# R STUDIO: MANUAL DEL BUEN PROGRAMADOR

¿Quieres ser eficiente y sentirte más cómodo programando? Quédate aquí que te enseñamos. Desde cambios en la configuración de Rstudio hasta una sección de preguntas y respuestas, leyendo esto, te darás cuenta de que hay vida más allá de las sentencias básicas y las funciones matemáticas. Bienvenido al manual del buen programador:

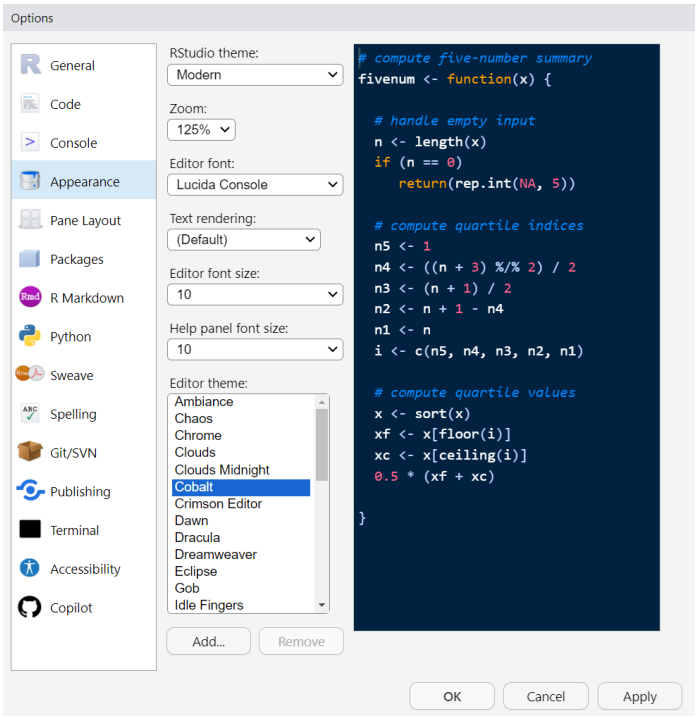
**ÍNDICE**



## ASPECTO DE RSTUDIO

Puede parecer una nimiedad, pero el aspecto de Rstudio influye en que te sientas más cómodo programando. Tal vez prefieres un aspecto más oscuro para que no se te canse la vista, o resaltar con colores más llamativos los tipos de elementos en R. En definitiva, te recomendamos que le eches un vistazo a las opciones que ofrece Rstudio y te enseñamos cómo hacerlo con una serie de pasos:

1. Dirígete a *Tools* en el menú principal.
2. *Global options*.
3. *Appearance*.
4. Una vez ahí podrás elegir de entre las opciones que se muestran en la imagen. En concreto, te recomendamos navegar por *Editor theme*.




## SOBRE DIRECTORIOS

Un directorio es una carpeta o ubicación específica en el sistema de archivos del ordenador. Aquí se almacenan tus archivos, incluidos los códigos de R.

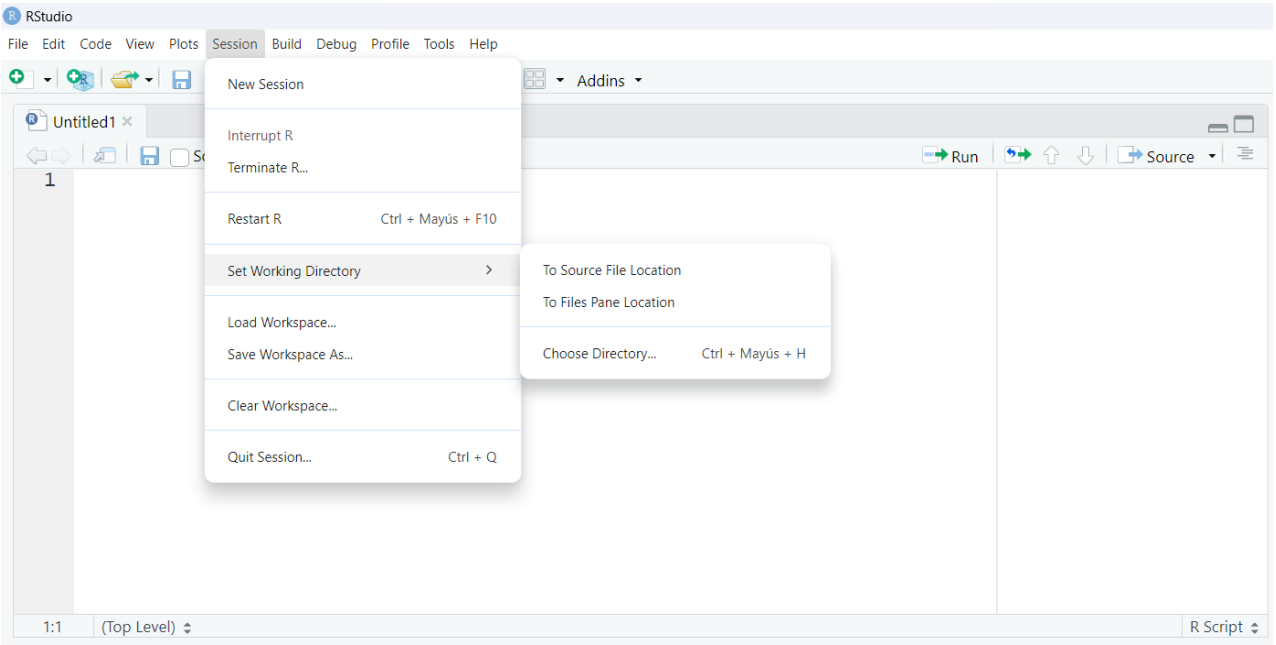
En primer lugar, te recomendamos mantener tus carpetas del ordenador organizadas. Esto facilita en gran medida el manejo de directorios y la tarea de encontrar códigos que

quieras utilizar. Puedes organizar tus códigos en carpetas según el nivel de dificultad, la tarea que realicen (interpolación derivación, integración...) o por fecha, tú decides.

A continuación, te explicamos cómo cambiar un directorio de trabajo para evitar los quebraderos de cabeza que surgen cuando no te encuentras en el adecuado.

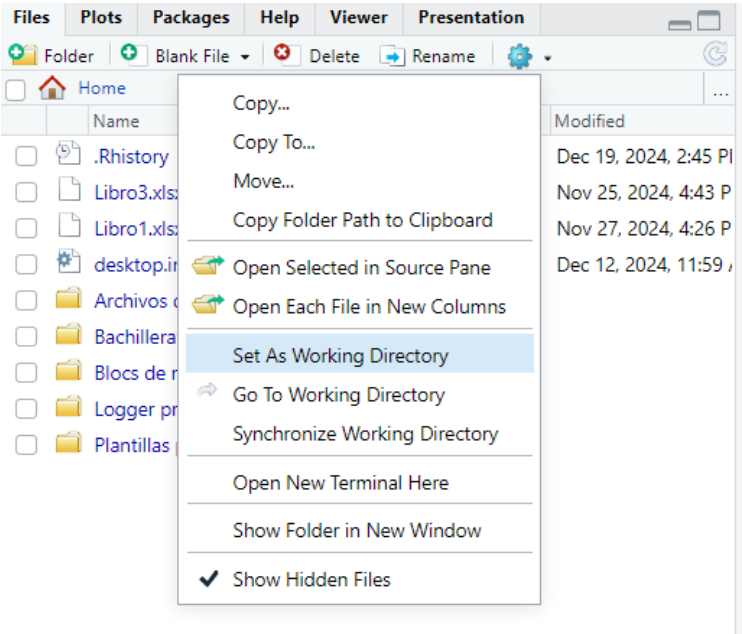
**Primera forma**

Dirígete a session, set working directory y choose directory. De esta manera, llegarás a una ventana que te permitirá elegir entre tus archivos.




**Segunda forma:**

En la sección de files navega hasta alcanzar el directorio en el que deseas trabajar. A continuación, haz clic en el engranaje y elige set as working directory.



**Tercera forma**:

Existen dos comandos extremadamente útiles para manejar los directorios de trabajo.

- **getwd ()**: Muestra el directorio de trabajo actual.
- **setwd("ruta/al/directorio")**: Cambia el directorio de trabajo al especificado.

**SI QUIERES HACER LAS MEJORES GRÁFICAS**

Graficar funciones es una herramienta muy valiosa para analizar el comportamiento de ciertas variables. Por ello, es importante representarlas de manera clara y visualmente apelativa. A continuación, te mostramos cómo incluyendo ejemplos de códigos de R generados por *copilot*.

**Gráfico de dispersión**

- **plot (x, y)**: Gráfico de dispersión con los valores en x e y.
- **Ejemplo:**

```
x <- 1:10

y <- x^2

plot(x, y, main = "Gráfico de Dispersión", xlab = "Eje X", ylab = "Eje Y")
```

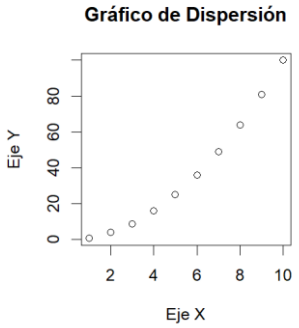



**Gráfico de barras**

- **barplot (x)**: Gráfico de barras con las alturas especificadas en x (barras separadas por espacios).
- **Ejemplo:**

```
counts <- table(mtcars$cyl)

barplot(counts, main = "Distribución de Cilindros", xlab = "Número de Cilindros", col = "lightblue")
```

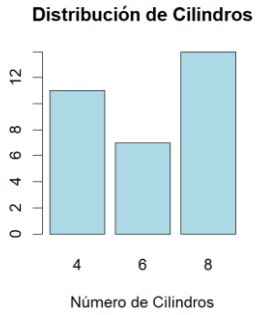
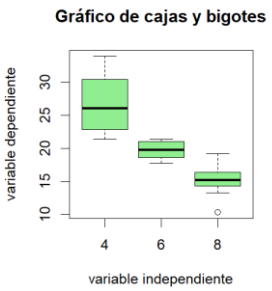


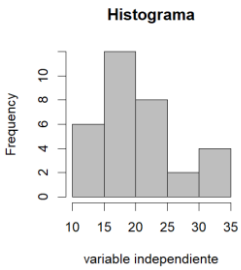

**Gráfico de cajas y bigotes**

- **boxplot (y ~ x, data = data)**: Gráfico de cajas y bigotes que compara la distribución de y a través de los niveles de x.
- **Ejemplo:**

```
boxplot(mpg ~ cyl, data = mtcars, main = "Gráfico de cajas y bigotes", xlab = "variable independiente", ylab = "variable dependiente", col = "lightgreen")
```



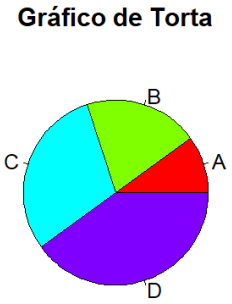
### Histogramas




- **hist(x)**: Histograma con los valores de x (barras juntas).
- **Ejemplo:**

```
hist(mtcars$mpg, main = "Histograma de MPG", xlab = "Millas por Galón", col = "gray")
```

### Gráfico de torta




- **pie(x)**: Pie chart con las proporciones especificadas en x.
- **Ejemplo:**

```
slices <- c(10, 20, 30, 40)

labels <- c("A", "B", "C", "D")

pie(slices, labels = labels, main = "Gráfico de Torta", col = rainbow(length(slices)))
```

### Configuración visual

- **type** =
  - "p": Puntos (default)
  - "l": Líneas
  - "b": Puntos y líneas
  - "c": Líneas sin puntos
  - "o": Puntos y líneas superpuestas
- **col=** Color de los elementos del gráfico (debes introducirlos en inglés).
- **lwd=** Grosor de las líneas
- **xlab= “”**: Etiqueta del eje X.
- **ylab= “”**: Etiqueta del eje Y.
- **main=** Título del gráfico
- **xlim**= Límite del eje X.
- **ylim**= Límite del eje Y.
- **legend =** Añadir una leyenda

### ATAJOS CON EL TECLADO

Te dejamos una serie de atajos que realizar con el teclado para que no parezcas una tortuga programando. Desde los más básicos, que comparten otros entornos digitales hasta los más específicos de R. Si bien, aquí se encuentran los que encontramos más útiles, puedes acceder a todos los que hay tecleando **Alt + Shift + K.**

#### Escritura y Edición

- **Alt -:** Para escribir una flecha (<-).
- **Ctrl + S:** Guarda el archivo actual.

- **Ctrl + Z:** Deshace la última acción.
- **Ctrl + Y:** Rehace la última acción deshecha.
- **Ctrl + D:** Elimina la línea actual.
- **Ctrl + Shift + N:** Crea un nuevo script.

**Ejecución**

- **Ctrl + Enter:** Ejecuta la línea de código donde se tenga el cursor o la parte seleccionada.
- **Ctrl + Shift + Enter:** Ejecuta todo el código del archivo.

**Navegación**

- **Ctrl + 1:** Cambia al script de código.
- **Ctrl + 2:** Cambia a la consola.
- **Ctrl +4:** Mostrar historial
- **Cursor:** al mover las flechas del teclado en sentido ascendente/descendente puedes navergar por las funciones ya definidas para hacer uso de ellas.

**Vista**

- **Ctrl + +:** Amplía el tamaño de la fuente.
- **Ctrl + -:** Reduce el tamaño de la fuente.

**Sesión**

- **Ctrl + Alt + R:** Reinicia la sesión R (limpia el entorno).

## R ME PERMITE UTILIZAR RETURN, SOLO UNA FUNCIÓN INTERNA O AMBAS ¿QUÉ HAGO?

Después de ver unos cuantos códigos, probablemente te hayas dado cuenta de que la gente parece utilizar estas opciones a su antojo para devolver el valor de una función. Si en algún momento esto te ha generado confusión, tranquilo, te ofrecemos una solución con ayuda de *copilot*:

**Uso implícito (solo argumento de la función)**

Cuando la lógica de tu función es sencilla y no necesitas devolver un resultado explícito.

```r
suma <- function(a, b) {
  a + b
}
print(suma(3, 5))  # Devuelve 8
```

**Uso explícito (solo return)**

Cuando quieres devolver un valor y finalizar la ejecución de la función en un punto específico. Es útil para funciones con lógica condicional.

```r
mi_funcion <- function(x) {
  if (x < 0) {
    return("Negativo")
  } else {
    return("Positivo o Cero")
  }
}
print(mi_funcion(-2))  # Devuelve "Negativo"
```

**Ambas**

Cuando quieres encapsular lógica en funciones internas para modularidad o reutilización, y además devolver un resultado explícito.

```r
mi_funcion <- function(name) {
  # Función interna para generar el saludo
  greet <- function(person_name) {
    return(paste("Hola,", person_name, "!"))
  }
  # Llamada a la función interna
  message <- greet(name)
  return(message)
}

# Llamada a la función externa e impresión del resultado
print(mi_funcion("Ana"))
```

**Conclusión**

Si bien es cierto que *copilot* nos ofrece una serie de recomendaciones acertada, nuestra recomendación es que utilices ambas para ir sobre seguro. Sin embargo, que no te extrañe si ves otras opciones y se consciente de que para R, no es siempre necesario utilizar return al final de una función, ya que devuelve automáticamente el último valor evaluado.

**¿CUÁNDO DEBO USAR STOP Y WARNING Y CON QUÉ FIN?**

Ambas funciones se utilizan para manejar y reportar condiciones específicas durante la ejecución de un script. En concreto condiciones no esperadas o no deseadas. A continuación, te mostramos lo que hace cada una específicamente:

- **stop ()**: Genera un mensaje de error y detiene la ejecución del código (+ if).
- **warning ()**: Genera un mensaje de advertencia que informa al usuario sobre un posible problema, pero no detiene la ejecución del código (+ if).

Para que entiendas mejor su funcionamiento, le hemos pedido ayuda a *copilot*, que ha generado un código en el que se pone de manifiesto su utilidad.

```r
# Función que utiliza stop, warning y message
mi_funcion <- function(x) {
  if (x < 0) {
    stop("El valor debe ser positivo")
  } else if (x == 0) {
    warning("El valor es cero, esto puede no ser deseado")
  } else {
    message("El valor es positivo y mayor que cero")
  }
  return(x)
}

# Llamada a la función con diferentes valores
mi_funcion(-1)  # Genera un error y detiene la ejecución
mi_funcion(0)   # Genera una advertencia pero continúa
mi_funcion(1)   # Genera un mensaje informativo
```

**¿POR QUÉ ME HA DADO ERROR?**

Una gran ventaja de Rstudio es que te indica dónde se encuentra tu error y la razón por la que falla el código en ese lugar. Sin embargo, en ocasiones las indicaciones que realiza pueden resultar confusas.

En primer lugar, sin leer tu código, estamos casi seguros de que tu error ha sido por alguna de las razones que vamos a mostrar a continuación. Te recomendamos seguir esta serie de pasos para ir confirmando que no se debe a ninguna de ellas.

1. Los temidos **paréntesis**. Comprueba que todos los paréntesis que estén abiertos también estén cerrados. Esto se aplica a otras estructuras como los corchetes, las comillas o las llaves. Además, como consejo, mejor que sobren paréntesis a que falten.
2. El **nombre que asignaste a tus variables y funciones**. Es increíble pero cierto el hecho de que muchos errores se deben a no recordar correctamente el nombre que elegiste. Si estás cansado o nervioso, la probabilidad de que te equivoques con esto incrementa mucho. Por eso, presta atención al nombre que tenía la variable o función que definiste anteriormente. Fíjate en las mayúsculas, en la abreviación que utilizaste o en si tenía un guion entre otras cosas.
3. **Espacios en blanco**. Digamos que R es muy sensible y le afectan mucho las cosas pequeñas. Por eso, puede que sea un simple espacio en blanco innecesario o inesperado lo que haya causado que no pueda ejecutar tu código
4. No es exactamente un error pero asegúrate de incluir **cat ("\014")** y **rm(list =ls ())** al principio de tu código. Esto te asegurará un entorno limpio y evitará confundir a R con elementos de otros códigos.

A continuación, te explicamos lo que quiere decir R con sus errores más comunes:

- **Unexpected symbol**: Esto quiere decir que hay un símbolo inesperado en el código, generalmente por un error de sintaxis. Comprueba si se trata de una coma, paréntesis o valor numérico donde no procede entre otras cosas.
- **Object "xyz" not found**: R no encuentra el objeto al que intentas acceder, Por ello, asegúrate de que el objeto ha sido creado previamente y que no hay errores tipográficos en su nombre.
- **Subscript out of bounds**: Estás intentando acceder a un índice de un vector, matriz u otro objeto que está fuera de su rango. Verifica los índices y asegúrate de que están dentro del rango del objeto.
- **Error in if (condition) { : argument is not interpretable as logical**: La condición en la sentencia if no es una expresión lógica (TRUE o FALSE). Asegúrate de que lo sea.
- **Warning: NAs introduced by coercion**: Se han introducido valores NA ("not available") en el proceso de conversión de tipos de datos. Esto puede pasar cuando intentas convertir cadenas de texto que no se pueden interpretar como números a un tipo numérico.