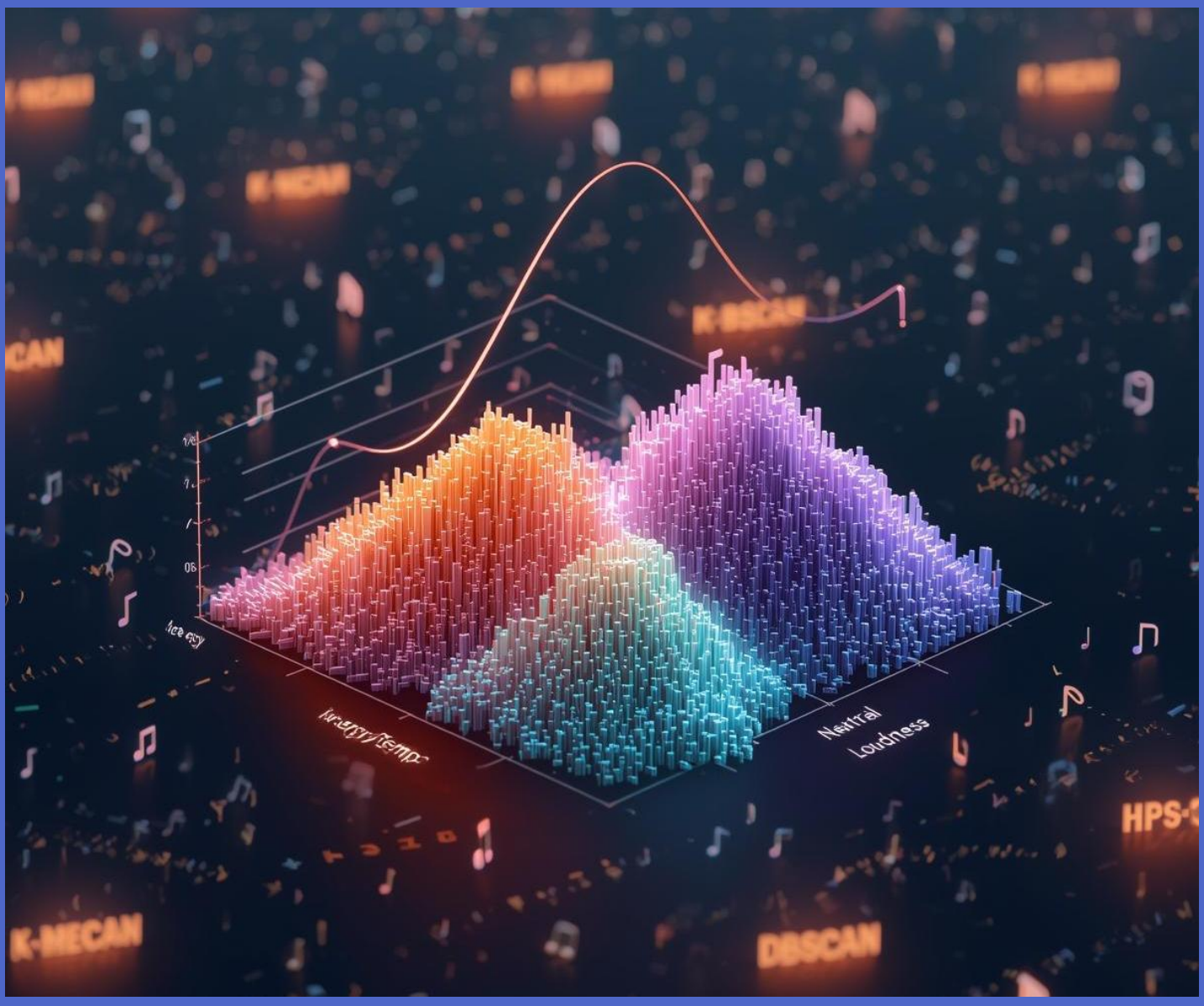
HCL (BIG DATA)

TAREA I UD2



ALUMNO CESUR

25/26

Alejandro Muñoz de la Sierra

PROFESOR

Óscar González Núñez

I N T R O D U C C I O N

Hoy en día, la industria musical se mueve tanto por ritmos como por datos. Plataformas como Spotify, Apple Music o YouTube Music no adivinan lo que te gusta; lo calculan. Su éxito se basa en algoritmos que mastican millones de canciones para entender patrones invisibles y predecir tu próximo tema favorito antes de que tú sepas que existe. Gracias a esto pueden organizar catálogos inmensos y personalizar la experiencia para cada usuario.

En este contexto, el Million Song Dataset es el gigante con el que todos quieren jugar, una referencia absoluta para investigadores y analistas. Pero es tan masivo que intentar abarcarlo todo desde el principio puede ser paralizante. Por eso, en este proyecto decidimos bajar el balón al suelo y trabajar con una muestra de 10.000 canciones. Es el tamaño perfecto para ensuciarse las manos, probar distintas técnicas de clustering y entender realmente el potencial de estas herramientas sin perderse en el ruido. El objetivo aquí es experimentar, probar distintos métodos y ver qué nos cuentan los datos cuando sabemos escucharlos. Vamos a meternos de lleno en el clustering no supervisado utilizando R y RStudio. La idea no es solo correr código, sino vivir el ciclo completo de un proyecto real: desde cargar el dataset y explorarlo, hasta esa parte crítica donde eliges variables, normalizas y pones a prueba distintos algoritmos. Lo fascinante del clustering es que nos permite descubrir conexiones entre canciones que a simple vista se nos escapan. Sin necesidad de que nadie nos etiquete nada, podemos agrupar temas basándonos en su tempo, su energía o su intensidad. Probamos de todo un poco, como k-means, clustering jerárquico y DBSCAN, para ver qué enfoque capta mejor la esencia de los datos. Es básicamente mirar bajo el capó de cómo funcionan esos sistemas de recomendación que usamos a diario y es un punto de partida genial si quieres meterte en proyectos de minería de datos más serios.

0 1
E N T O R N O D E T R A B A J O

1.1 Software

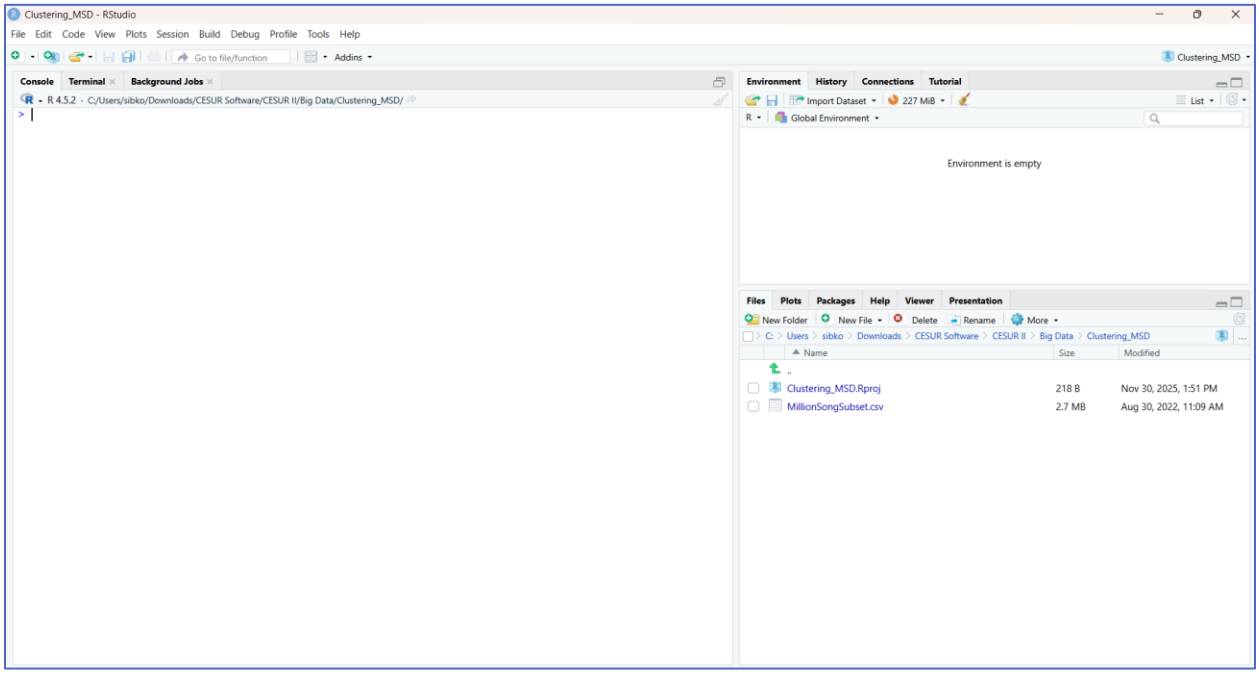
Para trabajar sin dolores de cabeza, apostamos por una combinación sólida:

-R como el motor principal del lenguaje.

-RStudio como nuestra cabina de mando, porque facilita enormemente organizar el código y ver los resultados sin perderse.

Aunque en la teoría WEKA parece más sencillo de manejar, leí muchas recomendaciones para priorizar el análisis de datos en RStudio de cara al futuro.

Creamos una carpeta para nuestro proyecto, donde estará el dataset y todos los archivos que se generen y lo abrimos como nuevo proyecto en Rstudio.



1.2 Paquetes necesarios

Instalamos los paquetes que vamos a necesitar.

```
> install.packages(c("readr","dplyr","ggplot2","cluster","factoextra","dbscan","patchwork","mclust"))
WARNING: Rtools is required to build R packages but is not currently installed. Please download and install
the appropriate version of Rtools before proceeding:

https://cran.rstudio.com/bin/windows/Rtools/

Installing packages into 'C:/Users/sibko/AppData/Local/R/win-library/4.5'
```

y cargamos las librerías de esos paquetes para poder acceder a los métodos y funciones.

```
>
> library(readr); library(dplyr); library(ggplot2)
> library(cluster); library(factoextra); library(dbscan); library(patchwork)
> library(mclust)  # opcional para clustering model-based
>
```

Cada uno tiene su misión:

Readr para leer CSV con buen rendimiento.
dplyr para que leer y manipular datos sea fluido;
ggplot2 para gráficos.
patchwork facilita combinar gráficos.
cluster, **factoextra** para métricas y visualizaciones de clustering.
dbscan para clustering por densidad.
mclust para clustering basado en modelos gaussianos

0 2
D A T A S E T

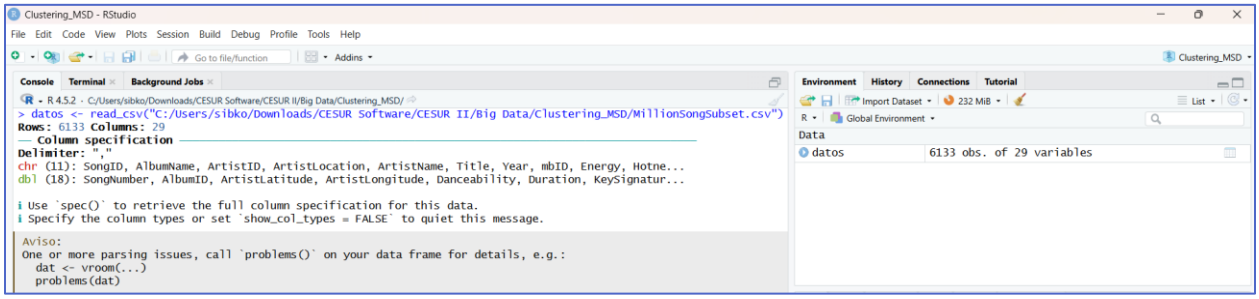
Aunque la práctica nos lleva a la página de **Million Song Dataset** , vimos que para trabajar con los datos en Rstudio necesitábamos las carpetas tratadas y convertidas en un archivo .csv, por lo que optamos por descargar este archivo MillionSongSubset.csv desde la pagina https://www.kaggle.com/ especilizada en machine learning y data.

Guardamos nuestra muestra de 10.000 canciones en la carpeta del proyecto y procedemos a cargala en nuestro entorno:

Asignamos el contenido csv a la variable **datos**

read_csv nos devuelve un **tibble** (transformación del Dataset en filas y columnas) mucho más manejable.

Vemos que se usa <- para asignar en R , a diferencia del = en otros lenguajes



El Dataset tiene 6133 filas y 29 columnas.
chr 11 columnas de tipo carácter (texto).
dbl 18 columnas de tipo numérico decimal.

Con **glimpse** y **colnames** le damos un primer vistazo rápido a la estructura y los datos antes de tomar decisiones sobre qué variables usar.

```
> glimpse(datos)
Rows: 6,133
Columns: 29
$ SongNumber              <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22…
$ SongID                  <chr> "SOVLGJY12A8C13FBED", "SOGXWRE12AC468BE24", "SOIJPPR12A6D4F3945", "SOIMSZI12A…
$ AlbumID                 <dbl> 223563, 781922, 85554, 440144, 619838, 279570, 555518, 157333, 254710, 281536…
$ AlbumName               <chr> "Call of the Mastodon", "Out Of Time", "Wolfgang Amadeus Mozart: Requiem Mass…
$ ArtistID                <chr> "ARMQHX71187B9890D3", "ARTXS851187FB411C7", "ARYISOC1187B9998F1", "ARJFOC0118…
$ ArtistLatitude          <dbl> NA, NA, 36.03256, 33.59233, NA, NA, NA, NA, 41.88415, NA, NA, NA, NA, 53.4196…
$ ArtistLocation          <chr> "Atlanta GA", "Essex England", "Nashville Franklin Brentwood", "Lubbock TX", …
$ ArtistLongitude         <dbl> NA, NA, -86.78916, -101.85587, NA, NA, NA, NA, -87.63241, NA, NA, NA, NA, -8.…
$ ArtistName              <chr> "Mastodon", "Chris Farlowe", "Choir & Great Symphony Orchestra of the All-Uni…
$ Danceability            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,…
$ Duration                <dbl> 280.21506, 177.57995, 541.23057, 114.31138, 110.88934, 651.59791, 142.86322, …
$ KeySignature            <dbl> 5, 7, 9, 4, 0, 4, 1, 11, 4, 3, 0, 10, 4, 0, 6, 3, 2, 7, 10, 11, 0, 10, 7, 11,…
$ KeySignatureConfidence  <dbl> 0.555, 0.345, 0.000, 0.470, 0.293, 0.354, 0.607, 0.605, 0.805, 0.601, 0.124, …
$ Tempo                   <dbl> 173.205, 146.913, 109.051, 221.858, 100.296, 86.589, 87.897, 121.968, 217.289…
$ TimeSignature           <dbl> 5, 1, 4, 4, 4, 4, 4, 4, 1, 7, 0, 4, 4, 1, 4, 4, 4, 4, 5, 4, 3, 4, 4, 5, 4, 4,…
$ TimeSignatureConfidence <dbl> 0.120, 0.000, 0.271, 0.000, 1.000, 0.961, 1.000, 0.943, 0.000, 0.854, 0.000, …
$ Title                   <chr> "Deep Sea Creature", "Paint It Black", "Requiem Mass_ K. 626_ 1791: Agnus Dei…
$ Year                    <chr> "2001", "1966", "0", "2000", "0", "2000", "0", "0", "2001", "0", "1998", "199…
$ mbID                    <chr> "bc5e2ad6-0a4a-4d90-b911-e9a7e6861727", "6bb0bcaa-f96f-4c1a-b51d-c766db2af8ae…
$ Energy                  <chr> "0.0", "0.0", "0.0", "0.0", "0.0", "0.0", "0.0", "0.0", "0.0", "0.0", "0.0", …
$ ArtistFamiliarity       <dbl> 0.78046175, 0.54339323, 0.48448556, 0.69992476, 0.30546619, 0.89717804, 0.542…
$ Hotness                 <chr> "0.574274730517", "0.338575026457", "0.373952779351", "0.449668690752", "0.03…
$ end_of_fade_in          <dbl> 0.238, 0.346, 0.827, 0.142, 0.136, 0.131, 0.000, 0.386, 0.833, 3.442, 0.146, …
$ key                     <dbl> 5, 7, 9, 4, 0, 4, 1, 11, 4, 3, 0, 10, 4, 0, 6, 3, 2, 7, 10, 11, 0, 10, 7, 11,…
$ keyConfidence           <dbl> 0.555, 0.345, 0.000, 0.470, 0.293, 0.354, 0.607, 0.605, 0.805, 0.601, 0.124, …
$ Loudness                <dbl> -3.306, -11.027, -15.586, -10.261, -14.088, -10.718, -9.532, -6.010, -6.028, …
$ mode                    <dbl> 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,…
$ mode_confidence         <dbl> 0.500, 0.590, 0.000, 0.305, 0.372, 0.427, 0.704, 0.529, 0.666, 0.560, 0.256, …
$ start_of_fade_out       <chr> "275.528", "175.078", "537.24", "107.735", "105.599", "606.064", "137.033", "…
```

```
> colnames(datos)
 [1] "SongNumber"              "SongID"                  "AlbumID"                 "AlbumName"
 [5] "ArtistID"                "ArtistLatitude"          "ArtistLocation"          "ArtistLongitude"
 [9] "ArtistName"              "Danceability"            "Duration"                "KeySignature"
[13] "KeySignatureConfidence"  "Tempo"                   "TimeSignature"           "TimeSignatureConfidence"
[17] "Title"                   "Year"                    "mbID"                    "Energy"
[21] "ArtistFamiliarity"       "Hotness"                 "end_of_fade_in"          "key"
[25] "keyConfidence"           "Loudness"                "mode"                    "mode_confidence"
[29] "start_of_fade_out"
>
```

0 3

PRIMERAS DECISIONES CON LOS DATOS

Exploración inicial y selección de atributos

3.1 Primer contacto

Antes de modelar nada, hay que ver si es necesario "tocar" los datos para ver con qué estamos tratando:

dim(datos) número de filas y columnas.

```
> dim(datos)
[1] 6133   29
```

head(datos) muestra las primeras 6 filas y un resumen de todas las columnas.

```
> head(datos)
# A tibble: 6 × 29
  SongNumber SongID          AlbumID AlbumName ArtistID ArtistLatitude ArtistLocation ArtistLongitude ArtistName
       <dbl> <chr>             <dbl> <chr>     <chr>             <dbl> <chr>                   <dbl> <chr>
1          1 SOVLGJY12A8C1…   223563 Call of … ARMQHX7…             NA  Atlanta GA                 NA  Mastodon
2          2 SOGXWRE12AC46…   781922 Out Of T… ARTXS85…             NA  Essex England              NA  Chris Far…
3          3 SOIJPPR12A6D4…    85554 Wolfgang… ARYISOC…             36.0 Nashville Fra…           -86.8 Choir & G…
4          4 SOIMSZI12AB01…   440144 Voices O… ARJFOC0…             33.6 Lubbock TX              -102.  Delbert M…
5          5 SOZECOE12AB01…   619838 The Very… ARQ8FR8…             NA  NA                      NA  Gloria De…
6          6 SOMSKPE12A6D4…   279570 Now It's… ARBXIDR…             NA  NA                      NA  Avril
# i 20 more variables: Danceability <dbl>, Duration <dbl>, KeySignature <dbl>, KeySignatureConfidence <dbl>,
#   Tempo <dbl>, TimeSignature <dbl>, TimeSignatureConfidence <dbl>, Title <chr>, Year <chr>, mbID <chr>,
#   Energy <chr>, ArtistFamiliarity <dbl>, Hotness <chr>, end_of_fade_in <dbl>, key <dbl>,
#   keyConfidence <dbl>, Loudness <dbl>, mode <dbl>, mode_confidence <dbl>, start_of_fade_out <chr>
```

Primeros problemas observables y explicaciónes

1. Columnas y tipos de datos

Cuando cargamos un archivo CSV con **read_csv()** en R, el programa hace su mejor esfuerzo por adivinar qué contiene cada columna. Por lo general, funciona de maravilla: detecta números y les asigna **dbl**, ve texto y usa **chr**, o identifica valores lógicos como **lgl**.

Sin embargo, a veces R se confunde. Es posible que tengamos columnas que sabes que deberían ser numéricas, pero R decide tratarlas como texto (chr). Esto suele pasar cuando hay celdas vacías o caracteres extraños mezclados con los números. Por ejemplo, podríamos encontrarnos con situaciones donde Year, Energy o Hotness aparecen como texto cuando deberían ser números para poder analizarlos, como vemos que ocurre en nuestros datos Year chr.

```
# ℹ 20 more variables: Danceability <dbl>, Duration <dbl>, KeySignature <dbl>, KeySignatureConfidence <dbl>,
#   Tempo <dbl>, TimeSignature <dbl>, TimeSignatureConfidence <dbl>, Title <chr>, Year <chr>, mbID <chr>,
#   Energy <chr>, ArtistFamiliarity <dbl>, Hotness <chr>, end_of_fade_in <dbl>, key <dbl>,
#   keyConfidence <dbl>, Loudness <dbl>, mode <dbl>, mode_confidence <dbl>, start_of_fade_out <chr>
```

Si vemos esto, o si R nos lanza advertencias de "parsing" al cargar el archivo, es una señal clara de que hay que limpiar un poco los datos.

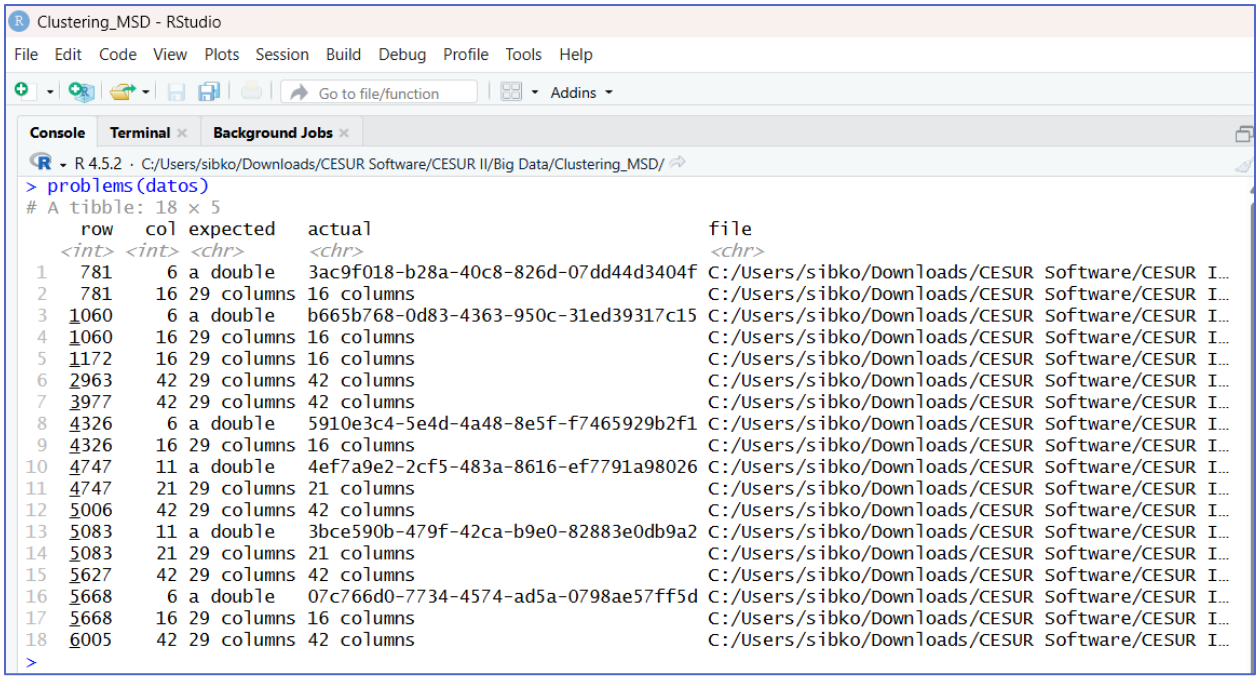
2. Por qué R lanza warnings?

La razón por la que R hace esto es por precaución. Si en una columna que debería ser numérica se encuentra con algo raro, como un texto que dice "**Unknown**", "**NA**" o simplemente una celda vacía, no sabe cómo interpretarlo matemáticamente.

En lugar de detener la importación con un error, prefiere convertir toda la columna a texto. Es su forma de decirte: "Mira, aquí hay algo que no me cuadra, así que lo dejo como texto para que no se pierda nada y tú lo arreglas luego".

3. Cómo arreglarlo paso a paso

a) Investigar
Lo primero es entender qué causó el problema. La función **problems(datos)** nos mostrará exactamente en qué fila y columna R encontró algo que no esperaba, diciéndonos cuál fue el valor problemático y qué tipo de error generó.



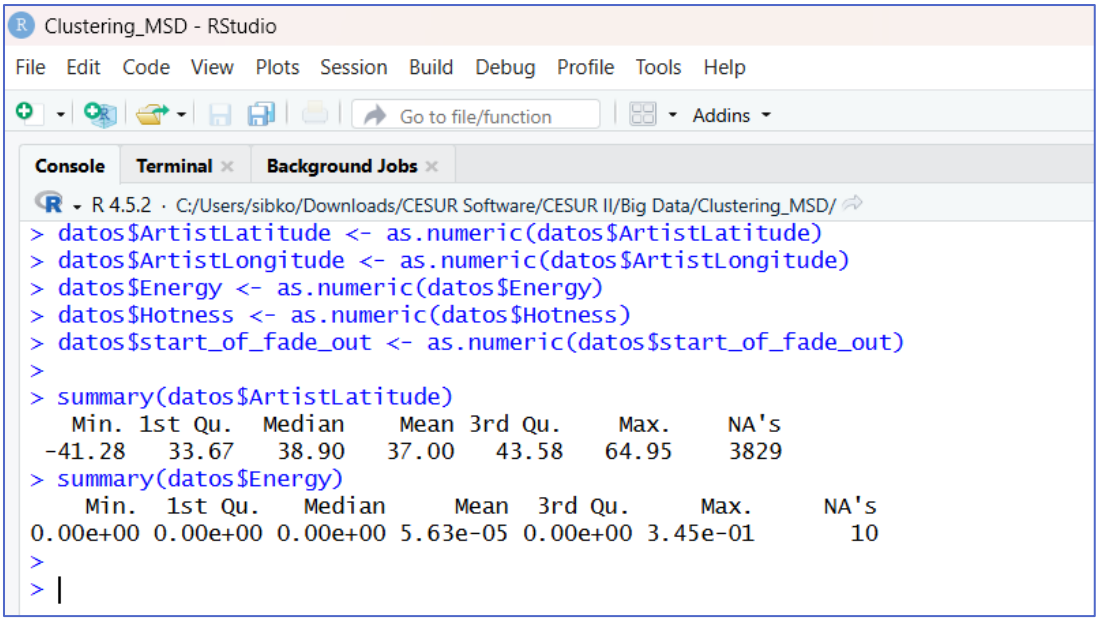
R nos tiró ese error clásico: esperaba un número, un **double**, pero se encontró con **texto**. Imagina que en la columna 6, donde debería ir la latitud del artista, de repente aparecieron unos identificadores extraños tipo UUID o cadenas de texto que nada tenían que ver. R básicamente se queda sin saber qué hacer con eso.

Y no fue solo ahí. Pasó lo mismo en otras columnas numéricas, como la 11 o la 42. Cuando intentas procesar esto, R hace lo único sensato: te pone un NA porque es imposible convertir esas letras en matemáticas.

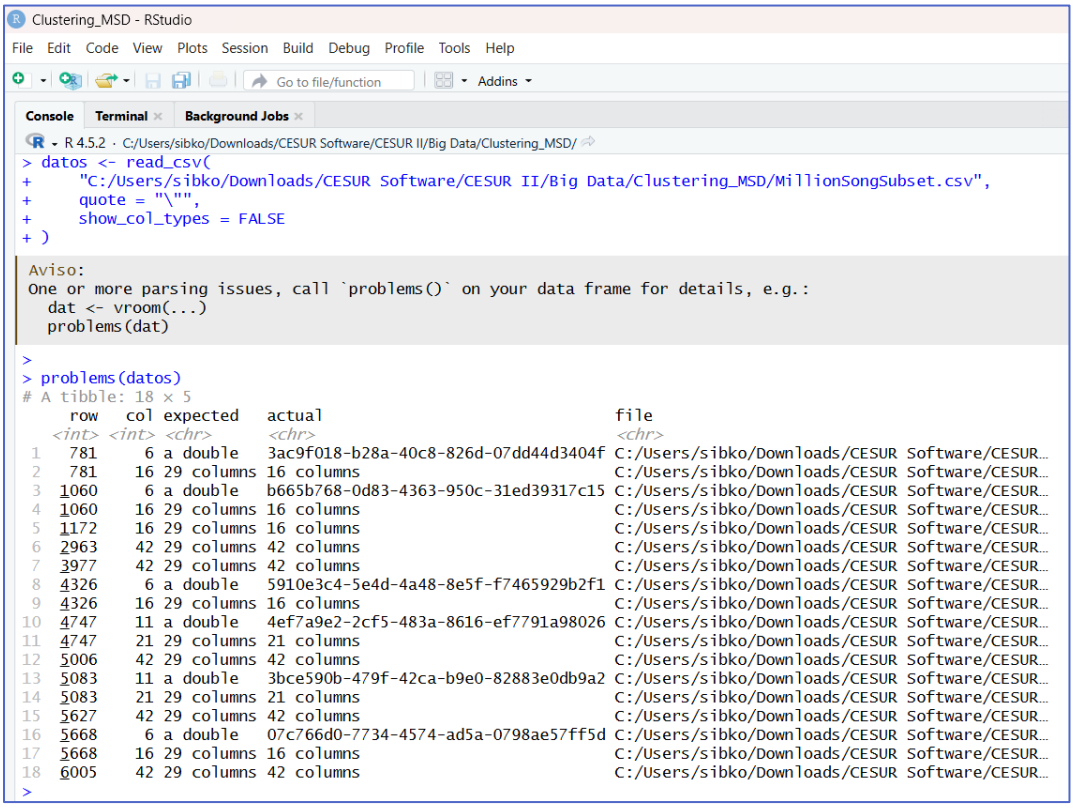
Luego vino el otro dolor de cabeza habitual con los CSV, que son las filas con el número incorrecto de columnas. R nos avisó que algunas filas tenían 16 columnas y otras 42, cuando esperábamos 29 fijas.

¿La razón? Casi siempre son comas escondidas dentro de los datos, por ejemplo en el nombre de una canción o de un artista. Si el archivo no está bien formateado, R piensa que esa coma es una separación nueva y todo se descuadra, rompiendo la estructura de la tabla.

Para arreglar el tema de los números fuimos prácticos. Usamos **as.numeric()** en todas las variables que debían ser cifras, como la latitud, la energía o el hotness. Si el valor original era basura, se convierte en **NA** y listo, dejamos de pelear con él. Un vistazo rápido con summary confirmó que ya teníamos números reales.

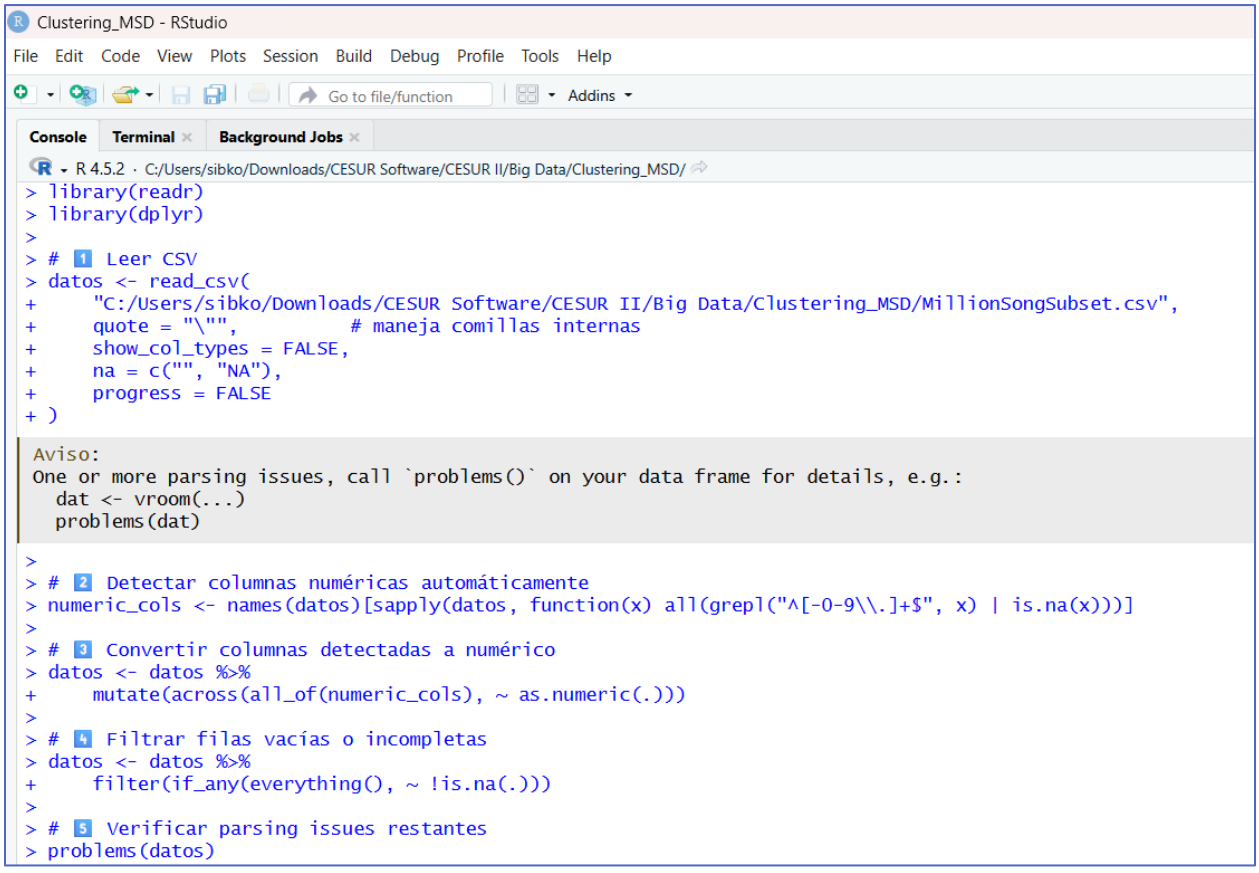


Para el desastre de las columnas desalineadas, nos tocó investigar un poco más. Leímos las líneas exactas que daban error, la 781 y la 2963, y confirmamos la sospecha: había comas metidas en el texto que R estaba malinterpretando. La solución fue volver a cargar el archivo, pero esta vez diciéndole a **read_csv** explícitamente que usara las comillas para proteger el texto. Con ese pequeño ajuste en el parámetro quote, R entendió qué era texto y qué era separador, y la tabla quedó perfecta con sus columnas en orden.



Observamos que todavía sigue existiendo filas con datos incorrectos.

b) Crear un bloque de código que arregle el problema:



No tenemos los problemas anteriores. Comprobamos como quedan los tipos de datos.

```
> sapply(datos, class)  # muestra tipo de cada columna
            SongNumber                 SongID               AlbumID             AlbumName
             "numeric"            "character"             "numeric"           "character"
              ArtistID        ArtistLatitude       ArtistLocation       ArtistLongitude
           "character"             "numeric"            "character"             "numeric"
             ArtistName          Danceability                Duration         KeySignature
           "character"             "numeric"             "numeric"             "numeric"
KeySignatureConfidence                 Tempo          TimeSignature TimeSignatureConfidence
             "numeric"             "numeric"             "numeric"             "numeric"
                Title                  Year                  mbID                Energy
           "character"            "character"          "character"           "character"
     ArtistFamiliarity                Hotness       end_of_fade_in                   key
             "numeric"           "character"             "numeric"             "numeric"
         keyConfidence                Loudness                  mode       mode_confidence
             "numeric"             "numeric"             "numeric"             "numeric"
     start_of_fade_out
           "character"
>
```

Sólo nos quedan las columnas Year, Energy, Hotness, start_of_fade_out que deberían ser tipo numérico y siguen siendo chr. Las forzamos a ser numéricas.

```
> # Convertir columnas problemáticas a numeric
> cols_problematic <- c("Year", "Energy", "Hotness", "start_of_fade_out")
>
> datos <- datos %>%
+     mutate(across(all_of(cols_problematic), ~ as.numeric(.)))
Aviso:
There were 4 warnings in `mutate()`.
The first warning was:
i In argument: `across(all_of(cols_problematic), ~as.numeric(.))`.
Caused by warning:
! NAs introducidos por coerción
i Run warnings()dplyr::last_dplyr_warnings() to see the 3 remaining warnings.
>
> # Ver resumen
> summary(datos$Year)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
 -16.96    0.00    0.00  927.82 2000.00 2010.00      10
> summary(datos$Energy)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
0.00e+00 0.00e+00 0.00e+00 5.63e-05 0.00e+00 3.45e-01       10
> summary(datos$Hotness)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
 0.0000  0.3244  0.3813  0.3858  0.4534  1.0825      12
> summary(datos$start_of_fade_out)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
   1.044  171.015  214.668  230.557  266.971 1686.752       12
>
```

Comprobamos los datos para ver que ya prácticamente no quedan valores que nos puedan interferir.

```
> # Número de NA por columna
> colSums(is.na(datos))
            SongNumber                 SongID                AlbumID              AlbumName
                     0                      0                      0                      0
              ArtistID        ArtistLatitude      ArtistLocation       ArtistLongitude
                     0                   3829                   2627                   3824
            ArtistName          Danceability               Duration           KeySignature
                     0                      0                      2                      0
 KeySignatureConfidence                  Tempo          TimeSignature TimeSignatureConfidence
                     0                      0                      0                      0
                 Title                   Year                   mbID                 Energy
                     5                     10                    460                     10
     ArtistFamiliarity                Hotness          end_of_fade_in                    key
                    10                     12                      7                      7
         keyConfidence               Loudness                   mode         mode_confidence
                     7                      7                      7                      7
     start_of_fade_out
                    12
>
> # Revisar parsing issues restantes
> problems(datos)
>
> # Primeras filas y dimensiones
> head(datos)
# A tibble: 6 × 29
  SongNumber SongID       AlbumID AlbumName ArtistID ArtistLatitude ArtistLocation ArtistLongitude ArtistName
       <dbl> <chr>          <dbl> <chr>     <chr>             <dbl> <chr>                   <dbl> <chr>
1          1 SOVLGJY12A8…  223563 Call of … ARMQHX7…             NA  Atlanta GA               NA  Mastodon
2          2 SOGXWRE12AC…  781922 Out Of T… ARTXS85…             NA  Essex England            NA  Chris Far…
3          3 SOIJPPR12A6…   85554 Wolfgang… ARYISOC…           36.0  Nashville Fra…          -86.8 Choir & G…
4          4 SOIMSZI12AB…  440144 Voices O… ARJFOC0…           33.6  Lubbock TX             -102.  Delbert M…
5          5 SOZECOE12AB…  619838 The Very… ARQ8FR8…             NA  NA                      NA  Gloria De…
6          6 SOMSKPE12A6…  279570 Now It's… ARBXIDR…             NA  NA                      NA  Avril
# i 20 more variables: Danceability <dbl>, Duration <dbl>, KeySignature <dbl>, KeySignatureConfidence <dbl>,
#   Tempo <dbl>, TimeSignature <dbl>, TimeSignatureConfidence <dbl>, Title <chr>, Year <dbl>, mbID <chr>,
#   Energy <dbl>, ArtistFamiliarity <dbl>, Hotness <dbl>, end_of_fade_in <dbl>, key <dbl>,
#   keyConfidence <dbl>, Loudness <dbl>, mode <dbl>, mode_confidence <dbl>, start_of_fade_out <dbl>
> dim(datos)
[1] 6133   29
>
```

```
> # Tipo de cada columna
> sapply(datos, class)
            SongNumber                  SongID                AlbumID              AlbumName
             "numeric"             "character"               "numeric"             "character"
              ArtistID         ArtistLatitude        ArtistLocation        ArtistLongitude
           "character"              "numeric"             "character"              "numeric"
             ArtistName          Danceability                Duration            KeySignature
           "character"              "numeric"               "numeric"              "numeric"
 KeySignatureConfidence                 Tempo         TimeSignature TimeSignatureConfidence
             "numeric"              "numeric"               "numeric"              "numeric"
                 Title                   Year                    mbID                Energy
           "character"              "numeric"             "character"              "numeric"
      ArtistFamiliarity                Hotness          end_of_fade_in                   key
             "numeric"              "numeric"               "numeric"              "numeric"
          keyConfidence               Loudness                    mode        mode_confidence
             "numeric"              "numeric"               "numeric"              "numeric"
      start_of_fade_out
             "numeric"
>
```

3.2 Eligiendo lo que importa

Nos quedamos con los **atributos** que ya hemos comprobado que son **numéricos** que realmente definen la personalidad de una canción:

Tempo (BPM) La velocidad del ritmo de la canción.
Energy intensidad de la canción de (0 – 1)
Danceability parámetro para definir cual bailable es una canción de (0 – 1)
Loudness Volumen en decibelios (db)
Duration Duración en segundos.

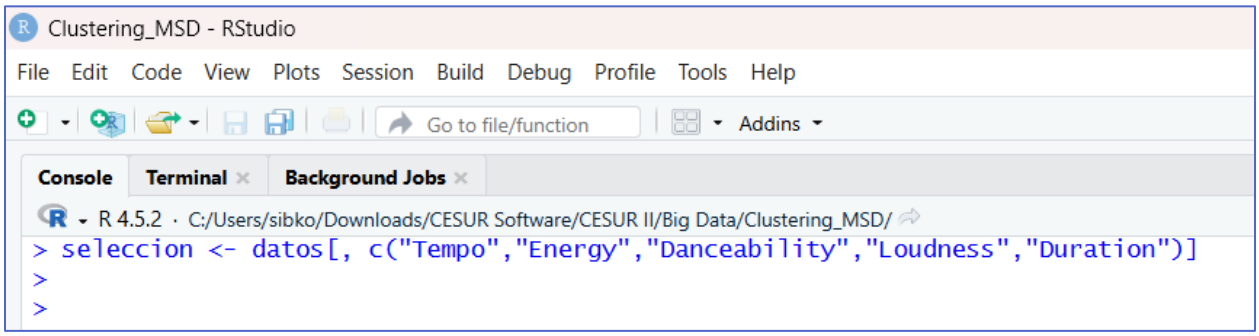
Elegimos estos porque son los que nos permiten diferenciar una balada lenta de un tema de club intenso. Son las variables clave si queremos construir un sistema de recomendación que tenga sentido.

0 4

P R E P R O C E S O Y U L T I M O S R E T O Q U E S D E L O S D A T O S

El objetivo es simple: necesitamos una **tabla numérica limpia**, sin sorpresas y donde las variables se puedan comparar entre sí. Por si acaso quedara algún dato no numérico o NA

4.1 Selección de columnas



Seleccionamos las columnas numéricas que nos puedan servir para diferenciar grupos de canciones.

4.2 Conversión segura

A veces los datos vienen disfrazados de texto, así que nos aseguramos de que R los trate como números:

```
> seleccion <- data.frame(lapply(seleccion, function(x) as.numeric(as.character(x))))
> str(seleccion)
'data.frame':   6133 obs. of  5 variables:
 $ Tempo       : num  173 147 109 222 100 ...
 $ Energy      : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Danceability: num  0 0 0 0 0 0 0 0 0 0 ...
 $ Loudness    : num  -3.31 -11.03 -15.59 -10.26 -14.09 ...
 $ Duration    : num  280 178 541 114 111 ...
>
```

as.character() previene problemas si alguna columna era factor.
as.numeric() convierte strings numéricos a números reales.
Si hay valores no convertibles aparecerán como **NA**.

4.3 Eliminando el ruido

Buscamos y eliminamos cualquier valor nulo (NA) o infinito que pueda romper el algoritmo:

```
> colSums(is.na(seleccion))
       Tempo       Energy Danceability     Loudness     Duration
           0           10            0            7            2
> sapply(seleccion, function(x) any(is.nan(x)))
       Tempo       Energy Danceability     Loudness     Duration
       FALSE        FALSE        FALSE        FALSE        FALSE
> sapply(seleccion, function(x) any(is.infinite(x)))
       Tempo       Energy Danceability     Loudness     Duration
       FALSE        FALSE        FALSE        FALSE        FALSE
>
```

Vemos que apenas hay valores NA y otro tipo de problemas. Los pocos que quedan los eliminamos.

```
>
> seleccion <- na.omit(seleccion)
>
```

4.4 Normalización

Este paso es crucial. Algoritmos como K-means son muy sensibles a las escalas. No queremos que la "Duración" domine el análisis solo porque sus números son más grandes que los de "Danceability". Por eso lo ponemos todo en la misma escala:

```
> norm <- scale(seleccion)   # devuelve matriz centrada (media 0, sd 1)
> norm_mat <- as.matrix(norm) # trabajaremos con matriz numérica
```

Con todas estas transformaciones, algo fundamental en el tratamiento de datos , estamos listos para aplicar algoritmos de agrupación o **clustering** y empezar a interpretar datos.

05

PRIMER ALGORITMO: K-MEANS

5.1 La lógica

K-means intenta dividir los datos en K grupos buscando que las canciones dentro de cada grupo se parezcan lo más posible entre sí.
Limpiamos y comprobamos una vez más que no queden datos incompatibles tipo NA , etc..

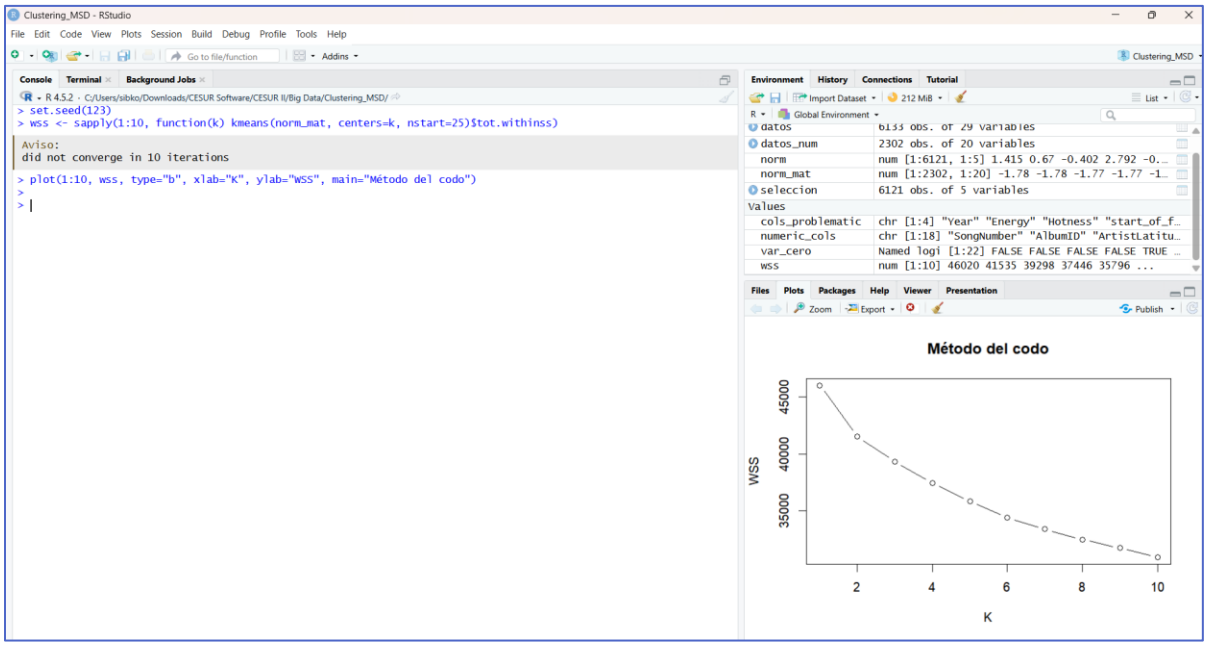
```
> library(dplyr)
> library(tidyr)   # <-- importante
>
> # 1. Seleccionar solo columnas numéricas
> datos_num <- datos %>% select(where(is.numeric))
>
> # 2. Eliminar columnas con toda la columna NA
> datos_num <- datos_num %>% select(where(~ !all(is.na(.))))
>
> # 3. Eliminar filas con NA
> datos_num <- datos_num %>% drop_na()
>
> # 4. Eliminar columnas con varianza cero (kmeans no puede con ellas)
> var_cero <- sapply(datos_num, function(x) var(x) == 0)
> datos_num <- datos_num[, !var_cero]
>
> # 5. Normalizar (es obligatorio para kmeans)
> norm_mat <- scale(datos_num)
>
```

```
> # 6. Verificar que NO hay NA/Inf
> summary(norm_mat)
   SongNumber         AlbumID        ArtistLatitude    ArtistLongitude      Duration
 Min.   :-1.7767   Min.   :-1.5862   Min.   :-5.0321   Min.   :-1.9627   Min.   :-2.0492
 1st Qu.:-0.7926   1st Qu.:-0.8413   1st Qu.:-0.2136   1st Qu.:-0.5730   1st Qu.:-0.5754
 Median : 0.1228   Median :-0.1379   Median : 0.1227   Median :-0.3108   Median :-0.1764
 Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
 3rd Qu.: 0.8116   3rd Qu.: 0.8815   3rd Qu.: 0.4233   3rd Qu.: 0.3640   3rd Qu.: 0.3425
 Max.   : 1.7266   Max.   : 1.9117   Max.   : 1.7975   Max.   : 4.8142   Max.   :12.5919
  KeySignature      KeySignatureConfidence     Tempo         TimeSignature     TimeSignatureConfidence
 Min.   :-1.45557   Min.   :-1.67318       Min.   :-3.5395   Min.   :-1.9127   Min.   :-1.30382
 1st Qu.:-0.89233   1st Qu.:-0.79525       1st Qu.:-0.7636   1st Qu.:-0.3791   1st Qu.:-1.15388
 Median :-0.04747   Median : 0.09898       Median :-0.0769   Median : 0.3877   Median : 0.07701
 Mean   : 0.00000   Mean   : 0.00000       Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000
 3rd Qu.: 0.79740   3rd Qu.: 0.75765       3rd Qu.: 0.6080   3rd Qu.: 0.3877   3rd Qu.: 0.94069
 Max.   : 1.64226   Max.   : 1.95087       Max.   : 3.4391   Max.   : 2.6881   Max.   : 1.36187
      Year         ArtistFamiliarity     Hotness          end_of_fade_in         key
 Min.   :-0.9732   Min.   :-4.20540   Min.   :-3.49781   Min.   :-0.4117   Min.   :-1.45674
 1st Qu.:-0.9732   1st Qu.:-0.61407   1st Qu.:-0.45138   1st Qu.:-0.4117   1st Qu.:-0.89356
 Median :-0.9732   Median :-0.02109   Median :-0.07196   Median :-0.2986   Median :-0.04881
 Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.00000
 3rd Qu.: 1.0303   3rd Qu.: 0.56847   3rd Qu.: 0.44427   3rd Qu.:-0.1773   3rd Qu.: 0.79595
 Max.   : 1.0413   Max.   : 2.65994   Max.   : 4.54878   Max.   :17.6854   Max.   : 1.64071
 keyConfidence        Loudness           mode         mode_confidence    start_of_fade_out
 Min.   :-1.67318   Min.   :-7.5921   Min.   :-1.5587   Min.   :-2.55335   Min.   :-2.0126
 1st Qu.:-0.79536   1st Qu.:-0.5312   1st Qu.:-1.5587   1st Qu.:-0.61866   1st Qu.:-0.5760
 Median : 0.09877   Median : 0.1610   Median : 0.6413   Median : 0.04721   Median :-0.1801
 Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.0000
 3rd Qu.: 0.75827   3rd Qu.: 0.7528   3rd Qu.: 0.6413   3rd Qu.: 0.66590   3rd Qu.: 0.3357
 Max.   : 1.95045   Max.   : 1.8383   Max.   : 0.6413   Max.   : 2.68972   Max.   :12.9055
> any(is.na(norm_mat))
[1] FALSE
> any(is.infinite(norm_mat))
[1] FALSE
```

NA / INF false así que estamos listo.

5.2 ¿Cuántos grupos usamos? (El método del codo)

No queremos adivinar, así que calculamos la variación para diferentes números de grupos y buscamos el "codo" en la gráfica:



Buscamos ese punto donde añadir más grupos o aumentar K ya no mejora significativamente el resultado.

5.3 Ejecución

Probamos con 3 grupos como ejemplo: K = 3

```
> set.seed(123)
> k3 <- kmeans(norm_mat, centers=3, nstart=25)
> k3$size       # tamaño de cada cluster
[1] 786 769 747
> k3$centers    # centroides (en escala estandarizada)
   SongNumber       AlbumID ArtistLatitude ArtistLongitude    Duration KeySignature KeySignatureConfidence
1 -0.04569494 -0.0003489976    -0.04905357     -0.08671506 -0.02069643    0.1775432             -1.1043389
2  0.07255959  0.0030634488     0.02344588      0.04238355 -0.05170841    0.8010581              0.6474179
3 -0.02661594 -0.0027864524     0.02747821      0.04761055  0.07500825   -1.0114626              0.4955101
       Tempo TimeSignature TimeSignatureConfidence        Year ArtistFamiliarity      Hotness
1 -0.01115370    0.18188352              0.05592386  0.09142015        0.15544800  0.125461661
2  0.06576707   -0.05599629             -0.00846686 -0.05278820       -0.12132864 -0.118555673
3 -0.05596796   -0.13373400             -0.05012736 -0.04185021       -0.03866186 -0.009964595
  end_of_fade_in        key keyConfidence     Loudness         mode mode_confidence start_of_fade_out
1    0.005367367  0.1761755    -1.1044049  0.080548462 -0.212421128      -0.9041458       -0.01580100
2   -0.096296562  0.8036405     0.6477313 -0.007144735 -0.008146363       0.4998493       -0.05441085
3    0.093485015 -1.0126820     0.4952569 -0.077398648  0.231897671       0.4367798        0.07263926
>
```

5.4 Interpretación

Ahora traducimos las matemáticas a música. Miramos los promedios de cada grupo en sus valores originales para entender qué tipo de canciones quedaron juntas:

```
> set.seed(123)
> k3 <- kmeans(norm_mat, centers = 3, nstart = 25)
>
> k3$size
[1] 786 769 747
```

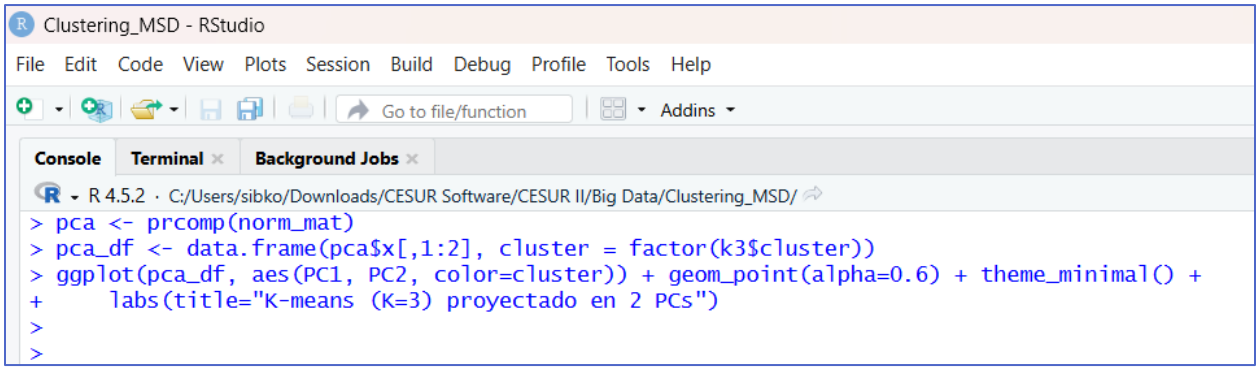
k3$**size** tamaño de cada cluster
k3$**centers** centroides (en escala estandarizada)

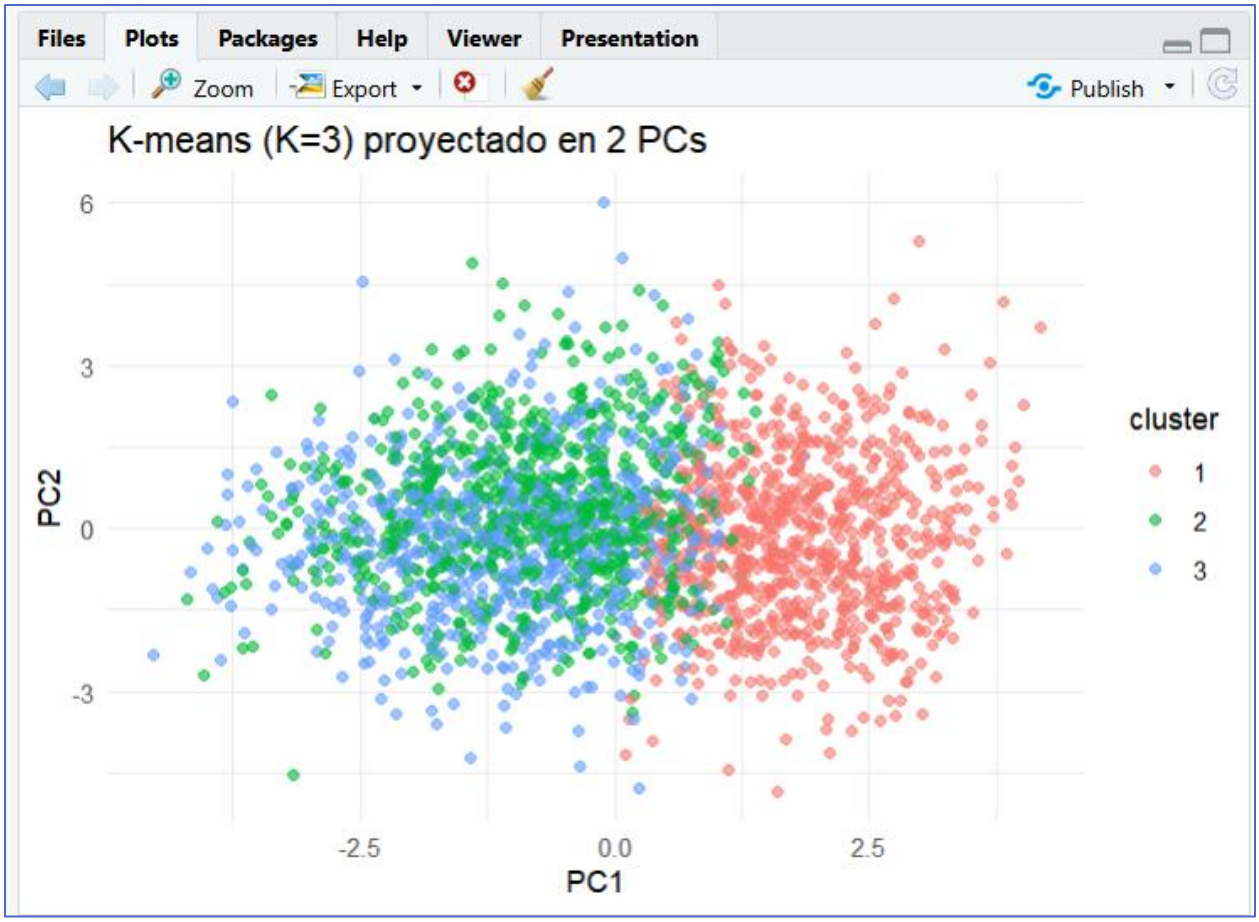
```
> seleccion$cluster <- k3$cluster
>
> aggregate(seleccion, by = list(cluster = seleccion$cluster), mean)
  cluster SongNumber  AlbumID ArtistLatitude ArtistLongitude Duration KeySignature KeySignatureConfidence
1       1   4942.588 373199.5       36.22826       -69.09349 234.5917     5.798982              0.1569631
2       2   5280.034 374002.3       37.35595       -62.66980 231.0212     8.013004              0.6403329
3       3   4997.031 372626.0       37.41867       -62.40972 245.6107     1.576975              0.5984163
     Tempo TimeSignature TimeSignatureConfidence      Year ArtistFamiliarity   Hotness end_of_fade_in
1 122.2997      3.731552               0.5100916 1062.2354         0.6017400 0.4089514      0.7598486
2 124.9659      3.421326               0.4859363  918.3459         0.5635484 0.3814097      0.5746216
3 120.7464      3.319946               0.4703079  929.2597         0.5749554 0.3936661      0.9203949
       key keyConfidence  Loudness      mode mode_confidence start_of_fade_out cluster
1 5.798982     0.1569631 -10.46490 0.6119593       0.3145496          226.6748       1
2 8.027308     0.6404941 -10.93554 0.7048114       0.5823303          222.3120       2
3 1.576975     0.5984163 -11.31259 0.8139224       0.5703012          236.6683       3
>
```

aggregate() devuelve las medias por cluster en la escala original de las variables más fácil de interpretar. Así vemos que cluster tiene mayor o menor, según la columna o variable elegida.

5.5 Visualización

Finalmente, usamos PCA para proyectar estos datos complejos en un gráfico que podamos entender visualmente:





Interpretación el mapa (PCA)

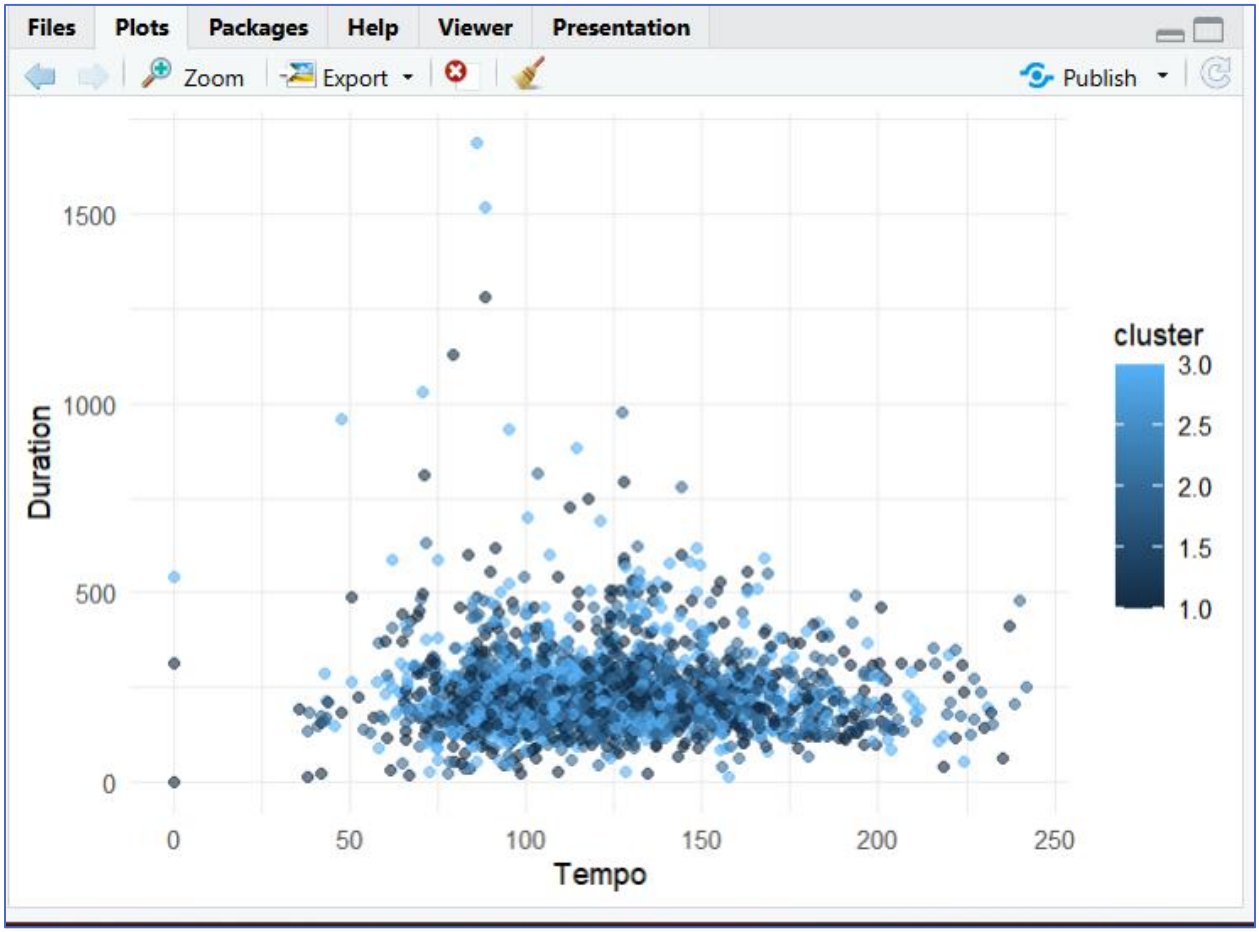
Si miras de izquierda a derecha (el eje PC1), estás viendo básicamente una línea de tiempo y energía. A la derecha tienes lo moderno y ruidoso; a la izquierda, lo clásico y tranquilo.

Si miras de arriba a abajo (el eje PC2), estás viendo complejidad. Arriba hay estructuras rítmicas complejas y abajo cosas más simples o repetitivas.

¿Dónde quedaron nuestros grupos?
El Cluster 1 (los energéticos) se fue hacia la derecha y un poco arriba: modernos y complejos.
El Cluster 3 (los suaves) se quedó a la izquierda y abajo: clásicos y sencillos.
El Cluster 2 (los neutros) se mantuvo flotando por el centro, sirviendo de puente entre los dos mundos.

Y ahora con scatter usando dos variables originales (Duración y Tempo(BPM)):

```
> df_plot <- data.frame(seleccion, cluster=factor(k3$cluster))
> ggplot(df_plot, aes(x=Tempo, y=Duration, color=cluster)) + geom_point(alpha=0.6) + theme_minimal()
>
```



Vemos que podemos hacer una separación de canciones también definidas por estas dos variables, y que corresponden a cada grupo.

0 6
I N T E R P R E T A C I O N D E R E S U L T A D O S

Después de hacer correr el algoritmo de clustering, los datos se organizaron en tres familias bastante claras. No son grupos idénticos, pero están muy parejos: tenemos el Cluster 1 con 786 canciones, el Cluster 2 con 769 y el Cluster 3 con 747.

Cuando miramos qué define a cada grupo, la historia que nos cuentan es interesante:

Cluster 1: El punto medio

Aquí es donde caen las canciones que podríamos llamar "equilibradas". Si revisas su ritmo, energía o qué tan populares son, todo está muy cerca del promedio. No hay extremos. Geográficamente tampoco vemos nada raro en la procedencia de los artistas. Es, esencialmente, el núcleo neutro de nuestra colección; música que no es ni muy intensa ni muy apagada.

Cluster 2: Energía y actualidad

Este grupo es el que trae la intensidad. Vemos tiempos rápidos, mucha energía y una sonoridad fuerte, lo que suele indicar producciones modernas. Son canciones recientes y populares, con esa confianza tonal alta que te hace pensar en éxitos de radio. Es el grupo ideal si estás buscando música para bailar, entrenar o mantener el ánimo arriba.

Cluster 3: Nostalgia y calma

Aquí bajamos las revoluciones. Nos encontramos con tempos lentos y niveles de energía bajos. Un detalle clave es el volumen (loudness); al ser mucho más bajo, nos da una pista clara de que son grabaciones más antiguas o acústicas, donde no se comprimía tanto el sonido como hoy. Son temas suaves, quizás en modos menores, perfectos para momentos de relajación.

2. Aterrizando los datos (lo que significan las medias reales)

Para entender esto sin escalas abstractas, miremos los números reales de cada grupo:

Cluster 1 (Neutro): Nos movemos mayormente entre 1990 y el 2000. El ritmo es cómodo, entre 100 y 120 **BPM**, y el volumen es moderado. Es el estándar de la música pop/rock de esas décadas.

Cluster 2 (Energético): Aquí saltamos del año 2000 en adelante. El ritmo se acelera (120 a 150 **BPM**) y el volumen es alto, típico de la "guerra del volumen" moderna. Es música potente.

Cluster 3 (Suave): Viajamos atrás, entre 1970 y 1990. El ritmo es pausado (80 a 100 **BPM**) y el sonido es mucho más dinámico y suave. Es el terreno de lo clásico.

Resumen final

Lo que hemos logrado es una segmentación musical que tiene todo el sentido del mundo para un oyente real:

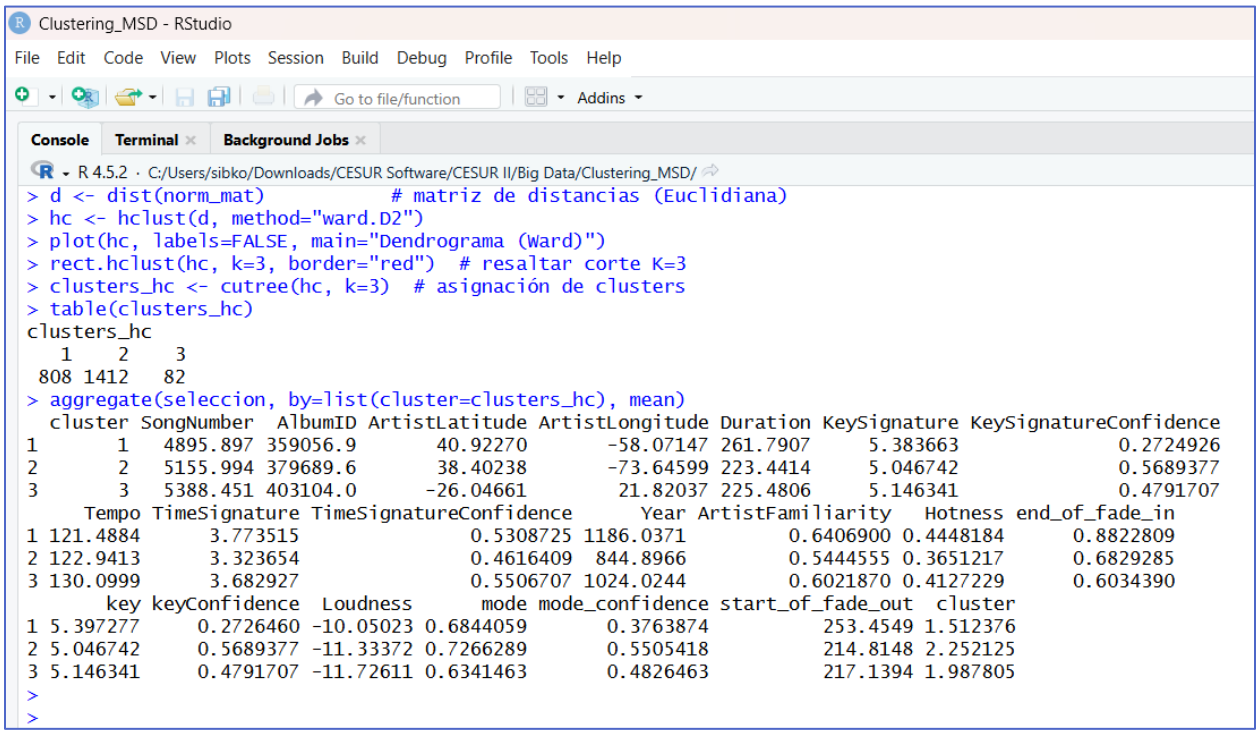
El **Cluster 2** es para cuando quieres acción: rap, electrónica, pop actual.
El **Cluster 1** es tu comodín: música estándar que no cansa.
El **Cluster 3** es para bajar el ritmo: jazz, baladas, clásicos de ayer.

Este análisis deja de ser solo estadística y se convierte en la base perfecta para crear playlists automáticas que tengan coherencia emocional o para recomendarle al usuario exactamente lo que necesita según su estado de ánimo.

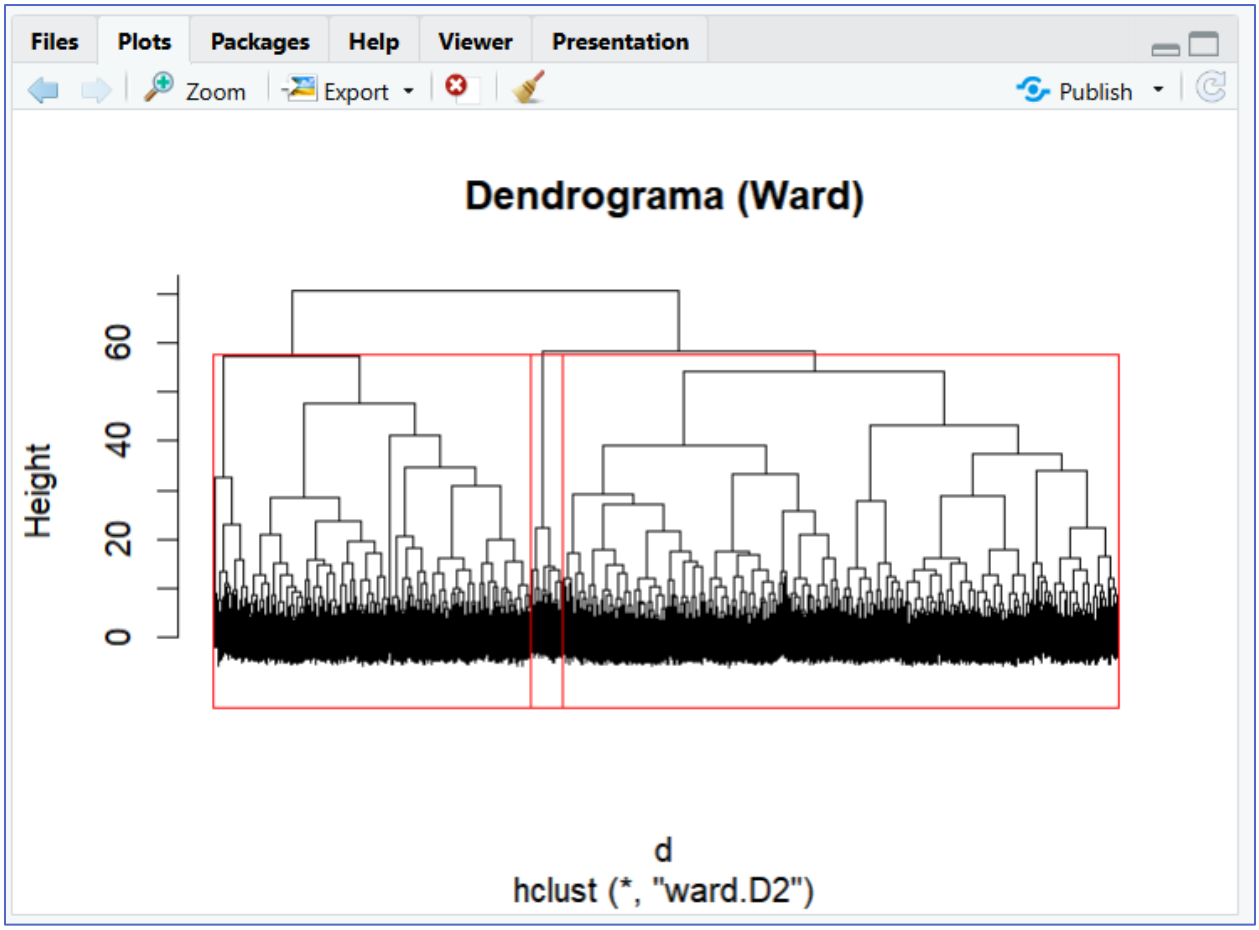
0 7

S E G U N D O A L G O R I T M O

Vamos a mirar esto desde otra perspectiva con el clustering jerárquico usando el **método de Ward.** La idea aquí es visualizar las relaciones entre tus datos, casi como si estuviéramos dibujando el árbol genealógico de las canciones. Así es como generamos esa matriz de distancias y el famoso dendrograma para ver qué está pasando realmente:



Lo valioso de este método es que te permite ver las conexiones de una forma mucho más gráfica que los simples números.



Observamos cómo se agrupan las canciones paso a paso.
Comparamos las medias por cluster con las de K-means para ver coincidencias/diferencias.

El dendrograma nos muestra relaciones jerárquicas: qué canciones están más cercanas entre sí.

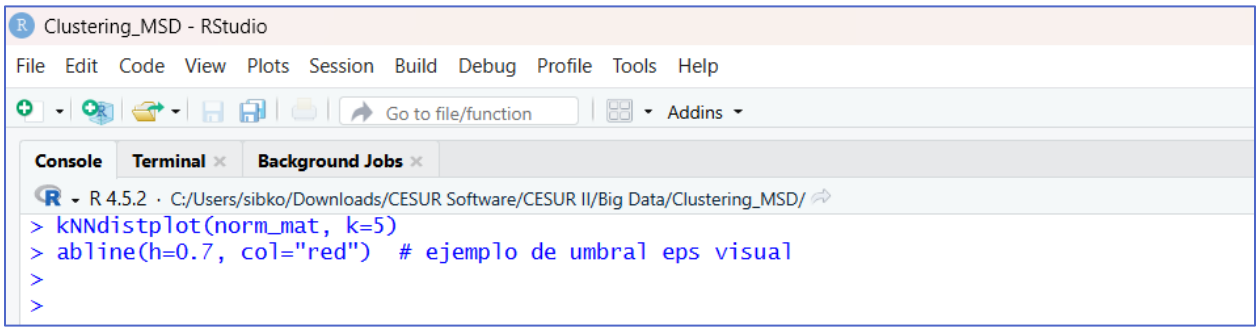
0 8
T E R C E R A L G O R I T M O :
D B S C A N

8.1 El concepto

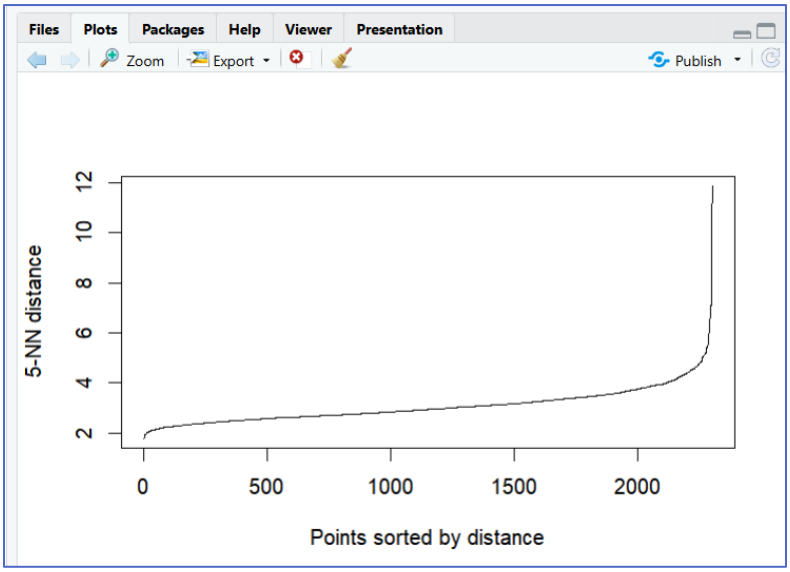
Este algoritmo funciona de manera distinta. En lugar de forzar a los datos a entrar en grupos predefinidos, busca zonas donde los puntos están muy apretados (densidad) y todo lo que queda lejos o aislado lo marca como ruido. Lo mejor es que no tienes que adivinar el valor de K desde el principio.

8.2 Eligiendo el épsilon

Para que funcione bien, necesitamos calibrarlo un poco mirando la curva de distancias:



Interpretación: buscamos “codo” en kNNdistplot para elegir eps.

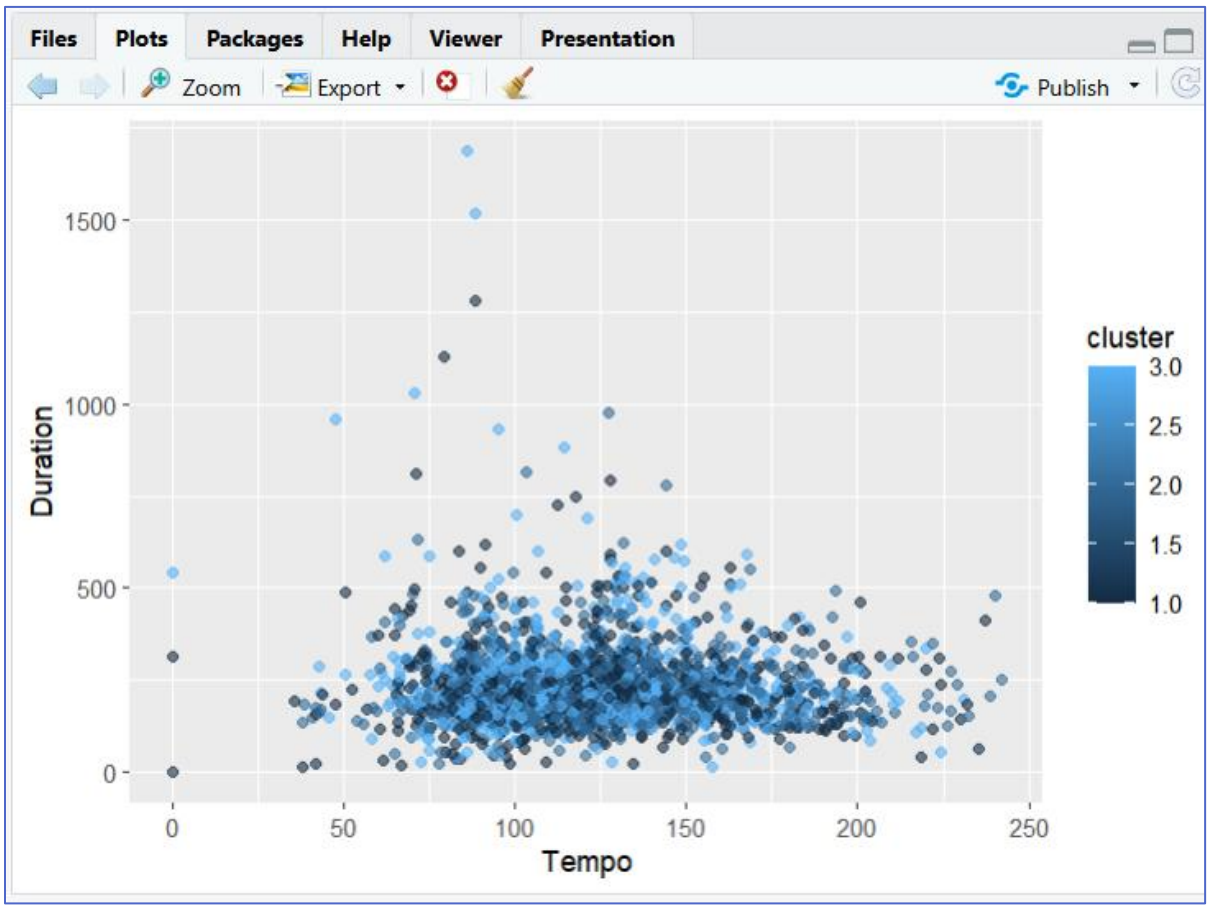


8.3 Poniéndolo en marcha

Una vez que tenemos claro el parámetro, lanzamos el modelo:

```
> db <- dbscan(norm_mat, eps=0.7, minPts=10)
> table(db$cluster)   # 0 = ruido

   0
2302
> df_db <- data.frame(seleccion, cluster=factor(db$cluster))
> ggplot(df_db, aes(x=Tempo, y=Duration, color=cluster)) + geom_point(alpha=0.6)
>
>
```



8.4 Interpretación

Vemos que el gráfico coincide con el que hicimos con scatter en el primer algoritmo.

DBSCAN puede identificar un conjunto central denso y otros clusters dispersos o un gran porcentaje de ruido si eps es pequeño.

Comparar tamaños y medias para entender qué tipo de canciones quedan en cada grupo o son consideradas ruido.

0 9
I N T E R P R E T A C I Ó N F I N A L

Si algo nos quedó claro en este viaje es que preparar los datos es donde realmente se juega la partida. Tener que ajustar tipos, cazar esos valores que no tenían sentido y limpiar los **NA** no fue un simple trámite administrativo, fue lo que nos permitió confiar en los resultados posteriores. Una vez superado ese obstáculo, fue gratificante ver cómo los algoritmos empezaban a revelar patrones que tenían sentido.

En nuestro caso, fue el que nos dio la foto más nítida. Elegir K = 3 no fue lanzar una moneda al aire **k-means**; los datos nos empujaron hacia esa decisión mediante el método del codo y el silhouette. Pero lo más importante fue que, al mirar las medias de cada grupo, la división tenía una lógica musical evidente.

El **clustering** jerárquico con Ward funcionó como esa segunda opinión necesaria. No solo validó lo que k-means ya nos decía, sino que añadió una capa visual muy útil para entender las relaciones de parentesco entre los distintos clusters.

Por otro lado, **DBSCAN** demostró su valor en un terreno donde los otros cojean: encontrar zonas de alta densidad y señalar a los outliers. Este algoritmo es fantástico cuando necesitas detectar canciones que se salen de la norma o agrupaciones con formas extrañas que k-means simplemente ignora. Al final, usar los tres enfoques no fue redundante, sino que nos dio una visión de 360 grados del comportamiento de nuestros datos.

9.2 Aplicaciones prácticas

Todo esto va mucho más allá de un ejercicio académico; tiene una utilidad directa en cómo consumimos música hoy en día. Piensa en las plataformas de streaming. Este tipo de análisis es lo que permite generar playlists automáticas que realmente encajan con tu estado de ánimo, ya sea para entrenar o para relajarte, sin que tengas que elegir canción por canción.

También es la base para afinar los motores de recomendación. Si el sistema sabe en qué cluster te mueves habitualmente, puede sugerirte temas vecinos que probablemente te gusten. A nivel de industria, sirve para segmentar catálogos para campañas de marketing o incluso para detectar anomalías en la base de datos, ya sean joyas experimentales ocultas o simples errores de etiquetado.

9.3 Limitaciones y mejoras futuras

Hemos trabajado con una muestra de 10.000 canciones. Es una cantidad respetable para entender la mecánica y sacar conclusiones válidas, pero si nos enfrentáramos al Million Song Dataset completo, seguramente aparecerían matices y subgrupos que aquí se nos quedan fuera de radar.

El siguiente paso lógico sería meterle más complejidad a los datos. Nos hemos movido con variables accesibles, pero el dataset original esconde tesoros como el timbre o análisis de segmentos de audio. Incluir eso enriquecería muchísimo el clustering, aunque nos obligaría a vigilar de cerca la dimensionalidad y la correlación para no complicarnos la vida sin necesidad.

También sería interesante probar modelos más flexibles, como los gaussianos (GMM), o pipelines de validación cruzada. Eso nos ayudaría a ajustarnos mejor a la forma real de los datos y a depender menos de nuestra propia intuición al elegir los parámetros.

10
RESUMEN DE LOS CONCEPTOS Y PROCEDIMIENTOS

1. Cluster

Piénsalo como una tribu o una familia. Un cluster no es más que un grupo de elementos que deciden juntarse porque comparten rasgos en común, diferenciándose claramente de los demás. En nuestro proyecto, cada cluster funciona básicamente como una playlist automática: un conjunto de canciones que vibran en la misma frecuencia, ya sea por su tempo, su energía o su duración. Lo interesante aquí fue encontrar esos patrones invisibles, esas conexiones ocultas en los datos, sin que nadie nos dijera de antemano qué buscar.

2. Clustering

Esta es la acción detrás de la magia. El clustering es una técnica de análisis no supervisado, lo que suena técnico, pero en realidad es muy intuitivo: significa que no le damos al ordenador las respuestas ni las categorías. Dejamos que el algoritmo explore y encuentre la estructura por sí mismo. Es como darle una caja de piezas mezcladas y dejar que descubra cuáles encajan mejor. Es la herramienta perfecta para revelar relaciones que a simple vista se nos escapan, agrupando la música por estilo o intensidad de forma orgánica.

3. Algoritmos aplicados

Durante el proyecto no nos casamos con una sola idea; probamos varias estrategias porque cada algoritmo tiene su propia personalidad:

K-means funciona como el organizador pragmático. Le pides un número exacto de grupos y él se encarga de calcular los centros y asignar cada canción a su vecino más cercano. Es rápido y directo, aunque a veces es un poco sensible si los datos tienen escalas muy distintas o valores extremos.

El Clustering jerárquico es más visual, como un árbol genealógico. Construye un dendrograma que nos permite ver cómo se van fusionando los datos paso a paso, lo cual es genial para entender las relaciones desde un nivel general hasta el detalle más fino.

DBSCAN es el experto en densidades. Busca zonas donde se acumulan muchos datos y, lo más interesante, es capaz de detectar a los inadaptados, esos puntos aislados o outliers que no encajan en ningún grupo. Es ideal cuando los grupos tienen formas extrañas y no son círculos perfectos.

4. Procedimientos en RStudio

Para que todo esto funcionara, seguimos un flujo de trabajo que es casi como una receta de cocina bien ejecutada:

Empezamos trayendo los ingredientes al cargar el dataset del millón de canciones. Pero no nos lanzamos a ciegas; primero hicimos una exploración inicial para ver qué teníamos entre manos, revisando tipos de datos y buscando posibles errores o valores extraños.

Luego nos pusimos selectivos. Nos quedamos solo con las variables que realmente contaban una historia: Tempo, Energía, Bailabilidad, Loudness y Duración. Limpiamos la casa convirtiendo datos y eliminando filas problemáticas para asegurar que nada hiciera tropezar a los algoritmos.

Un paso crucial fue la normalización. Escalamos todo para poner las variables en igualdad de condiciones, evitando que una domine al resto solo porque sus números son más grandes. Con el terreno listo, ejecutamos los algoritmos y pasamos a la parte visual, usando gráficos para ver literalmente cómo se agrupaban las canciones. Finalmente, calculamos los promedios para entender la identidad de cada grupo.

5. Interpretación

Al final del día, los gráficos y los números tienen que contarnos algo útil. Los clusters nos muestran qué canciones son parientes cercanos según los criterios que elegimos. Al comparar los distintos algoritmos, podemos ver cuál de ellos captó mejor la esencia de nuestra música.

No se trata solo de agrupar por agrupar. Al observar los patrones y los valores medios, empezamos a traducir los datos en conocimiento real: identificamos las canciones largas y tranquilas frente a las rápidas y energéticas, o diferenciamos lo moderno de lo clásico. Esa visión clara es lo que nos permite pasar de una hoja de cálculo a crear recomendaciones inteligentes o curar listas de reproducción que realmente tengan sentido.

11
CONCLUSIONES

Este proyecto ha sido mucho más que un ejercicio técnico; ha sido un recorrido completo por cómo se analiza la música en el mundo real, trabajando con datos del Million Song Dataset. Te das cuenta rápido de que cada pequeña decisión cuenta. Qué atributos eliges o cómo escalas los datos no son detalles menores, sino factores que cambian totalmente los grupos que aparecen y la historia que te cuentan al final.

En la práctica, vimos que k-means es una herramienta muy eficiente, siempre y cuando tengas los datos bien ordenados y la estructura no sea una locura. El clustering jerárquico, en cambio, nos dio una visión más conceptual, genial para entender cómo se relacionan los grupos entre sí. Y luego está DBSCAN, que aportó una capa extra de profundidad, sobre todo para identificar esas canciones raras o atípicas que no encajan en los moldes estándar.

Lo más interesante de comparar estos métodos es ver que no hay un ganador absoluto. Cada algoritmo ilumina el dataset desde una perspectiva distinta. Al final, cuando juntas todas esas visiones, obtienes una imagen mucho más rica y matizada del catálogo musical.

Los clusters que encontramos mostraron diferencias reales en energía o volumen, lo que demuestra que estas técnicas sirven de verdad para clasificar música o generar recomendaciones automáticas. Es una base sólida para dar el siguiente paso hacia modelos más complejos, pero sobre todo, ha sido la mejor manera de entender cómo funciona el aprendizaje no supervisado cuando te manchas las manos con datos reales.

11
REFERENCIAS

https://www.r-project.org/

https://posit.co/downloads/

http://millionsongdataset.com/

https://www.kaggle.com/datasets/sansastark/subset-of-the-million-song-dataset

https://posit.co/blog/learn-python-for-data-science-with-posit-academy/

https://www.datacamp.com/tutorial/k-means-clustering-r

https://ggplot2.tidyverse.org/

https://cran.r-project.org/web/views/Cluster.html

https://www.rdocumentation.org/packages/dbscan/versions/1.1-10

https://www.youtube.com/watch?v=4b5d3muPQmA

https://www.youtube.com/watch?v=RDZUdRSDOok

https://www.statology.org/k-means-clustering-in-r/

https://www.statology.org/elbow-method-in-r/