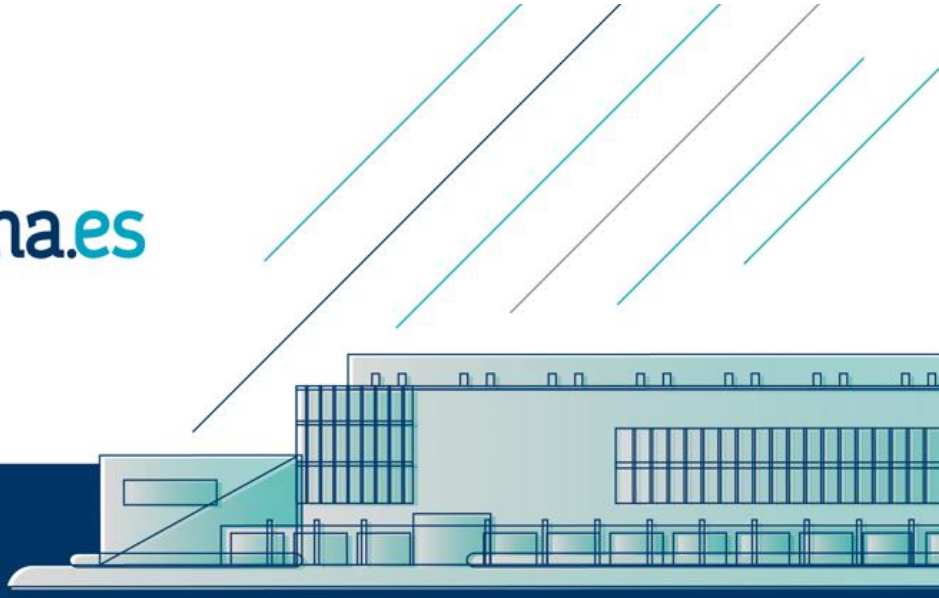
UNIVERSIDAD DE MÁLAGA | uma.es



Curso de iniciación a R y RStudio: Manual básico

Departamento de Economía Aplicada (Estadística y Econometría)

Tel.: 952 13 28 37 | ipsoler@uma.es

Facultad de Ciencias Económicas y Empresariales

Campus de El Ejido

Málaga.

UMA | Google Scholar | Orcid | ResearchGate | Linkedin

CURSO **2025/26**

Tabla de contenido

Presentación

Este curso está dirigido principalmente a estudiantes sin experiencia previa en programación que deseen iniciarse en R de manera clara y práctica. No se requiere conocimiento técnico previo. A través de ejemplos graduales, partiremos desde conceptos fundamentales como variables y operadores, avanzaremos por estructuras de datos básicas, y culminaremos con análisis estadístico real aplicado a datos concretos. El objetivo es que, al finalizar, se disponga de la confianza y las herramientas necesarias para importar y manipular datos externos, así como para realizar análisis estadísticos coherentes, sentando las bases para explorar posteriormente técnicas más avanzadas y paquetes especializados que expanden las posibilidades infinitas de R.

Introducción a R

R es un entorno y a la vez un lenguaje de programación especialmente diseñado para el análisis estadístico y gráfico. Se trata de un proyecto GNU que tiene su origen en el lenguaje S y el entorno que se desarrolló en *Bell Laboratories* (anteriormente *AT&T, ahora Lucent Technologies*) por John Chambers y sus colegas. Sin embargo, R puede considerarse como una implementación diferente de S existiendo algunas diferencias notables entre ellos.

Entre sus fortalezas, destaca su capacidad para trabajar con una amplia variedad de técnicas estadísticas, que incluyen modelado lineal y no lineal, pruebas estadísticas clásicas, análisis de series temporales, clasificación, y agrupación, entre otras, además de sus avanzadas herramientas gráficas. R sobresale por ofrecer un entorno planificado y coherente, lo que brinda una gran flexibilidad en su uso. Simultáneamente, al ser un lenguaje de programación completo y bien desarrollado en el que incluyen características avanzadas como condicionales, bucles, funciones recursivas definidas por el usuario y capacidades robustas de entrada y salida. Estas cualidades permiten que, además de las herramientas integradas, R sea un entorno dinámico donde cualquier usuario programar nuevas aplicaciones (paquetes) y ponerlas a disposición de la comunidad.

Por otro lado, uno de sus desafíos iniciales es su aparente dificultad de manejo y su interfaz poco amigable para el usuario, especialmente en comparación con otros programas. Sin embargo, esta percepción suele disiparse tras superar la curva de aprendizaje inicial, momento en el que se aprecia su enorme potencia estadística, estabilidad, bajo consumo de recursos, flexibilidad y simplicidad en el uso. Además, al ser Software Libre, R está disponible de manera gratuita y funciona en una amplia gama de plataformas, incluidas UNIX y sistemas similares (como FreeBSD y Linux), Windows y MacOS. Estas características explican su creciente popularidad dentro de la comunidad científica. Su evolución constante está respaldada por una activa comunidad de usuarios que contribuyen al desarrollo y análisis de sus aplicaciones. Actualmente, su mantenimiento y avance están a cargo del *R Development Core Team*.

Podemos encontrar la página web de R en la dirección: http://www.r-project.org/. Allí veremos un enlace (download R) para la instalación de R.

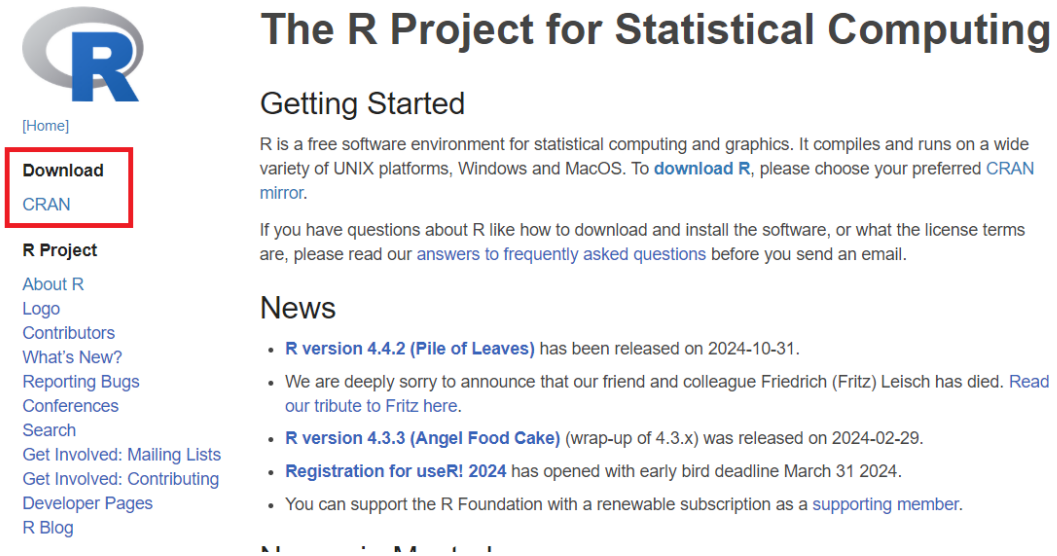


Figura 1: Web del Proyecto R

El enlace pide una localización (CRAN Mirror): elegimos generalmente la más próxima, por ejemplo, Spain https://cran.rediris.es/

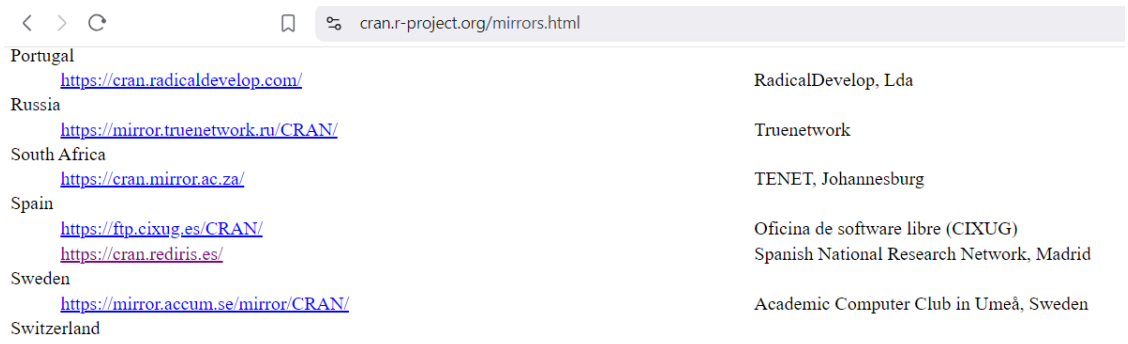


Figura 2: Descargar a través del Mirror

Seleccionamos el sistema operativo:

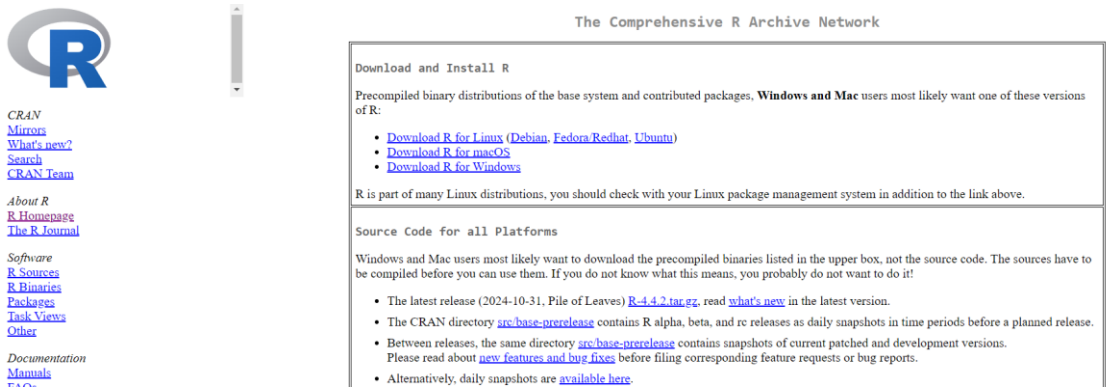


Figura 3: Elegimos el sistema operativo

Una vez descargado el archivo, basta con ejecutarlo y seguir las instrucciones del instalador. Es recomendable aceptar las opciones predeterminadas, asegurándose de que la opción "crear un acceso directo en el escritorio" esté seleccionada, y evitando modificar otras configuraciones. Por lo general, el proceso de instalación es relativamente rápido y no presenta complicaciones. No obstante, una instalación completa de R requiere la descarga de múltiples archivos, lo que podría activar las medidas de seguridad de nuestro sistema, como antivirus o cortafuegos. Estos podrían interpretar el acceso continuo y la descarga de archivos como una posible amenaza, limitando o bloqueando el proceso. En ese caso —identificable por la aparición de mensajes de error durante la instalación—, puede ser necesario reducir temporalmente el nivel de protección del sistema. Cabe destacar que todos los archivos de R son completamente seguros y están libres de virus. Finalmente, es importante esperar con paciencia hasta que la instalación se complete por completo.

R y RStudio

Una vez R se encuentre correctamente instalada en nuestro equipo, podemos usar la consola de R e introducir los comandos[1].

[1] R se ejecuta en un entorno gráfico integrado automáticamente en Windows denominado RGui, el cual dispone de un pequeño panel de herramientas.

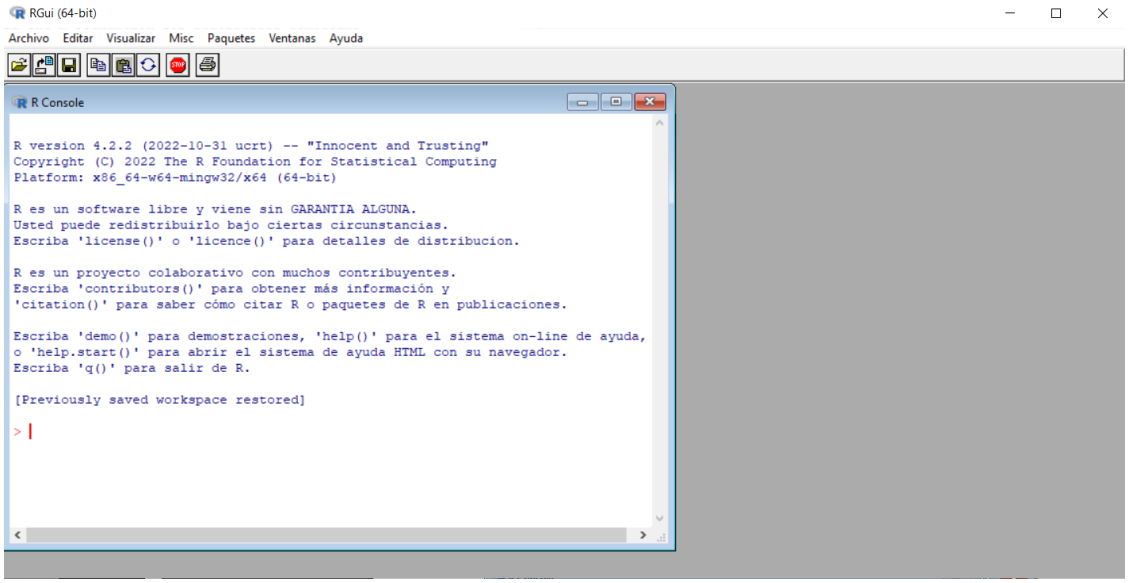


Figura 4: Consola de R con RGui

Este método suele ser incómodo y poco atractivo para muchos usuarios. El símbolo > de la consola es el punto a partir el cual deben escribirse las instrucciones de R con las que realizar cualquier operación. Una vez se introduce la instrucción se ejecuta al pulsar `<INTRO>`. A modo de ejemplo, podrías introducir la siguiente instrucción usando R como una calculadora.

```
2+2
```

```
## [1] 4
```

O emplear cualquiera de sus funciones.

```
print("Hola Mundo")
```

```
## [1] "Hola Mundo"
```

Los nostálgicos de MS-DOS posiblemente no necesiten nada más.

Es común usar una analogía con lo que sería un coche donde R sería el motor y todo el complejo sistema interno que lo hace funcionar y el entorno gráfico integrado sería el cuadro de mando con el que manipulamos el coche. Con los conocimientos necesarios, uno podría conducir el coche interactuando directamente con las entrañas del coche, pero la mayoría de los usuarios prefieren un salpicadero moderno, asientos de cuero climatizados y, por supuesto, un buen portavasos para el café. Por este motivo, existen alternativas que integran diferentes entornos gráficos y que facilitan la interacción con el usuario. Entre los entornos de desarrollo integrados más populares se encuentra

RStudio, el cual incluye una consola que a su vez incorpora con un editor resaltado de síntesis y facilita la gestión del espacio de trabajo.

En esta introducción usaremos únicamente RStudio. Para instalar RStudio acudimos a: https://posit.co/download/rstudio-desktop/

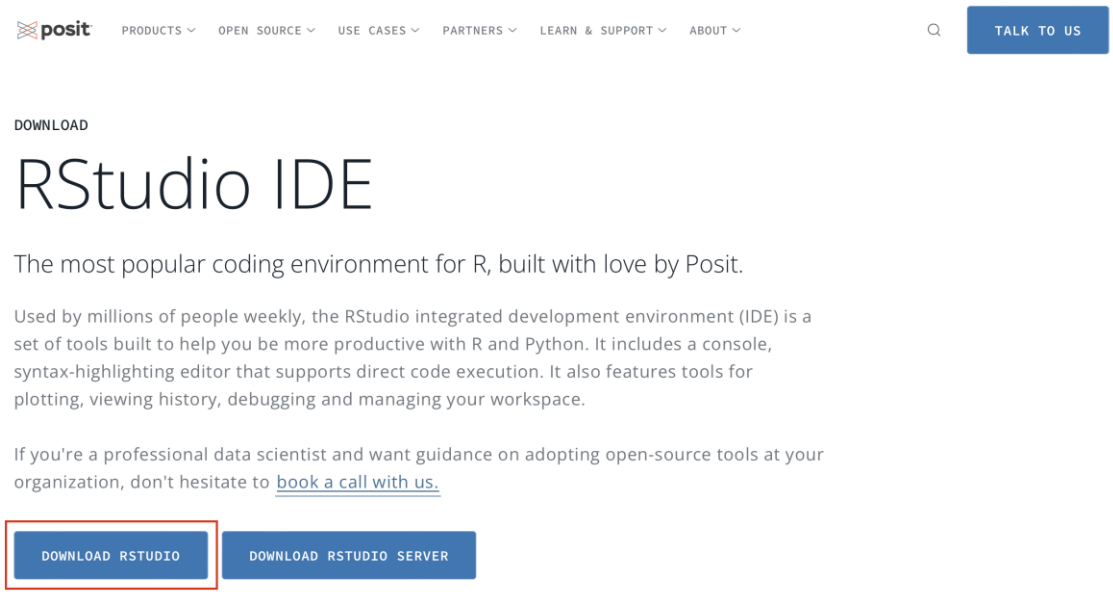


Figura 5: Web de RStudio

Descargamos e instalamos la versión de RStudio para escritorio. En la misma página, también encontrarás un enlace para instalar la última versión de R, en caso de que aún no lo hayas hecho.

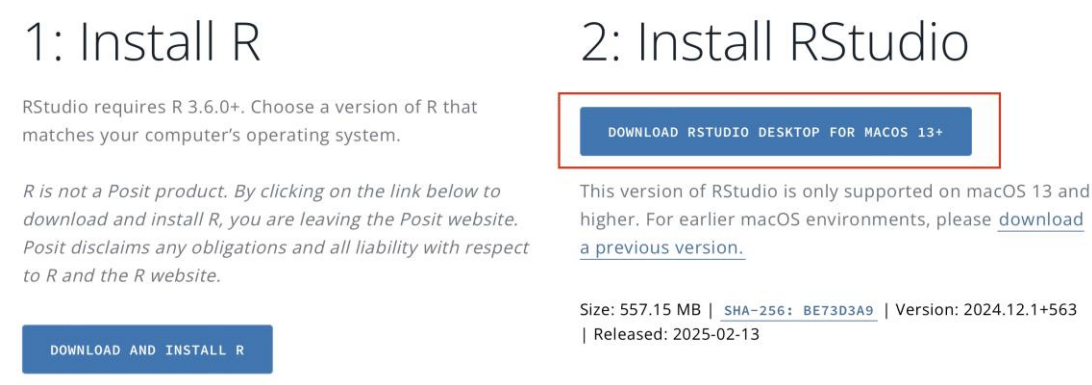


Figura 6: Descarga de R y RStudio

Si has llegado aquí sin seguir los pasos previos, no te preocupes, todavía estás a tiempo de ponerte al día. Lo importante es que primero instales R y después instales RStudio.

Entorno de Trabajo en RStudio

RStudio se compone de cuatro paneles principales:

Script (Editor de Código): Es la ventana usualmente situada en la esquina superior izquierda. En ella se escriben y se guardan los scripts de R cuya extensión es `.R`. Estos scripts permiten escribir, editar y ejecutar código de manera más organizada en comparación con la ejecución directa en la consola. Cumpliendo además una función similar a los archivos `.do` de Stata o los archivos de sintaxis `.sps` en SPSS. Permiten automatizar el análisis, garantizando la reproducibilidad del código y facilitando la depuración de errores en el análisis de datos.

Entre las principales ventajas de utilizar el editor de scripts en RStudio:

- Autocompletado de funciones y nombres de variables: A medida que escribe código, RStudio sugiere funciones y variables disponibles, reduciendo errores tipográficos y mejorando la eficiencia.
- Gestión de errores mejorada: El editor muestra advertencias y errores con mensajes descriptivos antes de ejecutar el código, lo que ayuda a depurar problemas.
- Resaltado de sintaxis con colores diferenciados: Diferentes elementos del código (funciones, variables, cadenas de texto, comentarios) se muestran en colores distintos, facilitando la lectura y comprensión del código.
- Ejecución de líneas o bloques de código específicos: Se puede seleccionar un fragmento de código y ejecutarlo haciendo clic en Run o mediante el atajo `Ctrl + Enter` (Windows/Linux) o `Cmd + Enter` (Mac) sin necesidad de ejecutar todo el script.
- Historial de cambios y recuperación de código: Permite guardar versiones del script y retroceder en caso de errores o modificaciones accidentales.
- Compatibilidad con Markdown y Notebooks: RStudio permite trabajar con R Markdown para combinar código con texto explicativo y generar informes reproducibles como este mismo documento.

Puedes crear un nuevo Script mediante la barra de herramientas clicando en File > New File > R Script o usando los atajos de teclado `Ctrl + Shift + N` en Windows/Linux o `Cmd + Shift + N` en Mac. Para guardar un Script puedes emplear la barra de herramientas File > Save As... o con `Ctrl + S` o `Cmd + S` en función de tu sistema operativo.

Consola: Se encuentra en la esquina inferior izquierda. Es la ventana donde se ejecutan los comandos y se visualizan los resultados en tiempo real. Básicamente la consola original de R. Podemos escribir

código directamente en la consola sin guardarlo en el script. Un interesante atajo para limpiar esta consola es `Ctrl + L`.

Entorno/Historial: El *Environment* en RStudio es el espacio de trabajo donde se almacenan y administran los objetos creados durante una sesión de R. Ubicada generalmente en la esquina superior derecha de la ventana. Estos objetos pueden ser de lo más diversos: desde un número, textos, tablas, bases de datos, funciones, resultados de un análisis, gráficos, matrices, etc.

Archivos/Gráficos/Paquetes/Ayuda: Ventana multifunción que facilita la navegación de archivos, la visualización de gráficos, la gestión de paquetes y la consulta de documentación. Aquí es donde el entorno se vuelve verdaderamente práctico y adaptable a distintas necesidades del análisis de datos. Motivo por el que merece la pena detenerse en las pestañas que la integran.

- *Archivos*: funciona como un pequeño explorador integrado dentro de RStudio. Permite navegar por las carpetas de tu proyecto y acceder rápidamente a los documentos, scripts y bases de datos que estés utilizando. Este acceso directo no solo facilita la carga de archivos al entorno de trabajo, sino que también fomenta una mejor organización. En el contexto de proyectos en R, mantener una estructura clara de carpetas y archivos es crucial: garantiza que tu análisis sea reproducible, escalable y fácil de compartir o retomar más adelante.
- *Gráficos*: recoger las salidas gráficas generadas. Es posible revisar gráficos anteriores, exportarlos o guardarlos en distintos formatos.
- *Paquetes*: Los paquetes son fundamentales para entender R. En este curso trabajamos con las herramientas incluidas en el entorno base. Sin embargo, R realmente se potencia gracias a su comunidad. Miles de paquetes han sido desarrollados por usuarios y organizaciones de todo el mundo, y cada uno de ellos amplía las capacidades del lenguaje. Desde visualización avanzada hasta análisis de redes, minería de texto o bioinformática, los paquetes permiten que R evolucione constantemente. Esta pestaña te permite ver qué paquetes tienes instalados, activarlos en tu sesión de trabajo o instalar nuevos con unos pocos clics. Aprender a identificar y usar los paquetes adecuados es una de las habilidades más valiosas que adquirirás con el tiempo.
- *Ayuda*: La comunidad también ha tratado de guiar el proceso de aprendizaje en ese mar de paquetes y funciones. Cada vez que tengas dudas sobre un paquete o función, puedes consultar su documentación oficial directamente desde aquí. Dichas ayudas incluyen ejemplos prácticos que puedes ejecutar para conocer mejor la función. Es muy recomendable acostumbrarse a visitar estas ayudas: son una fuente de información detallada, precisa y muchas veces más útil que cualquier búsqueda rápida en internet.

  Existen varias formas de acceder a la documentación de ayuda, tanto desde el propio RStudio como desde la consola.

- **Desde RStudio**

  La más obvia de todas las alternativas es precisamente la pestaña Help. En ella podemos navegar entre funciones, buscar en la caja de búsqueda, e incluso abrir la documentación en una pestaña del navegador interno.

  Otra alternativa bastante ágil es usar el atajo de teclado `F1` o `Fin+F1` mientras el cursor se encuentre en la función sobre la que queremos ayuda.

- **Desde la consola**

  Podemos utilizar la función `help(función)` con la función sobre la que solicita ayuda. Esto devolverá la documentación oficial sobre la función. Otra alternativa equivalente es `?función`. Puede ser útil la alternativa `??keyword` cuando queremos hacer una búsqueda global en toda la ayuda, bien porque buscamos ayuda sobre un paquete o cuando no recordamos el nombre exacto de la función. Para buscar ayuda específica sobre un paquete podemos usar `help(package = "paquete")`.

- *Viewer* Al principio quizá no se utilice mucho. Es aquí donde se visualizan aplicaciones interactivas hechas con Shiny, informes generados con R Markdown o visualizaciones interactivas que responden al usuario. Esta capacidad de integrar contenido dinámico dentro del entorno convierte a RStudio en una poderosa herramienta no solo para análisis, sino también para comunicación de resultados.

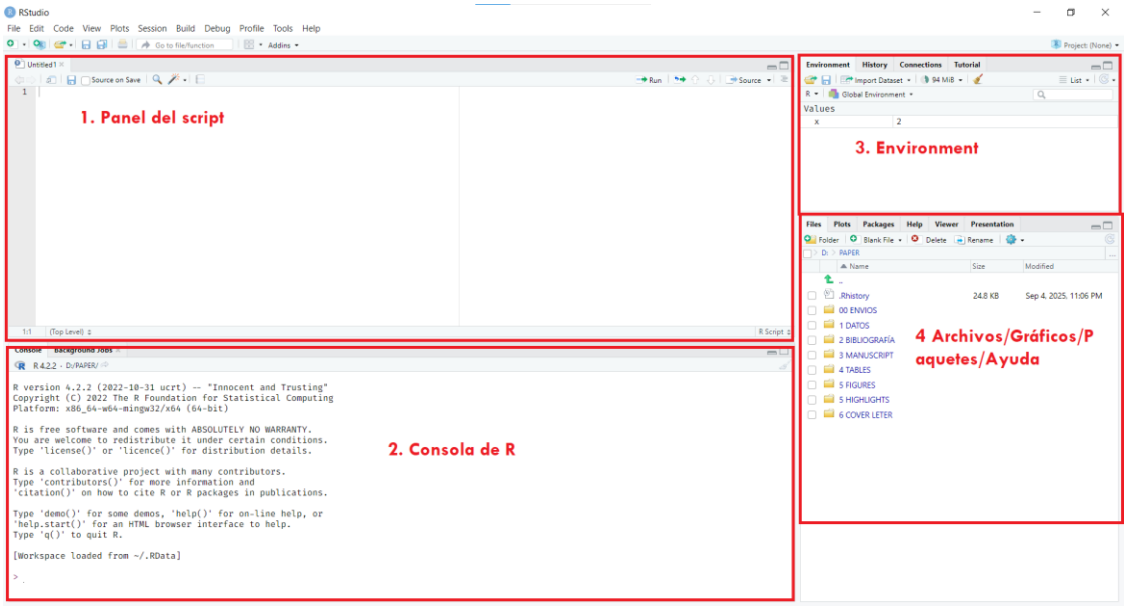


Figura 7: Paneles de RStudio

Sintaxis básica: variables, operadores, funciones

R puede funcionar como una simple calculadora, pero su verdadero poder radica en la creación, manipulación y administración de objetos. Estos objetos pueden ser tan diversos como:

- Vectores (`numeric`, `character`, `logical`, etc.)
- Matrices (`matrix`)
- Listas (`list`)
- Dataframes (`data.frame`)
- Factores (`factor`)
- Funciones (`function`)

La mejor manera de entender su sintaxis es practicando. Por ello, para esta sección vamos a abrir un nuevo script. Se recuerda que se puede crear un nuevo Script mediante la barra de herramientas clicando en File > New File > R Script o usando los atajos de teclado `Ctrl + Shift + N` en Windows/Linux o `Cmd + Shift + N` en Mac.

En R, las variables almacenan valores y se asignan con <- (`< + -`) o =. Aunque ambos funcionan, se recomienda <- por convención. En RStudio la flecha puede escribirse más rápido usando `Alt + -` o `Option + -`\. Para crear ese objeto tendríamos que escribir `a \< - 2 + 2`. Crearemos

nuestro primer objeto llamado `a` como el producto de 2 * 3 y ejecutamos con `Ctrl + Intro` o `Cmd + Intro`.

```
a <- 2*3
```

Al ejecutar el código suceden dos hechos importantes. El primero es que el resultado de la operación ya no aparece en la consola. Para ver ese resultado debemos llamar al objeto por su nombre.

```
a

## [1] 6
```

El segundo es que nuestro primer objeto se incorpora a nuestro *Environment*.

Llegados a este punto, es importante tener en cuenta que R tiene por norma el reciclaje y el ahorro. Veamos este script línea a línea.

```
# Este signo se usa para comentar el código.

a <- 2 # Para generar un objeto se usan los símbolos "<" y "-".
a # Ahora podemos llamar a nuestro objeto a cuyo valor es 2.

## [1] 2

a <- c(3,4,5) # Ahora R sobrescribirá el objeto a.
              # La función "c" sirve para definir un conjunto de valores.
a # Aquí vemos como ahora a es un vector.

## [1] 3 4 5

a <- rep(1:3, length = 16)  # Ahora con rep() pedimos repetir una secuenc
ia.
                            # 1:3 genera una secuencia desde 1 hasta 3.
                            # con length = 16 queremos fijar la longitud
de a
a # Vemos que mantiene la repetición de la secuencia hasta completar los
16.

##  [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1

rm(a) # Esta función elimina un objeto del environment.
```

También es importante reseñar que R distingue entre mayúsculas y minúsculas.

```r
a <- "Blanco" # Esto creará un objeto character.
A <- F # Esto creará un objeto lógico. F = FALSE
str(A) # Función para mostrar de forma compacta la estructura interna de un objeto
```

```
##  logi FALSE
```

```r
class(a) # Especificará la clase del objeto
```

```
## [1] "character"
```

```r
rm(a)
rm(A)
# Se empieza a vislumbra la estructura de las funciones
# Básicamente: nombre_funcion(argumentos)
```

Operadores aritméticos

```r
x <- 10
y <- 5
x + y      # Suma (10 + 5 = 15)
```

```
## [1] 15
```

```r
x - y     # Resta (10 - 5 = 5)
```

```
## [1] 5
```

```r
x * y  # Multiplicación (10 * 5 = 50)
```

```
## [1] 50
```

```r
x / y   # División (10 / 5 = 2)
```

```
## [1] 2
```

```r
x^2     # Exponenciación (10^2 = 100)
```

```
## [1] 100
```

```r
x %% 3     # Resto de la división (resto de 10/3 = 1)
```

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

Operadores de comparación

```
x > y   # ¿x es mayor que y?
## [1] TRUE
x < y   # ¿x es menor que y?
## [1] FALSE
x == y  # ¿x es igual a y?
## [1] FALSE
x != y  # ¿x es diferente de y? El símbolo se genera con ! =
## [1] TRUE
x >= y  # ¿x es mayor o igual a y?
## [1] TRUE
x <= y  # ¿x es menor o igual a y?
## [1] FALSE
```

Operadores lógicos

Los operadores lógicos permiten combinar múltiples condiciones y tomar decisiones complejas sobre datos. Son fundamentales para filtrar, validar y manipular información.

```
(x > 5) & (y < 10)  # AND: ¿x > 5 y y < 10?
## [1] TRUE
(x > 5) | (y == 5)  # OR: ¿x > 5 o y == 5?
## [1] TRUE
!(x == 10)          # NOT: ¿x NO es 10?
## [1] FALSE
# Estas estructuras pueden ser muy útiles para desarrollar medidas de con
trol o funciones.
```

Álgebra matricial

```
X <- matrix(c(rep(1,10),sample(10:20,10), sample(25:70, 10)), ncol=3)

# Aquí hay varias funciones nuevas. Puedes conocer más sobre una función
de varias formas:

# Al escribir una función RStudio de dará una pequeña descripción de la f
unción.
# Pulsando F1 abrirás la ayuda en la ventana de ayuda.
# El la propia ventana de ayuda hay un buscador de funciones y paquetes.
# Usando la función help(nombredelafuncion)
# Escribiendo ?nombredelafuncion

# Las ayudas de las funciones son muy útiles, con ejemplos que se pueden
ejecutar para conocer más sobre ellas.

Y <- c(10.2, 12.5, 7.9, 6.9, 3.8, 4.5, 6.3, 14.6, 9, 12.4) # Para la sepa
ración decimal se usa el punto y no la coma que se reserva para separar.

tX <- t(X) # Transpuesta de una matriz.
           # Recuerda que R distingue entre mayúsculas y minúsculas.

X_X <- tX%*%X # Producto de matrices. El símbolo para multiplicar matrice
s es %*%
              # Su usásemos únicamente * intentaría multiplicar cada elem
ento ij
              # de una matriz con su homónimo en la otra matriz.

X_Y <- t(X)%*%Y # No tienes por qué crear un objeto siempre.
                # Puedes saltarte todos los pasos intermedios que necesit
es

X_X_1 <- solve(X_X) # Sirve para muchos propósitos entre ellos para inver
tir matrices

determinante <- det(X_X) # Para calcular el determinante

Beta <- X_X_1%*%X_Y
```

```
Beta # ¡Oh, sorpresa! Este vector parece contener información clave 👀
```

```
##              [,1]
## [1,] 10.65019086
## [2,]  0.16752751
## [3,] -0.08929524
```

```
    # Pero no adelantemos conclusiones.
    # Un saludo a los compañeros del departamento.
    # Estoy seguro de que lo vieron venir desde lejos 🤣
```

Funciones

Ya hemos visto algunas de la infinidad de funciones existentes en R. Entre las más utilizadas a efectos de cálculo quedarían:

```
sqrt(144) # Raíz cuadrada
```

```
## [1] 12
```

```
log(100) # Logaritmo Neperiano.
```

```
## [1] 4.60517
```

```
log(100, base=10) # Logaritmo en base 10
```

```
## [1] 2
```

```
round(200/14,3) # Redondear un resultado a un número de decimales.
```

```
## [1] 14.286
```

R cuenta con una enorme colección de funciones y paquetes creados por la comunidad para realizar casi cualquier tarea. Sin embargo, en algunos casos, la función que necesitamos puede no existir o simplemente no queremos perder tiempo buscándola. La verdadera potencia de R radica en su flexibilidad: podemos crear nuestras propias funciones para automatizar tareas y adaptarlas a nuestras necesidades.

```
area <- function(base,altura) { # Los parámetros requeridos separados por comas.
                                # Se abre y se cierra la función con llaves.
```

```
  return(base*altura) # Operaciones necesarias.
                      # La función return indica que proporcione el resul
tado.
} # Fin de la función

# El último comentario de fin de la función se suele indicar por cortesía
# para facilitar la lectura del script.

area(2,4) # La función aparece en el environment como un objeto más

## [1] 8
```

En este ejemplo hemos creado un objeto del tipo función llamado *area*. Para que calcule el área de un rectángulo. Luego lo hemos puesto a prueba indicando como argumentos la *base* y la *altura* como 2 y 4 respectivamente. La salida indicada con la función *return* es obviamente 8.

Existen otro tipo de funciones en R conocidas como funciones anónimas. Funciones que pueden resultar útiles cuando necesitamos realizar una operación rápida sin definir una función con nombre. Son empleadas comúnmente en situaciones donde la función solo se usará una vez o dentro de otra función como argumento.

```
# Ejemplo de función anónima

(function(x, y){ z <- x^2 + y^2; x+y+z })(0:7, 1)

## [1]  2  4  8 14 22 32 44 58
```

Manipulación de datos básicos (vectores, listas, matrices, dataframes)

R puede trabajar con una enorme variedad de objetos prácticamente sin esfuerzo. A continuación, veremos algunos objetos algo más complejos.

Vectores

```
# Ejecuta el código línea a línea y comprueba qué pasa con el objeto v

v <- c(1,2,3,4,5,6,7,8,9,10) # Crea un vector de 10 elementos.
length(v)  # Es la función para obtener la longitud de un vector.

## [1] 10
```

```
v <- seq(from=1, to=20, by=2) # Función para establecer una secuencia.
                              # El argumento by indica el incremento en l
a secuencia.

v[5] # Los corchetes se usan para indicar un orden. La posición 5 del vec
tor v.

## [1] 9

v[7] <- 21     # Cambio un valor concreto del vector.  v[7] ahora es 21.
v[-7] # El signo - en la posición genera un objeto con todos los datos sa
lvo ese.

## [1]  1  3  5  7  9 11 15 17 19

which(v==21) # Localiza las posiciones que cumplen una determinada condic
ión.

## [1] 7

v <- c(rep(7,10),rep(12,15),rep(15,5)) # rep crea repeticiones
unique(v) # Obtiene los valores únicos de nuestro vector.

## [1]  7 12 15

table(v) # Genera una tabla de frecuencias de nuestro vector.

## v
##  7 12 15
## 10 15  5

t <- table(v) # Puedo crear un objeto con esa tabla.
```

Matrices

Ya en la operativa matricial se ha introducido el objeto *matrix*. A continuación, vamos a ahondar en su manipulación.

```
# Ejecuta el código línea a línea y comprueba qué pasa con el objeto m

m <- matrix(c(2022,2023,2024, 11,12,9, 4,7,3), nrow = 3, ncol=3)
```

```
m # El resultado sería el mismo manteniendo únicamente uno de los argumen
tos: nrow o ncol.
```

```
##      [,1] [,2] [,3]
## [1,] 2022   11    4
## [2,] 2023   12    7
## [3,] 2024    9    3
```

```
m[5] # Si usamos el corchete localizará la posición 5 como si fuera un ve
ctor.
```

```
## [1] 12
```

```
m[1,3] # Proporciona el valor de la matriz en la fila 1, columna 3
```

```
## [1] 4
```

```
m[1,] # Proporciona los datos de la fila 1
```

```
## [1] 2022   11    4
```

```
m[,3] # Proporciona los datos de la columna 3
```

```
## [1] 4 7 3
```

```
m <- matrix(c(2022,2023,2024, 11,12,9, 4,7,3), nrow = 3, ncol=3, byrow =
TRUE)
m # El argumento byrow sirve para establecer los datos por filas.
```

```
##      [,1] [,2] [,3]
## [1,] 2022 2023 2024
## [2,]   11   12    9
## [3,]    4    7    3
```

```
m <- matrix(c(11,12,9, 4,7,3), nrow = 2, ncol=3, byrow = TRUE,
              dimnames = list(c("Producción", "Exportaciones"),
                              c("2022", "2023", "2024")))
m # Con el argumento dimnames le hemos dado nombre a las filas y las colu
mnas mediante una lista.
```

```
##                2022 2023 2024
## Producción       11   12    9
## Exportaciones     4    7    3
```

```
m[,1] # Podemos filtrar marcando en orden de la matriz

##    Producción Exportaciones
##            11             4

subset(m, select = "2022") # O utilizar un filtro.

##                2022
## Producción       11
## Exportaciones     4

rownames(m) # Para obtener el nombre de las filas

## [1] "Producción"    "Exportaciones"

colnames(m) # Para obtener el nombre de las columnas

## [1] "2022" "2023" "2024"
```

Listas

No es casualidad que al final del ejemplo anterior aparezca la función `list()`. Las listas en R son estructuras flexibles que pueden almacenar distintos tipos de datos (vectores, matrices, otros dataframes, etc.). Una lista en R es como ese cajón de tu casa o el trastero: puedes meter de todo, desde un destornillador hasta una bicicleta, pasando por un táper sin tapa. Lo malo es que, cuando necesitas algo, tienes que rebuscar bien para encontrarlo.

```
# Ejecuta poco a poco el código y comprueba los resultados.

lista <- list( # Saltos introducidos por consenso para hacer más legible
el código.
  Nombres = c("Lucia", "Pedro","Miguel","Rosa", "Sergio"), # Fíjate como
cada grupo                                                          s
e separa por comas.
  Edad = c(22, 25, 12, 7, 9), # Se podría usar la flecha, pero el igual a
quí si                                  está aceptado.
  Grado =c(F,T,F,F) # Puede escribirse FALSE O F, TRUE O T.
  ) # Cierra de la lista

lista[1] # Proporciona la caja 1 con las cosas dentro. Fuera tiene escrit
o: Nombres
```

```
## $Nombres
## [1] "Lucia"  "Pedro"  "Miguel" "Rosa"   "Sergio"

lista[[1]] # Abre la primera caja. Da el contenido sin la caja de cartón.

## [1] "Lucia"  "Pedro"  "Miguel" "Rosa"   "Sergio"

lista[[1]][2] # De los elementos dentro de la caja da el segundo.

## [1] "Pedro"

lista[["Grado"]] # Contenido de la caja que tiene escrito: Grado. Sin usa
r posiciones.

## [1] FALSE  TRUE FALSE FALSE

lista$Grado # La forma más común de buscar.

## [1] FALSE  TRUE FALSE FALSE

lista$Nombres[2] # Da el segundo objeto de la caja nombres.

## [1] "Pedro"

length(lista) # cuenta cuantas cajas tiene la lista.

## [1] 3

names(lista) # Proporciona los nombres de las cajas

## [1] "Nombres" "Edad"    "Grado"

cosas <- c("Destornillador","Taladro","Llave Inglesa", "Amoladora")

lista <- c(lista,list(cosas)) # Hemos metido las cosas en una caja y la h
emos                                    guardado en el trastero junto c
on todo lo demás.

names(lista) # La última caja no tiene nombre.

## [1] "Nombres" "Edad"    "Grado"   ""

names(lista)[4] <- "Cosas" # También podemos usar names para nombrar las
cajas.
names(lista)
```

```
## [1] "Nombres" "Edad"    "Grado"   "Cosas"
```

Dataframes

Los dataframe un tipo especial de lista con ciertas propiedades especiales en las que cada elemento es un vector de la misma longitud. Representan la clásica estructura de datos tabulares, al estilo de una hoja de cálculo de Excel.

- Cada fila representa una observación (como una fila en Excel que describe un individuo, una flor, un coche...).
- Cada columna representa una variable (como una columna en Excel: edad, altura, especie, etc.).
- Las columnas pueden tener distintos tipos de datos: números, texto, valores lógicos, factores, etc.)

Su facilidad de lectura, versatilidad y simplicidad han impulsado su uso generalizado en R, haciendo que muchas funciones estén especialmente diseñadas para trabajar con ellos.

```
dataset <- data.frame(
  Nombres = c("Lucia", "Pedro","Miguel","Rosa", "Sergio"), # Recuerda las comas.
  Edad = c(22, 25, 12, 7, 9), # Recuerda las comas. No está de más recordarlo.
  Grado =c(F,T,F,F,"")
  ) # La estructura es similar a la lista.

# Al ser una tabla todos los elementos deben tener el mismo número de filas.
# Observa las "" finales de Grado.
```

Exploración de un dataframe

Existen algunas funciones interesantes para los dataframe.

```
names(dataset) # Ya es una habitual en los ejemplos.

## [1] "Nombres" "Edad"    "Grado"

head(dataset) # Proporciona las primeras filas del dataframe

##   Nombres Edad Grado
## 1   Lucia   22 FALSE
```

```
## 2   Pedro   25  TRUE
## 3  Miguel   12 FALSE
## 4    Rosa    7 FALSE
## 5  Sergio    9
```

```
tail(dataset) # Proporciona las últimas filas del dataframe
```

```
##   Nombres Edad Grado
## 1   Lucia   22 FALSE
## 2   Pedro   25  TRUE
## 3  Miguel   12 FALSE
## 4    Rosa    7 FALSE
## 5  Sergio    9
```

```
nrow(dataset) # Proporciona el número de filas
```

```
## [1] 5
```

```
ncol(dataset) # Proporciona el número de columnas
```

```
## [1] 3
```

```
summary(dataset) # Resumen atendiendo a la naturaleza de las variables
```

```
##    Nombres               Edad       Grado
##  Length:5           Min.   : 7   Length:5
##  Class :character   1st Qu.: 9   Class :character
##  Mode  :character   Median :12   Mode  :character
##                     Mean   :15
##                     3rd Qu.:22
##                     Max.   :25
```

Acceso a los elementos del dataframe

Se puede acceder al dataframe por posiciones como en los anteriores elementos haciendo uso de los corchetes o mediante la estructura vista también en las listas `dataset$variable`. Algunos ejemplos:

```
# Ejecuta el código línea a línea para comprobar los resultados.

# Por posición:
```

```
dataset[1] # Proporciona la primera columna del dataframe manteniendo la
estructura              tabla propia del dataframe

##   Nombres
## 1   Lucia
## 2   Pedro
## 3  Miguel
## 4    Rosa
## 5  Sergio

dataset[[1]] # Proporciona el contenido de la primera columna como un vec
tor.

## [1] "Lucia"  "Pedro"  "Miguel" "Rosa"   "Sergio"

dataset[1,] # Primera fila como un dataframe con un único elemento.

##   Nombres Edad Grado
## 1   Lucia   22 FALSE

dataset[,1] # Primera columna como vector. Sin la condición de dataframe.

## [1] "Lucia"  "Pedro"  "Miguel" "Rosa"   "Sergio"

dataset[1,1] # Primera posición de la primera columna. El equivalente a l
a casilla               A1 de Excel.

## [1] "Lucia"

dataset[[1]][1] # Lo mismo pero con un formato distinto. Pruébalo en otro
s objetos.

## [1] "Lucia"

# Por nombre de variables

dataset$Nombres # Usando el $. En RStudio se debería abrir un desplegable
con los                 posibles nombres.

## [1] "Lucia"  "Pedro"  "Miguel" "Rosa"   "Sergio"

# Mezcla de ambos.
```

```
dataset$Edad[1] # Dentro de la variable edad, el valor del primer element
o observado.

## [1] 22
```

Añadir filas, columnas o combinar dataframes

```
# Unir al mismo dataframe

nuevo <- data.frame(Nombres ="Virginia",
                    Edad = 45,
                    Grado = F
                    )

dataset <- rbind(dataset,nuevo) # Para unir filas.

sueldo <- c(2200,1700,1800,2100,1500,2000)

dataset <- cbind(dataset,sueldo) # Para unir columnas.

# Unir dos dataframes

data1 <- dataset[1:3,] # Separo dataset en las tres primeras observacione
s
data2 <- dataset[4:6,] # Separo dataset en las tres últimas observaciones

data3 <- dataset[-(1:3),] # ¿A qué se parece esto?

rm(data3) # Es la función para borrar objetos del environment

data <- rbind(data1,data2)

# De forma análoga para unir dataframes por columnas.
```

Filtrar y ordenar

```
subset(dataset, Grado==F) # Filtra con los elementos que cumplen la condi
ción.

##    Nombres Edad Grado sueldo
## 1    Lucia   22 FALSE   2200
```

```
## 3   Miguel   12 FALSE   1800
## 4     Rosa    7 FALSE   2100
## 6 Virginia   45 FALSE   2000
```

```
dataset[dataset$Grado==F,] # Filtrado con operadores lógicos.
```

```
##    Nombres Edad Grado sueldo
## 1    Lucia   22 FALSE   2200
## 3   Miguel   12 FALSE   1800
## 4     Rosa    7 FALSE   2100
## 6 Virginia   45 FALSE   2000
```

Ordenar

```
dataset_ord <- dataset[order(dataset$sueldo),]
dataset_ord
```

```
##    Nombres Edad Grado sueldo
## 5   Sergio    9         1500
## 2    Pedro   25  TRUE   1700
## 3   Miguel   12 FALSE   1800
## 6 Virginia   45 FALSE   2000
## 4     Rosa    7 FALSE   2100
## 1    Lucia   22 FALSE   2200
```

```
# La función order da las posiciones.
# Usándola dentro del corchete indicamos el orden que deseamos.
```

```
dataset_ord <- dataset[order(-dataset$sueldo),] # Si quisiéramos ordenar
en sentido descendente.
```

```
dataset_ord
```

```
##    Nombres Edad Grado sueldo
## 1    Lucia   22 FALSE   2200
## 4     Rosa    7 FALSE   2100
## 6 Virginia   45 FALSE   2000
## 3   Miguel   12 FALSE   1800
## 2    Pedro   25  TRUE   1700
## 5   Sergio    9         1500
```

Factores

Los factores son una estructura de datos especial en R que representa variables categóricas nominales u ordinales. Aunque pueden parecer similares a vectores de texto, tienen características únicas que los hacen poderosos para análisis estadístico pero que también pueden causar sorpresas si no se entienden bien.

```
ejemplo <- factor(c("rojo", "verde", "azul", "azul", "rojo", "rojo"))
ejemplo # Para que nos muestre el objeto debemos llamarlo por su nombre.

## [1] rojo  verde azul  azul  rojo  rojo
## Levels: azul rojo verde

levels(ejemplo) # Indica las diferentes modalidades encontradas.

## [1] "azul"  "rojo"  "verde"
```

¿Por qué son importantes los factores?

Los factores son esenciales en análisis estadístico porque R los trata de manera especial en modelos, gráficos y tablas. Sin embargo, su naturaleza puede sorprender:

Es posible recodificar los valores de los niveles de un factor con el argumento `labels`. Veamos un ejemplo:

```
ejemplo <- c(0,1,1,0,1,0,0,1) # Reciclamos ejemplo
unique(ejemplo) # Función parecida a levels aplicable con más tipos de ob
jetos.

## [1] 0 1

ejemplo <- factor(ejemplo, labels = c("hombre", "mujer")) # Recodificamos
.
ejemplo

## [1] hombre mujer  mujer  hombre mujer  hombre hombre mujer
## Levels: hombre mujer

levels(ejemplo)

## [1] "hombre" "mujer"
```

```
levels(ejemplo) <- c("Hombre", "Mujer") # con levels también se pueden ca
mbiar los                                                    nombres.
```

Aunque su utilidad es fundamental, su manipulación puede ser peligrosa. Una vez que los datos están como factores, es difícil incorporar modalidades diferentes. Por ello, es recomendable la conversión una vez la base de datos esté cerrada o utilizar funciones temporales.

```
ejemplo <- c("rojo", "verde", "azul", "azul", "rojo", "rojo") # Reciclamo
s otra vez
levels(as.factor(ejemplo)) # Usamos una función sobre el resultado de otr
a función

## [1] "azul"  "rojo"  "verde"
```

Análisis Estadístico.

R permite realizar todos análisis estadísticos (tablas, gráficos y cálculo de estadísticas) de forma sencilla sin apenas gastar recursos. Hasta este punto hemos aprendido a crear objetos, manipularlos y entender las estructuras de datos fundamentales. Llegados a este punto vamos a importar datos externos, algo realmente habitual en este y otros tantos programas y trabajar sobre el ejemplo.

Ejemplo con RStudio

Para esta guía descargaremos a modo de ejemplo una base de datos pública (clic aquí) subida a GitHub por Douae-Naciri correspondiente a un proyecto de minería de datos sobre emprendimiento que lleva por título: *"Prédiction du potentiel entrepreneurial des étudiants: Une approche basée sur KNN et Random Forest"* @naciri2025 . Dicho proyecto pretende analizar si un estudiante tiene un perfil emprendedor a partir de datos recogidos a través de un formulario. Entre las variables recogidas se incluye información sobre diversos factores personales, demográficos y psicológicos.

Estructura del fichero:

Cada fila representa un estudiante y contiene las siguientes variables:

- **EducationSector**: Sector educativo al que pertenece el estudiante.
- **IndividualProject**: Indica si el estudiante ha desarrollado un proyecto individual (Sí/No).
- **Age**: Edad del estudiante.
- **Gender**: Género del estudiante.
- **City**: Ciudad de residencia.
- **Influenced**: Indica si el estudiante ha sido influenciado por alguien para emprender (ej. familia, amigos, profesores).

- **Perseverance**: Nivel de perseverancia del estudiante.
- **DesireToTakeInitiative**: Grado de deseo de tomar la iniciativa en proyectos o negocios.
- **Competitiveness**: Nivel de competitividad del estudiante.
- **SelfReliance**: Nivel de confianza en sí mismo para resolver problemas.
- **StrongNeedToAchieve**: Intensidad de la necesidad de lograr objetivos.
- **SelfConfidence**: Nivel de autoconfianza del estudiante.
- **GoodPhysicalHealth**: Estado de salud física general.
- **MentalDisorder**: Indica si el estudiante ha sido diagnosticado con algún trastorno mental (Sí/No).
- **KeyTraits**: Rasgos clave asociados con el potencial emprendedor del estudiante.
- **ReasonsForLack**: Motivos por los cuales el estudiante no ha desarrollado habilidades emprendedoras.
- **y**: Variable objetivo que puede representar si el estudiante tiene potencial emprendedor (Sí/No o una escala de evaluación).

Carga del fichero:

La forma más sencilla de importar archivos externos es a través de la ruta File > Import Dataset > From...

En nuestro caso, al tratarse de un archivo con extensión csv usaremos From Text (base)... Esto abrirá el explorador del sistema en el que tendremos que localizar el archivo y, una vez seleccionado, la interfaz que aparece se refleja en la Figura 8.

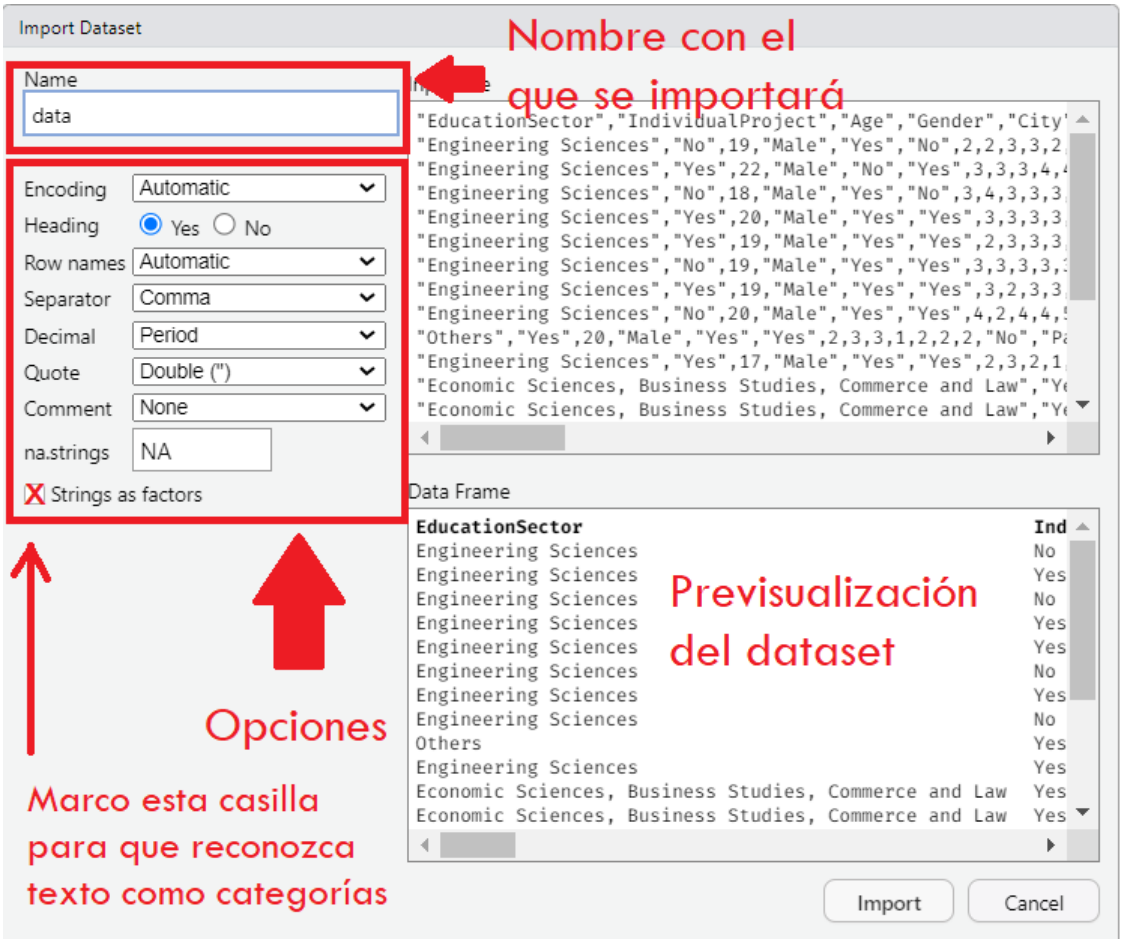


Figura 8: Explorador de archivos

Otra alternativa para la importación sería acudir al panel de **Archivos/Gráficos/Paquetes/Ayuda** situado en de la parte inferior derecha y localizar el archivo. Se recomienda crear estructuras de trabajo sencillas. En la Figura 9 se plantea una estructura de artículo ubicándose para este ejemplo el archivo en la carpeta datos. Si queréis acudir a otra carpeta podéis usar en explorador propio de RStudio o hacer clic en los tres puntos horizontales visibles en la parte superior derecha del panel. Con ello, se abrirá el explorador del propio sistema operativo y podremos localizar el archivo en cuestión.

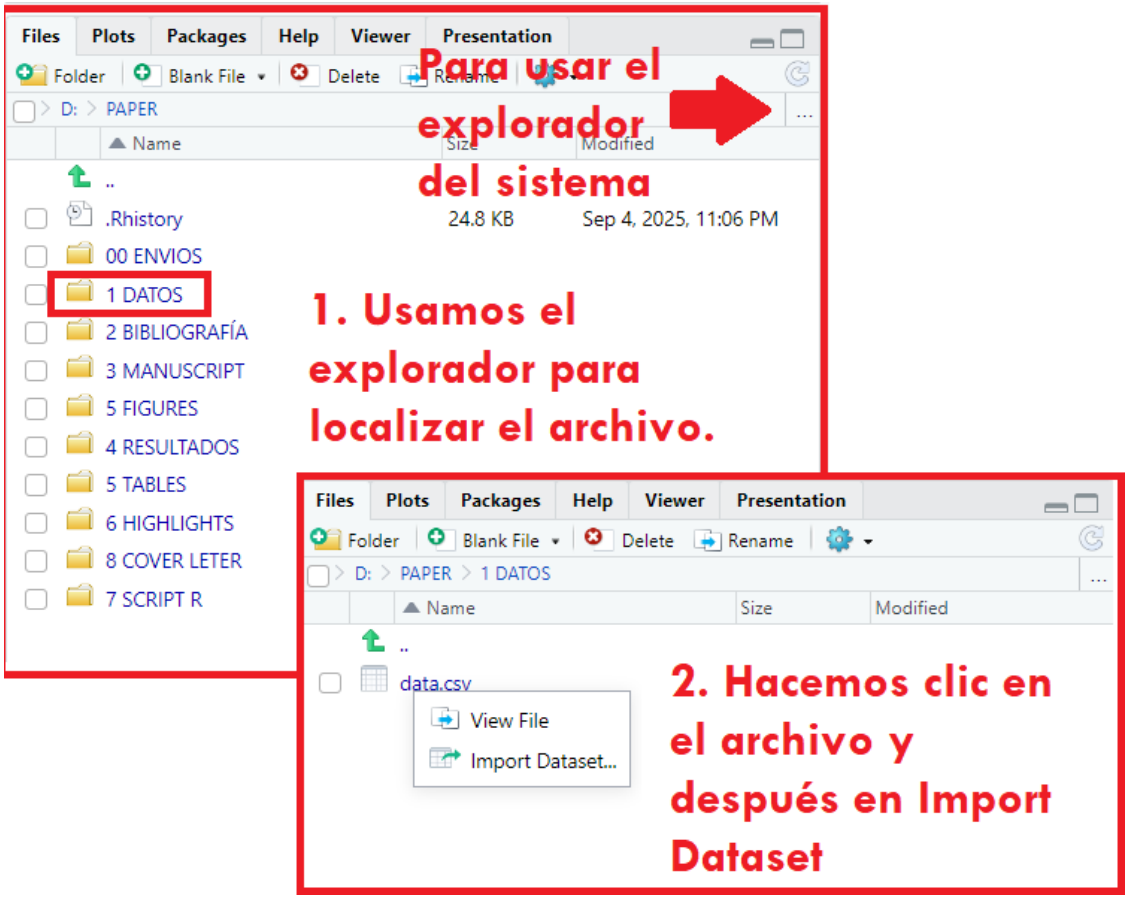


Figura 9: Explorador de archivos

Una vez localizado hacemos clic en él y seleccionamos Import Dataset lo que nos llevará a la Figura 10. En esta pantalla podemos ver la ruta del fichero y comprobar que, en este caso, la importación se ha llevado a cabo directamente desde la carpeta de descargas. Además, existe una ventana de previsualización con la que comprobar si la importación es o no correctas. En dicha ventana podríamos modificar los tipos variables clicando en las flechas que aparecen al lado del tipo de variable: double, character, etc. Por ejemplo, podríamos como antes convertir las variables que queramos en factores.

En la parte inferior izquierda existen una serie de opciones para configurar correctamente la importación. Por un lado, el nombre con el que se importará la base de datos al *Environment*, el cual por defecto es `data` y por otro las distintas opciones para una correcta importación. Por último, en la parte inferior derecha encontramos el código con el que se importará la base de datos. Existen dos funciones que merecen especial mención. La primera es `library(readr)`. Es una función para llamar a una librería externa de funciones. Los paquetes de los que hemos hablado

anteriormente. Está función está estrechamente ligada a otra con la que se instalan paquetes externos `install.packages()` a la que puedes también acceder a través de la pestaña de paquetes. Dado el carácter introductorio del presente manual no se va a ampliar las posibilidades que ofrecen los paquetes de R.

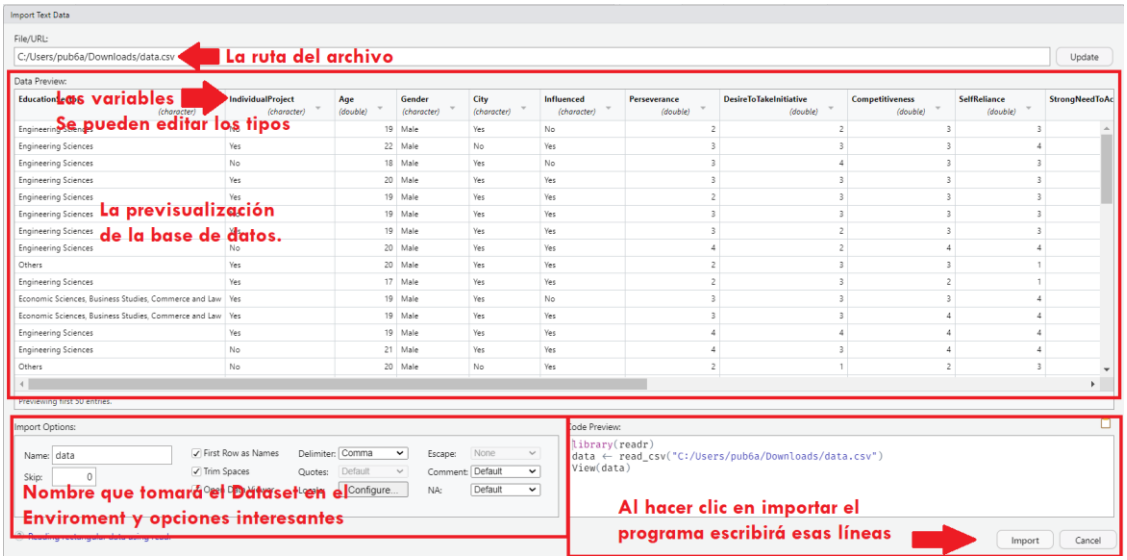


Figura 10: Importar archivo

Otra función es `View(data)`. Esta función realmente no es necesaria para la importación. Se incluye para abrir una pestaña con la visualización de los datos. Si la borramos antes de importar veremos que la pestaña que aparece a la izquierda en la Figura 10 no aparecería. Si hiciéramos clic en el *Enviroment* a data, veremos que en la consola automáticamente se escribe `View(data)`.

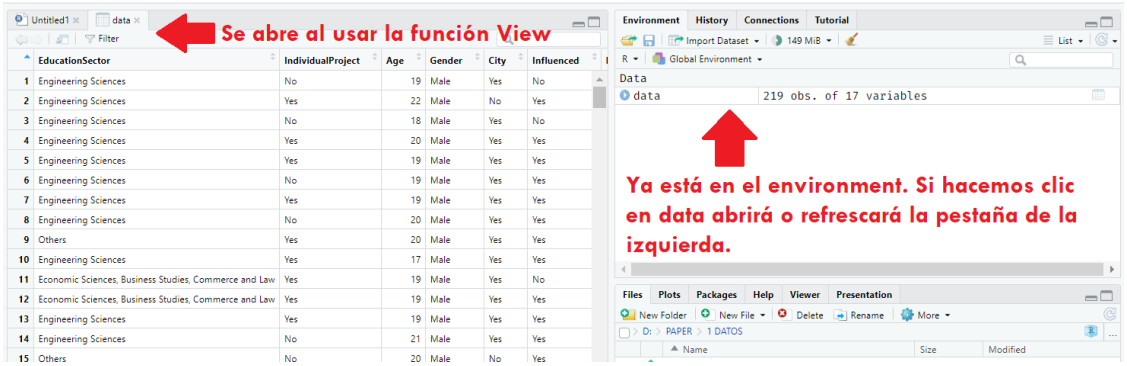


Figura 11: Archivo importado

```
summary(data)
```

```
##                                                EducationSector Individu
alProject
##  Engineering Sciences                                    :123    No : 95
##  Economic Sciences, Business Studies, Commerce and Law: 32    Yes:124
##  Art, Music or Design                                    : 21
##  Others                                                  : 20
##  Medicine, Health Sciences                               : 10
##  Humanities and Social Sciences                          :  5
##  (Other)                                                 :  8
##       Age           Gender     City     Influenced  Perseverance
##  Min.   :17.00   Female: 57   No : 36   No : 61    Min.   :1.000
##  1st Qu.:19.00   Male  :162   Yes:183   Yes:158    1st Qu.:3.000
##  Median :20.00                                     Median :3.000
##  Mean   :19.75                                     Mean   :3.352
##  3rd Qu.:20.00                                     3rd Qu.:4.000
##  Max.   :26.00                                     Max.   :5.000
##
##  DesireToTakeInitiative Competitiveness  SelfReliance   StrongNeedToAc
hieve
##  Min.   :1.000          Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:3.000          1st Qu.:3.000   1st Qu.:3.000   1st Qu.:3.000
##  Median :4.000          Median :4.000   Median :4.000   Median :4.000
##  Mean   :3.621          Mean   :3.589   Mean   :3.721   Mean   :3.909
##  3rd Qu.:5.000          3rd Qu.:4.500   3rd Qu.:5.000   3rd Qu.:5.000
##  Max.   :5.000          Max.   :5.000   Max.   :5.000   Max.   :5.000
##
##  SelfConfidence  GoodPhysicalHealth MentalDisorder      KeyTraits
##  Min.   :1.000   Min.   :1.000      No :155        Passion   :62
##  1st Qu.:3.000   1st Qu.:3.000      Yes: 64        Positivity:73
##  Median :4.000   Median :4.000                     Resilience:10
##  Mean   :3.575   Mean   :3.562                     Vision    :35
##  3rd Qu.:4.000   3rd Qu.:4.000                     Work Ethic:39
##  Max.   :5.000   Max.   :5.000
##
##
ReasonsForLack
##
:91
##  Just not interested! (Want to work in the corporate sector, or for th
```

```
e government or pursue research or something else):41
##  Academic Pressure
:11
##  Not willing to start a venture in India and waiting for future reloca
tion                                                  :10
##  Lack of Knowledge
: 7
##  Academic Pressure, Unwillingness to take risk, Lack of Knowledge
: 4
##  (Other)
:55
##        y
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.4155
##  3rd Qu.:1.0000
##  Max.   :1.0000
##
```

```
summary(data$Age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   17.00   19.00   20.00   19.75   20.00   26.00
```

```
summary(data[["Age"]])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   17.00   19.00   20.00   19.75   20.00   26.00
```

```
summary(data[2:4])
```

```
##  IndividualProject      Age           Gender
##  No : 95           Min.   :17.00   Female: 57
##  Yes:124           1st Qu.:19.00   Male  :162
##                    Median :20.00
##                    Mean   :19.75
##                    3rd Qu.:20.00
##                    Max.   :26.00
```

Resúmenes y tablas

Para seguir con el ejemplo, realizamos la importación tal cual aparece en la Figura 8. Una de las primeras funciones que debamos conocer es *summary*. Esta función no es específica del análisis estadístico ya que se adapta al objeto sobre el que las estemos aplicando. En este caso, al aplicarla sobre un conjunto de datos, lo que arroja son la distribución de frecuencias, en el caso de variables categóricas y los estadísticos descriptivos elementales. Vamos a verlo.

```
summary(data) # Simple, pero puede hacerse difícil de leer si hay muchas variables
```

```
##                                                EducationSector Individu
alProject
##  Engineering Sciences                                    :123    No : 95
##  Economic Sciences, Business Studies, Commerce and Law: 32    Yes:124
##  Art, Music or Design                                    : 21
##  Others                                                  : 20
##  Medicine, Health Sciences                               : 10
##  Humanities and Social Sciences                          :  5
##  (Other)                                                 :  8
##       Age           Gender     City     Influenced  Perseverance
##  Min.   :17.00   Female: 57   No : 36   No : 61    Min.   :1.000
##  1st Qu.:19.00   Male  :162   Yes:183   Yes:158    1st Qu.:3.000
##  Median :20.00                                     Median :3.000
##  Mean   :19.75                                     Mean   :3.352
##  3rd Qu.:20.00                                     3rd Qu.:4.000
##  Max.   :26.00                                     Max.   :5.000
##
##  DesireToTakeInitiative Competitiveness  SelfReliance   StrongNeedToAc
hieve
##  Min.   :1.000          Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:3.000          1st Qu.:3.000   1st Qu.:3.000   1st Qu.:3.000
##  Median :4.000          Median :4.000   Median :4.000   Median :4.000
##  Mean   :3.621          Mean   :3.589   Mean   :3.721   Mean   :3.909
##  3rd Qu.:5.000          3rd Qu.:4.500   3rd Qu.:5.000   3rd Qu.:5.000
##  Max.   :5.000          Max.   :5.000   Max.   :5.000   Max.   :5.000
##
##  SelfConfidence  GoodPhysicalHealth MentalDisorder      KeyTraits
##  Min.   :1.000   Min.   :1.000      No :155        Passion   :62
##  1st Qu.:3.000   1st Qu.:3.000      Yes: 64        Positivity:73
```

```
##  Median :4.000   Median :4.000                    Resilience:10
##  Mean   :3.575   Mean   :3.562                    Vision    :35
##  3rd Qu.:4.000   3rd Qu.:4.000                    Work Ethic:39
##  Max.   :5.000   Max.   :5.000
##
##
ReasonsForLack
##
:91
##  Just not interested! (Want to work in the corporate sector, or for th
e government or pursue research or something else):41
##  Academic Pressure
:11
##  Not willing to start a venture in India and waiting for future reloca
tion                                                :10
##  Lack of Knowledge
: 7
##  Academic Pressure, Unwillingness to take risk, Lack of Knowledge
: 4
##  (Other)
:55
##        y
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.4155
##  3rd Qu.:1.0000
##  Max.   :1.0000
##
```

```
summary(data$Age) # Una forma de centrarnos exclusivamente en una variabl
e.
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   17.00   19.00   20.00   19.75   20.00   26.00
```

```
summary(data[["Age"]]) # Otra forma con la que hacer lo mismo.
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   17.00   19.00   20.00   19.75   20.00   26.00
```

```
summary(data[2:4]) # Otra forma, esta vez empleando posiciones.
```

```
##  IndividualProject      Age           Gender
##  No : 95           Min.   :17.00   Female: 57
##  Yes:124           1st Qu.:19.00   Male  :162
##                    Median :20.00
##                    Mean   :19.75
##                    3rd Qu.:20.00
##                    Max.   :26.00
```

Otra posibilidad es crear tablas de recuento o de proporciones para las variables de interés, para lo que podemos usar la función *table*.

```
tabla1 <- table(data$KeyTraits, dnn = "Key Traits") # Creo una tabla agregándola al Environment
tabla1 # Recuerda que para ver el resultado tenemos que llamar al objeto
```

```
## Key Traits
##    Passion Positivity Resilience     Vision Work Ethic
##         62         73         10         35         39
```

```
prop.table(tabla1) # Tabla con las proporciones
```

```
## Key Traits
##    Passion Positivity Resilience     Vision Work Ethic
##  0.2831050  0.3333333  0.0456621  0.1598174  0.1780822
```

```
tabla2 <- table(data$Age) # También sería válido para datos cuantitativos
cumsum(tabla2) # Proporciona las frecuencias absolutas acumuladas
```

```
##  17  18  19  20  21  22  23  24  25  26
##   7  25  95 174 202 214 216 217 218 219
```

```
tabla3 <- table(data$Gender, data$KeyTraits) # Tabla de doble entrada
tabla3
```

```
##
##          Passion Positivity Resilience Vision Work Ethic
##   Female      19         22          3      4          9
##   Male        43         51          7     31         30
```

Seguramente utilizaremos los datos con algún propósito específico. Tal vez hayamos generado estas tablas para incluirlas en un informe, un trabajo académico o un artículo. También es posible redactar directamente en RStudio, como ocurre en este manual, aunque quizá todavía no estés tan familiarizado con R y RStudio y prefieras exportar los resultados de una manera más práctica que simplemente copiarlos y pegarlos. Lo haremos con la función `write` que veremos a continuación, pero antes veremos dos funciones que pueden ser de utilidad.

```
names(tabla1) # Muestra las modalidades de la variable que se reflejan en la tabla.
```

```
## [1] "Passion"    "Positivity" "Resilience" "Vision"     "Work Ethic"
```

```
names(tabla1) = c("Pasión", "Positividad", "Resiliencia", "Visión", "Trabajo Ético")
tabla1 # Ahora aparecen en español
```

```
##        Pasión   Positividad    Resiliencia        Visión Trabajo Ético
##            62            73            10            35            39
```

```
dimnames(tabla1) # Hace algo parecido y puede ser útil para definir tablas complejas
```

```
## [[1]]
## [1] "Pasión"        "Positividad"   "Resiliencia"    "Visión"
## [5] "Trabajo Ético"
```

```
names(dimnames(tabla1)) # La combinación muestra el nombre de la variable.
```

```
## NULL
```

```
                        # Podemos cambiar dicho nombre si no nos gusta.
names(dimnames(tabla1)) = "Rasgos clave"
tabla1
```

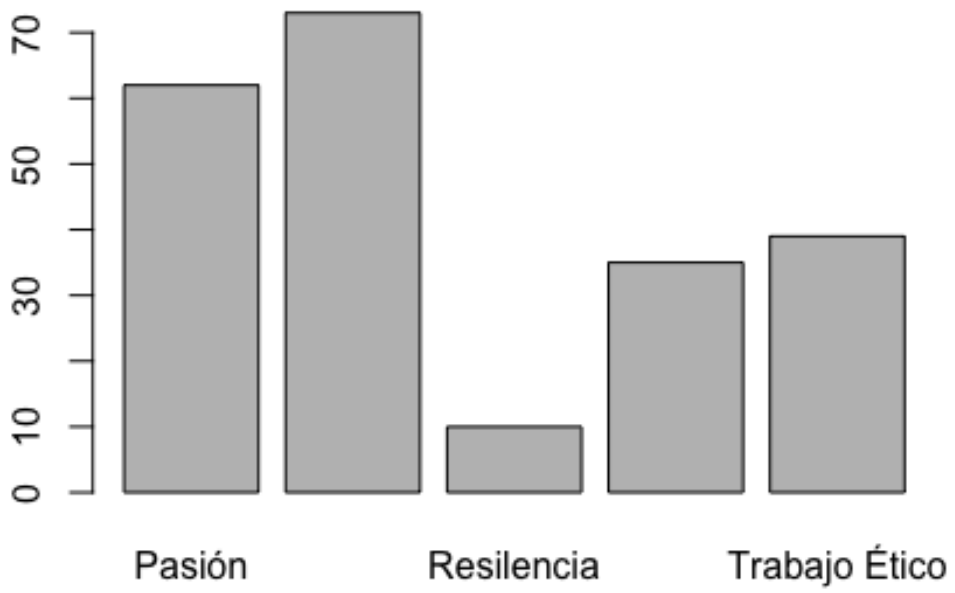
```
## Rasgos clave
##        Pasión   Positividad    Resiliencia        Visión Trabajo Ético
##            62            73            10            35            39
```

Ahora vamos a guardar nuestro archivo de tabla para incorporarlo posteriormente a nuestro trabajo.

Gráficos

Pero qué sería de la estadística sin un bonito gráfico. Existen multitud de paquetes para hacer gráficos preciosos en R. En este curso nos quedamos con los más simples. Al igual que sucedía con las tablas y otros objetos, podemos ejecutarlos directamente o guardarlos en el entorno para ser llamados posteriormente. A continuación, vamos a generar un gráfico de barras mediante la función `barplot` tanto por ejecución directa como cambiando aspectos estéticos y asignándole un nombre. Las posibilidades de personalización son enormes.

```
barplot(tabla1) # Este es el gráfico por defecto
```



Ejemplo 1 de diagrama de barras

```
plot1 <- barplot(tabla1, col = c("green","blue","red","orange", "lightpin
k"),
                 main = "Gráfico de ejemplo")
```

Ejemplo 2 de diagrama de barras

```
plot1

##      [,1]
## [1,]  0.7
## [2,]  1.9
## [3,]  3.1
## [4,]  4.3
## [5,]  5.5
```

Vamos a exportar este gráfico para incorporarlo a nuestro trabajo. El gráfico debería aparecer en la pestaña plot del panel de exploración por lo que solo hay que hacer clic derecho sobre él y guardarlo como queramos. También podemos usar el icono de Export que aparece en la pestaña.

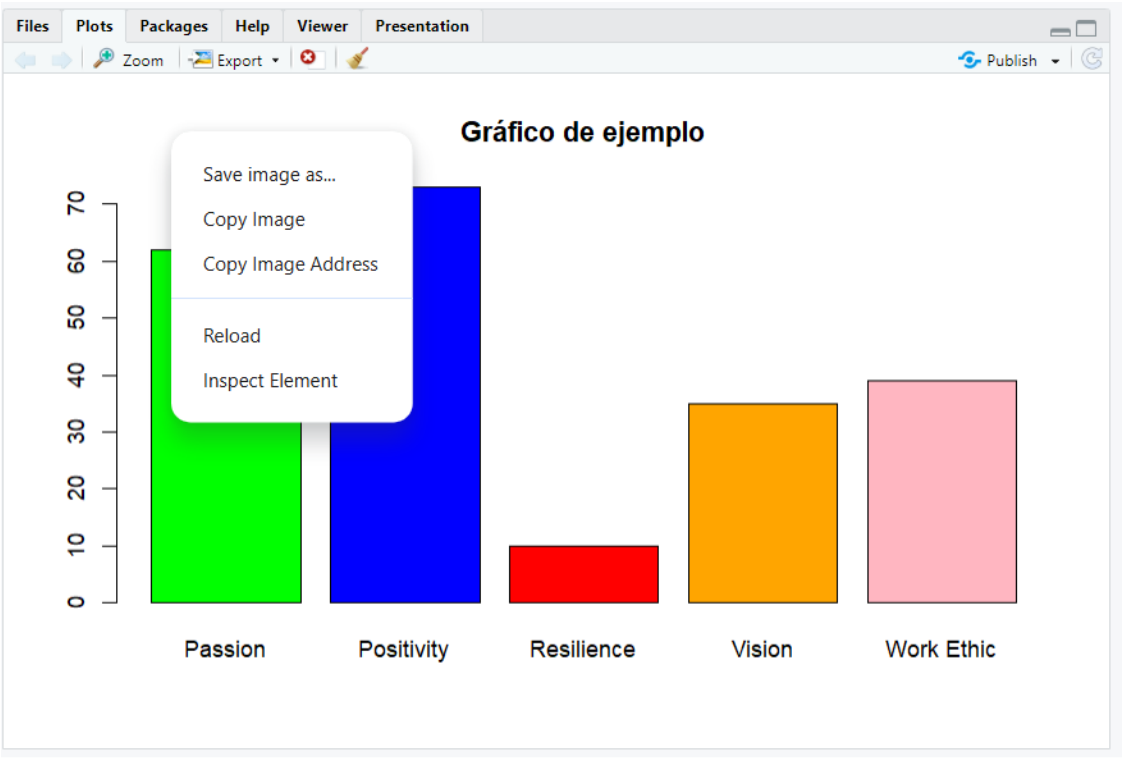
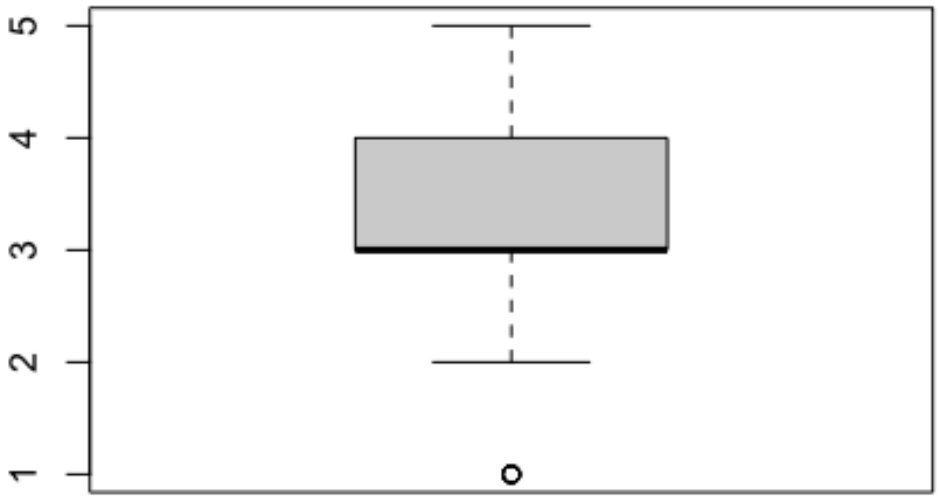


Figura 12: Exportar gráfico mediante clic derecho

Otro gráfico muy usado es el gráfico de cajas, cuya utilidad es bien conocida para detectar *outliers*.

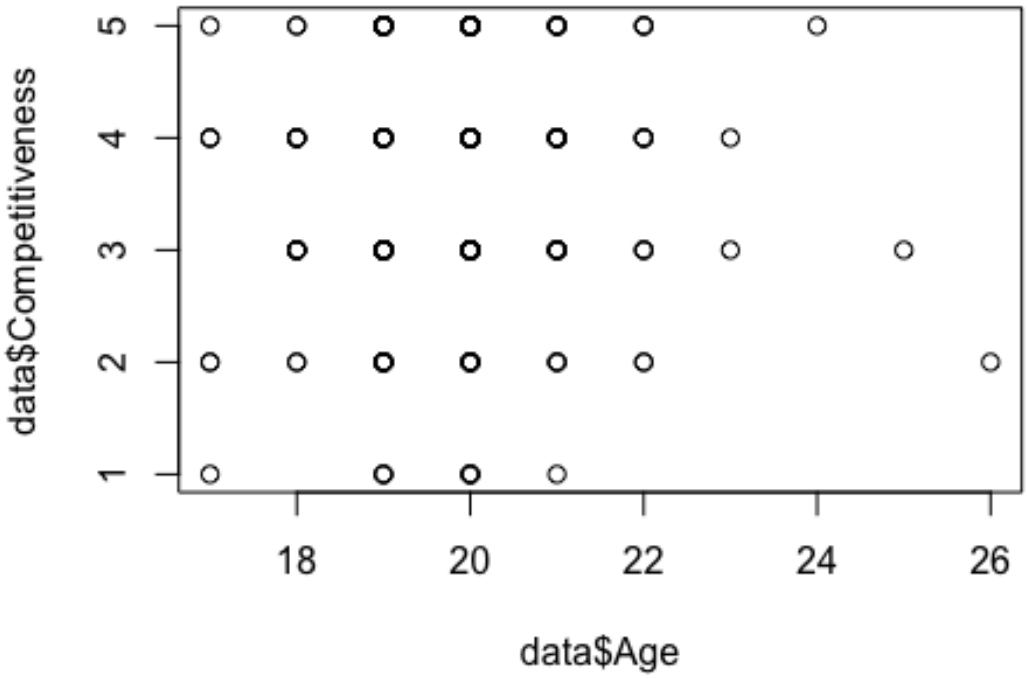
```
boxplot(data$Perseverance)
```



Ejemplo de diagrama de cajas

Para gráficos bivariados podemos sugerir dos de los más comunes. Por un lado, la clásica nube de puntos aplicable sobre variables cuantitativas.

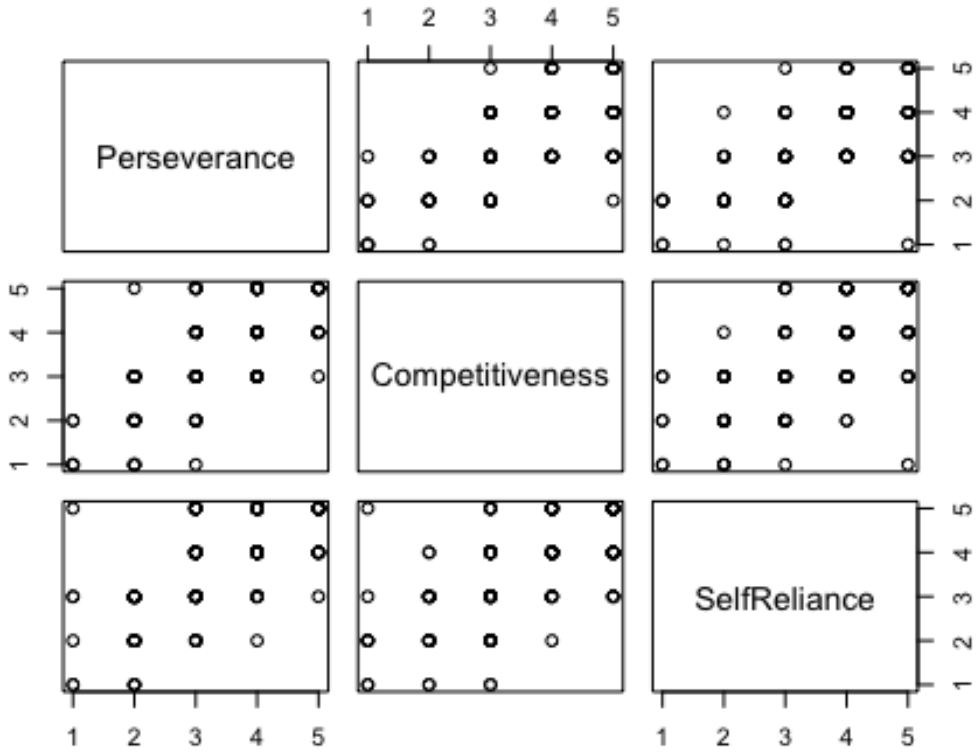
```
plot(data$Age, # La función plot tiene muchos usos
     data$Competitiveness, # sirve tanto para una como dos variables
     type="p") # En type = "p" sirve para indicar que use puntos.
```



Ejemplo de nube de puntos

También podríamos usar la función `pairs`, la cual ofrece muchas posibilidades. Te sugiero que visites la ayuda para ver algunos ejemplos más avanzados sobre lo que se puede hacer.

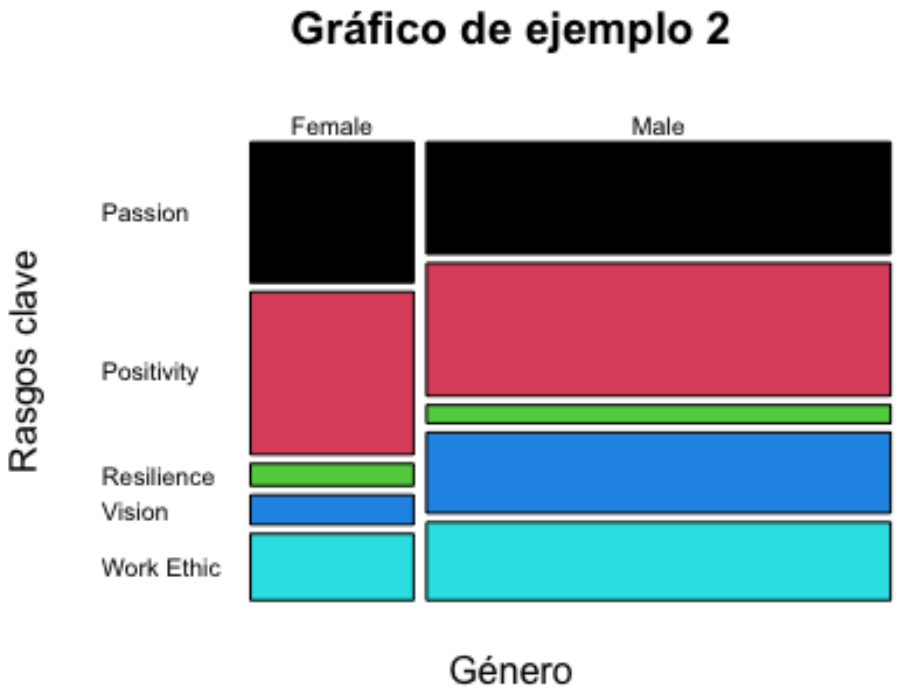
```
pairs(~Perseverance+Competitiveness+SelfReliance, data = data)
```



Ejemplo simple con pairs

Para variables cualitativas, uno de los más utilizados es el mosaico.

```
mosaicplot(tabla3, color = 1:5, las = 1, # Añadimos argumentos de persona
lización
           main = "Gráfico de ejemplo 2", # Título del gráfico
           xlab = "Género", # Etiqueta del eje x
           ylab = "Rasgos clave") # Etiqueta del eje y
```



Ejemplo de gráfico de mosaico

Medidas descriptivas

Existen múltiples medidas descriptivas, cada una con un propósito, unas ventajas y unos inconvenientes. La forma más común de clasificarlas es mediante su funcionalidad. Así podemos encontrar, medidas de posición, dispersión, forma, concentración o asociación. En este bloque, vamos a ver algunos ejemplos de ellas.

Medidas de posición

Intentan resumir los datos en un único valor que los representa.

```
mean(data$Age) # La media
```

```
## [1] 19.75342
```

```r
median(data$Age) # La mediana
```

```
## [1] 20
```

```r
quantile(data$Age) # Proporciona los cuartiles
```

```
##   0%  25%  50%  75% 100%
##   17   19   20   20   26
```

```r
quantile(data$Age, probs = 0.3) # Con el argumento probs forzamos uno o v
arios percentiles específicos. Por ejemplo, el percentil 30
```

```
## 30%
##  19
```

La moda, no tiene una función propia en los paquetes básicos integrados en R. Pero es un buen ejemplo para pensar en cómo funciona el lenguaje de R.

```r
moda1 <- names(sort(tabla1, decreasing = T))[1]
moda1
```

```
## [1] "Positividad"
```

El primer ejemplo crea el objeto *moda1*, el cual ordena la tabla de forma decreciente y proporciona el nombre de la primera fila, es decir el nombre de la modalidad de la variable con mayor frecuencia. Aunque válida para este ejemplo, podría presentar algunos problemas en los casos con más de una moda o en variables cuantitativas. Otra posibilidad podría ser el contemplado en la función creada *moda*.

```r
moda <- function(x)  names(x[which.max(x)])
moda2 <- moda(tabla1)
```

Aquí se ha aprovechado para crear una función y luego aplicarla en el objeto *tabla1*. Ambas opciones están pensadas para aplicarse a un objeto de tipo tabla, aunque también podríamos buscar una manera de utilizarlas directamente sobre la base de datos.

Cuando la comunidad crea funciones, estas se ponen a prueba con el objetivo de evaluar cuál resulta mejor: la que ofrece más garantías, la que se ejecuta con mayor eficiencia o la que es más sencilla de usar.

Medidas de dispersión

Proporciona una medida sobre la variabilidad de los datos.

```
var(data$Age)          # Varianza

## [1] 1.663692

sd(data$Age)           # Desviación estándar

## [1] 1.289842

range(data$Age)        # Rango (mínimo y máximo)

## [1] 17 26

IQR(data$Age)          # Rango intercuartílico

## [1] 1
```

Medidas de asociación

Tratan de medir la relación entre dos variables.

```
cov(data$Perseverance, data$DesireToTakeInitiative)      # Covarianza

## [1] 0.7393699

cor(data$Perseverance, data$DesireToTakeInitiative)      # Correlación de
Pearson

## [1] 0.6446263

cor(data$Perseverance, data$DesireToTakeInitiative, method = "spearman")
# Correlación de Spearman

## [1] 0.6306326
```

Regresión

Modelo que explica la relación lineal entre dos variables cuantitativas.

```
modelo <- lm(DesireToTakeInitiative~Age, data=data)
# Modelo y~x La y se explica en función de la x
# La función es mucho más completa y personalizable.
modelo # Llamamos a un objeto modelo

##
## Call:
```

```
## lm(formula = DesireToTakeInitiative ~ Age, data = data)
##
## Coefficients:
## (Intercept)          Age
##     4.68120     -0.05367
```

```
class(modelo) # Como vemos la clase del objeto modelo es lm
```

```
## [1] "lm"
```

```
summary(modelo) # Recuperamos summary. Aplicado a un objeto distinto gene
ra resultados distintos.
```

```
##
## Call:
## lm(formula = DesireToTakeInitiative ~ Age, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7151 -0.6614  0.3386  1.2580  1.6069
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.68120    1.19850   3.906 0.000125 ***
## Age         -0.05367    0.06054  -0.886 0.376345
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.153 on 217 degrees of freedom
## Multiple R-squared:  0.003608,   Adjusted R-squared:  -0.0009834
## F-statistic: 0.7858 on 1 and 217 DF,  p-value: 0.3763
```

Conclusión

El propósito de esta guía ha sido servir como un primer acercamiento a R y a su entorno de trabajo, RStudio. A través de ejemplos sencillos y ejercicios prácticos, el lector ha podido familiarizarse con la lógica del lenguaje, su sintaxis y las estructuras de datos fundamentales que constituyen la base para cualquier análisis. El recorrido planteado no pretende abarcar la amplitud del lenguaje, sino ofrecer un punto de partida claro y accesible que permita ganar confianza en el uso de R. Esta guía llega a su fin, pero el viaje con R no ha hecho más que empezar. Lo que comienza como ejercicios

simples puede convertirse en una puerta hacia el análisis riguroso y la exploración creativa de datos. Con dedicación y curiosidad, R dejará de ser una herramienta desconocida para convertirse en un aliado indispensable en el trabajo académico, profesional o personal.

Comenzar a programar en R puede parecer desafiante, pero con práctica constante y exploración progresiva, se transforma en una experiencia enriquecedora constituye una base sobre el cual construir conocimientos más avanzados. A medida que se adquiera fluidez, el siguiente paso natural será enfrentarse a conjuntos de datos propios, importarlos y analizarlos con las herramientas aprendidas, para después avanzar hacia técnicas más complejas y especializadas. A partir de aquí, el siguiente paso será profundizar en el uso de librerías externas que expanden enormemente las posibilidades de análisis y visualización.

Más allá de la práctica, es fundamental desarrollar la capacidad de formular preguntas y de traducirlas en análisis concretos dentro de R. El valor de dominar este lenguaje no radica únicamente en conocer sus funciones, sino en comprender cómo aplicarlas para resolver problemas reales. En este sentido, la guía busca no solo enseñar comandos, sino también orientar hacia una forma de pensar en términos de datos, relaciones y modelos.

R es una herramienta viva, respaldada por una comunidad amplia y en constante desarrollo. Explorar sus posibilidades abre el camino a campos tan diversos como la estadística avanzada, la visualización de datos, la ciencia de datos o el aprendizaje automático. Esta guía aspira a ser una invitación a seguir aprendiendo, a profundizar en nuevas librerías, a experimentar con proyectos propios y a descubrir todo el potencial que ofrece este lenguaje. Un servidor se despide, esperando haber cumplido tales propósitos y deseándole suerte en su viaje.

Ejercicios de Práctica

1. Se dispone de la siguiente información relativa a las edades de un grupo de personas.

   25 30 18 40 22 35 65 53 49

   1.1. Crea un vector llamado edad y ordénalo de forma creciente.

   1.2. Calcula la media.

   1.3. Calcula el máximo.

2. Crea un vector x con los siguientes valores: 1, 3, 5, 7, 9 y otro llamado y, cuyos valores sean 2, 4, 6, 8, 10.

   2.1. Suma ambos vectores

   2.2. Multiplica los valores de x e y elemento a elemento.

   2.3. Multiplica el traspuesto del vector x por el vector y de forma que quede un escalar.

3. Crea una matriz llamada m1 con los números del 1 al 12 y organízala en 3 filas y 4 columnas.

   3.1. Multiplica la matriz anterior por 2 y guárdala como un objeto llamado m2.

   3.2. Crea una matriz llamada m3 con únicamente dos primeras filas y las tres primeras columnas de m2.

4. A continuación, se presenta una tabla con las calificaciones de tres alumnos en diferentes asignaturas.

	Biología	Matemáticas	Historia	Inglés
Pilar	7.5	5.2	6.5	7.9
Ana	8.1	9.2	8.2	9
José	5.8	6.4	9.2	8.5

   4.1. Crea una matriz llamada calificaciones con la estructura de la tabla.

   4.2. Calcula la media para cada asignatura.

   4.3. Crea una lista con la información y filtra las calificaciones solo para Pilar y Ana.

5. Usa la base de datos `mtcars` integrada en R.

5.1. Abre la ayuda de la base de datos. ¿De dónde fueron extraídos los datos?

5.2. Crea una tabla de doble entrada con el tipo de motor (línea o en forma de V) y el tipo de cambio (manual o automático).

5.3. Filtra los coches con más de 100cv.

5.4. ¿Cuál sería el modelo como mayor peso?

6. Usa la base de datos integrada `iris`, integrada en R.

6.1. Calcula la media y la desviación típica de la variable `Sepal.Length`

6.2. Representa un gráfico simple que relacione `Sepal.Length` y `Petal.Length`

SOLUCIONES PARA LOS EJERCICIOS

Ejercicio 1: Operaciones con vectores

1.1. Crea un vector llamado edad y ordénalo de forma creciente.

```
edad <- c(25, 30, 18, 40, 22, 35, 65, 53, 49)
edad_ordenada <- sort(edad)
edad_ordenada

## [1] 18 22 25 30 35 40 49 53 65
```

1.2. Calcula la media.

```
media_edad <- mean(edad)
media_edad

## [1] 37.44444
```

1.3. Calcula el máximo.

```
maximo_edad <- max(edad)
maximo_edad

## [1] 65
```

Ejercicio 2: Operaciones con vectores

2.1. Suma ambos vectores

```
x <- c(1, 3, 5, 7, 9)
y <- c(2, 4, 6, 8, 10)

suma <- x + y
suma

## [1]  3  7 11 15 19
```

2.2. Multiplica los valores de x e y elemento a elemento.

```
multiplicacion_elemento <- x * y
multiplicacion_elemento
```

```
## [1]  2 12 30 56 90
```

2.3. Multiplica el traspuesto del vector x por el vector y de forma que quede un escalar.

```
escalar <- t(x) %*% y
escalar

##      [,1]
## [1,]  190

# Resultado: 165 (1*2 + 3*4 + 5*6 + 7*8 + 9*10)
```

Ejercicio 3: Matrices

3.1. Crea una matriz llamada m1 con los números del 1 al 12 organizados en 3 filas y 4 columnas.

```
m1 <- matrix(1:12, nrow = 3, ncol = 4)
m1

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

3.2. Multiplica la matriz anterior por 2 y guárdala como un objeto llamado m2.

```
m2 <- m1 * 2
m2

##      [,1] [,2] [,3] [,4]
## [1,]    2    8   14   20
## [2,]    4   10   16   22
## [3,]    6   12   18   24
```

3.3. Crea una matriz llamada m3 con las dos primeras filas y las tres primeras columnas de m2.

```
m3 <- m2[1:2, 1:3]
m3
```

```
##      [,1] [,2] [,3]
## [1,]    2    8   14
## [2,]    4   10   16
```

Ejercicio 4: Matriz de calificaciones

4.1. Crea una matriz llamada calificaciones con la estructura de la tabla.

```
calificaciones <- matrix(
  c(7.5, 5.2, 6.5, 7.9,
    8.1, 9.2, 8.2, 9.0,
    5.8, 6.4, 9.2, 8.5),
  nrow = 3,
  ncol = 4,
  byrow = TRUE,
  dimnames = list(
    c("Pilar", "Ana", "José"),
    c("Biología", "Matemáticas", "Historia", "Inglés")
  )
)
calificaciones
```

```
##       Biología Matemáticas Historia Inglés
## Pilar      7.5         5.2      6.5    7.9
## Ana        8.1         9.2      8.2    9.0
## José       5.8         6.4      9.2    8.5
```

4.2. Calcula la media para cada asignatura.

```
media_asignaturas <- colMeans(calificaciones)
media_asignaturas
```

```
##    Biología Matemáticas    Historia      Inglés
##    7.133333    6.933333    7.966667    8.466667
```

4.3. Crea una lista con la información y filtra las calificaciones solo para Pilar y Ana

```
lista_calificaciones <- list(
  matriz_completa = calificaciones,
  pilar_ana = calificaciones[c("Pilar", "Ana"), ]
```

```
)

lista_calificaciones$pilar_ana
```

```
##       Biología Matemáticas Historia Inglés
## Pilar      7.5         5.2      6.5    7.9
## Ana        8.1         9.2      8.2    9.0
```

Ejercicio 5: Base de datos mtcars

5.1. Abre la ayuda de la base de datos mtcars.

Puede utilizarse `?mtcars` o `help(mtcars)`. Esto abrirá la documentación en la pestaña Help donde podremos leer:

> *"The data was extracted from the 1974 Motor Trend US magazine"*

5.2. Crea una tabla de doble entrada con el tipo de motor (línea o en forma de V) y el tipo de cambio (manual o automático).

```
tabla_cruzada <- table(mtcars$vs, mtcars$am)
dimnames(tabla_cruzada) <- list(
  c("Motor en V", "Motor en línea"),
  c("Automático", "Manual")
)
tabla_cruzada
```

```
##                Automático Manual
## Motor en V             12      6
## Motor en línea          7      7
```

5.3. Filtra los coches con más de 100cv.

```
coches_100cv <- subset(mtcars, hp > 100)
coches_100cv
```

```
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear car
b
## Mazda RX4          21.0   6 160.0 110 3.90 2.620 16.46  0  1    4
4
## Mazda RX4 Wag      21.0   6 160.0 110 3.90 2.875 17.02  0  1    4
4
```

```
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3
1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3
2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3
1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3
4
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4
4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4
4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3
3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3
3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3
3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3
4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3
4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3
4
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3
2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3
2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3
4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3
2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5
2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5
4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5
6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5
```

```
8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4
2

# También podría filtrase como:
coches_100cv_v2 <- mtcars[mtcars$hp > 100, ]
coches_100cv_v2

##                      mpg cyl  disp  hp drat    wt  qsec vs am gear car
b
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4
4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4
4
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3
1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3
2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3
1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3
4
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4
4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4
4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3
3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3
3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3
3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3
4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3
4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3
4
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3
2
```

```
## AMC Javelin          15.2   8 304.0 150 3.15 3.435 17.30  0  0    3
2
## Camaro Z28           13.3   8 350.0 245 3.73 3.840 15.41  0  0    3
4
## Pontiac Firebird     19.2   8 400.0 175 3.08 3.845 17.05  0  0    3
2
## Lotus Europa         30.4   4  95.1 113 3.77 1.513 16.90  1  1    5
2
## Ford Pantera L       15.8   8 351.0 264 4.22 3.170 14.50  0  1    5
4
## Ferrari Dino         19.7   6 145.0 175 3.62 2.770 15.50  0  1    5
6
## Maserati Bora        15.0   8 301.0 335 3.54 3.570 14.60  0  1    5
8
## Volvo 142E           21.4   4 121.0 109 4.11 2.780 18.60  1  1    4
2
```

5.4. ¿Cuál sería el modelo con mayor peso?

```
modelo_mayor_peso <- rownames(subset(mtcars, wt == max(mtcars$wt)))
modelo_mayor_peso

## [1] "Lincoln Continental"

# También podría filtrase como:
modelo_mayor_peso_v2 <- rownames(mtcars)[which.max(mtcars$wt)]
modelo_mayor_peso_v2

## [1] "Lincoln Continental"
```

Ejercicio 6: Base de datos iris

6.1. Calcula la media y la desviación típica de la variable ***Sepal.Length***

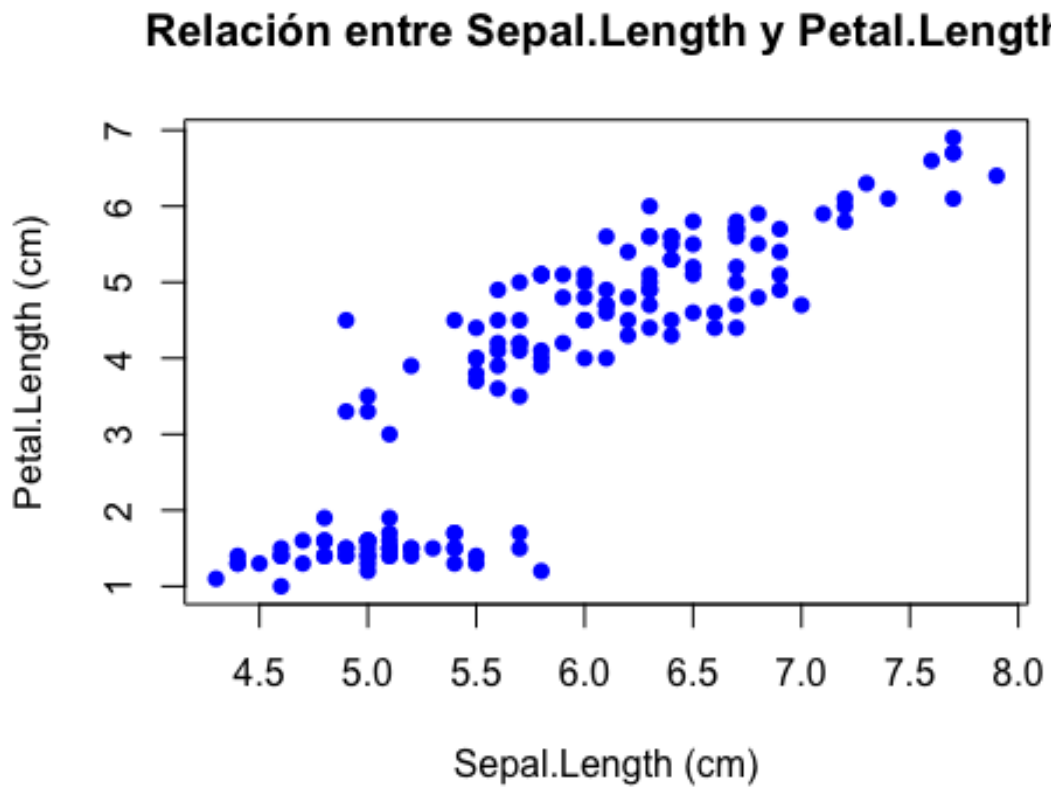
```
media_sepal <- mean(iris$Sepal.Length)
desv_sepal <- sd(iris$Sepal.Length)

media_sepal

## [1] 5.843333
```

```
desv_sepal
```

```
## [1] 0.8280661
```

6.2. Representa un gráfico simple que relacione ***Sepal.Length*** y ***Petal.Length***

```
plot(iris$Sepal.Length, iris$Petal.Length,
     main = "Relación entre Sepal.Length y Petal.Length",
     xlab = "Sepal.Length (cm)",
     ylab = "Petal.Length (cm)",
     col = "blue",
     pch = 16)
```



.