	1449.5000	1248.6389
4	0.1772596	146.3586	13.40505	5.36919	5.337317	307.7397	328.9588
5	14.4041451	1276.1865	4747.27461	231.62694	90.943005	1417.5596	1285.3523

```r
# Gráfico de barras para las medias de cada variable por clúster
resumen_por_cluster_media5 %>%
  pivot_longer(cols = -cluster_5, names_to = "Variable", values_to = "Media") %>%
  ggplot(aes(x = factor(cluster_5), y = Media, fill = Variable)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Media de las Variables por Clúster", x = "Clúster", y = "Media") +
  theme_minimal()
```

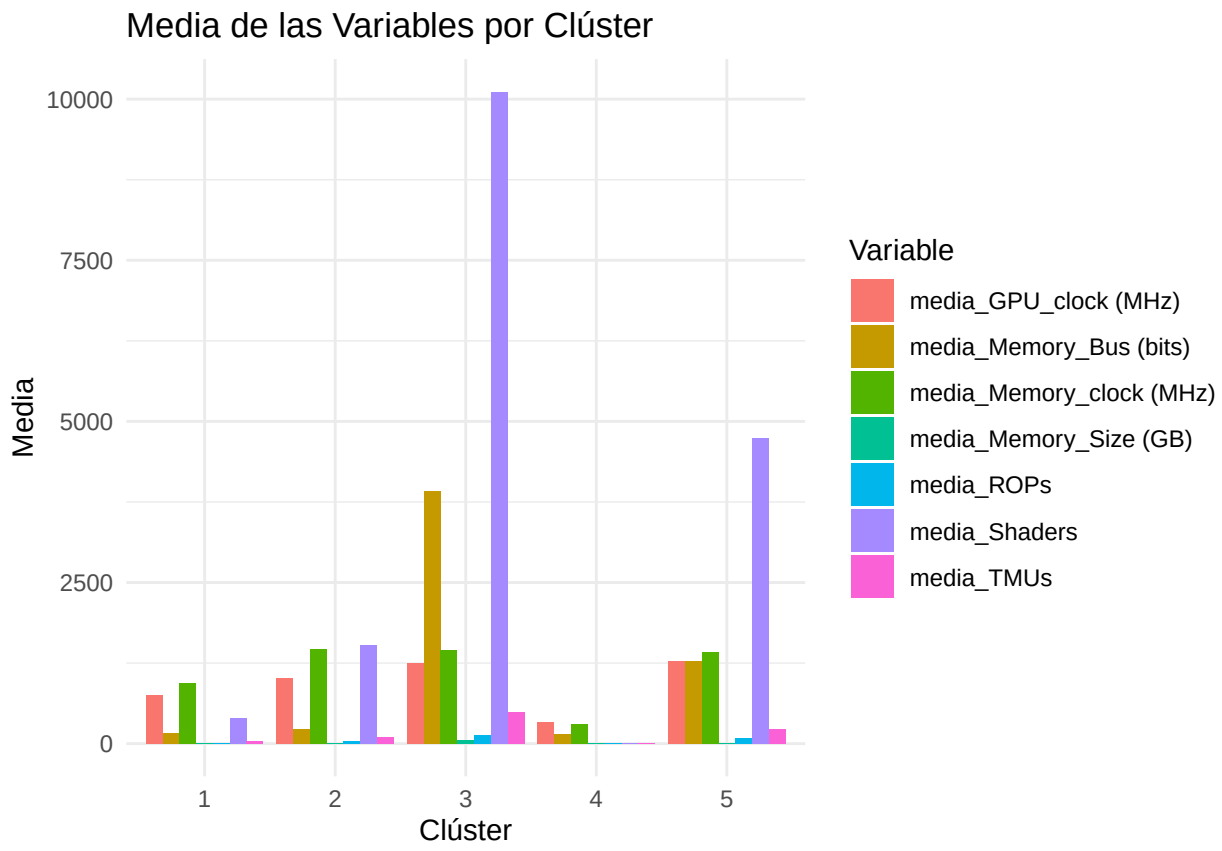

Figura 18: Media de las Variables por Clúster (K=5)

Observaciones:

- Aquí sí que encontramos un grupo (grupo3) cuya característica principal es el ancho de banda alto. También es el que más número de Shaders tiene, aunque esto podría deberse a ciertos valores muy altos que suben la media. Por eso más adelante vamos a ver la varianza de cada grupo. Este grupo podría estar formado en gran parte por las tarjetas gráficas de tipo HBM, que como habíamos mencionado anteriormente, están diseñadas para ofrecer un mayor ancho de banda.

Para confirmar si realmente el grupo 3 está formado por tarjetas gráficas de tipo HBM, vamos a mostrar el número de tarjetas gráficas de tipo HBM, HBM2 y HBM2e que hay en cada grupo.

```r
# Contar el número de tarjetas gráficas de tipo HBM, HBM2 y HBM2e en cada grupo
conteo_hbm <- datos %>%
  group_by(cluster_5, Memory_Type) %>%
  summarise(n = n(), .groups = 'drop') %>%
  filter(Memory_Type %in% c("HBM", "HBM2", "HBM2e", "HBM3"))

# Hacer una tabla con kable
kable(conteo_hbm,
      caption = "\\label{tab:conteo_hbm} Número de tarjetas gráficas de tipo HBM, HBM2,
                 HBM2e y HBM3 por clúster",
      col.names = c("Clúster", "Tipo de Memoria", "Número de Tarjetas Gráficas"),
      format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("!H", "striped", "scale_down", "hold_position"))
```

Tabla 3: Número de tarjetas gráficas de tipo HBM, HBM2, HBM2e y HBM3 por clúster

Clúster	Tipo de Memoria	Número de Tarjetas Gráficas
1	HBM2	3
2	HBM2	2
3	HBM2	7
3	HBM2e	15
3	HBM3	4
5	HBM	23
5	HBM2	24
5	HBM2e	7

La mayoría de las tarjetas gráficas de tipo HBM se clasifican en los grupo 3 y 5, aunque de entre las HBM, las más potentes (HBM2e y HBM3) se agrupan en su mayoría en el grupo 3, lo que explicaría el alto ancho de banda de este grupo. El grupo 5 presentaba el segundo mayor número de ancho de banda.

Vamos a ver la varianza de cada grupo con k=5.

```r
# Agrupar los datos por clúster y calcular la varianza

resumen_por_cluster_varianza5 <- datos %>%
  group_by(cluster_5) %>%
  summarise(across(c(`Memory_Size (GB)`, `Memory_Bus (bits)`, Shaders, TMUs, ROPs,
                     `Memory_clock (MHz)`, `GPU_clock (MHz)`), var,
                     .names = "var_{.col}"))

# Generar una tabla con kable

kable(resumen_por_cluster_varianza5,
      caption = "\\label{tab:resumen_por_cluster_varianza5} Resumen de las varianzas por clúster",
```

```
    col.names = c("Clúster", "Memoria (GB)", "Bus (bits)",
    "Shaders", "TMUs", "ROPs", "Reloj Memoria (MHz)", "Reloj GPU (MHz)"),
    format = "latex", booktabs = TRUE) %>%
kable_styling(latex_options = c("!H", "striped", "scale_down", "hold_position"))
```

Tabla 4: Resumen de las varianzas por clúster

Clúster	Memoria (GB)	Bus (bits)	Shaders	TMUs	ROPs	Reloj Memoria (MHz)	Reloj GPU (MHz)
1	1.5249875	11755.98	70559.348	337.40933	57.41111	26168.18	28147.59
2	7.9321378	10737.16	454439.611	1745.46797	205.72062	56619.84	84275.02
3	910.4253968	6036008.63	25948029.968	21821.51111	4651.88571	157241.34	190159.44
4	0.0387919	36646.34	1452.154	34.91139	21.02170	25738.94	25201.28
5	63.2837327	2544950.72	4290858.888	2670.19344	518.96028	268289.30	178457.64

```r
# Gráfico de barras para las varianzas de cada variable por clúster
resumen_por_cluster_varianza5 %>%
  pivot_longer(cols = -cluster_5, names_to = "Variable", values_to = "Varianza") %>%
  ggplot(aes(x = factor(cluster_5), y = Varianza, fill = Variable)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Varianza de las Variables por Clúster", x = "Clúster", y = "Varianza") +
  theme_minimal()
```

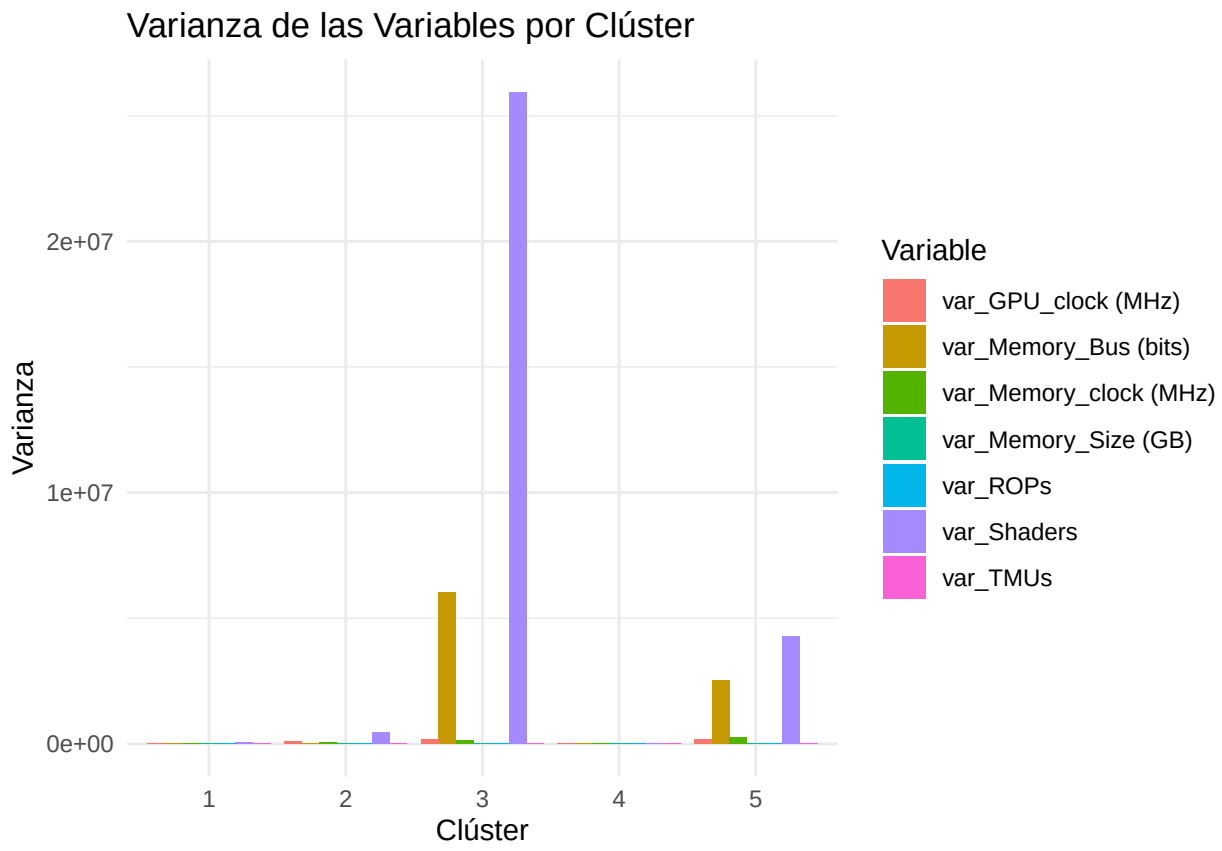

Figura 19: Varianza de las Variables por Clúster (K=5)

Observaciones:

- En el grupo 3 vemos una varianza muy alta en el número de Shaders, lo que sugiere que este grupo está formado por tarjetas gráficas de diferentes generaciones o arquitecturas. Esto podría ser un indicativo de que el grupo 3 está formado por tarjetas gráficas de gama alta, pero con diferentes arquitecturas. El ancho de banda también tiene una varianza elevada, lo que podría explicarse por la presencia de tarjetas gráficas de tipo HBM, que como hemos mencionado anteriormente, están diseñadas para ofrecer un mayor ancho de banda.
- Los grupos 1, 2 y 4 tienen una varianza baja en todas las variables, lo que sugiere que estos grupos están muy homogéneos y que las tarjetas gráficas de estos grupos tienen características similares.

Conclusiones:

- Pese a que el modelo con k=6 tiene una variación intra-clúster total menor, para nuestro análisis nos interesa más tener las tarjetas gráficas con la tecnología HBM juntas, por lo que el modelo con k=5 es más adecuado para nuestro análisis.

  No obstante, vamos a ver si con el modelo conjunto de variables numéricas y categóricas obtenemos un mejor resultado.

Vamos a guardar los resultados obtenidos en una variable para poder compararlos con los resultados obtenidos con el modelo conjunto de variables numéricas y categóricas.

```
# Crear un vector de conclusiones
conclusiones_k5 <- c(
  "El grupo 3 se caracteriza por tener un ancho de banda alto,",
  "lo que podría indicar que está formado por tarjetas gráficas de tipo HBM.",
  "Los grupos 1, 2 y 4 tienen una varianza baja en todas las variables,",
  "lo que sugiere que estos grupos están muy homogéneos y bien definidos."
)
# Crear un data frame resumen para el modelo k=5
tabla_resultados_kmeans_5 <- data.frame(
  Modelo = "Solo numéricas",
  Numero_Clusters = length(unique(kmeans_result_5$cluster)),
  Variacion_Intra_Cluster = intra_cluster_variation_5,
  Conclusiones = paste(conclusiones_k5, collapse = " ")
)
```

4.3.2 Clustering con variables numéricas y categóricas

En este apartado, incluiermos algunas variables categóricas al método k-means, para así poder comparar los resultados obtenidos con el modelo anterior y ver si, al incluir variables categóricas, obtenemos un mejor resultado.

En la base de datos tenemos las siguientes variables categóricas:

- **GPU_Chip**: Chip de la GPU.
- **Bus**: Tipo de bus.
- **Memory_Type**: Tipo de memoria.
- **Released**: Fecha de lanzamiento.

De estas variables nos interesa sólamente usar la variable "Memory_Type", ya que es la que más puede influir en el rendimiento de la tarjeta gráfica. La variable "GPU_Chip" no nos interesa porque, como se ha podido observar en el **apartado 2.1**, tenemos 335 tipos de chip diferentes, lo que nos generaría un número muy elevado de variables dummy. La variable "Bus" tampoco nos interesa porque no tiene un impacto significativo en el rendimiento de la tarjeta gráfica, pues la mayoría de las tarjetas gráficas utilizan el bus PCIe.

4.3.2.1 Preparación de los datos

Para incluir variables categóricas en el método K-means, es necesario convertirlas en variables dummy. Esto se logra utilizando la función `dummy_cols` de la librería `fastDummies`, que crea variables dummy para las variables categóricas seleccionadas. Las variables dummy son variables binarias que indican la presencia o ausencia de una categoría específica.

Después de crear las variables dummy, combinaremos las variables numéricas y categóricas en un solo conjunto de datos. Esto nos permitirá aplicar el método K-means a un conjunto de datos que incluye tanto variables numéricas como categóricas. Finalmente, escalaremos los datos para normalizar las variables para que todas tengan la misma importancia en el análisis.

```
library(fastDummies)

# Transformar la vaiable memory_type a factor
datos_transform <- datos %>%
  mutate(Memory_Type = as.factor(Memory_Type))

# Crear variables dummy para las variables categóricas seleccionadas
datos_dummy <- datos_transform %>%
  dummy_cols(select_columns = c("Memory_Type"), remove_first_dummy = TRUE)

# Combinar numéricas y categóricas
datos_combinados <- datos_transform %>%
  select(`Memory_Size (GB)`, `Memory_Bus (bits)`, Shaders, TMUs, ROPs,
         `Memory_clock (MHz)`, `GPU_clock (MHz)`) %>%
  cbind(datos_dummy %>% select(starts_with("Memory_Type_")))

# Escalar los datos para normalizar las variables

datos_combinados_scaled <- scale(datos_combinados)

# Ver cuántas variables nuevas se han creado al hacer dummies

nvariables <- ncol(datos_combinados)

nvariables
```

```
## [1] 32
```

Después de crear las variables dummy, vemos que el número total de variables ha ascendido a 32 variables.

Tantas variables podrían hacer que el algoritmo K-means no funcione correctamente, por lo que vamos a aplicar el método PCA para reducir la dimensionalidad de los datos.

4.3.2.2 Fundamento teórico del análisis de componentes principales (PCA)

(Manual) explica que el PCA es un método no supervisado que se utiliza para resumir un conjunto grande de variables correlacionadas mediante un número menor de variables representativas llamadas componentes principales. Estas componentes son direcciones en el espacio de características a lo largo de las cuales los datos originales presentan una alta variabilidad, y definen líneas y subespacios que están lo más cerca posible del conjunto de datos. PCA se usa tanto para producir variables derivadas que pueden emplearse en modelos de aprendizaje supervisado como para la visualización de datos o para la imputación de valores faltantes en una matriz de datos.

¿Cómo funciona PCA?

El PCA encuentra una representación de los datos en un espacio de menor dimensión, donde cada dimensión es una combinación lineal de las variables originales.

Imaginemos que tenemos un conjunto de datos con p variables originales $X_1, X_2, \dots, X_p$.

La primera componente se calcula como:

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \cdots + \phi_{p1}X_p$$

donde los coeficientes ϕ_{ij} se denominan **cargas**, y cumplen la condición:

$$\sum_{j=1}^{p} \phi_{j1}^2 = 1$$

PCA tiene como objetivo **maximizar la varianza** de la nueva variable Z_1, es decir, encontrar la dirección en el espacio de los datos a lo largo de la cual las observaciones presentan la mayor variabilidad posible. Esto se hace resolviendo un problema de optimización bajo la restricción mencionada anteriormente.

El conjunto de valores z_{i1} obtenidos para cada observación se denominan **scores** del primer componente.

Una vez determinado el primer componente principal Z_1, se puede calcular el **segundo componente principal** Z_2. Este es otra combinación lineal de las variables originales $X_1, ..., X_p$, pero con dos condiciones:

1. **Debe tener la máxima varianza posible**, igual que Z_1.
2. **Debe ser no correlacionado con** Z_1, lo cual equivale a que su vector de cargas ϕ_2 sea **ortogonal** al vector de cargas del primer componente, ϕ_1.

La forma general del segundo componente es:

$$z_{i2} = \phi_{12}x_{i1} + \phi_{22}x_{i2} + \cdots + \phi_{p2}x_{ip}$$

donde $\phi_2 = (\phi_{12}, \phi_{22}, ..., \phi_{p2})$ es el **vector de cargas del segundo componente principal**, y cumple:

- $\sum_{j=1}^{p} \phi_{j2}^2 = 1$ (normalización), y

- $\phi_2 \perp \phi_1$ (ortogonalidad).

Este proceso se repite para obtener más componentes principales $Z_3, Z_4, ..., Z_M$ (con $M \leq p$), cada uno maximizando la varianza restante y siendo ortogonal a los anteriores.

4.3.2.3 Aplicación del PCA

Vamos a aplicar el PCA a los datos combinados. Para ello, vamos a usar la función `PCA` de la librería `FactoMineR`, que permite realizar un análisis de componentes principales de manera sencilla.

La función `PCA` toma como argumentos los datos, el número de componentes principales a calcular y el gráfico a generar. En este caso, vamos a calcular todos los componentes principales y no vamos a generar gráficos. Con esta función, podemos obtener fácilmente la varianza explicada por cada componente principal con la función `eig`, que devuelve un data frame con la varianza explicada por cada componente principal.

```
library(FactoMineR)
# Realizar PCA
pca_result <- PCA(datos_combinados_scaled, graph = FALSE)

# Ver la varianza explicada por cada componente principal

pca_var <- pca_result$eig
pca_var
```

```
##         eigenvalue percentage of variance cumulative percentage of variance
## comp 1  5.32700624            16.64689451                           16.64689
## comp 2  2.24726968             7.02271776                           23.66961
## comp 3  1.39849281             4.37029004                           28.03990
## comp 4  1.23319452             3.85373288                           31.89364
## comp 5  1.18271645             3.69598891                           35.58962
## comp 6  1.15939412             3.62310661                           39.21273
## comp 7  1.07048094             3.34525292                           42.55798
## comp 8  1.04989121             3.28091002                           45.83889
## comp 9  1.03425554             3.23204857                           49.07094
## comp 10 1.01924387             3.18513710                           52.25608
## comp 11 1.01143834             3.16074480                           55.41682
## comp 12 1.01124085             3.16012766                           58.57695
## comp 13 1.01015492             3.15673413                           61.73369
## comp 14 1.00439888             3.13874650                           64.87243
## comp 15 1.00288571             3.13401784                           68.00645
## comp 16 1.00259693             3.13311540                           71.13957
## comp 17 1.00245519             3.13267245                           74.27224
## comp 18 1.00202792             3.13133724                           77.40358
## comp 19 1.00132527             3.12914147                           80.53272
```

```
## comp 20 1.00106724            3.12833513                83.66105
## comp 21 1.00078643            3.12745760                86.78851
## comp 22 1.00048747            3.12652336                89.91503
## comp 23 1.00040438            3.12626368                93.04130
## comp 24 1.00032843            3.12602634                96.16732
## comp 25 0.39834403            1.24482510                97.41215
## comp 26 0.35610868            1.11283961                98.52499
## comp 27 0.18335509            0.57298465                99.09797
## comp 28 0.10156016            0.31737552                99.41535
## comp 29 0.07146214            0.22331919                99.63867
## comp 30 0.04620136            0.14437924                99.78305
## comp 31 0.04385966            0.13706145                99.92011
## comp 32 0.02556555            0.07989233               100.00000
```

Vemos que con 19 componentes principales explicamos el 80% de la varianza total. Como queremos obtener el menor número de componentes principales que expliquen el 80% de la varianza total, vamos a intentar agrupar los tipos de memoria menos comunes en una sola categoría llamada “Otros”.

4.3.2.4 Agrupación de tipos de memoria y aplicación del PCA

Como habíamos hecho en el **apartado 2.4.2**, vamos a filtrar los tipos de memoria, pero esta vez vamos a filtrar los menos comunes. Luego vamos a agrupas estas tarjetas gráficas en una sola categoría llamada “Otros”. Vamos a llamarla datos2 para diferenciarla con la base de datos original.

Además, vamos a agrupar HBM, HBM2, HBM2e y HBM3 en una sola categoría llamada “HBM”

```
# Crear una copia de los datos originales para trabajar con ella
datos2 <- datos

# Filtrar los tipos de memoria menos comunes
tipos_memoria_no_comunes <- datos2 %>%
  count(Memory_Type, sort = TRUE) %>%
  filter(n < 15) %>%
  pull(Memory_Type)

# Agrupar los tipos de memoria menos comunes en una sola categoría llamada "Otros"

datos2$Memory_Type <- ifelse(datos2$Memory_Type %in% tipos_memoria_no_comunes, "Otros",
datos2$Memory_Type)

# Agrupar HBM, HBM2 y HBM2e en una sola categoría llamada "HBM"

datos2$Memory_Type <- ifelse(datos2$Memory_Type %in% c("HBM", "HBM2", "HBM2e", "HBM3"), "HBM",
datos2$Memory_Type)
```

Ahora que hemos agrupado los tipos de memoria menos comunes, vamos a volver a crear las variables dummy y aplicar el PCA.

```
# Transformar la variable memory_type a factor
datos2$Memory_Type <- as.factor(datos2$Memory_Type)

# Crear variables dummy para las variables categóricas seleccionadas
datos_dummy2 <- datos2 %>%
  dummy_cols(select_columns = c("Memory_Type"), remove_first_dummy = TRUE)

# Combinar numéricas y categóricas
datos_combinados2 <- datos2 %>%
  select(`Memory_Size (GB)`, `Memory_Bus (bits)`, Shaders, TMUs, ROPs,
         `Memory_clock (MHz)`, `GPU_clock (MHz)`) %>%
  cbind(datos_dummy2 %>% select(starts_with("Memory_Type_")))

# Escalar los datos para normalizar las variables

datos_combinados_scaled2 <- scale(datos_combinados2)

# Ver cuántas variables nuevas se han creado al hacer dummies
nvariables2 <- ncol(datos_combinados_scaled2)
nvariables2
```

```
## [1] 19
```

Ahora que hemos agrupado los tipos de memoria menos comunes, el número total de variables ha descendido a 19 variables.

Vamos a aplicar de nuevo el PCA con la nueva base de datos reducida para ver cuántos componentes principales necesitamos para poder realizar el clustering.

```
# Realizar PCA
pca_result2 <- PCA(datos_combinados_scaled2, graph = FALSE)

# Ver la varianza explicada por cada componente principal

pca_var2 <- pca_result2$eig
pca_var2
```

```
##         eigenvalue percentage of variance cumulative percentage of variance
## comp 1  5.23247717             27.5393535                           27.53935
## comp 2  2.20352143             11.5974812                           39.13683
## comp 3  1.38728320              7.3014905                           46.43833
```

```
## comp 4  1.22149152          6.4289027                52.86723
## comp 5  1.16513867          6.1323088                58.99954
## comp 6  1.06365013          5.5981586                64.59770
## comp 7  1.05243183          5.5391149                70.13681
## comp 8  1.03011566          5.4216614                75.55847
## comp 9  1.02250842          5.3816233                80.94009
## comp 10 1.01130568          5.3226615                86.26276
## comp 11 1.00887570          5.3098721                91.57263
## comp 12 0.51775051          2.7250027                94.29763
## comp 13 0.41195491          2.1681837                96.46581
## comp 14 0.30355378          1.5976515                98.06347
## comp 15 0.11690998          0.6153157                98.67878
## comp 16 0.09556421          0.5029695                99.18175
## comp 17 0.07010610          0.3689795                99.55073
## comp 18 0.04938565          0.2599245                99.81066
## comp 19 0.03597546          0.1893445               100.00000
```

Aquí vemos que con 9 componentes principales explicamos el 80% de la varianza total, pero nos interesa más tener el menor número posible de componentes principales, por lo que vamos a quedarnos con 8 componentes principales, que explican el 75% de la varianza total.

4.3.2.5 Método del codo con PCA

Vamos a aplicar el método del codo con 8 componentes principales para determinar el número óptimo de clústeres (K). Para ello, vamos a aplicar de nuevo la función `PCA`, donde podremos seleccionar 8 componentes principales introduciendo el argumento `ncp = 9` seguido de $\$ind$coord, que nos devolverá las coordenadas de los individuos en el espacio de las componentes principales. Con estas coordenadas, vamos a aplicar el método del codo para determinar el número óptimo de clústeres (K).

```r
# Seleccionar las primeras 9 componentes principales
pca_data <- PCA(datos_combinados_scaled2, ncp = 8, graph = FALSE)$ind$coord

# Hacer el método del codo

fviz_nbclust(pca_data, kmeans, method = "wss") +
  labs(title = "Método del Codo para Determinar K",
       x = "Número de Clústeres (K)",
       y = "Suma de Distancias al Cuadrado (WCSS)")
```

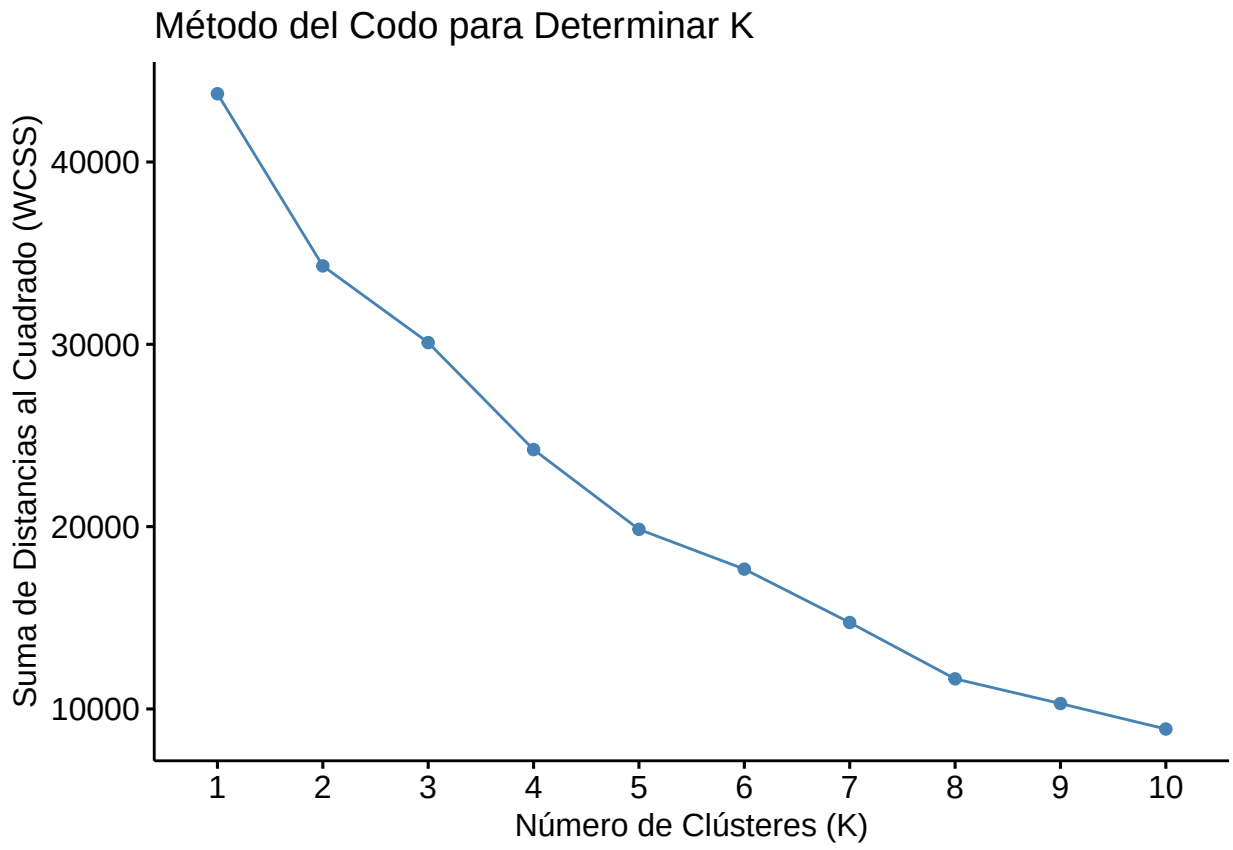

Figura 20: Método del Codo para Determinar K con PCA

Vemos un ligero codo en k=8, por lo que vamos a aplicar el método k-means con k=8.

4.3.2.6 Aplicación del método K-means con PCA

En este apartado, vamos a aplicar el método k-means con k=8. Para ello, similar a lo que hicimos en el **apartado 3.3.1.3**, vamos a usar la función `kmeans` de R.

Esta vez vamos a usar el argumento `centers = 8` para indicar que queremos 8 clústeres. También vamos a usar el

argumento `nstart = 25` para evitar que el algoritmo se quede atrapado en un mínimo local. Luego vamos a calcular la variación intra-clúster total y graficar los clústeres obtenidos. Para ello, vamos a usar la función `fviz_cluster` de la librería `factoextra`, que permite visualizar los resultados del clustering de manera intuitiva.

```
# Aplicar K-means con K=8
kmeans_result_8 <- kmeans(pca_data, centers = 8, nstart = 25)

# Calcular la variación intra-clúster
intra_cluster_variation_8 <- sum(kmeans_result_8$withinss)
intra_cluster_variation_8
```

```
## [1] 9501.153
```

```r
# Graficar los clústeres obtenidos con K=8
fviz_cluster(kmeans_result_8, data = pca_data) +
  labs(title = "Clústeres de Tarjetas Gráficas (K=8)",
       x = "Componente 1",
       y = "Componente 2")
```

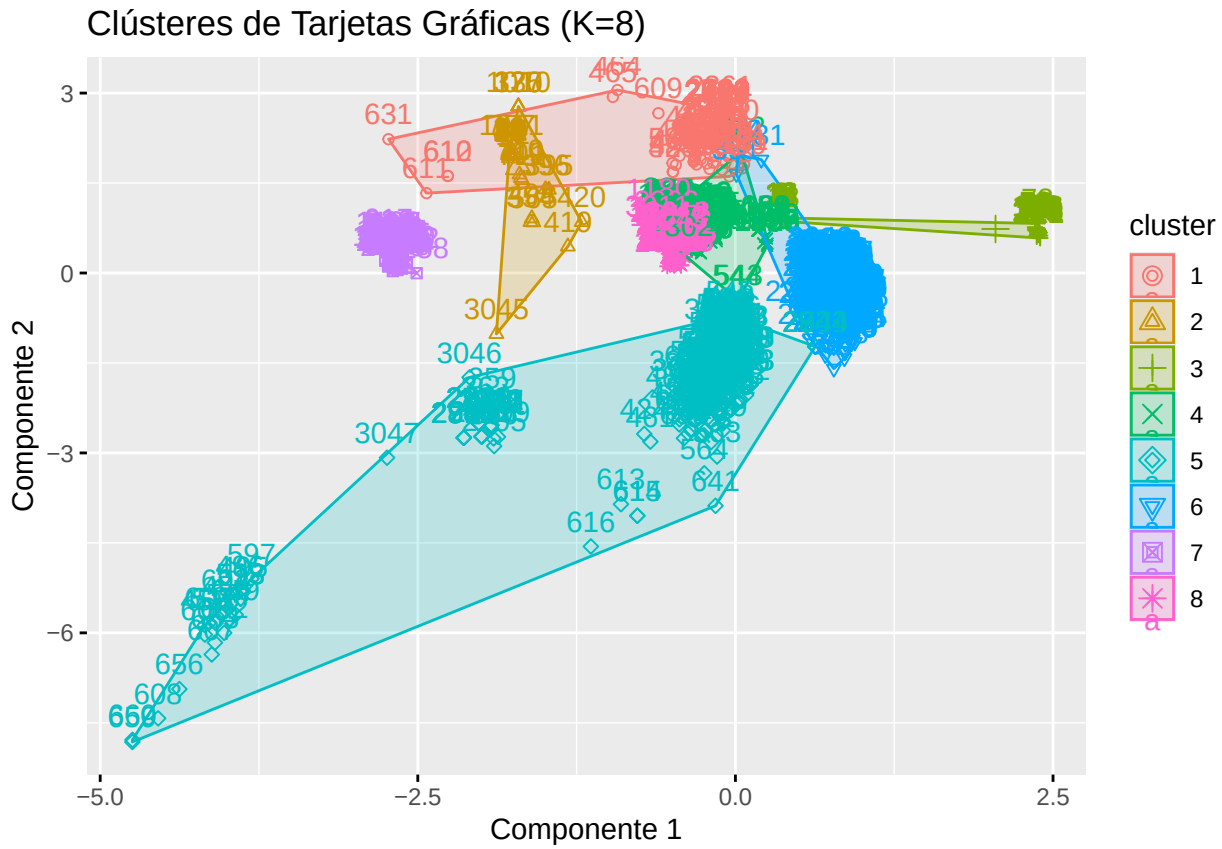

Figura 21: Clústeres de Tarjetas Gráficas (K=8)

Resultados:

- La variación intra-clúster total es de 9501.1530549.
- El gráfico de la **Figura 21** muestra la distribución de los clústeres obtenidos con K=8. Se observa que los grupos 1 y 2 tienen cierta superposición, igual que los grupos 3 y 6.
- Los grupos 7 y 8 son los que mejor definidos están, pues muestran muy poca variación interna y están bien separados entre sí.
- El grupo 5 es el que más disperso está, lo que sugiere que este grupo puede contener tarjetas gráficas de diferentes generaciones o arquitecturas.

4.3.2.7 Análisis de los grupos obtenidos en el clustering

Vamos a analizar los grupos obtenidos en el clustering para ver si podemos identificar patrones o características comunes entre las tarjetas gráficas de cada grupo.

Para ello, primero calcularemos la media y la varianza de cada variable por clúster. Para facilitar la visualización de los resultados, generaremos tablas con `kable` para mostrar las medias y varianzas de cada variable por clúster.

Resumen por clúster con medias:

```
# Añadir la variable de clúster al conjunto de datos original
datos2$cluster_8 <- kmeans_result_8$cluster

# Agrupar los datos por clúster y calcular la media
resumen_por_cluster_media8 <- datos2 %>%
  group_by(cluster_8) %>%
  summarise(across(c(`Memory_Size (GB)`, `Memory_Bus (bits)`, Shaders, TMUs, ROPs,
                     `Memory_clock (MHz)`, `GPU_clock (MHz)`), mean,
                     .names = "media_{.col}"))

# Generar una tabla con kable
kable(resumen_por_cluster_media8,
      caption = "\\label{tab:resumen_por_cluster_media8} Resumen de las medias por clúster",
      col.names = c("Clúster", "Memoria (GB)", "Bus (bits)",
      "Shaders", "TMUs", "ROPs", "Reloj Memoria (MHz)", "Reloj GPU (MHz)"),
      format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("!H", "striped", "scale_down", "hold_position"))
```

Tabla 5: Resumen de las medias por clúster

Clúster	Memoria (GB)	Bus (bits)	Shaders	TMUs	ROPs	Reloj Memoria (MHz)	Reloj GPU (MHz)
1	27.9250000	4249.60000	5185.600000	315.5000000	89.100000	944.7125	1075.6625
2	1.4474910	364.30769	237.538462	11.5128205	7.282051	387.6154	279.2051
3	0.0196149	91.07692	1.615385	0.9807692	1.634615	114.2500	109.8269
4	0.4341728	194.48471	95.876471	16.8858824	10.109412	558.4812	462.8282
5	11.9962406	221.23308	4032.962406	174.0150376	72.541353	1666.2632	1397.7932
6	3.3391608	209.87413	1111.342657	75.1923077	26.870629	1274.4231	890.5140
7	0.3263393	105.60000	37.307143	7.2428571	5.085714	403.4786	496.8071
8	1.9559257	93.88679	342.971698	23.8726415	10.033019	905.2099	821.5212

Resumen por varianzas de cada grupo:

```
# Agrupar los datos por clúster y calcular la varianza
resumen_por_cluster_varianza8 <- datos2 %>%
  group_by(cluster_8) %>%
  summarise(across(c(`Memory_Size (GB)`, `Memory_Bus (bits)`, Shaders, TMUs, ROPs,
                     `Memory_clock (MHz)`, `GPU_clock (MHz)`), var,
```

```
                    .names = "var_{.col}"))
# Generar una tabla con kable
kable(resumen_por_cluster_varianza8,
      caption = "\\label{tab:resumen_por_cluster_varianza8} Resumen de las varianzas por clúster",
      col.names = c("Clúster", "Memoria (GB)", "Bus (bits)",
      "Shaders", "TMUs", "ROPs", "Reloj Memoria (MHz)", "Reloj GPU (MHz)"),
      format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("!H", "striped", "scale_down", "hold_position"))
```

Tabla 6: Resumen de las varianzas por clúster

Clúster	Memoria (GB)	Bus (bits)	Shaders	TMUs	ROPs	Reloj Memoria (MHz)	Reloj GPU (MHz)
1	976.8550633	1356512.243	5.271498e+06	20086.025316	2023.2810127	147042.840	26835.543
2	12.7982443	576167.903	9.809682e+05	1022.887989	190.4183536	370341.559	216202.115
3	0.0003874	1605.975	6.467513e-01	3.397685	0.8166542	3493.451	3974.824
4	0.2792182	11811.430	2.631748e+04	361.302627	51.6122788	82301.089	28289.146
5	88.7358349	11264.044	1.247734e+07	12335.728075	1390.8001702	108285.221	216185.667
6	7.5704555	11775.620	4.438798e+05	2127.039101	257.2010964	74074.159	30369.897
7	0.0505883	3163.350	4.568315e+03	54.242754	10.4961973	11348.194	12172.013
8	1.5803612	1757.869	2.924894e+04	102.120857	19.8097819	10845.258	28051.253

Observaciones:

- El grupo 1 tiene con diferencia la media más alta de ancho de bus, lo que podría sugerir que está formado principalmente por tarjetas gráficas de memoria *HBM*. También tiene la media más alta de memoria, que indica que este grupo también está formado por tarjetas gráficas de gama alta a parte de las tarjetas gráficas con memoria *HBM*.
- El grupo 2 no destaca en ninguna variable, lo que podría sugerir que estaría formado por tarjetas con el tipo de memoria *Otros* o por tarjetas de gama media.
- El grupo 3 tiene la media y varianza más baja en la mayoría de las variables, lo que sugiere que este grupo está formado por tarjetas gráficas de gama baja.
- Aún con estas tablas, no podemos concluir por qué hay cierta superposición entre los grupos 1 y 2, y entre los grupos 3 y 6. Para ello, vamos a ver si podemos encontrar alguna relación entre los grupos obtenidos y el tipo de memoria de las tarjetas gráficas.

Vamos a ver si podemos encontrar alguna relación entre los grupos obtenidos y el tipo de memoria de las tarjetas gráficas. Para ello, vamos a contar el número de tarjetas gráficas por tipo de memoria en cada clúster y también el número total de tarjetas gráficas por tipo de memoria en la base de datos. Esto nos permitirá ver si hay algún tipo de memoria que esté más presente en un clúster que en otro.

```
# Combinar las tablas de resumen por clúster y resumen total
resumen_comparacion <- resumen_por_cluster_memoria %>%
  left_join(resumen_total_memoria, by = "Memory_Type", suffix = c("_cluster", "_total")) %>%
  mutate(porcentaje_cluster = n_cluster / n_total * 100)
```

```r
# Generar una tabla con kable
tabla_comparacion <- kable(resumen_comparacion,
      caption = "\\label{tab:comparacion_memoria} Comparación del número de tarjetas gráficas por tipo d
      col.names = c("Clúster", "Tipo de Memoria", "Número en Clúster",
      "Número Total", "Porcentaje en Clúster"),
      format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("!H", "striped", "scale_down", "hold_position"))

tabla_comparacion
```

Tabla 7: Comparación del número de tarjetas gráficas por tipo de memoria

Clúster	Tipo de Memoria	Número en Clúster	Número Total	Porcentaje en Clúster
1	HBM	76	81	93.8271605
1	Otros	4	45	8.8888889
2	Otros	39	45	86.6666667
3	SDR	87	87	100.0000000
3	DRAM	17	17	100.0000000
4	GDDR3	508	508	100.0000000
4	DDR	314	314	100.0000000
4	GDDR4	25	25	100.0000000
4	GDDR6	2	211	0.9478673
4	HBM	1	81	1.2345679
5	GDDR6	209	211	99.0521327
5	GDDR5X	30	30	100.0000000
5	GDDR6X	20	20	100.0000000
5	GDDR5	5	1145	0.4366812
5	Otros	2	45	4.4444444
6	GDDR5	1140	1145	99.5633188
6	HBM	4	81	4.9382716
7	DDR2	140	140	100.0000000
8	DDR3	424	424	100.0000000

Observaciones:

- El grupo 1 está formado principalmente por las tarjetas gráficas con memoria HBM, lo que quiere decir que este grupo debería tener un ancho de bus más alto que el resto de grupos. Esto se puede ver en la **Tabla 1** de resumen de medias, donde el grupo 1 tiene la media más alta de ancho de bus.
- El grupo 2 está formado principalmente por las tarjetas gráficas con "Otros" tipos de memoria, lo que podría explicar la superposición con el grupo 1, ya que este grupo tiene una media de ancho de bus más baja que el grupo 1, pero ligeramente más alta que el resto de grupos. Además, el tipo de memoria *Otros* sólamente está presente en estos dos grupos.
- El grupo 3 está predominado por las tarjetas gráficas con memoria SDR y DRAM, lo que podría sugerir que estas dos arquitecturas son muy similares entre sí. Como habíamos visto en la **Tabla 1**, el grupo 3 tiene las medias más bajas en todas las variables, lo que seguramente se podría deber a que estas tecnologías son más antiguas.

- El grupo 4 está formado por 5 tipos de memoria diferentes, siendo la mayoría GDDR3, DDR y GDDR6, donde la totalidad de estos tipos de memoria están en este grupo. En la **Tabla 1** vemos que el grupo 4 no tiene ninguna variable que destaque sobre el resto, esto quiere decir que este grupo está formado por tarjetas gráficas de gama media-baja.
- El grupo 5 está formado por básicamente todas las tarjetas gráficas con memoria GDDR6, GDDR5X y GDDR6X, que son memorias de gama alta, lo que se puede observar también en la **Tabla 1**, donde el grupo 5 presenta las medias más altas, sólamente superadas por el grupo 1, aunque en reloj de memoria y reloj de GPU, el grupo 5 tiene las medias más altas. Esto tiene sentido, ya que, como habíamos mencionado anteriormente, las tarjetas gráficas con memoria HBM están especializadas en tareas que requieren un alto ancho de banda, mientras que las tarjetas gráficas con memoria GDDR6, GDDR5X y GDDR6X están diseñadas para ofrecer un alto rendimiento en juegos y aplicaciones gráficas.
- El grupo 6 está formado mayoritariamente por tarjetas gráficas con memoria GDDR5, que es una memoria de gama media-alta.
- Los grupos 7 y 8 están formados únicamente por una memoria cada uno, siendo la memoria DDR2 y DDR3 respectivamente, donde ambos tienen el 100% de las tarjetas gráficas con ese tipo de memoria. Esto podría explicar por qué estos grupos están tan bien definidos, ya que al ser un único tipo de memoria, no hay variación interna entre las tarjetas gráficas de estos grupos.

Guardamos los resultados obtenidos en una variable para poder compararlos con los resultados obtenidos con el modelo conjunto de variables numéricas y categóricas.

```
# Crear un vector de conclusiones
conclusiones_k8 <- c(
  "Se tuvo que hacer un PCA para reducir la dimensionalidad de los datos.",
  "Después de aplicar el PCA y realizar el clustering,",
  "se observó que los grupos 1 y 2 tienen cierta superposición,",
  "igual que los grupos 3 y 6.",
  "Había un par de grupos bien definidos y otros que estaban más dispersos.",
  "Luego se concluyó que los grupos estaban principalmente definidos",
  "por el tipo de memoria de las tarjetas gráficas."
)
# Crear un data frame resumen para el modelo k=5
tabla_resultados_kmeans_8 <- data.frame(
  Modelo = "Numéricas y categóricas",
  Numero_Clusters = length(unique(kmeans_result_8$cluster)),
  Variacion_Intra_Cluster = intra_cluster_variation_8,
  Conclusiones = paste(conclusiones_k8, collapse = " ")
)
```

4.3.3 Comparación de los dos modelos

En este apartado, vamos a comparar los resultados obtenidos con el modelo de sólo variables numéricas y el modelo de variables numéricas y categóricas. Para ello, vamos a crear una tabla con los resultados obtenidos en ambos modelos,

donde incluiremos el número de clústeres, la variación intra-clúster total y las conclusiones obtenidas en cada modelo.

```
# Unimos los resultados de ambos modelos en una sola tabla
tabla_resultados_kmeans <- rbind(tabla_resultados_kmeans_5, tabla_resultados_kmeans_8)

# Generar una tabla con kable
kable(tabla_resultados_kmeans,
      caption = "\\label{tab:comparacion_modelos} Comparación de los resultados de los modelos K-means"
      col.names = c("Modelo", "Número de Clústeres",
      "Variación Intra-Cluster", "Conclusiones"),
      format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("!H", "striped", "scale_down", "hold_position")) %>%
  column_spec(4, width = "8cm")
```

Tabla 8: Comparación de los resultados de los modelos K-means

Modelo	Número de Clústeres	Variación Intra-Cluster	Conclusiones
Solo numéricas	5	6157.303	El grupo 3 se caracteriza por tener un ancho de banda alto, lo que podría indicar que está formado por tarjetas gráficas de tipo HBM. Los grupos 1, 2 y 4 tienen una varianza baja en todas las variables, lo que sugiere que estos grupos están muy homogéneos y bien definidos.
Numéricas y categóricas	8	9501.153	Se tuvo que hacer un PCA para reducir la dimensionalidad de los datos. Después de aplicar el PCA y realizar el clustering, se observó que los grupos 1 y 2 tienen cierta superposición, igual que los grupos 3 y 6. Había un par de grupos bien definidos y otros que estaban más dispersos. Luego se concluyó que los grupos estaban principalmente definidos por el tipo de memoria de las tarjetas gráficas.

Frente a los resultados obtenidos, el modelo con sólo variables numéricas resulta ser más sencillo de interpretar, con una variación intra-clúster total más baja, grupos mejor definidos y menos dispersos. Además, este modelo tiene en cuenta más las especificaciones técnicas de las tarjetas gráficas que el modelo con el tipo de memoria, donde este clasificaba las tarjetas gráficas principalmente por el tipo de memoria.

4.4 Implementación manual del algoritmo K-means

En este apartado, vamos a implementar el algoritmo k-means de manera manual, lo que nos permitirá utilizar distintas medidas de distancia y analizar si los grupos resultantes difieren en función de la métrica empleada.

En el análisis anterior, hemos utilizado la función `kmeans` de R, que implementa el algoritmo k-means clásico utilizando la distancia euclidiana como medida de proximidad entre observaciones y centroides. Sin embargo, como se mencionó en el **apartado 3.2.1**, existen otras medidas de distancia que pueden ser útiles en función de la naturaleza de los datos y de los objetivos del análisis, como la `distancia de Manhattan`, la `distancia de Chebyshev` o la `distancia de Minkowski`. Esta última es especialmente interesante, ya que generaliza tanto la distancia euclidiana (cuando $p = 2$) como la de Manhattan (cuando $p = 1$), permitiendo ajustar el parámetro p para controlar la sensibilidad a valores extremos.

En nuestro caso, el análisis exploratorio ha mostrado la presencia de valores atípicos, especialmente en las tarjetas gráficas más modernas **(apartado 2.3.3)**. En estos escenarios, la distancia euclidiana puede verse muy influida por estos valores extremos, lo que podría afectar a la formación de los clústeres. Por el contrario, la distancia de Manhattan y, en general, la distancia de Minkowski con (p < 2), son menos sensibles a los valores atípicos, lo que podría dar lugar a agrupamientos más robustos.

Por lo tanto, la estrategia será la siguiente:

1. Implementaremos el algoritmo k-means manualmente utilizando la distancia euclidiana. Esto nos permitirá comparar directamente los resultados con los obtenidos previamente mediante la función kmeans de R y validar que la implementación es correcta.
2. Aplicaremos el algoritmo utilizando la distancia de Minkowski, ajustando el parámetro (p) para explorar si se obtienen agrupamientos diferentes o incluso mejores, especialmente en presencia de valores atípicos. De este modo, podremos evaluar si la elección de la métrica de distancia influye en la calidad y la interpretación de los clústeres obtenidos.

4.4.1 Repaso teórico del algoritmo K-means

En el **apartado 3.2.1** se ha explicado el algoritmo k-means, pero aquí vamos a profundizar un poco más en su expresión matemática y los pasos que sigue el algoritmo.

Como bien habíamos mencionado, el algoritmo k-means, para minimizar la suma total de distancias al cuadrado entre los puntos y los centroides sigue los siguientes pasos:

1. Inicialización: Se seleccionan aleatoriamente (k) puntos del conjunto de datos como centroides iniciales.

Matemáticamente, esto se puede expresar como:

$$C_1, C_2, \dots, C_k \in \mathbb{R}^p$$

donde C_i es el i-ésimo centroide y p es el número de variables.

Esto en R, podemos hacerlo con la función `sample`, que selecciona aleatoriamente (k) puntos del conjunto de datos. En este caso, sólo queremos las variables numéricas.

```
#  Centroides iniciales aleatorios de las variables numéricas
centroides_iniciales <- datos_num[sample(1:nrow(datos_num), k), ]
```

2. Encontrar la distancia entre cada punto y cada centroide.

Primero tendremos que inicializar una matriz para las distancias, donde cada fila representa un punto y cada columna representa un centroide.

```r
# Inicializar una matriz para las distancias
distancias <- matrix(0, nrow = nrow(datos_num), ncol = k)
```

Una vez tenemos la matriz de distancias, tendremos que, en cada iteración, calcular la distancia de cada punto a cada centroide. En nuestro caso, como habíamos mencionado, vamos a usar la distancia euclidiana y la distancia de Minkowski. Para diferenciar si estamos usando la distancia euclidiana o la de Minkowski, vamos a usar un argumento `method` que nos permita elegir entre ambas, y luego en el bucle `for` que calcula las distancias, vamos a usar un `if` para elegir la distancia a calcular.

- Fórmula para la distancia euclidiana:

$$d_{euclidean}(x_i, C_j) = \sqrt{\sum_{l=1}^{p}(x_{il} - C_{jl})^2}$$

- Fórmula para la distancia de Minkowski:

$$d_{minkowski}(x_i, C_j) = \left(\sum_{l=1}^{p}|x_{il} - C_{jl}|^p\right)^{\frac{1}{p}}$$

La única diferencia entre ambas fórmulas es la potencia a la que elevamos las diferencias entre los puntos y los centroides. En el caso de la distancia euclidiana, elevamos al cuadrado, y en el caso de Minkowski, elevamos a (p), donde el valor de (p) lo elegimos nosotros. En este caso, vamos a usar el valor de (p = 1.5), que es un valor intermedio entre la distancia euclidiana y la de Manhattan.

```r
  for (iter in 1:max_iter) {
    # Calcular las distancias euclidianas
    if (method == "euclidean") {
      for (j in 1:k) {
        distancias[, j] <- sqrt(rowSums((as.matrix(datos_num) -
                                         matrix(rep(as.numeric(centroides[j, ]),
                                                nrow(datos_num)),
                                                ncol = p, byrow = TRUE))^2))
      }
    # Calcular las distancias de Minkowski
    } else if (method == "minkowski") {
      for (j in 1:k) {
        distancias[, j] <- (rowSums(abs(as.matrix(datos_num) -
                                         matrix(rep(as.numeric(centroides[j, ]),
                                                nrow(datos_num)),
                                                ncol = p,
                                                byrow = TRUE))^p_minkowski))^(1/p_minkowski)
```

```
      }
    }
    }
```

3. Asignar cada punto al clúster correspondiente al centroide más cercano. Esto se puede hacer utilizando la función `apply` de R, que aplica una función a cada fila o columna de un data frame.

```
# Asignar cada punto al clúster correspondiente al centroide más cercano
nueva_asignacion <- apply(distancias, 1, which.min)
```

Además queremos que el bucle pare si la asignación de clústeres no cambia, es decir, si la nueva asignación es igual a la anterior. Para ello, vamos a usar un `if` que compare ambas asignaciones.

```
# Si la asignación no cambia, salir del bucle
if (all(nueva_asignacion == asignacion_clusters)) break
```

4. Actualizar los centroides de cada clúster. Para ello, vamos a calcular la media de cada clúster y asignarla al nuevo centroide. Esto se puede hacer utilizando la función `colMeans` de R, que calcula la media de cada columna de un data frame.

```
# Actualizar los centroides de cada clúster
for (idx in 1:k) {
  puntos_cluster <- datos_num[asignacion_clusters == idx, , drop = FALSE]
  if (nrow(puntos_cluster) > 0) {
    centroides[idx, ] <- colMeans(puntos_cluster)
  }
}
```

Esta parte del algoritmo funciona de la siguiente manera:

- Para cada clúster, seleccionamos los puntos que pertenecen a ese clúster (asignacion_clusters == idx).
- Si hay puntos en ese clúster (nrow(puntos_cluster) > 0), calculamos la media de cada variable y la asignamos al nuevo centroide.
- Si no hay puntos en ese clúster, no hacemos nada y el centroide se queda como estaba.

4.4.2 Creación del algoritmo K-means manual

Ahora que hemos explicado el algoritmo k-means, vamos a implementarlo manualmente en R. Para ello, vamos a crear una función `kmeans_manual` que implemente el algoritmo k-means siguiendo los pasos descritos anteriormente.

```
kmeans_manual <- function(datos, k = 5, max_iter = 100, method = "euclidean",
                          p_minkowski = 2, seed = 123) {
```

```r
set.seed(seed)

# Seleccionar solo variables numéricas menos las variables cluster_5, cluster_6 y
#cluster_8 y estandarizar
datos_num <- datos %>%
  select(-c(cluster_5, cluster_6)) %>%
  select(where(is.numeric)) %>%
  scale() %>%
  as.data.frame()

n <- nrow(datos_num)
p <- ncol(datos_num)

# Inicializar centroides aleatorios
centroides <- as.matrix(datos_num[sample(1:n, k), ])
asignacion_clusters <- rep(0, n)
distancias <- matrix(0, nrow = n, ncol = k)

for (iter in 1:max_iter) {
  # Calcular las distancias euclidianas
  if (method == "euclidean") {
    for (j in 1:k) {
      distancias[, j] <- sqrt(rowSums((as.matrix(datos_num) -
                                       matrix(rep(as.numeric(centroides[j, ]),
                                       nrow(datos_num)),
                                       ncol = p, byrow = TRUE))^2))
    }
  # Calcular las distancias de Minkowski
  } else if (method == "minkowski") {
    for (j in 1:k) {
      distancias[, j] <- (rowSums(abs(as.matrix(datos_num) -
                                       matrix(rep(as.numeric(centroides[j, ]),
                                              nrow(datos_num)),
                                              ncol = p,
                                              byrow = TRUE))^p_minkowski))^(1/p_minkowski)
    }
  }
  nueva_asignacion <- apply(distancias, 1, which.min)
  if (all(nueva_asignacion == asignacion_clusters)) break
  asignacion_clusters <- nueva_asignacion
  for (idx in 1:k) {
    puntos_cluster <- datos_num[asignacion_clusters == idx, , drop = FALSE]
    if (nrow(puntos_cluster) > 0) {
```

```
        centroides[idx, ] <- colMeans(puntos_cluster)
      }
    }
  }
  withinss <- numeric(k)
  for (idx in 1:k) {
    puntos_cluster <- datos_num[asignacion_clusters == idx, , drop = FALSE]
    if (nrow(puntos_cluster) > 0) {
      withinss[idx] <- sum(rowSums((as.matrix(puntos_cluster) -
                                    matrix(rep(as.numeric(centroides[idx, ]),
                                           nrow(puntos_cluster)),
                                           ncol = p, byrow = TRUE))^2))
    }
  }
  list(cluster = asignacion_clusters,
       centers = centroides,
       withinss = withinss,
       tot.withinss = sum(withinss),
       iter = iter,
       datos_num = datos_num)
}
```

Con la función `kmeans_manual` implementada, ahora podemos aplicarla a nuestro conjunto de datos. Vamos a aplicar el algoritmo k-means manualmente utilizando la distancia euclidiana y la distancia de Minkowski.

4.4.2.1 Uso de la función `kmeans_manual` con distancia euclidiana

Uso de la función para distancia euclidiana:

```
# Aplicar el algoritmo k-means manual
kmeans_result_manual <- kmeans_manual(datos, k = 5)
table(kmeans_result_manual$cluster)
```

```
##
##    1    2    3    4    5
##  778 1231   82  208  748
```

```
# variación intra-clúster total
intra_cluster_variation_euclidiana <- kmeans_result_manual$tot.withinss
cat("Variación intra-clúster total (Euclidiana):",
intra_cluster_variation_euclidiana, "\n")
```

```
## Variación intra-clúster total (Euclidiana): 6160.19
```

```r
# Graficar los clústeres obtenidos
fviz_cluster(list(data = kmeans_result_manual$datos_num,
                  cluster = kmeans_result_manual$cluster))
```

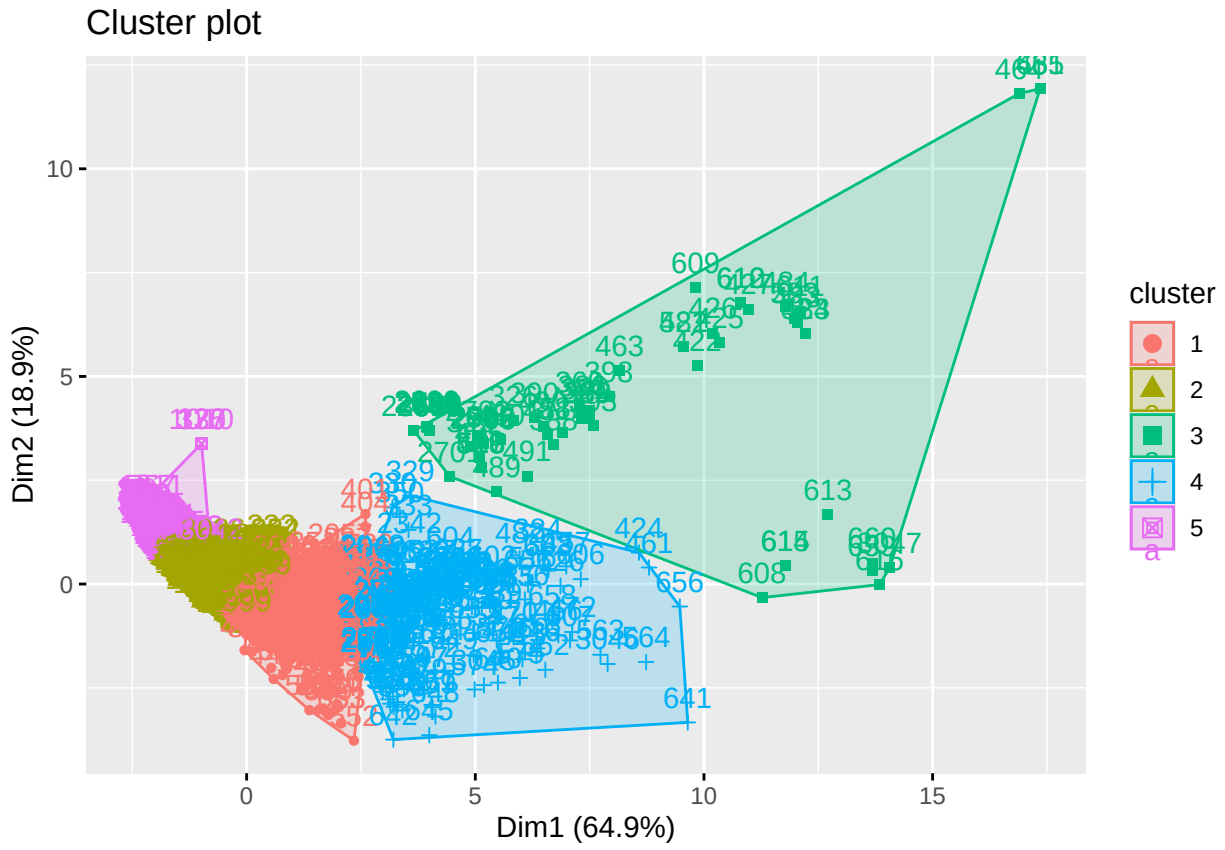

Figura 22: Clústeres de Tarjetas Gráficas (Euclidiana)

Resumen por cluster con medias:

```r
# Añadir la variable de clúster al conjunto de datos original
datos$cluster_manual <- kmeans_result_manual$cluster

# Agrupar los datos por clúster y calcular la media
resumen_por_cluster_media_manual <- datos %>%
  group_by(cluster_manual) %>%
  summarise(across(c(`Memory_Size (GB)`, `Memory_Bus (bits)`, Shaders, TMUs, ROPs,
                     `Memory_clock (MHz)`, `GPU_clock (MHz)`), mean,
                     .names = "media_{.col}"))

# Generar una tabla con kable
kable(resumen_por_cluster_media_manual,
```

```
    caption = "\\label{tab:resumen_por_cluster_media_manual} Resumen de las medias por clúster (Euclid
    col.names = c("Clúster", "Memoria (GB)", "Bus (bits)",
    "Shaders", "TMUs", "ROPs", "Reloj Memoria (MHz)", "Reloj GPU (MHz)"),
    format = "latex", booktabs = TRUE) %>%
kable_styling(latex_options = c("!H", "striped", "scale_down", "hold_position"))
```

Tabla 9: Resumen de las medias por clúster (Euclidiana)

Clúster	Memoria (GB)	Bus (bits)	Shaders	TMUs	ROPs	Reloj Memoria (MHz)	Reloj GPU (MHz)
1	4.0086761	223.6298	1407.32134	89.789203	31.300771	1429.3946	971.9460
2	1.5751929	163.5093	379.07717	31.305443	13.912266	928.4614	750.0991
3	30.1707317	4035.1220	6623.21951	349.365854	100.585366	1035.7317	1143.5732
4	13.8750000	322.3077	4261.23077	204.692308	84.115385	1683.2548	1403.5433
5	0.1734311	146.1390	12.81016	5.308823	5.287433	305.7420	327.4746

Tabla comparativa de las medias de kmeans manual con Euclidiana y el kmeans de R:

Tabla 10: Resumen de las medias por clúster

Clúster	Memoria (GB)	Bus (bits)	Shaders	TMUs	ROPs	Reloj Memoria (MHz)	Reloj GPU (MHz)
1	1.6396161	163.2756	394.39685	32.11417	14.102362	939.9283	758.9039
2	4.4273585	223.1950	1521.00629	96.32201	34.193711	1468.6025	1013.4201
3	56.4444444	3921.7778	10115.55556	485.55556	125.333333	1449.5000	1248.6389
4	0.1772596	146.3586	13.40505	5.36919	5.337317	307.7397	328.9588
5	14.4041451	1276.1865	4747.27461	231.62694	90.943005	1417.5596	1285.3523

Tabla 11: Resumen de las medias por clúster (Euclidiana)

Clúster	Memoria (GB)	Bus (bits)	Shaders	TMUs	ROPs	Reloj Memoria (MHz)	Reloj GPU (MHz)
1	4.0086761	223.6298	1407.32134	89.789203	31.300771	1429.3946	971.9460
2	1.5751929	163.5093	379.07717	31.305443	13.912266	928.4614	750.0991
3	30.1707317	4035.1220	6623.21951	349.365854	100.585366	1035.7317	1143.5732
4	13.8750000	322.3077	4261.23077	204.692308	84.115385	1683.2548	1403.5433
5	0.1734311	146.1390	12.81016	5.308823	5.287433	305.7420	327.4746

Si nos fijamos bien en las tablas de medias, vemos que los resultados son muy similares entre el k-means de R y el k-means manual con distancia euclidiana. Podemes identificar las siguientes relaciones entre los grupos de ambos modelos:

- El grupo 1 del k-means de R corresponde al grupo 2 del k-means manual con distancia euclidiana,
- el grupo 2 corresponde al grupo 1 del k-means manual con distancia euclidiana,
- el grupo 3 corresponde al grupo 3 del k-means manual con distancia euclidiana,
- el grupo 4 corresponde al grupo 5 del k-means manual con distancia euclidiana y,
- el grupo 5 corresponde al grupo 4 del k-means manual con distancia euclidiana.

Esto indica que la implementación del algoritmo k-means manual es correcta y que los resultados obtenidos son consistentes con los obtenidos con la función `kmeans` de R. Este resultado es esperado, ya que la función `kmeans` de R utiliza la distancia euclidiana por defecto para calcular las distancias entre los puntos y los centroides.

Para una visualización más clara, vamos a crear una tabla que muestre la correspondencia entre los grupos del k-means de R y los grupos del k-means manual con distancia euclidiana.

```
# Crear una tabla de correspondencia entre los grupos del k-means de R y
#los grupos del k-means manual con distancia euclidiana
tabla_correspondencia <- data.frame(
  Grupo_kmeans_R = 1:5,
  Grupo_kmeans_manual = c(2, 1, 3, 5, 4)
)

# Unir la tabla de correspondencia con las medias de ambos modelos
tabla_correspondencia_medias <- tabla_correspondencia %>%
  left_join(resumen_por_cluster_media5, by = c("Grupo_kmeans_R" = "cluster_5")) %>%
  left_join(resumen_por_cluster_media_manual, by = c("Grupo_kmeans_manual" = "cluster_manual"),
                                                     suffix = c("_R", "_Manual"))

# Seleccionar y renombrar columnas para mayor claridad
tabla_correspondencia_medias <- tabla_correspondencia_medias %>%
  select(
    Grupo_kmeans_R, Grupo_kmeans_manual,
    starts_with("media_")
  )

# Cambia los nombres de las columnas para que sean más descriptivos
colnames(tabla_correspondencia_medias) <- c(
  "Grupo K-means R", "Grupo K-means Manual",
  "Memoria (GB) R", "Bus (bits) R", "Shaders R", "TMUs R", "ROPs R",
  "Reloj Memoria (MHz) R", "Reloj GPU (MHz) R",
  "Memoria (GB) Manual", "Bus (bits) Manual", "Shaders Manual",
  "TMUs Manual", "ROPs Manual", "Reloj Memoria (MHz) Manual", "Reloj GPU (MHz) Manual"
)

# Poner la tabla en formato largo
tabla_vertical <- tabla_correspondencia_medias %>%
  pivot_longer(
    cols = -c(`Grupo K-means R`, `Grupo K-means Manual`),
    names_to = c("Variable", "Modelo"),
    names_pattern = "(.*) (R|Manual)",
    values_to = "Media"
  ) %>%
```

```
  pivot_wider(
    names_from = Modelo,
    values_from = Media
  )

# Opcional: ordenar por grupo y variable
tabla_vertical <- tabla_vertical %>%
  arrange(`Grupo K-means R`, Variable) %>%
  mutate(
    `R` = round(`R`, 2),
    `Manual` = round(`Manual`, 2)
  )

# Mostrar la tabla en vertical con kable
kable(tabla_vertical,
      caption = "\\label{tab:comparacion_medias_vert} Comparación de medias por grupo
                 y variable: K-means R vs K-means manual",
      col.names = c("Grupo K-means R", "Grupo K-means Manual", "Variable", "Media R", "Media Manual"),
      format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("!H", "striped", "scale_down", "hold_position"))
```

Tabla 12: Comparación de medias por grupo y variable: K-means R vs K-means manual

Grupo K-means R	Grupo K-means Manual	Variable	Media R	Media Manual
1	2	Bus (bits)	163.28	163.51
1	2	Memoria (GB)	1.64	1.58
1	2	ROPs	14.10	13.91
1	2	Reloj GPU (MHz)	758.90	750.10
1	2	Reloj Memoria (MHz)	939.93	928.46
1	2	Shaders	394.40	379.08
1	2	TMUs	32.11	31.31
2	1	Bus (bits)	223.19	223.63
2	1	Memoria (GB)	4.43	4.01
2	1	ROPs	34.19	31.30
2	1	Reloj GPU (MHz)	1013.42	971.95
2	1	Reloj Memoria (MHz)	1468.60	1429.39
2	1	Shaders	1521.01	1407.32
2	1	TMUs	96.32	89.79
3	3	Bus (bits)	3921.78	4035.12
3	3	Memoria (GB)	56.44	30.17
3	3	ROPs	125.33	100.59
3	3	Reloj GPU (MHz)	1248.64	1143.57
3	3	Reloj Memoria (MHz)	1449.50	1035.73
3	3	Shaders	10115.56	6623.22
3	3	TMUs	485.56	349.37
4	5	Bus (bits)	146.36	146.14
4	5	Memoria (GB)	0.18	0.17
4	5	ROPs	5.34	5.29
4	5	Reloj GPU (MHz)	328.96	327.47
4	5	Reloj Memoria (MHz)	307.74	305.74
4	5	Shaders	13.41	12.81
4	5	TMUs	5.37	5.31
5	4	Bus (bits)	1276.19	322.31
5	4	Memoria (GB)	14.40	13.88
5	4	ROPs	90.94	84.12
5	4	Reloj GPU (MHz)	1285.35	1403.54
5	4	Reloj Memoria (MHz)	1417.56	1683.25
5	4	Shaders	4747.27	4261.23
5	4	TMUs	231.63	204.69

Vemos que las medias son muy similares entre ambos modelos, lo que confirma que la implementación del algoritmo k-means manual es correcta y que los resultados obtenidos son consistentes con los obtenidos con la función `kmeans` de R.

```
# Crear un vector de conclusiones
conclusiones_manual_euclidiana <- c(
  "La implementación manual del algoritmo k-means con distancia euclidiana",
  "ha dado resultados muy similares a los obtenidos con la función kmeans de R.",
  "Los grupos obtenidos son consistentes y las medias de las variables son muy parecidas.",
```

```
  "Esto indica que la implementación es correcta y que los resultados son válidos."
)
# Crear un data frame resumen para el modelo k=5
tabla_resultados_manual_euclidiana <- data.frame(
  Modelo = "K-means manual (Euclidiana)",
  Numero_Clusters = length(unique(kmeans_result_manual$cluster)),
  Variacion_Intra_Cluster = intra_cluster_variation_euclidiana,
  Conclusiones = paste(conclusiones_manual_euclidiana, collapse = " ")
)
```

4.4.2.2 Uso de la función `kmeans_manual` con distancia de Minkowski

Ahora que hemos establecido que la implementación manual del algoritmo k-means otorga resultados muy parecidos a los obtenidos con la función `kmeans` de R, vamos a ver si usando la distancia de Minkowski obtenemos resultados diferentes o mejores, especialmente en presencia de valores atípicos.

Uso de la función para Minkowski:

```
# Aplicar el algoritmo k-means manual con Minkowski
kmeans_result_minkowski <- kmeans_manual(datos, k = 5, method = "minkowski",
                                         p_minkowski = 1.5)
table(kmeans_result_minkowski$cluster)
```

```
##
##    1    2    3    4    5
##  778 1231   74  216  748
```

```
# variación intra-clúster total
intra_cluster_variation_minkowski <- kmeans_result_minkowski$tot.withinss
cat("Variación intra-clúster total (Minkowski):", intra_cluster_variation_minkowski, "\n")
```

```
## Variación intra-clúster total (Minkowski): 6155.124
```

```r
# Graficar los clústeres obtenidos
fviz_cluster(list(data = kmeans_result_minkowski$datos_num,
                  cluster = kmeans_result_minkowski$cluster))
```

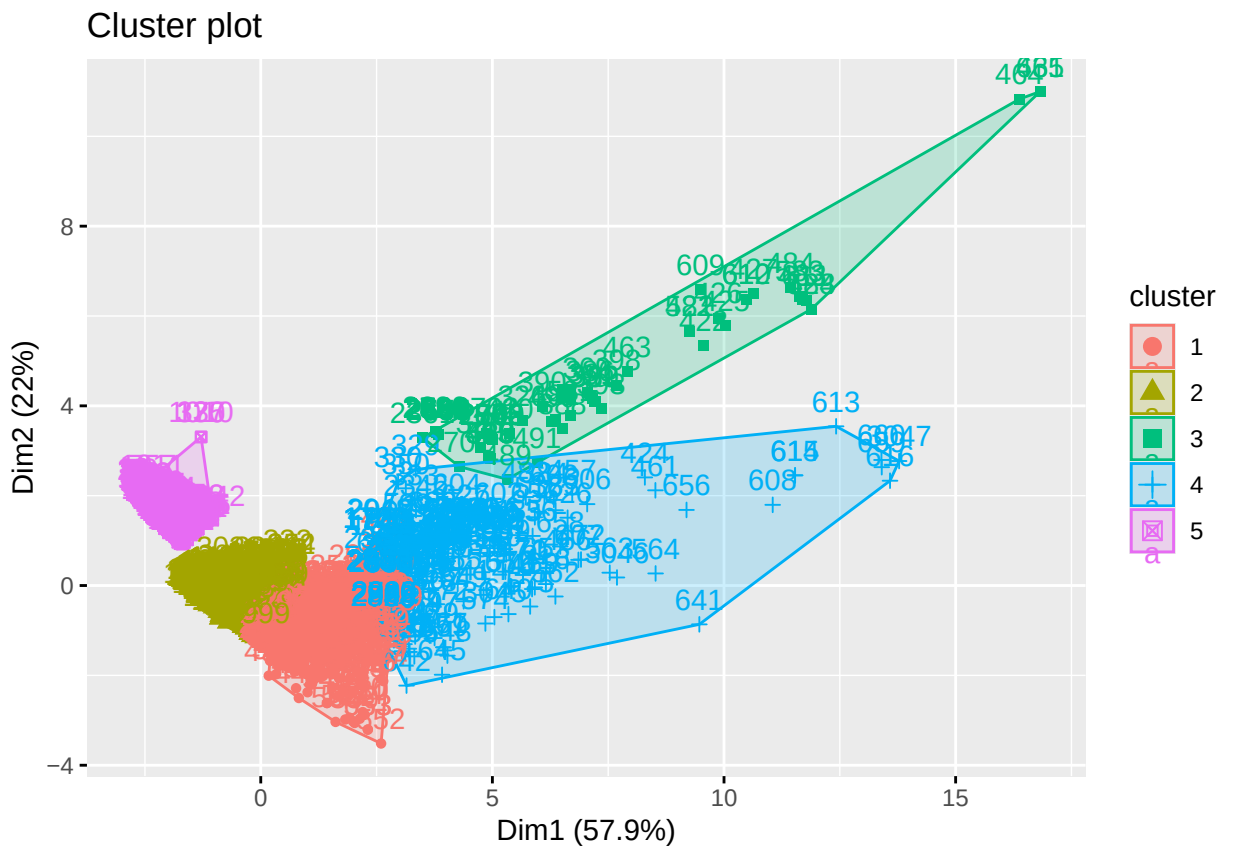

Figura 23: Clústeres de Tarjetas Gráficas (K=5) con K-means Manual (Minkowski)

Como podemos ver en el gráfico, los clústeres obtenidos son también muy similares a los obtenidos con la función `kmeans` de R y con la implementación manual del algoritmo k-means con distancia euclidiana. Sin embargo, la variación intra-clúster total es ligeramente más baja, lo que sugiere que los clústeres son un poco más compactos.

Aquí también vemos que el grupo 3 está más disperso que el resto de grupos. En el análisis de los grupos obtenidos con la función de R, habíamos concluido que se debía a que este grupo estaba principalmente formado por tarjetas gráficas de tipo de memoria HBM y de gama alta, pero con arquitecturas diferentes. Vamos a ver si esto se repite en el análisis de los grupos obtenidos con la distancia de Minkowski. Para ello, vamos a ver contar el número de tarjetas que hay por tipo de memoria en cada clúster, igual que hicimos en el **apartado 3.3.1.4**.

```r
# Contar el número de tarjetas gráficas por tipo de memoria en cada clúster
resumen_por_cluster_memoria_minkowski <- datos %>%
  group_by(cluster_manual = kmeans_result_minkowski$cluster, Memory_Type) %>%
  summarise(n = n()) %>%
  ungroup() %>%
```

```
  arrange(cluster_manual, desc(n))
# Contar el número total de tarjetas gráficas por tipo de memoria
resumen_total_memoria_minkowski <- datos %>%
  group_by(Memory_Type) %>%
  summarise(n = n()) %>%
  ungroup() %>%
  arrange(desc(n))
# Mostrar una tabla con kable
tabla_comparacion_minkowski <- resumen_por_cluster_memoria_minkowski %>%
  left_join(resumen_total_memoria_minkowski, by = "Memory_Type", suffix = c("_cluster", "_total")) %>%
  mutate(porcentaje_cluster = n_cluster / n_total * 100)
# Generar una tabla con kable
tabla_comparacion_minkowski <- kable(tabla_comparacion_minkowski,
      caption = "\\label{tab:comparacion_memoria_minkowski} Comparación del número de
                 tarjetas gráficas por tipo de memoria (Minkowski)",
      col.names = c("Clúster", "Tipo de Memoria", "Número en Clúster",
      "Número Total", "Porcentaje en Clúster"),
      format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("!H", "striped", "scale_down", "hold_position"))
tabla_comparacion_minkowski
```

Tabla 13: Comparación del número de tarjetas gráficas por tipo de memoria (Minkowski)

Clúster	Tipo de Memoria	Número en Clúster	Número Total	Porcentaje en Clúster
1	GDDR5	672	1145	58.6899563
1	GDDR6	96	211	45.4976303
1	HBM2	4	36	11.1111111
1	GDDR5X	2	30	6.6666667
1	LPDDR4X	2	2	100.0000000
1	DDR3	1	424	0.2358491
1	LPDDR5	1	1	100.0000000
2	GDDR5	423	1145	36.9432314
2	DDR3	414	424	97.6415094
2	GDDR3	352	508	69.2913386
2	GDDR4	25	25	100.0000000
2	DDR2	7	140	5.0000000
2	DDR4	4	4	100.0000000
2	GDDR6	3	211	1.4218009
2	HBM2	3	36	8.3333333
3	HBM2	25	36	69.4444444
3	HBM	23	23	100.0000000
3	HBM2e	22	22	100.0000000
3	HBM3	4	4	100.0000000
4	GDDR6	112	211	53.0805687
4	GDDR5	49	1145	4.2794760
4	GDDR5X	28	30	93.3333333
4	GDDR6X	20	20	100.0000000
4	HBM2	4	36	11.1111111
4	GDDR7	3	3	100.0000000
5	DDR	314	314	100.0000000
5	GDDR3	156	508	30.7086614
5	DDR2	133	140	95.0000000
5	SDR	87	87	100.0000000
5	DRAM	17	17	100.0000000
5	EDO	11	11	100.0000000
5	DDR3	9	424	2.1226415
5	VRAM	7	7	100.0000000
5	eDRAM	6	6	100.0000000
5	GDDR2	3	3	100.0000000
5	SGR	2	2	100.0000000
5	FPM	1	1	100.0000000
5	GDDR5	1	1145	0.0873362
5	SGRAM	1	1	100.0000000

Observaciones

- Se observa claramente que el grupo 3 agrupa la mayoría de las tarjetas gráficas con memoria HBM. Esto contrasta con los resultados obtenidos anteriormente (Tabla 3 del **apartado 3.3.1.4**), donde las tarjetas con memoria HBM estaban repartidas en varios grupos. La utilización de la distancia de Minkowski ha permitido que estas tarjetas

se agrupen de forma más coherente, lo que indica que esta métrica es más robusta frente a valores atípicos.

- La mayor dispersión observada en el grupo 3 puede explicarse porque, aunque todas las tarjetas comparten la característica de un ancho de bus elevado (propio de la memoria HBM), pueden diferir en otras especificaciones técnicas debido a pertenecer a distintas generaciones o arquitecturas. Por ello, dentro del grupo existe una mayor variabilidad en el resto de variables.
- El grupo 4 también está más disperso que el resto de grupos. Vemos que en general está formado por tarjetas de gama alta, pero no podemos explicar con estos datos por qué hay cierta dispersión entre las tarjetas gráficas. Por ello, vamos a analizar más a fondo este grupo.

Resumen por cluster con medias:

```
# Añadir la variable de clúster al conjunto de datos original
datos$cluster_minkowski <- kmeans_result_minkowski$cluster
# Agrupar los datos por clúster y calcular la media
resumen_por_cluster_media_minkowski <- datos %>%
  group_by(cluster_minkowski) %>%
  summarise(across(c(`Memory_Size (GB)`, `Memory_Bus (bits)`, Shaders, TMUs, ROPs,
                     `Memory_clock (MHz)`, `GPU_clock (MHz)`), mean,
                     .names = "media_{.col}"))
# Generar una tabla con kable
kable(resumen_por_cluster_media_minkowski,
      caption = "\\label{tab:resumen_por_cluster_media_minkowski} Resumen de las medias por clúster (Mi
      col.names = c("Clúster", "Memoria (GB)", "Bus (bits)",
      "Shaders", "TMUs", "ROPs", "Reloj Memoria (MHz)", "Reloj GPU (MHz)"),
      format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("!H", "striped", "scale_down", "hold_position"))
```

Tabla 14: Resumen de las medias por clúster (Minkowski)

Clúster	Memoria (GB)	Bus (bits)	Shaders	TMUs	ROPs	Reloj Memoria (MHz)	Reloj GPU (MHz)
1	4.0086761	223.6298	1407.32134	89.789203	31.300771	1429.3946	971.9460
2	1.5751929	163.5093	379.07717	31.305443	13.912266	928.4614	750.0991
3	29.4324324	4428.1081	5346.59459	324.864865	91.135135	947.5270	1081.2297
4	14.7314815	325.1852	4786.07407	218.444444	87.962963	1689.4907	1415.2731
5	0.1734311	146.1390	12.81016	5.308823	5.287433	305.7420	327.4746

```
# Gráfico de barras de medias de cada variable por clúster
resumen_por_cluster_media_minkowski %>%
  pivot_longer(cols = -cluster_minkowski, names_to = "Variable", values_to = "Media") %>%
  ggplot(aes(x = factor(cluster_minkowski), y = Media, fill = Variable)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Medias de cada variable por clúster (Minkowski)",
       x = "Clúster", y = "Media") +
```

```
  theme_minimal() +
  scale_fill_brewer(palette = "Set3")
```

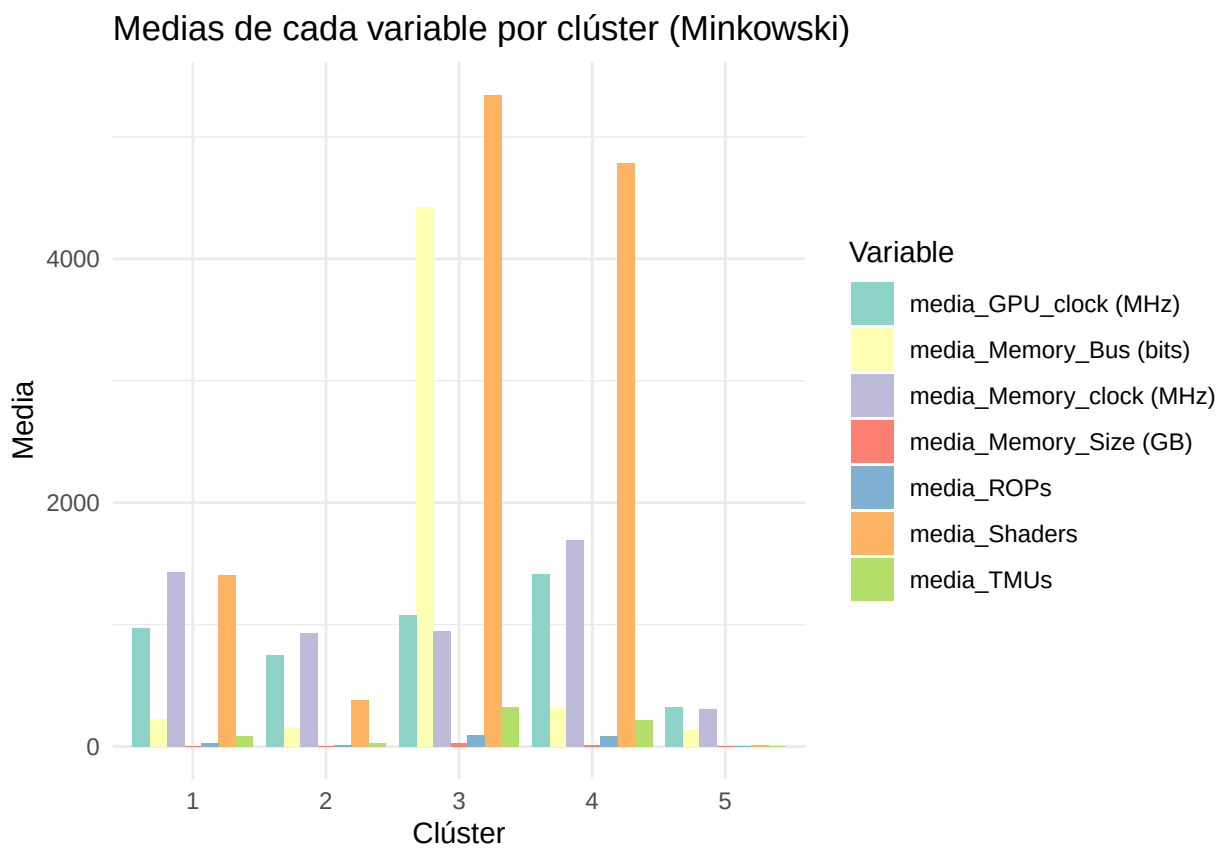

Figura 24: Medias de cada variable por clúster (Minkowski)

Viendo las medias, en efecto podemos observar que el grupo 3 tiene la media más alta de ancho de bus, y que junto con el grupo 4 presentan en general las medias más altas de todas las variables, lo que indica que estos grupos están formados por tarjetas gráficas de gama alta.

Vamos a ver en qué fechas se lanzaron las tarjetas gráficas del grupo 4, para ver si podemos entender mejor por qué hay cierta dispersión entre las tarjetas gráficas de este grupo. Para ello, vamos a crear una tabla con las fechas de lanzamiento de las tarjetas gráficas del grupo 4.

Tabla 15: Tarjetas gráficas del grupo 4 por fecha de lanzamiento

Fecha de lanzamiento	Número de tarjetas gráficas
Después de 2020	104
2016-2020	84
2010-2015	28

Aquí podemos ver que la gran mayoría de las tarjetas gráficas del grupo 4 fueron lanzadas después de 2020 y entre

2016 y 2020, con unas pocas lanzadas entre 2015 y 2015. Esto indica que el grupo 4 está formado por tarjetas gráficas de gama alta, pero de distintas generaciones, lo que podría explicar la dispersión observada, ya que entre 2010-2015 y la actualidad ha habido un gran avance en la tecnología de las tarjetas gráficas, lo que ha permitido mejorar las especificaciones técnicas de las mismas.

Entonces la dispersión podría estar causada por las tarjetas gráficas lanzadas entre 2010 y 2015, que son de generaciones más antiguas y por tanto tienen especificaciones técnicas más bajas que las tarjetas gráficas lanzadas después de 2020.

Dicho esto, parece interesante ver cómo se comportan los demás grupos en cuanto a fechas de lanzamiento. Para ello, vamos a crear un gráfico de barras agrupadas que muestre el número de tarjetas gráficas por grupo y fecha de lanzamiento, donde además añadiremos una etiqueta con el porcentaje del total de tarjetas gráficas por grupo y fecha de lanzamiento, pero sólo en el menor valor de cada grupo.

Con esto, podremos ver cómo se distribuyen las tarjetas gráficas por grupo y fecha de lanzamiento, y si hay una relación entre la dispersión de los grupos y la fecha de lanzamiento de las tarjetas gráficas.

```r
# Calcular el total y el porcentaje por grupo
tarjetas_por_fecha <- datos %>%
  group_by(cluster_minkowski, Released) %>%
  summarise(n = n(), .groups = "drop") %>%
  group_by(cluster_minkowski) %>%
  mutate(total_grupo = sum(n),
         porcentaje = round(100 * n / total_grupo, 1),
         min_n = min(n),
         label = ifelse(n == min_n, paste0(porcentaje, "%"), NA))
```

```r
# Gráfico de barras agrupadas con etiqueta solo en el menor valor
ggplot(tarjetas_por_fecha, aes(x = factor(cluster_minkowski), y = n, fill = Released)) +
  geom_bar(stat = "identity", position = position_dodge(width = 0.8)) +
  geom_text(aes(label = label),
            position = position_dodge(width = 0.8),
            vjust = -0.5,
            color = "black",
            size = 3,
            na.rm = TRUE) +
  labs(
    title = "Tarjetas gráficas por grupo y fecha de lanzamiento",
    x = "Grupo",
    y = "Número de tarjetas gráficas",
    fill = "Fecha de lanzamiento"
  ) +
  theme_minimal() +
  scale_fill_brewer(palette = "Set2")
```

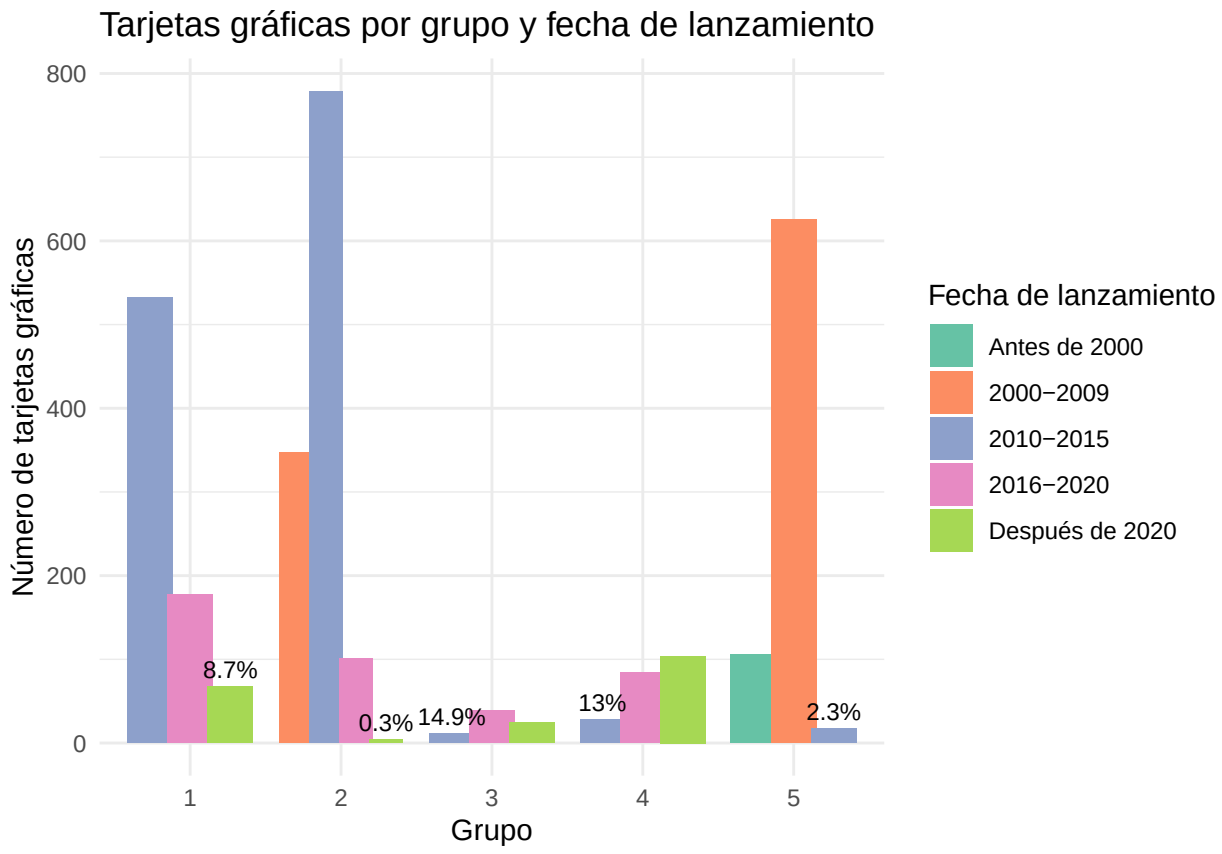

Figura 25: Tarjetas gráficas por grupo y fecha de lanzamiento

Según el gráfico de la **Figura 25**, podemos ver que en los grupos 3 y 4, que eran los que más dispersos estaban, tienen los porcentajes más altos del periodo de fecha de lanzamiento que menor número de tarjetas gráficas tienen en sus respectivos grupos. Además, en estos dos grupos, la mayoría de las tarjetas gráficas son de entre 2016 y después de 2020, lo que podría explicar la dispersión observada en estos grupos, ya que las tarjetas gráficas de estas fechas son de gama alta y tienen especificaciones técnicas más altas que las tarjetas gráficas de generaciones anteriores.

Para confirmar esta hipótesis, vamos a ver si los grupos 3 y 4 tienen una mayor variabilidad en las especificaciones técnicas de las tarjetas gráficas. Para ello, vamos a calcular la desviación estándar de cada variable por grupo y ver si hay una mayor variabilidad en los grupos 3 y 4.

```
# Calcular la desviación estándar de cada variable por grupo
desviacion_estandar_por_grupo <- datos %>%
  group_by(cluster_minkowski) %>%
  summarise(across(c(`Memory_Size (GB)`, `Memory_Bus (bits)`, Shaders, TMUs, ROPs,
                     `Memory_clock (MHz)`, `GPU_clock (MHz)`), sd,
                     .names = "sd_{.col}"))
```

```r
# Gráfico de barras de desviación estándar de cada variable por clúster
ggplot(desviacion_estandar_por_grupo %>%
       pivot_longer(cols = -cluster_minkowski, names_to = "Variable",
                    values_to = "Desviación Estándar"),
     aes(x = factor(cluster_minkowski), y = `Desviación Estándar`, fill = Variable)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Desviación estándar de cada variable por clúster (Minkowski)",
       x = "Clúster", y = "Desviación Estándar") +
  theme_minimal() +
  scale_fill_brewer(palette = "Set3")
```

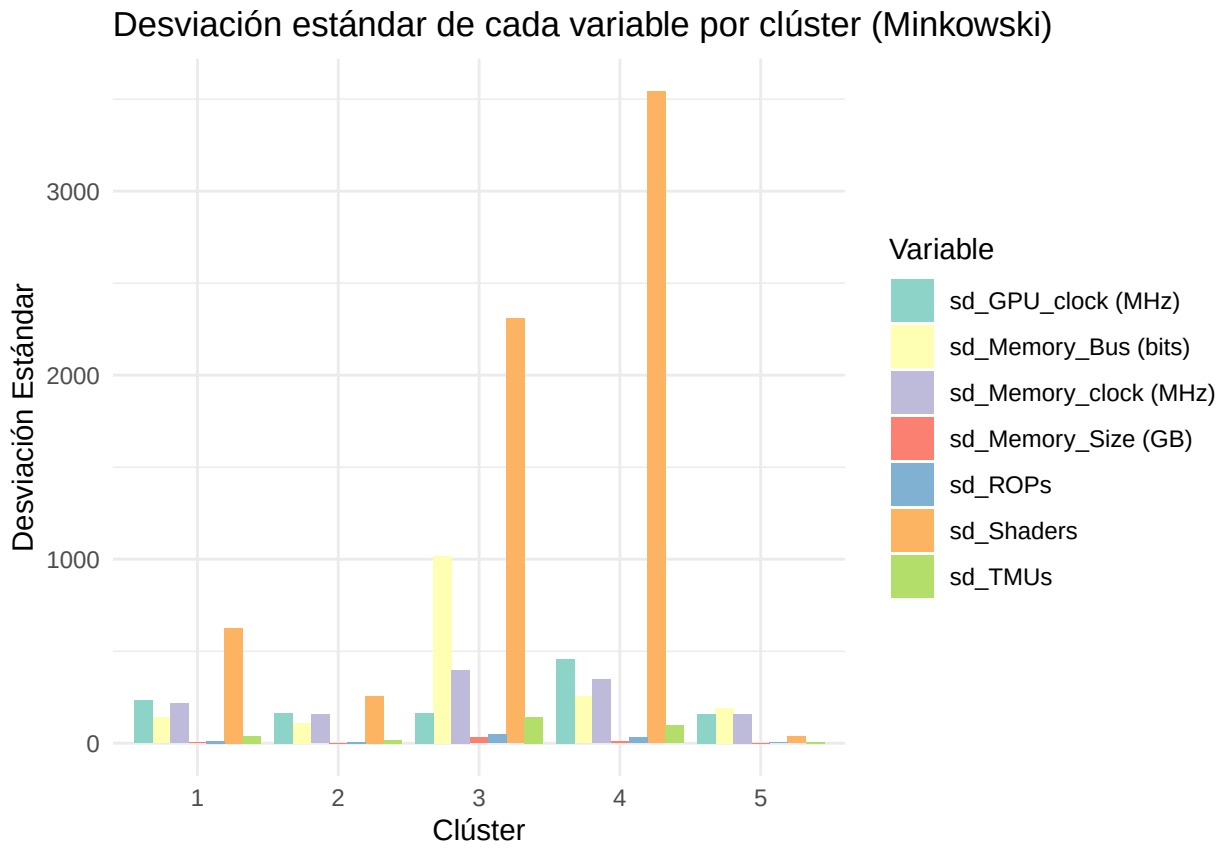

Figura 26: Desviación estándar de cada variable por clúster (Minkowski)

Vemos que efectivamente los grupos 3 y 4 tienen una mayor variabilidad en las especificaciones técnicas de las tarjetas gráficas, lo que confirma nuestra hipótesis de que la dispersión observada en estos grupos se debe a la presencia de tarjetas gráficas de distintas generaciones y arquitecturas.

```r
# Crear un vector de conclusiones
conclusiones_manual_minkowski <- c(
  "La implementación manual del algoritmo k-means con distancia de Minkowski",
```

```
  "ha dado resultados muy similares a los obtenidos con la función kmeans de R.",
  "Los grupos obtenidos son consistentes y las medias de las variables son muy parecidas.",
  "La variación intra-clúster total es ligeramente más baja que con la distancia euclidiana,",
  "lo que sugiere que los clústeres son un poco más compactos.",
  "El grupo 3 agrupa la mayoría de las tarjetas gráficas con memoria HBM,",
  "mientras que el grupo 4 está formado por tarjetas gráficas de gama alta, pero de distintas generacio
  "La dispersión observada en estos grupos se debe a la presencia de tarjetas gráficas de distintas gen
)
# Crear un data frame resumen para el modelo k=5
tabla_resultados_manual_minkowski <- data.frame(
  Modelo = "K-means manual (Minkowski)",
  Numero_Clusters = length(unique(kmeans_result_manual$cluster)),
  Variacion_Intra_Cluster = intra_cluster_variation_minkowski,
  Conclusiones = paste(conclusiones_manual_minkowski, collapse = " ")
)
```

5 Conclusiones

5.1 Valoración del cumplimiento de los objetivos

A lo largo de este trabajo se han cumplido los objetivos planteados al inicio del estudio:

1. **Describir y preparar la base de datos:** Se realizó una exhaustiva limpieza y transformación de los datos, identificando y tratando valores atípicos, variables irrelevantes y datos faltantes. Esto permitió obtener un conjunto de datos adecuado y fiable para el análisis posterior.

2. **Explorar y analizar las variables clave:** Se llevó a cabo un análisis exploratorio detallado de las variables numéricas y categóricas, identificando patrones, tendencias y relaciones relevantes entre las diferentes características de las tarjetas gráficas. Este análisis facilitó la interpretación de los resultados obtenidos en las fases posteriores.

3. **Implementar y comparar métodos de agrupamiento:** Se aplicó el algoritmo K-means en diferentes escenarios: utilizando únicamente variables numéricas, combinando variables numéricas y categóricas, y mediante una implementación manual que permitió experimentar con distintas métricas de distancia (euclidiana y Minkowski). Además, para poder reducir la dimensionalidad del conjunto de datos en el caso del modelo combinado de variables numéricas y categóricas, se empleó el análisis de componentes principales (PCA), lo que permitió sintetizar la información y facilitar la interpretación de los resultados.

4. **Interpretar y validar los grupos obtenidos:** Se analizaron en profundidad las características de los clústeres resultantes, identificando patrones comunes y diferencias entre los grupos. Además, se evaluó la robustez y coherencia de los agrupamientos en función de las especificaciones técnicas y la evolución tecnológica de las tarjetas gráficas.

5. **Extraer conclusiones relevantes:** Finalmente, se sintetizaron los hallazgos obtenidos, destacando las implicaciones prácticas y teóricas del agrupamiento de tarjetas gráficas.

En definitiva, el trabajo ha permitido alcanzar todos los objetivos propuestos, proporcionando una visión integral y objetiva sobre la clasificación y segmentación de tarjetas gráficas en función de sus características técnicas, y demostrando la utilidad del PCA como herramienta para el análisis y reducción de la dimensionalidad en conjuntos de datos complejos. Además, la implementación manual del algoritmo K-means ha permitido profundizar en el entendimiento del funcionamiento de este método de agrupamiento y su aplicación en el contexto de las tarjetas gráficas.

5.2 Conclusiones finales

En primer lugar, se presenta una tabla con los resultados obtenidos en cada uno de los modelos:

```
# Combinar todos los data frames de resultados
tabla_resultados_comparativa <- rbind(
  tabla_resultados_kmeans,
  tabla_resultados_manual_euclidiana,
  tabla_resultados_manual_minkowski
)

# Generar una tabla con kable
kable(tabla_resultados_comparativa,
      caption = "\\label{tab:comparacion_modelos_todos} Comparación de los resultados de los modelos K-m
      col.names = c("Modelo", "Número de Clústeres",
      "Variación Intra-Cluster", "Conclusiones"),
      format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("!H", "striped", "scale_down", "hold_position")) %>%
  column_spec(4, width = "8cm")
```

Tabla 16: Comparación de los resultados de los modelos K-means

Modelo	Número de Clústeres	Variación Intra-Cluster	Conclusiones
Solo numéricas	5	6157.303	El grupo 3 se caracteriza por tener un ancho de banda alto, lo que podría indicar que está formado por tarjetas gráficas de tipo HBM. Los grupos 1, 2 y 4 tienen una varianza baja en todas las variables, lo que sugiere que estos grupos están muy homogéneos y bien definidos.
Numéricas y categóricas	8	9501.153	Se tuvo que hacer un PCA para reducir la dimensionalidad de los datos. Después de aplicar el PCA y realizar el clustering, se observó que los grupos 1 y 2 tienen cierta superposición, igual que los grupos 3 y 6. Había un par de grupos bien definidos y otros que estaban más dispersos. Luego se concluyó que los grupos estaban principalmente definidos por el tipo de memoria de las tarjetas gráficas.
K-means manual (Euclidiana)	5	6160.190	La implementación manual del algoritmo k-means con distancia euclidiana ha dado resultados muy similares a los obtenidos con la función kmeans de R. Los grupos obtenidos son consistentes y las medias de las variables son muy parecidas. Esto indica que la implementación es correcta y que los resultados son válidos.
K-means manual (Minkowski)	5	6155.124	La implementación manual del algoritmo k-means con distancia de Minkowski ha dado resultados muy similares a los obtenidos con la función kmeans de R. Los grupos obtenidos son consistentes y las medias de las variables son muy parecidas. La variación intra-clúster total es ligeramente más baja que con la distancia euclidiana, lo que sugiere que los clústeres son un poco más compactos. El grupo 3 agrupa la mayoría de las tarjetas gráficas con memoria HBM, mientras que el grupo 4 está formado por tarjetas gráficas de gama alta, pero de distintas generaciones. La dispersión observada en estos grupos se debe a la presencia de tarjetas gráficas de distintas generaciones y arquitecturas.

En la **Tabla 16** podemos observar que el único modelo que incluía variables categóricas (modelo combinado) no ha sido capaz de encontrar una estructura clara en los datos, lo que indica que las variables categóricas no aportan información relevante para el agrupamiento de tarjetas gráficas para nuestro caso. Por tanto, los modelos que únicamente utilizan variables numéricas (tanto el K-means de R como la implementación manual) han sido los más efectivos para identificar patrones y segmentar las tarjetas gráficas en clústeres coherentes, donde los tres modelos han obtenido resultados muy similares en cuanto al número de clústeres y la variación intra-clúster, así como la clasificación de las tarjetas gráficas. No obstante, cabe destacar que el modelo con la distancia de Minkowski, ha presentado una variación intra-cluster ligeramente menor, lo que sugiere que los clústeres son un poco más compactos y homogéneos, especialmente en el caso de las tarjetas gráficas con memoria HBM.

Hablando de las tarjetas gráficas con memoria HBM, durante todo el trabajo hemos visto que estas tarjetas gráficas se diferencian mucho más de las demás, ya que tienen un uso muy especial comparado con el uso habitual de las tarjetas gráficas, que es el gaming. Este uso especial se resume en que las tarjetas gráficas con memoria HBM están diseñadas para tareas que requieren un alto rendimiento en el procesamiento de datos, como la inteligencia artificial, el aprendizaje automático, la computación científica y el procesamiento de datos biomédicos. Esto lo consiguen con un ancho de bus elevado, que se ha podido observar a lo largo del análisis, tanto en el análisis exploratorio de datos como en los modelos de agrupamiento, donde hemos visto que las tarjetas gráficas con memoria HBM se agrupan en un clúster propio, lo que indica que estas tarjetas gráficas tienen unas especificaciones técnicas muy diferentes al resto

de tarjetas gráficas.

5.3 Reflexión personal

La realización de este Trabajo de Fin de Grado ha supuesto un importante proceso de aprendizaje y desarrollo personal en varios ámbitos. En primer lugar, he adquirido una mayor destreza en el uso de R Markdown como herramienta integral para la elaboración de documentos científicos y técnicos. Gracias a este proyecto, he aprendido a gestionar referencias bibliográficas mediante archivos .bib, a utilizar el sistema de índices y referencias cruzadas de manera automática, y a estructurar un documento extenso de forma clara y profesional. Considero que estas competencias serán muy valiosas tanto en futuros estudios como en el ámbito laboral, donde la capacidad de documentar y presentar resultados de manera formal y reproducible es cada vez más demandada.

Otro aspecto fundamental ha sido la mejora en la presentación y comunicación de resultados. El hecho de tener que exponer los análisis y conclusiones de forma coherente, estructurada y comprensible me ha obligado a reflexionar sobre la mejor manera de transmitir la información, adaptando el lenguaje y los recursos gráficos al público objetivo. Esta habilidad es esencial en cualquier entorno profesional, ya que la claridad en la comunicación de resultados puede marcar la diferencia en la toma de decisiones.

Asimismo, este trabajo me ha permitido desarrollar la capacidad de investigar a fondo, enfrentándome a problemas nuevos y buscando soluciones de manera autónoma. He aprendido a consultar fuentes especializadas, comparar enfoques y adaptar las metodologías a las características concretas de los datos y los objetivos del análisis.

En el plano técnico, la implementación manual del algoritmo K-means ha sido especialmente enriquecedora. No solo me ha permitido comprender en profundidad el funcionamiento interno del método, sino también experimentar con diferentes métricas de distancia y valorar su impacto en los resultados. Esta experiencia me ha dado una visión más crítica y flexible sobre el uso de algoritmos de agrupamiento en la práctica.

Por último, he ampliado notablemente mis conocimientos sobre tarjetas gráficas y sus aplicaciones, descubriendo que su utilidad va mucho más allá del ámbito de los videojuegos. Ahora comprendo mejor su papel en áreas como la inteligencia artificial, la computación científica o el procesamiento de datos biomédicos, lo que me ha permitido valorar la importancia de la tecnología en contextos muy diversos.

En definitiva, este TFG ha supuesto un reto que me ha permitido crecer tanto a nivel técnico como personal, y me ha motivado a seguir aprendiendo y profundizando en el análisis de datos y sus aplicaciones reales.

6 Líneas futuras

A partir de los resultados obtenidos en este trabajo, se abren varias líneas interesantes para continuar y profundizar en el análisis:

- **Optimización de los modelos de agrupamiento:** Sería interesante explorar técnicas avanzadas de optimización de agrupamiento para la selección automática del número óptimo de clústeres y la mejora de la robustez de los resultados. Además, se podrían comparar otros algoritmos de clustering, como DBSCAN que se había mencionado en el **apartado 4.1**, para evaluar su desempeño frente a K-means en este contexto.

- **Exploración de nuevos ámbitos de aplicación:**
  Los resultados del clustering han puesto de manifiesto la existencia de grupos de tarjetas gráficas con características técnicas diferenciadas, especialmente aquellas con memoria HBM. En el futuro, me gustaría profundizar en el análisis de otros segmentos identificados por los modelos de agrupamiento, investigando sus aplicaciones específicas en ámbitos como el *gaming*, la computación científica, el procesamiento de datos biomédicos o incluso el renderizado profesional. Esto permitiría comprender mejor el impacto de la evolución tecnológica de las GPUs en diferentes sectores y orientar el análisis hacia casos de uso concretos.

Estas líneas futuras permitirían enriquecer el estudio, aportar una visión más completa sobre la segmentación tecnológica de las tarjetas gráficas y abrir nuevas oportunidades de investigación aplicada en el ámbito del análisis de datos.

Referencias

Cui, Mengyao, et al. “Introduction to the k-Means Clustering Algorithm Based on the Elbow Method.” *Accounting, Auditing and Finance*, vol. 1, no. 1, 2020, pp. 5–8.

Eklund, Anders, et al. “fMRI Analysis on the GPU—Possibilities and Challenges.” *Computer Methods and Programs in Biomedicine*, vol. 105, no. 2, 2012, pp. 145–61.

Kalaiselvi, T., et al. “Survey of Using GPU CUDA Programming Model in Medical Image Analysis.” *Informatics in Medicine Unlocked*, vol. 9, 2017, pp. 133–44.

Manual, A. Beginner’s. *An Introduction to Statistical Learning with Applications in r*. 2013.

Neshat, Mehdi, et al. “Wind Turbine Power Output Prediction Using a New Hybrid Neuro-Evolutionary Method.” *Energy*, vol. 229, 2021, p. 120617.

Oyelade, Olanrewaju Jelili, et al. “Application of k Means Clustering Algorithm for Prediction of Students Academic Performance.” *arXiv Preprint arXiv:1002.2425*, 2010.

Saxena, Amit, et al. “A Review of Clustering Techniques and Developments.” *Neurocomputing*, vol. 267, 2017, pp. 664–81.

Singh, Archana, et al. “K-Means with Three Different Distance Metrics.” *International Journal of Computer Applications*, vol. 67, no. 10, 2013.

Zhang, Yichen, et al. “A GPU-Based Computational Framework That Bridges Neuron Simulation and Artificial Intelligence.” *Nature Communications*, vol. 14, no. 1, 2023, p. 5798.

Zhao, Ying, et al. “Hierarchical Clustering Algorithms for Document Datasets.” *Data Mining and Knowledge Discovery*, vol. 10, 2005, pp. 141–68.