

PrimerosPasosEnR

Escuela de datos - Fundación SOL

miércoles, 23 oct , 2024

Tabla de contenidos

Inicio	5
Una breve guía	5
1 Para comenzar	6
1.1 Instalar R	6
2 Un IDE	7
3 Conocer sobre las versiones	8
4 Tipos de datos	9
4.1 Números	9
4.2 Cadena de texto	9
4.3 Valores lógicos	10
4.4 Vectores	10
4.5 Coerción	11
4.5.1 1. Coerción de Logical a Integer	12
4.5.2 2. Coerción de Integer a Numeric	12
4.5.3 3. Coerción de Numeric a Character	12
4.5.4 4. Coerción de diferentes tipos en una lista	13
4.6 Coerción Explícita	13
4.7 Manipular vectores	14
4.7.1 Suma de vectores	14
4.7.2 Filtrar vectores	15
4.8 Lista	15
4.9 data.frame	17
5 Historia	18
5.1 Un poco de historia de la programación	18
5.2 Algunos puntos adicionales a considerar:	19
6 Un poco de historia de R (o R en la historia)	20
6.0.1 Les invitamos a reflexionar sobre las siguientes preguntas:	21
6.1 Orígenes de R	21
6.1.1 Principales Hitos	21
6.2 Usabilidad	22

6.3	Usos Más Populares	22
6.4	Para profundizar..	22
6.4.1	Algunos textos y libros adicionales:	22
7	¿Qué es R Base?	24
7.1	Características de R Base	24
7.2	Ejemplo R base	24
7.3	Manipulaciones de tablas con R base	25
7.3.1	Manipulación de tablas con paquete ‘dplyr’	26
7.4	Guardar o exportar una tabla o data.frame	28
7.5	Ciclo for	30
7.5.1	Estructura básica del bucle for	31
7.6	Sintaxis básica de una función en R	33
7.6.1	Funciones con valores predeterminados	34
7.6.2	Funciones dentro de funciones	34
7.6.3	Funciones anónimas	35
7.6.4	Ejemplo avanzado con purrr	35
8	Presentaciones en RMD	36
8.1	Presentaciones en RMarkdown	36
8.2	Funciones con kableExtra	36
9	Flujo de trabajo con R	40
10	Algunos paquetes sugeridos	41
10.1	Tidyverse	41
10.1.1	Comenzaremos conociendo funciones del paquete ‘dplyr’	41
10.1.2	filter	42
10.1.3	select	45
10.1.4	arrange	46
10.1.5	mutate	46
10.1.6	summarise	50
10.1.7	group_by	50
10.1.8	rename	51
10.1.9	distinct	52
10.1.10	slice	52
10.1.11	count	53
10.1.12	pull	53
10.1.13	across	53
10.1.14	case_when	54
10.1.15	if_else	55
10.1.16	Joins	57

Referencias	62
Referencias principales para Aprender R	62
Libros	62
Tutoriales en Línea	62
Canales de YouTube	63
Documentación Oficial	63
Blogs y Comunidades	63
Recursos Adicionales	64

Inicio

Una breve guía

Este documento te permitirá dar los primeros pasos en R. La idea es que puedas comprender los aspectos generales de la historia, estructura de datos y objetos básicos en R.

La guía tiene siete capítulos orientados a conocer aquellos códigos que te servirán para *soltar la mano* y no perder la confianza en lo útil que puede llegar a ser aprender sobre pensamiento crítico, programación y datos.

El primer capítulo se llama **Para comenzar** está orientado a que tengas las referencias si es que no has instalado el R en tu computador. También contiene orientaciones generales sobre conceptos como IDE y versión.

El segundo capítulo llamado **Primeros pasos** busca entregar una panorámica general sobre los tipos de datos en R. Te ayudará a comprender que diferencia hay entre un número y una letra. Está más enfocado a la programación básica en R. Cuando estamos comenzando puede ser un capítulo al que volvamos frecuentemente, mientras avanzamos en la práctica de R.

El tercer capítulo **Historia** está pensado para que tengas una idea de los orígenes de R y algunos hitos importantes.

En el cuarto capítulo, **¿Qué es R Base?** se revisan conceptos generables de R algunos códigos de utilidad y la creación de funciones, algo muy importante en el uso de R.

Para comenzar a entrenar, en el quinto capítulo **Presentaciones con RMD** se proponen algunas utilidades para implementar el código y darle estilos a los trabajos escritos o presentaciones.

En el capítulo 6 podremos conocer las principales funciones de ‘dplyr’ librería que vienen cargada con ‘tidyverse’.

El capítulo siguiente (séptimo capítulo) siempre está en construcción y pronto será publicado

Las referencias te permitirán conocer algunos sitios recomendables para tu autoaprendizaje.

1 Para comenzar

1.1 Instalar R

Si nunca has instalado un programa de computadora en tu vida, quizás sea recomendable que pidas apoyo para la instalación. Si has instalado un programa antes, hacerlo con R no debería ser más complejo.

Un fuente “oficial” para descargar R es CRAN (del inglés Comprehensive R Archive Network) algo así como el gran repositorio de R. Puedes acceder haciendo clic acá: [CRAN](#).

En ese sitio puedes descargar R, para Linux, Windows o Mac. Luego puedes trabajar con tu IDE favorito. No sabes qué es un IDE, para más referencias puede visitar la clase abierta [la clase abierta](#) de nuestra Escuela de Datos.

2 Un IDE

En pocas palabras el IDE es donde escribes el código y trabajar la mayor parte del tiempo. Su aspecto positivo es que permite visualizar resultados y realizar tareas utilizando botones y una interfaz integradas, por lo que puede ser útil cuando estás comenzando.

Uno de los IDE más populares de R, es RStudio. El IDE es un programa, por lo que también lo debes descargar e instalar. Puedes hacerlo acá: [POSIT - RSTUDIO](#). Posit es la empresa que creó RStudio.

También puede usar otros IDE's, o escribir en tu consola o en un libro de notas, el objetivo es que utilices lo que te sea más cómodo para trabajar.

3 Conocer sobre las versiones

Tanto R como RStudio, tienen versiones. Una versión es generalmente asociada a un número. dependiendo de la cantidad de cambios significativos para funcionar que tenga el programa en el tiempo, es probable que tenga muchas o pocas versiones.

Recordemos que R es parte de lo que podemos llamar *software libre* o *código abierto* es decir, puedes descargar el código desde CRAN y modificarlo. Eso ha sido positivo para que muchas personas puedan crear sus propios avances o programas con R.

Puede que tu interés no vaya a esos aspectos de R, de todas formas, saber la versión de tu programa es importante. Si instalaste R hace años y luego no lo seguiste usando y ahora lo quieres usar para seguir estos primeros pasos, puedes tomar en cuenta el código:

```
version
```

```
-
platform      x86_64-w64-mingw32
arch           x86_64
os             mingw32
crt            ucrt
system        x86_64, mingw32
status
major         4
minor         4.1
year          2024
month         06
day           14
svn rev       86737
language      R
version.string R version 4.4.1 (2024-06-14 ucrt)
nickname      Race for Your Life
```

Soló con que escribas `version` en la consola y presiones `enter`, podrás tener información de la versión. Debes fijarte en `version.string` y podrás saber la versión.

4 Tipos de datos

#Primeros pasos

En R, hay varios tipos de datos básicos que puedes usar. Aquí hay una descripción de los tipos de datos más comunes.

4.1 Números

Los números pueden ser enteros o números decimales.

```
# Número entero  
x <- 5  
print(x)
```

```
[1] 5
```

```
# Número decimal  
y <- 5.5  
print(y)
```

```
[1] 5.5
```

4.2 Cadena de texto

Las cadenas de texto se representan usando comillas simples o dobles.

```
texto <- "Hola, mundo!"  
  
print(texto)
```

```
[1] "Hola, mundo!"
```

4.3 Valores lógicos

Los valores lógicos (booleanos) pueden ser TRUE o FALSE. Los valores lógicos se usan comúnmente para realizar operaciones condicionales.

```
# Valores lógicos
verdadero <- TRUE
falso <- FALSE
print(verdadero)
```

```
[1] TRUE
```

```
print(falso)
```

```
[1] FALSE
```

4.4 Vectores

Un vector es una secuencia de datos del mismo tipo. Puedes crear vectores usando la función `c()`.

```
# Vector numérico
numeros <- c(1, 2, 3, 4, 5)
print(numeros)
```

```
[1] 1 2 3 4 5
```

```
# Vector de caracteres
cadenas <- c("uno", "dos", "tres")
print(cadenas)
```

```
[1] "uno" "dos" "tres"
```

```
# Vector lógico
logicos <- c(TRUE, FALSE, TRUE)
print(logicos)
```

```
[1] TRUE FALSE TRUE
```

```
# Nuevo vector
# ¿Qué ocurre si en un mismo vector dejamos character y numeric?
combinado <- c(1,2,3,"cuatro", "cinco")

# El resultado es un vector de "character".
combinado
```

```
[1] "1"      "2"      "3"      "cuatro" "cinco"
```

```
# Confirmamos la clase del vector con la función `class()`
class(combinado)
```

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

```
# A lo anterior se le ha llamado, "coerción"
```

4.5 Coerción

En R, la coerción de vectores se refiere al proceso de conversión automática de los elementos de un vector a un tipo de datos común cuando los elementos originales son de tipos diferentes. Esto se hace para asegurar que todas las operaciones en los vectores se realicen de manera consistente y sin errores. R sigue una jerarquía específica de tipos de datos para realizar esta conversión.

Jerarquía de Coerción R tiene una jerarquía de tipos de datos que determina cómo se realiza la coerción. La jerarquía de coerción en R es la siguiente (de menor a mayor):

1. Logical (Lógico): TRUE, FALSE
2. Integer (Entero): Números enteros
3. Numeric (Numérico o Double): Números reales
4. Complex (Complejo): Números complejos
5. Character (Carácter): Cadenas de texto
6. List (Lista): Colecciones de elementos

Cuando se combinan elementos de diferentes tipos en un vector, R los convierte automáticamente al tipo de datos “más alto” en la jerarquía.

```
** lógico -> entero -> numérico -> cadena de texto (logical -> integer -> numeric -> character)
**
```

4.5.1 1. Coerción de Logical a Integer

```
vec <- c(TRUE, FALSE, 1)
print(vec) # Output: 1 0 1
```

```
[1] 1 0 1
```

En este ejemplo, TRUE se convierte a 1 y FALSE se convierte a 0.

4.5.2 2. Coerción de Integer a Numeric

```
vec <- c(1L, 2.5)
print(vec) # Output: 1.0 2.5
```

```
[1] 1.0 2.5
```

El número entero 1L se convierte a 1.0 para coincidir con el número numérico 2.5.

4.5.3 3. Coerción de Numeric a Character

```
vec <- c(1.5, "a")
print(vec) # Output: "1.5" "a"
```

```
[1] "1.5" "a"
```

El número 1.5 se convierte a la cadena de texto "1.5".

4.5.4 4. Coerción de diferentes tipos en una lista

```
vec <- list(1, "a", TRUE)
print(vec) # Output: [[1]] 1 [[2]] "a" [[3]] TRUE
```

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

```
[[2]]
[1] "a"
```

```
[[3]]
[1] TRUE
```

En una lista, los elementos no se fuerzan a un tipo común; cada elemento puede mantener su tipo de datos original.

4.6 Coerción Explícita

Además de la coerción automática, R también permite realizar coerciones explícitas usando funciones específicas como `as.numeric()`, `as.character()`, `as.integer()`, etc.

Ejemplo de Coerción Explícita

```
vec <- c("1", "2", "3")
numeric_vec <- as.numeric(vec)
print(numeric_vec) # Output: 1 2 3
```

```
[1] 1 2 3
```

Aquí, `as.numeric(vec)` convierte explícitamente el vector de caracteres `vec` en un vector numérico.

Notas Importantes

NA (Not Available): Durante la coerción, si algún elemento no puede convertirse correctamente, R genera un valor NA y generalmente emite una advertencia.

Factores: Cuando se trabaja con factores, coercionarlos a otros tipos puede requerir pasos adicionales, como primero convertir el factor a carácter antes de convertirlo a numérico.

Ejemplo

```
factor_vec <- factor(c("1", "2", "3"))
char_vec <- as.character(factor_vec)
numeric_vec <- as.numeric(char_vec)
print(numeric_vec) # Output: 1 2 3
```

```
[1] 1 2 3
```

4.7 Manipular vectores

Puedes manipular vectores de varias maneras, incluyendo seleccionar elementos, sumar y restar elementos, y más.

```
# Selección del primer elemento
primer_elemento <- numeros[1]
print(primer_elemento)
```

```
[1] 1
```

```
# Selección de múltiples elementos
primeros_tres <- numeros[1:3]
print(primeros_tres)
```

```
[1] 1 2 3
```

4.7.1 Suma de vectores

```
# Suma de vectores
suma <- numeros + c(1, 1, 1, 1, 1)
print(suma)
```

```
[1] 2 3 4 5 6
```

```
# Producto de vectores
producto <- numeros * c(2, 2, 2, 2, 2)
print(producto)
```

```
[1] 2 4 6 8 10
```

4.7.2 Filtrar vectores

```
# Filtrar valores mayores que 2
mayores_que_dos <- numeros[numeros > 2]
print(mayores_que_dos)
```

```
[1] 3 4 5
```

```
# Filtrar valores iguales a TRUE
verdaderos <- logicos[logicos == TRUE]
print(verdaderos)
```

```
[1] TRUE TRUE
```

4.8 Lista

En R, una lista es una estructura de datos versátil que puede contener elementos de diferentes tipos, por ejemplo, vectores, matrices, data frames, e incluso otras listas. Esto la vuelve óptima para organizar y manipular datos heterogéneos.

Concepto de list()

Una lista en R se crea utilizando la función `list()`. A diferencia de los vectores, que deben contener elementos del mismo tipo, las listas pueden contener elementos de diferentes tipos. Cada elemento de una lista puede ser referenciado mediante índices o nombres.

Ejemplo de Creación de una Lista

```
lista1 <- list(
  numero = 1,
  nombre = "Juan",
  vector = c(1, 2, 3),
  matriz = matrix(1:4, nrow = 2),
  lista_anidada = list(a = 10, b = 20)
)

print(lista1)
```

```
$numero
```

```
[1] 1
```

```
$nombre
```

```
[1] "Juan"
```

```
$vector
```

```
[1] 1 2 3
```

```
$matriz
```

```
      [,1] [,2]  
[1,]    1    3  
[2,]    2    4
```

```
$lista_anidada
```

```
$lista_anidada$a
```

```
[1] 10
```

```
$lista_anidada$b
```

```
[1] 20
```

Acceso a Elementos de una Lista Puedes acceder a los elementos de una lista utilizando el operador de doble corchete `[[]]` o el operador de signo de dólar `$`.

Acceso por Índice

```
print(lista1[[1]]) # [1] 1
```

```
[1] 1
```

```
print(lista1[[3]]) # [1] 1 2 3
```

```
[1] 1 2 3
```

Acceso por nombre

```
print(lista1$nombre) # [1] "Juan"
```

```
[1] "Juan"
```

```
print(lista1$vector) # [1] 1 2 3
```

```
[1] 1 2 3
```

4.9 data.frame

Un data.frame en R es una estructura de datos fundamental y muy versátil que se utiliza para almacenar datos tabulares. Es similar a una tabla en una base de datos o una hoja de cálculo en Excel, donde cada columna puede contener un tipo diferente de datos (numéricos, caracteres, factores, etc.), pero todos los elementos de una columna deben ser del mismo tipo.

Concepto de data.frame

Un data.frame es esencialmente una lista de vectores de igual longitud, donde cada vector representa una columna de datos y cada elemento dentro del vector representa una fila. Los data.frames son utilizados ampliamente en R para la manipulación y análisis de datos.

Creación de un data.frame

Puedes crear un data.frame utilizando la función data.frame().

```
# Crear un data frame simple
df <- data.frame(
  nombre = c("Ana", "Luis", "Marta", "Juan"),
  edad = c(23, 35, 29, 40),
  salario = c(50000, 60000, 70000, 80000)
)
print(df)
```

```
  nombre edad salario
1   Ana   23  50000
2  Luis   35  60000
3  Marta   29  70000
4   Juan   40  80000
```

5 Historia

5.1 Un poco de historia de la programación

La narrativa tradicional presenta la historia de la programación como una marcha triunfal hacia la eficiencia y la automatización. Sin embargo, esta visión ignora las complejas relaciones sociales, económicas y políticas que han moldeado este campo.

La historia de la programación es una narrativa intrínsecamente ligada a las dinámicas sociales y económicas que han moldeado nuestro mundo contemporáneo. Desde sus inicios, la programación ha servido no solo como una herramienta técnica, sino también como un reflejo de las estructuras de poder y las relaciones de producción imperantes.

Este contexto histórico ha dejado una huella profunda en la cultura de la programación. La búsqueda de la eficiencia y la optimización, a menudo a expensas de la legibilidad y la accesibilidad del código, refleja la lógica capitalista de maximizar la producción y minimizar los costos.

Además, la programación ha estado tradicionalmente dominada por una élite técnica, en su mayoría hombres blancos de países desarrollados. Esta falta de diversidad ha perpetuado sesgos y desigualdades en el acceso y uso de la tecnología.

En sus albores, la programación surgió en un contexto de industrialización y expansión capitalista, donde la eficiencia y la productividad eran imperativos. Los primeros lenguajes de programación, como Fortran y COBOL, fueron desarrollados para optimizar procesos industriales y administrativos, facilitando así la consolidación de grandes corporaciones y la centralización del control económico. Este enfoque técnico reflejaba una visión mecanicista de la sociedad, donde el ser humano se veía subordinado a las máquinas y a los imperativos del capital.

La programación no es neutral. Las decisiones sobre qué problemas se abordan, cómo se diseñan las interfaces y quién tiene acceso a la tecnología reflejan las agendas de quienes la crean. En muchos casos, estas agendas están ligadas al poder, el control y el beneficio económico. La creciente influencia de las grandes empresas tecnológicas plantea preguntas sobre la privacidad, la manipulación y el impacto social de la programación.

En este sentido, es crucial mantener una mirada crítica sobre el desarrollo y uso de R, y de cualquier lenguaje de programación. Debemos preguntarnos: **¿quién controla el acceso a estas herramientas? ¿Cómo se utilizan para generar y distribuir conocimiento? ¿Quiénes se benefician de su uso?**

5.2 Algunos puntos adicionales a considerar:

- **El papel del colonialismo:** La historia de la programación está entrelazada con la expansión colonial. La necesidad de administrar y controlar vastos territorios impulsó el desarrollo de tecnologías de la información, como el telégrafo y las máquinas tabuladoras.
- **La brecha digital:** A pesar de los avances tecnológicos, el acceso a la programación y a las herramientas digitales sigue siendo desigual. La brecha digital perpetúa la exclusión y limita las oportunidades para las comunidades marginadas.
- **El impacto ambiental:** La producción de hardware y el consumo energético de las computadoras tienen un impacto significativo en el medio ambiente. Una mirada crítica debe considerar la sostenibilidad de la industria tecnológica.

6 Un poco de historia de R (o R en la historia)

El lenguaje R, creado en la década de 1990 por Ross Ihaka y Robert Gentleman, emerge en un contexto diferente, marcado por la globalización y la revolución informática. R fue diseñado como una herramienta para el análisis estadístico y la visualización de datos, promoviendo una **democratización del conocimiento técnico**. Sin embargo, desde una perspectiva crítica (o marxista), es crucial cuestionar quién controla y se beneficia de estas herramientas. Aunque R es de código abierto y accesible, su adopción está en gran medida mediada por instituciones académicas y corporativas que **dominan la producción de conocimiento y, por ende, perpetúan ciertas relaciones de poder**.

Además, el auge de lenguajes de programación como R puede interpretarse como parte de una lógica capitalista que busca constantemente innovar y optimizar, pero que también puede conducir a una precarización laboral en sectores tecnológicos. La constante demanda de nuevos conocimientos y habilidades en programación refleja una dinámica de explotación donde quien trabaja debe **adaptarse continuamente** para mantenerse ‘relevante’ en el *mercado laboral*.

Si bien R es un lenguaje de código abierto, su uso se ha visto influenciado por la creciente mercantilización del conocimiento y la proliferación de software propietario en el ámbito del análisis de datos. Las grandes empresas tecnológicas han invertido en el desarrollo de herramientas y plataformas comerciales que, si bien se basan en R, buscan capturar y controlar el flujo de datos y análisis. La proliferación de datos masivos (Big Data) y la inteligencia artificial plantean nuevas formas de explotación y vigilancia, que deben ser analizadas desde una perspectiva crítica.

La estandarización de ciertos lenguajes, la propiedad intelectual y las licencias de software son aspectos que revelan cómo el poder se configura y se mantiene en el ámbito tecnológico. En este sentido, la historia de la programación es una ventana para comprender las intersecciones entre tecnología, economía y sociedad, y para cuestionar las estructuras que determinan quién tiene acceso y control sobre las herramientas que moldean nuestro mundo.

La división del trabajo intelectual y manual, característica del capitalismo, se replica en el ámbito de la programación. Los “arquitectos” del software, en su mayoría provenientes de élites académicas, diseñan sistemas que luego son implementados por programadores, quienes a menudo enfrentan **condiciones laborales precarias y alienantes**. Esta división jerárquica perpetúa la desigualdad y limita la participación democrática en la creación tecnológica.

En síntesis el enfoque que proponemos considera crucial desmitificar la idea de que la tecnología es neutral o inherentemente beneficiosa. **La programación, como herramienta, puede ser utilizada para fines emancipatorios o para reforzar las estructuras de opresión.**

6.0.1 Les invitamos a reflexionar sobre las siguientes preguntas:

- ¿Cómo podemos construir una tecnología que esté al servicio de la clase trabajadora y no de la acumulación incesante de capital?
- ¿Cuál es el objetivo de aprender programación en un contexto donde la Inteligencia Artificial puede resolver problemas básicos de forma eficiente?

Recordemos las palabras de Marx:

“Los filósofos no han hecho más que interpretar de diversos modos el mundo, pero de lo que se trata es de transformarlo”.

6.1 Orígenes de R

R es un lenguaje y entorno de programación para análisis estadístico y gráficos. Surgió a principios de la década de 1990 como una implementación del lenguaje S, que fue desarrollado en los Laboratorios Bell por John Chambers y sus colegas. Como ya se dijo, R fue creado por Ross Ihaka y Robert Gentleman en la Universidad de Auckland, Nueva Zelanda, y fue lanzado como software libre bajo la licencia GNU General Public License en 1995.

6.1.1 Principales Hitos

- 1995: Primera versión pública de R.
- 2000: Se lanza la versión 1.0.0 de R, marcando su madurez - como herramienta de análisis estadístico.
- 2004: R se convierte en el lenguaje más utilizado en el Proyecto de Análisis Estadístico de Google.
- 2010: Revolution Analytics (ahora parte de Microsoft) comienza a ofrecer soporte comercial para R.
- 2015: Microsoft adquiere Revolution Analytics, impulsando el uso de R en el ámbito empresarial.
- 2017: RStudio (el IDE más usado para R) lanza RStudio Server Pro, facilitando el uso de R en entornos de producción.

- 2023: Rstudio (Antes así se llamaba la empresa y el IDE) cambia su nombre a POSIT y comienza a desarrollar RStudio con orientación al uso multilingüaje.

6.2 Usabilidad

R es conocido por su facilidad para el análisis estadístico y la visualización de datos. Su usabilidad se ve reforzada por una comunidad activa que contribuye con paquetes adicionales que extienden su funcionalidad. R es especialmente apreciado en la academia y en la industria por su capacidad de manejar grandes conjuntos de datos y por sus capacidades gráficas avanzadas.

6.3 Usos Más Populares

Análisis Estadístico: R ofrece una amplia gama de técnicas estadísticas, incluyendo modelos lineales y no lineales, pruebas estadísticas clásicas, análisis de series temporales, clasificación y agrupamiento.

Visualización de Datos: Paquetes como ggplot2 permiten crear visualizaciones de datos sofisticadas y personalizadas.

Ciencia de Datos: R es ampliamente utilizado para el análisis de datos en la ciencia de datos, incluyendo minería de datos y machine learning.

6.4 Para profundizar..

Si quieres conocer más sobre los enfoques críticos en cuanto a comunicación, tecnología, programación y lucha de clases, puedes revisar el trabajo de [Wendy Hui Kyong Chun](#), [Safiya Noble](#), [Judith Sutz](#), [Pedro Demo](#) y no olvides que: siempre es recomendable volver a los clásicos: [El Capital](#)

6.4.1 Algunos textos y libros adicionales:

- [Cyber-Marx: Cycles and circuits of struggle in high technology capitalism](#) - Nick Dyer-Witheford.
- [La Condición de la Posmodernidad](#) - David Harvey
- [A dependência tecnologica segundo a dialéctica da dependência](#) de Ruy Mauro Marini
- [Desenvolvimento e dependência](#) - Ruy Mauro Marini

También puede ser útil visitar [Revista Chasqui de Ecuador](#)

7 ¿Qué es R Base?

R base se refiere al conjunto fundamental de funciones y paquetes que se instalan con R. Este núcleo incluye funciones para manipulación básica de datos, operaciones aritméticas, funciones estadísticas básicas, y gráficos simples. R base proporciona las herramientas necesarias para comenzar con el análisis de datos, y su funcionalidad puede ser ampliada mediante la instalación de paquetes adicionales desde CRAN (Comprehensive R Archive Network).

7.1 Características de R Base

Manipulación de Datos: Funciones para manejar estructuras de datos como vectores, matrices, listas y data frames. Estadísticas Básicas: Funciones para cálculos estadísticos como media, mediana, varianza, y desviación estándar. Gráficos: Herramientas para crear gráficos básicos como histogramas, diagramas de dispersión, y gráficos de líneas. Programación: Funciones de control de flujo, bucles, y la capacidad de definir funciones personalizadas.

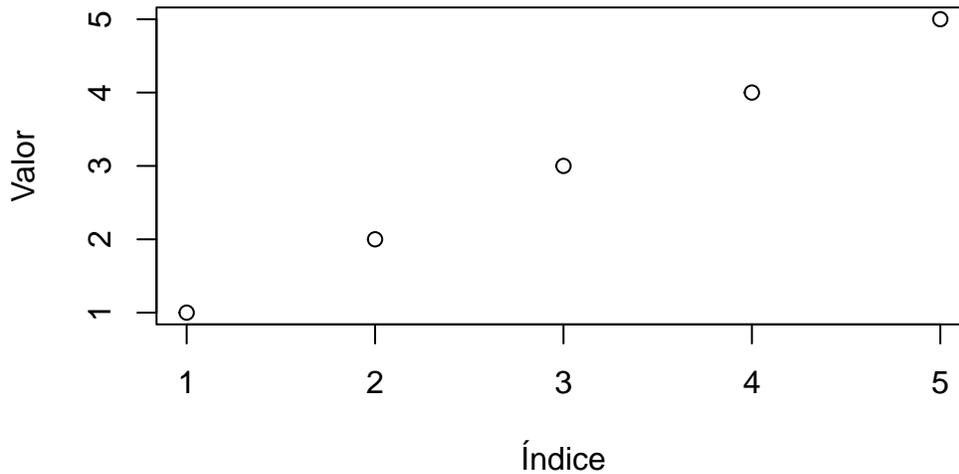
7.2 Ejemplo R base

```
# Crear un vector numérico
numeros <- c(1, 2, 3, 4, 5)

# Calcular la media del vector
media_numeros <- mean(numeros)

# Crear un gráfico de dispersión
plot(numeros, main="Gráfico de Dispersión",
      xlab="Índice", ylab="Valor")
```

Gráfico de Dispersión



7.3 Manipulaciones de tablas con R base

```
# Crear un data.frame o tabla

df <- data.frame(
  nombre = c("Ana", "Luis", "Marta", "Juan"),
  edad = c(23, 35, 29, 40),
  salario = c(50000, 60000, 70000, 80000)
)

# Crear un data frame
tabla1 <- data.frame(dia = c("Lunes", "Martes", "Miercoles", "Jueves"),
  obs = c(10, 11, "hola", 22))

tabla1
```

```
      dia  obs
1  Lunes  10
2  Martes  11
3 Miercoles hola
4  Jueves  22
```

Algo importante a tener en cuenta es que para construir un data.frame o tabla es necesario tener vectores de una longitud similar. Los vectores pueden tener distintos elementos como 'numeric' o 'character' sin embargo, se aplicarán las reglas de coerción.

```
# Para ver una columna específica de una tabla podemos utilizar el operador '$'  
class(tabla1$dia)
```

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

```
class(tabla1$obs)
```

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

```
# En este caso utilizamos un función de la familia 'as.' para convertir el vector en numeric  
#esta palabra se convierte en NA pues no se puede transformar a numeric.  
class(as.numeric(tabla1$obs))
```

```
Warning: NAs introducidos por coerción
```

```
[1] "numeric"
```

```
# Convertir numerico obs  
tabla1$obs <- as.numeric(tabla1$obs)
```

```
Warning: NAs introducidos por coerción
```

```
# Revisamos la clase de la columna del objeto.  
class(tabla1$obs)
```

```
[1] "numeric"
```

```
#Promedio  
mean(tabla1$obs, na.rm = TRUE)
```

```
[1] 14.33333
```

7.3.1 Manipulación de tablas con paquete 'dplyr'

```
# Cargamos la libreria 'dplyr' y 'clipr'
library(dplyr)
```

Adjuntando el paquete: 'dplyr'

The following objects are masked from 'package:stats':

```
filter, lag
```

The following objects are masked from 'package:base':

```
intersect, setdiff, setequal, union
```

```
library(clipr)
```

Welcome to clipr. See ?write_clip for advisories on writing to the clipboard in R.

```
# Operador 'pipe' tiene uso en "Rbase" y en "tidyverse" con el paquete magrittr
#rbase      # magrittr
# |>       # %>%

# Creamos un nuevo objeto utilizando la pipe
objeto1 <- tabla1 |> # acá asignamos un nuevo objeto "objeto1", desde "tabla1"
dplyr::filter(!is.na(obs)) |> # se realiza un filtro para quitar "NA" en col "obs"
dplyr::summarise(# Se realiza resumen con medidas
                 promedio = mean(obs, na.rm = TRUE), # promedio
                 mediana = median(obs, na.rm = TRUE), # mediana
                 numero = n()) |> #conteo
dplyr::rename(Media = promedio) # se renombra la columna promedio como "Media"

# También es posible utilizar pipe en rbase, pero solo la del estilo " |> "
objeto1$Media |> class()
```

```
[1] "numeric"
```

```
# Con clipr se puede copiar el objeto rapidamente para pegar en excel y otro
# clipr::write_clip(objeto1)

# Con esta función se puede conocer el directorio de trabajo actual
getwd()
```

```
[1] "C:/Users/rgalv/Nextcloud/2024/EscuelaDeDatos/PrimeroPasosEnR"
```

7.4 Guardar o exportar una tabla o data.frame

Es posible guardar un objeto de R en diversos formatos. En el caso de que sean objetos como tablas de datos o bases de datos, que no se trabajarán otro lenguaje, puede ser recomendable almacenarlos en formato `´.rds´` que es el nativo de R y permite mejor velocidad de lectura y escritura, además menos tamaño en disco.

También es posible guardar los archivos en formato `´.csv´` u otros. Generalmente esto se puede hacer con las funciones `´write.´` o `´save´`

```
# Guardar RDS
saveRDS(objeto1, "objeto1.rds")

# Guardar objeto1 como un csv
write.csv2(x = objeto1, # corresponde al objeto a exportar
           file = "objeto1.csv", # define el nombre que tendrá el archivo exportado
           dec = ",", # define que para decimales se utilice ´,´
           sep = ";", # define que el separador sea ´;´
           row.names = FALSE)
```

```
Warning in write.csv2(x = objeto1, file = "objeto1.csv", dec = ",", sep = ";",
: attempt to set 'sep' ignored
```

```
Warning in write.csv2(x = objeto1, file = "objeto1.csv", dec = ",", sep = ";",
: attempt to set 'dec' ignored
```

7.4.0.1 Manipulación Rbase

Para los siguientes ejercicios se utilizará el paquete `´cars´` que viene incorporado en R.

```
# Manipular datos
class(cars$dist)
```

```
[1] "numeric"
```

```
# Filtro
cars$speed[cars$speed > 10 & cars$dist > 20]
```

```
[1] 11 12 12 13 13 13 13 14 14 14 14 15 15 16 16 17 17 17 18 18 18 18 19 19 19
[26] 20 20 20 20 20 22 23 24 24 24 24 25
```

```
# Crea nuevo objeto a partir de filtro
cars2 <- cars[1:15, 1:2]

# Los valores mayores a 10 de la columna speed se reemplazan con 'NA'
cars2$speed[cars2$speed>10] <- NA

# Para trabajar con esta sintaxis es importante considerar que
# en el interior de los corchetes se sigue el orden [filas, columnas]
# objeto[f,c]

# El dato que está en la tercera fila, de la segunda columna
cars[3,2]
```

```
[1] 4
```

```
# Filtro de rango de los datos que están entre las filas 1 a la 4 y columnas 1 y 2
cars[1:4, 1:2]
```

```
  speed dist
1     4     2
2     4    10
3     7     4
4     7    22
```

```
# Usando vector
cars[c(1,4), c(1,2)]
```

```

speed dist
1      4      2
4      7     22

```

```

# Para crear una tabla de contingencia se puede utilizar la función `table()`
table(cars2$speed)

```

```

4  7  8  9 10
2  2  1  1  3

```

```

prop.table(table(iris$Species))

```

```

      setosa versicolor virginica
0.3333333 0.3333333 0.3333333

```

```

table(iris$Species, iris$Petal.Length)

```

```

      setosa versicolor virginica
1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.9 3 3.3 3.5 3.6 3.7 3.8 3.9 4
setosa      1  1  2  7 13 13  7  4  2  0  0  0  0  0  0  0  0
versicolor  0  0  0  0  0  0  0  0  0  1  2  2  1  1  1  3  5
virginica   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

      setosa versicolor virginica
4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5 5.1 5.2 5.3 5.4 5.5 5.6 5.7
setosa      0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
versicolor  3  4  2  4  7  3  5  2  2  1  1  0  0  0  0  0  0
virginica   0  0  0  0  1  0  0  2  3  3  7  2  2  2  3  6  3

      setosa versicolor virginica
5.8 5.9 6 6.1 6.3 6.4 6.6 6.7 6.9
setosa      0  0  0  0  0  0  0  0  0
versicolor  0  0  0  0  0  0  0  0  0
virginica   3  2  2  3  1  1  1  2  1

```

7.5 Ciclo for

Un bucle for en R permite ejecutar repetidamente un bloque de código un número específico de veces.

7.5.1 Estructura básica del bucle for

La estructura básica del ciclo `for` se describe de la siguiente forma: *for (variable in secuencia){ operaciones }*

```
# for (variable in secuencia) {  
#     Código a ejecutar  
# }
```

- *Variable*: Es una variable que toma los valores de cada elemento en la secuencia.
- *Secuencia*: Es un vector que contiene los valores que la variable tomará sucesivamente.
- *Código a ejecutar*: Es el bloque de código que se ejecutará en cada iteración del bucle.

Ejemplo 1: Iteración en una secuencia

```
for (i in 1:5) {  
  print(i)  
}
```

```
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5
```

En este caso, `i` tomará cada valor en la secuencia 1, 2, 3, 4, 5, y `print(i)` se ejecutará en cada iteración, imprimiendo cada número.

Ejemplo 2: Iteración en un vector

```
# Crea un vector numérico  
numeros <- c(2, 4, 6, 8, 10)  
# Realiza el bucle para imprimir los valores del vector  
for (num in numeros) {  
  print(num)  
}
```

```
[1] 2  
[1] 4  
[1] 6  
[1] 8  
[1] 10
```

Ejemplo 3: Operaciones dentro del bucle

```
numeros <- c(1, 2, 3, 4, 5)
for (num in numeros) {
  cuadrado <- num^2 # Operación
  print(paste("El cuadrado de", num, "es", cuadrado))
}
```

```
[1] "El cuadrado de 1 es 1"
[1] "El cuadrado de 2 es 4"
[1] "El cuadrado de 3 es 9"
[1] "El cuadrado de 4 es 16"
[1] "El cuadrado de 5 es 25"
```

Ejemplo 4: Bucles anidados

```
for (i in 1:5) {
  for (j in 1:5) {
    resultado <- i * j
    print(paste(i, "x", j, "=", resultado))
  }
}
```

```
[1] "1 x 1 = 1"
[1] "1 x 2 = 2"
[1] "1 x 3 = 3"
[1] "1 x 4 = 4"
[1] "1 x 5 = 5"
[1] "2 x 1 = 2"
[1] "2 x 2 = 4"
[1] "2 x 3 = 6"
[1] "2 x 4 = 8"
[1] "2 x 5 = 10"
[1] "3 x 1 = 3"
[1] "3 x 2 = 6"
[1] "3 x 3 = 9"
[1] "3 x 4 = 12"
[1] "3 x 5 = 15"
[1] "4 x 1 = 4"
[1] "4 x 2 = 8"
[1] "4 x 3 = 12"
```

```
[1] "4 x 4 = 16"  
[1] "4 x 5 = 20"  
[1] "5 x 1 = 5"  
[1] "5 x 2 = 10"  
[1] "5 x 3 = 15"  
[1] "5 x 4 = 20"  
[1] "5 x 5 = 25"
```

En este caso, el bucle interno recorre los valores del 1 al 5 para cada valor del bucle externo, generando una tabla de multiplicar.

7.6 Sintaxis básica de una función en R

Las funciones en R son fundamentales para estructurar el código y hacerlo reutilizable. Utilizar paquetes como *purrr* y *kableExtra* nos permite ampliar las capacidades de R para manejar listas y generar tablas con estilo, respectivamente. Al seguir una estructura clara y consistente, podemos desarrollar código R eficiente y fácil de mantener.

Para definir una función en R, Se utiliza la palabra clave `function`. Aquí tienes la estructura básica:

```
nombre_de_la_funcion <- function(argumento1, argumento2, ...) {  
  # Cuerpo de la función  
  # Código a ejecutar  
  resultado <- argumento1 + argumento2  
  return(resultado)  
}
```

Ejemplo: Crear una función para sumar dos números

```
suma <- function(a, b) {  
  resultado <- a + b  
  return(resultado)  
}  
  
# Usar la función  
suma(3, 5) # Devuelve 8
```

```
[1] 8
```

7.6.1 Funciones con valores predeterminados

Se pueden establecer valores predeterminados para los argumentos de una función:

```
suma <- function(a = 1, b = 1) {  
  resultado <- a + b  
  return(resultado)  
}  
  
# Usar la función con valores predeterminados  
suma() # Devuelve 2
```

```
[1] 2
```

7.6.2 Funciones dentro de funciones

En R, también se puede definir funciones dentro de otras funciones:

```
operacion <- function(a, b) {  
  suma <- function(x, y) {  
    return(x + y)  
  }  
  producto <- function(x, y) {  
    return(x * y)  
  }  
  resultado_suma <- suma(a, b)  
  resultado_producto <- producto(a, b)  
  return(list(suma = resultado_suma, producto = resultado_producto))  
}  
  
# Usar la función  
operacion(3, 5) # Devuelve una lista con la suma y el producto
```

```
$suma  
[1] 8
```

```
$producto  
[1] 15
```

7.6.3 Funciones anónimas

Las funciones anónimas (también conocidas como funciones lambda) son útiles para operaciones rápidas y se utilizan a menudo con funciones de orden superior como `apply`, `lapply`, `sapply`, entre otras.

```
# Usar una función anónima para elevar al cuadrado los elementos de un vector
vector <- c(1, 2, 3, 4)
resultado <- sapply(vector, function(x) x^2)
print(resultado) # Devuelve c(1, 4, 9, 16)
```

```
[1] 1 4 9 16
```

7.6.4 Ejemplo avanzado con purrr

El paquete `purrr` de *tidyverse* proporciona una forma funcional de trabajar con listas y vectores. Aquí hay un ejemplo de cómo usar una función personalizada con `map`:

```
library(purrr)

# Definir una función para multiplicar por 2
multiplicar_por_dos <- function(x) {
  return(x * 2)
}

# Usar map para aplicar la función a cada elemento de una lista
lista <- list(1, 2, 3, 4)
resultado <- map(lista, multiplicar_por_dos)
print(resultado) # Devuelve una lista con elementos multiplicados por 2
```

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

```
[[2]]
[1] 4
```

```
[[3]]
[1] 6
```

```
[[4]]
[1] 8
```

8 Presentaciones en RMD

RMD es la extensión de los archivos creados en Rmarkdown es una combinación de códigos de R y Markdown ([Revisa acá ¿Qué es markdown?](#))

8.1 Presentaciones en RMarkdown

Para crear una presentación en RMarkdown, utilizando RStudio, se debe seleccionar un nuevo archivo tipo RMarkdown, luego el tipo de presentación que se quiere desarrollar, para este ejemplo seleccionamos **beamer**, luego se abra una ventana con un código escrito similar al siguiente:

```
# ---
# title: "Tu Título"
# author: "Tu Nombre"
# date: "Fecha"
# output: beamer_presentation
# ---
#
# ## Diapositivas con viñeta
#
# - Viñeta 1
# - Viñeta 2
# - Viñeta 3
```

8.2 Funciones con kableExtra

Para crear tablas en formato *LaTeX* o *HTML* con el paquete *kableExtra*, primero definimos los datos y luego aplicamos las funciones del paquete para formatear la tabla.

```
# Para instalar paquete "kableExtra" en caso de ser necesario

paquetes <- c("kableExtra", "DT", "htmlwidgets")
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

```
# Función para instalar y cargar paquetes
install_carga <- function(paquetes) {
  if (!require(paquetes, character.only = TRUE)) {
    install.packages(paquetes)
    library(paquetes, character.only = TRUE)
  }
}

# Aplicar la función a cada paquete de la lista
lapply(paquetes, install_carga)
```

```
[[1]]
NULL
```

```
[[2]]
NULL
```

```
[[3]]
NULL
```

```
# Carga libreria
library(kableExtra)

# Definir datos
dt <- mtcars[1:5, 1:6]

# Crear una tabla con estilo en LaTeX
kbl(dt, booktabs = TRUE) %>%
  kable_styling(latex_options = "striped")
```

Ejemplo de tabla con estilos

	Grupo 1			Grupo 2		
	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

```
# Definir los datos
dt <- mtcars[1:5, 1:6]

# Crear una tabla con múltiples estilos
kbl(dt,
  booktabs = TRUE) %>%
kable_styling(
  bootstrap_options = c("striped", "hover", "condensed"),
  full_width = FALSE,
  position = "center",
  font_size = 10
) %>%
add_header_above(c(" ", "Grupo 1" = 3, "Grupo 2" = 3)) %>%
column_spec(1, bold = TRUE, color = "red",
  background = "yellow") %>%
column_spec(2, width = "5em") %>%
row_spec(0, bold = TRUE, color = "white",
  background = "#D7261E") %>%
row_spec(1:5, background = "#F7F7F7")
```

Explicación de los Estilos

- `booktabs`: Utiliza el paquete `booktabs` para mejorar el diseño de la tabla.
- `bootstrap_options`: Añade varias opciones de estilo como `striped` (filas alternas de color), `hover` (resaltar filas al pasar el mouse) y `condensed` (espaciado reducido).
- `full_width`: Ajusta la tabla para que no ocupe todo el ancho disponible.
- `position`: Define la posición de la tabla, en este caso centra la tabla en la página.
- `font_size`: Cambia el tamaño de la fuente de la tabla.
- `add_header_above`: Añade un encabezado adicional encima de la tabla para agrupar columnas.

- `column_spec`: Especifica estilos para columnas individuales. En este caso, la primera columna es negrita, roja y con fondo amarillo, y la segunda columna tiene un ancho específico.
- `row_spec`: Especifica estilos para filas individuales. La fila de encabezado (0) es negrita, blanca con fondo rojo oscuro, y las filas 1 a 5 tienen un fondo gris claro.

9 Flujo de trabajo con R

En este capítulo veremos algunas alternativas para realizar transformaciones utilizando R. Utilizaremos de distintas sintaxis de programación mediante el uso de paquetes y R base.

10 Algunos paquetes sugeridos

A continuación veremos el uso de dos paquetes recomendados para análisis de datos ‘Tidyverse’ y ‘data.table’

10.1 Tidyverse

Este es un “*paquete de paquetes*” que permite cargar una serie de herramientas útiles para el tratamiento de datos. Para profundizar sobre los usos de este paquete se recomienda la lectura de [R for Data science](#).

Es un paquete que puede facilitar el uso inicial de R porque la sintaxis en inglés de compone de funciones como: ‘mutate’ que sirve para crear o editar columnas (*mutar la tabla de datos*), ‘arrange’ que sirve para *ordenar*, ‘group_by’ para agrupar por categorías y así. Estos ejemplos corresponden a funciones del paquete ‘dplyr’ que se puede cargar de forma individual o con ‘tidyverse’.

10.1.1 Comenzaremos conociendo funciones del paquete ‘dplyr’

- Características principales:
 1. Enfoque en verbos: dplyr se basa en verbos que describen acciones comunes en la manipulación de datos, como filtrar, seleccionar, ordenar, etc. Esto hace que el código sea más legible y fácil de entender.
 2. Sintaxis “pipe” (%>% o |>): Permite encadenar múltiples operaciones de forma secuencial, lo que facilita la lectura y la escritura del código.
 3. Integración con tidyverse: dplyr funciona a la perfección con otros paquetes del tidyverse, como ggplot2 para visualización y tidyr para la limpieza de datos.
- Funciones principales:

10.1.2 filter

Cundo cargas el paquete tidyverse aparece un print de pantalla como el siguiente:

```
— Attaching core tidyverse packages ————— tidyverse 2.0.0 —
✓ dplyr      1.1.4    ✓ readr      2.1.5
✓ forcats    1.0.0    ✓ stringr    1.5.1
✓ ggplot2    3.5.1    ✓ tibble     3.2.1
✓ lubridate  1.9.3    ✓ tidyr      1.3.1
✓ purrr      1.0.2
— Conflicts ————— tidyverse_conflicts() —
✗ dplyr::filter() masks stats::filter()
✗ dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

Esto nos indica una alerta de conflicto: ‘filter()’ entra en conflicto con la función ‘stats::filter()’ del paquete stats. Esto se debe a que ambas funciones tienen el mismo nombre, ‘stats’ es un paquete que se carga automáticamente con R base.

¿Cómo resolverlo?

Si necesitas usar la función ‘filter()’ de stats, puedes llamarla explícitamente usando ‘stats::filter()’. Si cargas dplyr después de stats, la función ‘filter()’ de dplyr tendrá prioridad. Si consideramos que por lo general ‘dplyr’ o ‘tidyverse’ serán cargados luego de abrir ‘R’, por lo general se cargará posteriormente la función de ‘dplyr’.

- Ahora veamos un ejemplo del uso de la función con la tabla *mtcars* que viene cargada en R.

```
# Primero cargamos la librería tidyverse
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.4    v readr      2.1.5
v forcats    1.0.0    v stringr    1.5.1
v ggplot2    3.5.1    v tibble     3.2.1
v lubridate  1.9.3    v tidyr      1.3.1
v purrr      1.0.2
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
# Filtrar los coches con 4 cilindros
dplyr::filter(mtcars, cyl == 4)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

```
# Filtrar los coches con más de 100 caballos de fuerza y cambio automático (am = 0)
dplyr::filter(mtcars, hp > 100 & am == 0)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
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

Como ves las funciones anteriores no utilizan la ‘pipe’ (%>% o |>) una alternativa para trabajar con la pipe haciendo lo mismo anterior corresponde a:

```
# Filtrar los coches con 4 cilindros
mtcars %>%
  dplyr::filter(cyl == 4)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

```
# Filtrar los coches con más de 100 caballos de fuerza y cambio automático (am = 0)
mtcars %>%
  dplyr::filter(hp > 100 & am == 0)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
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

En los ajemplos anteriores se imprimen todas las filas de filtradas. Como la tabla es corta no resulta problemático, pero con una tabla más grande puede ser poco útil, en ese caso es posible agregar la función 'head()'

```
mtcars %>%
  dplyr::filter(hp > 100 & am == 0) %>%
  head(n = 10)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
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

Se puede definir el número de filas a mostrar en 'head()' head(n = 10). Si el argumento está vacío se imprimirán las 6 primeras filas.

10.1.3 select

La función 'select' permite seleccionar columnas específicas de un data frame o tabla.

```
# Seleccionar las columnas "mpg" (millas por galón), "cyl" (cilindros) y "hp" (caballos de fuerza)
select(mtcars, mpg, cyl, hp) %>%
  head(n = 5)
```

	mpg	cyl	hp
Mazda RX4	21.0	6	110
Mazda RX4 Wag	21.0	6	110
Datsun 710	22.8	4	93
Hornet 4 Drive	21.4	6	110
Hornet Sportabout	18.7	8	175

```
# Seleccionar todas las columnas EXCEPTO "qsec" (tiempo en recorrer 1/4 de milla) y "carb" (combustible)
mtcars %>%
  select(-qsec, -carb) %>%
  head(n = 5)
```

	mpg	cyl	disp	hp	drat	wt	vs	am	gear
Mazda RX4	21.0	6	160	110	3.90	2.620	0	1	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	0	1	4
Datsun 710	22.8	4	108	93	3.85	2.320	1	1	4
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	1	0	3
Hornet Sportabout	18.7	8	360	175	3.15	3.440	0	0	3

10.1.4 arrange

Esta función sirve para ordenar una tabla según una variable.

```
# Ordenar los coches por consumo de combustible ("mpg") de forma descendente
arrange(mtcars, desc(mpg)) %>%
  head(n = 5)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1

```
# Ordenar los coches por número de cilindros y luego por caballos de fuerza (ascendente)
mtcars %>%
  arrange(cyl, hp) %>%
  head(n = 5)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1

10.1.5 mutate

Esta función es útil para crear columnas o editar las ya existentes.

```
# Crear una columna "relacion_hp_peso" (caballos de fuerza / peso)
mutate(mtcars, relacion_hp_peso = hp / wt)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
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
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
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 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
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
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
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
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	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

relacion_hp_peso

Mazda RX4	41.98473
Mazda RX4 Wag	38.26087
Datsun 710	40.08621
Hornet 4 Drive	34.21462
Hornet Sportabout	50.87209
Valiant	30.34682
Duster 360	68.62745
Merc 240D	19.43574
Merc 230	30.15873
Merc 280	35.75581
Merc 280C	35.75581
Merc 450SE	44.22604
Merc 450SL	48.25737
Merc 450SLC	47.61905
Cadillac Fleetwood	39.04762

Lincoln Continental	39.63864
Chrysler Imperial	43.03087
Fiat 128	30.00000
Honda Civic	32.19814
Toyota Corolla	35.42234
Toyota Corona	39.35091
Dodge Challenger	42.61364
AMC Javelin	43.66812
Camaro Z28	63.80208
Pontiac Firebird	45.51365
Fiat X1-9	34.10853
Porsche 914-2	42.52336
Lotus Europa	74.68605
Ford Pantera L	83.28076
Ferrari Dino	63.17690
Maserati Bora	93.83754
Volvo 142E	39.20863

```
# Convertir la columna "mpg" a kil6metros por litro (kpl)
mtcars %>%
  mutate(kpl = mpg * 0.425144)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
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
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
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 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
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
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2

Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
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
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	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

kpl

Mazda RX4	8.928024
Mazda RX4 Wag	8.928024
Datsun 710	9.693283
Hornet 4 Drive	9.098082
Hornet Sportabout	7.950193
Valiant	7.695106
Duster 360	6.079559
Merc 240D	10.373514
Merc 230	9.693283
Merc 280	8.162765
Merc 280C	7.567563
Merc 450SE	6.972362
Merc 450SL	7.354991
Merc 450SLC	6.462189
Cadillac Fleetwood	4.421498
Lincoln Continental	4.421498
Chrysler Imperial	6.249617
Fiat 128	13.774666
Honda Civic	12.924378
Toyota Corolla	14.412382
Toyota Corona	9.140596
Dodge Challenger	6.589732
AMC Javelin	6.462189
Camaro Z28	5.654415
Pontiac Firebird	8.162765
Fiat X1-9	11.606431
Porsche 914-2	11.053744
Lotus Europa	12.924378
Ford Pantera L	6.717275

```
Ferrari Dino          8.375337
Maserati Bora        6.377160
Volvo 142E          9.098082
```

10.1.6 summarise

Con dos “m” summarise es una función útil para el calculo de estadísticos de las variables.

```
# Calcular el promedio de millas por galón ("mpg")
summarise(mtcars, mpg_promedio = mean(mpg))
```

```
mpg_promedio
1      20.09062
```

```
# Calcular la desviación estándar de la potencia ("hp")
summarise(mtcars, hp_sd = sd(hp))
```

```
hp_sd
1 68.56287
```

10.1.7 group_by

Permite agrupar variables y realizar operaciones en esos grupos.

```
# Agrupar los coches por número de cilindros ("cyl")
group_by(mtcars, cyl)
```

```
# A tibble: 32 x 11
# Groups:   cyl [3]
  mpg   cyl  disp    hp  drat    wt  qsec    vs  am  gear  carb
<dbl> <dbl>
1  21     6  160   110  3.9   2.62  16.5    0    1     4     4
2  21     6  160   110  3.9   2.88  17.0    0    1     4     4
3 22.8    4  108    93  3.85  2.32  18.6    1    1     4     1
4 21.4    6  258   110  3.08  3.22  19.4    1    0     3     1
5 18.7    8  360   175  3.15  3.44  17.0    0    0     3     2
6 18.1    6  225   105  2.76  3.46  20.2    1    0     3     1
7 14.3    8  360   245  3.21  3.57  15.8    0    0     3     4
8 24.4    4  147.    62  3.69  3.19  20     1    0     4     2
```

```

  9 22.8    4 141.    95 3.92  3.15 22.9    1    0    4    2
10 19.2    6 168.   123 3.92  3.44 18.3    1    0    4    4
# i 22 more rows

```

```

# Calcular el promedio de millas por galón para cada número de cilindros
mtcars %>%
  group_by(cyl) %>%
  summarise(mpg_promedio = mean(mpg))

```

```

# A tibble: 3 x 2
  cyl mpg_promedio
<dbl> <dbl>
1     4         26.7
2     6         19.7
3     8         15.1

```

10.1.8 rename

Esta función permite cambiar el nombre de las columnas, primero debes indicar el nuevo nombre y luego un '=' seguido del nombre actual.

```

# Cambiar el nombre de la columna "mpg" a "millas_por_galon"
rename(mtcars, millas_por_galon = mpg) %>%
  head(n = 5)

```

	millas_por_galon	cyl	disp	hp	drat	wt	qsec	vs	am	gear
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3

	carb
Mazda RX4	4
Mazda RX4 Wag	4
Datsun 710	1
Hornet 4 Drive	1
Hornet Sportabout	2

10.1.9 distinct

```
# Eliminar las filas duplicadas en función de la columna "cyl"  
distinct(mtcars, cyl)
```

	cyl
Mazda RX4	6
Datsun 710	4
Hornet Sportabout	8

10.1.10 slice

Esta función permite seleccionar filas específicas de una tabla, complementado con ‘group_by’ permite desarrollar seleccionar elementos por grupos.

```
# Seleccionar el primer elemento de cada grupo  
mtcars %>%  
  group_by(cyl) %>%  
  slice(1)
```

```
# A tibble: 3 x 11  
# Groups:   cyl [3]  
  mpg   cyl  disp    hp  drat    wt  qsec    vs  am  gear  carb  
<dbl>  
1  22.8     4  108    93  3.85  2.32  18.6     1   1     4     1  
2   21      6  160   110  3.9   2.62  16.5     0   1     4     4  
3  18.7     8  360   175  3.15  3.44  17.0     0   0     3     2
```

```
# Seleccionar las filas 10, 20 y 30  
slice(mtcars, c(10, 20, 30))
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Merc 280	19.2	6	167.6	123	3.92	3.440	18.3	1	0	4	4
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.9	1	1	4	1
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.5	0	1	5	6

10.1.11 count

Cuenta el número de filas para cada combinación de variables. Similar a `group_by()` y `summarise(n = n())`.

```
# Contar el número de vehiculos para cada combinación de cilindros ("cyl") y cambio ("am")
count(mtcars, cyl, am)
```

```
   cyl am  n
1    4  0  3
2    4  1  8
3    6  0  4
4    6  1  3
5    8  0 12
6    8  1  2
```

10.1.12 pull

Permite extraer una columna o valor como un vector de datos.

```
# Extraer la columna "hp" como un vector
hp_vector <- pull(mtcars, hp)
```

10.1.13 across

Es una función que ayuda a aplicar una función a múltiples columnas.

```
# Calcular la media de todas las columnas numéricas, usando 'where' y la función 'is.numeric'
mtcars %>%
  summarise(across(where(is.numeric), mean))
```

```
   mpg   cyl  disp    hp  drat    wt  qsec    vs  am
1 20.09062 6.1875 230.7219 146.6875 3.596563 3.21725 17.84875 0.4375 0.40625
   gear  carb
1 3.6875 2.8125
```

10.1.14 case_when

Esta función permite crear nuevas columnas basadas en condiciones, permite recodificar variables. Se puede combinar con el uso de 'mutate'

```
# Crear una columna "categoria_hp" que clasifica los coches según su potencia
mtcars %>%
  mutate(categoria_hp = case_when(
    hp < 100 ~ "Baja",
    hp >= 100 & hp < 200 ~ "Media",
    hp >= 200 ~ "Alta"
  ))
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
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
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
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 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
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
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
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
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	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

categoria_hp

Mazda RX4	Media
Mazda RX4 Wag	Media
Datsun 710	Baja
Hornet 4 Drive	Media
Hornet Sportabout	Media
Valiant	Media
Duster 360	Alta
Merc 240D	Baja
Merc 230	Baja
Merc 280	Media
Merc 280C	Media
Merc 450SE	Media
Merc 450SL	Media
Merc 450SLC	Media
Cadillac Fleetwood	Alta
Lincoln Continental	Alta
Chrysler Imperial	Alta
Fiat 128	Baja
Honda Civic	Baja
Toyota Corolla	Baja
Toyota Corona	Baja
Dodge Challenger	Media
AMC Javelin	Media
Camaro Z28	Alta
Pontiac Firebird	Media
Fiat X1-9	Baja
Porsche 914-2	Baja
Lotus Europa	Media
Ford Pantera L	Alta
Ferrari Dino	Media
Maserati Bora	Alta
Volvo 142E	Media

10.1.15 if_else

Esta función es similar a ‘case_when’ solo que sirve para dos condicionales, a menos que se aniden varios ‘if_else’, pero eso puede ser menos legible que la sintaxis de ‘case_when’.

```
# Crear una columna "potente" que indica si el coche tiene más de 150 caballos de fuerza
mtcars %>%
  mutate(potente = if_else(hp > 150, TRUE, FALSE))
```

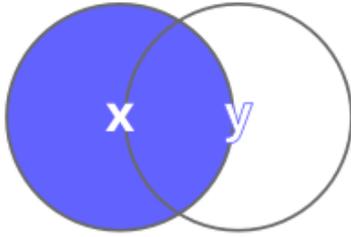
	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	potente
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4	FALSE
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4	FALSE
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1	FALSE
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1	FALSE
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2	TRUE
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1	FALSE
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4	TRUE
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2	FALSE
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2	FALSE
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4	FALSE
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4	FALSE
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3	TRUE
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3	TRUE
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3	TRUE
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4	TRUE
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4	TRUE
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4	TRUE
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1	FALSE
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2	FALSE
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1	FALSE
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1	FALSE
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2	FALSE
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2	FALSE
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4	TRUE
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2	TRUE
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1	FALSE
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2	FALSE
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2	FALSE
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4	TRUE
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6	TRUE
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8	TRUE
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2	FALSE

10.1.16 Joins

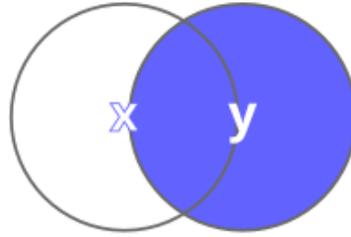
Existen diversas funciones de 'dplyr' que permiten 'fusionar' bases de datos o planillas, segun sus variables. Estas funciones se puyede agrupar en una familia de 'join' dependiendo del uso especifico que queramos podemos utilizar un '_join' adecuado.

dplyr *joins*

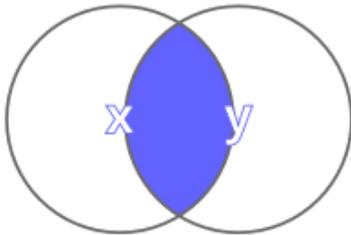
left_join(x, y)



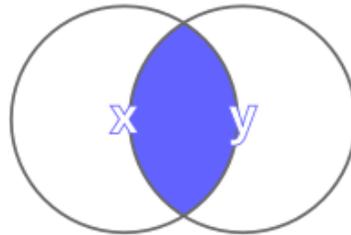
right_join(x, y)



inner_join(x, y)

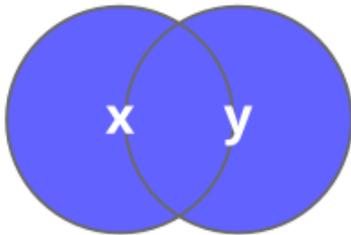


semi_join(x, y)

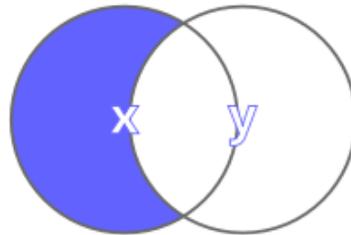


(never duplicate rows of x)

full_join(x, y)



anti_join(x, y)



[Visualización de joins en diagrama de venn disponible en etse enlace](#)

Como ejemplo del diagrama anterior podemos tomar el caso de 'left_join' al realizar este tipo de unión lo que hacemos es que aquellos elementos del objeto 'y' que coinciden con alguno de 'x' quedarán en el conjunto, podemos pensar en esto como una variable llamada x, que tiene valores semejantes en y, entonces al hacer 'left_join' de x con y, la tabla se quedará con los datos de x más aquellos de y que coincidan.

Si la columna que se usa para unir las tablas tiene diferente nombre en cada data frame, puedes especificar los nombres en el argumento by: by = c("columna_df1" = "columna_df2").

Para ver un ejemplo sencillo utilizaremos dos data.frame o tablas, creadas para estos efectos. *df1* y *df2*

```
df1 <- data.frame(
  ID = c(1, 2, 3, 4),
  nombre = c("Ana", "Juan", "Pedro", "Maria"),
  edad = c(20, 22, 21, 23)
)

df2 <- data.frame(
  ID = c(1, 2, 5),
  calificacion = c(85, 90, 78)
)
```

1. Ahora haremos un cruce por la izquierda

```
left_join(df1, df2, by = "ID")
```

	ID	nombre	edad	calificacion
1	1	Ana	20	85
2	2	Juan	22	90
3	3	Pedro	21	NA
4	4	Maria	23	NA

- Conserva todas las filas de df1 (la izquierda).
- Agrega las columnas de df2 que coinciden con df1 según la columna "ID".
- Si no hay coincidencia en df2, las nuevas columnas tendrán valores NA.

2. Ahora uno por la derecha

```
right_join(df1, df2, by = "ID")
```

	ID	nombre	edad	calificacion
1	1	Ana	20	85
2	2	Juan	22	90
3	5	<NA>	NA	78

- Conserva todas las filas de df2 (la derecha).
 - Agrega las columnas de df1 que coinciden con df2 según la columna “ID”.
 - Si no hay coincidencia en df1, las nuevas columnas tendrán valores NA.
3. Un cruce que mantiene solo lo que está a la izquierda y derecha.

```
inner_join(df1, df2, by = "ID")
```

	ID	nombre	edad	calificacion
1	1	Ana	20	85
2	2	Juan	22	90

- Conserva solo las filas donde hay coincidencia en ambas tablas según la columna “ID”.
4. Un cruce que conserva todas las filas de ambas tablas

```
full_join(df1, df2, by = "ID")
```

	ID	nombre	edad	calificacion
1	1	Ana	20	85
2	2	Juan	22	90
3	3	Pedro	21	NA
4	4	Maria	23	NA
5	5	<NA>	NA	78

- Conserva todas las filas de ambas tablas.
 - Si no hay coincidencia, las columnas de la otra tabla tendrán valores NA.
5. El anti_join devuelve las filas de la primera tabla (df1 en este caso) que no tienen coincidencias en la segunda tabla (df2) según la columna especificada en by (en este caso, “ID”).

```
anti_join(df1, df2, by = "ID")
```

	ID	nombre	edad
1	3	Pedro	21
2	4	Maria	23

Esta función es útil para:

- Identificar las filas que faltan en una tabla con respecto a otra.
- Filtrar las filas de una tabla que no se encuentran en otra.
- Encontrar valores únicos en una tabla que no están presentes en otra.

Referencias

A continuación podrás revisar fuentes útiles y referencias.

Referencias principales para Aprender R

Aquí tienes una lista de recursos esenciales para aprender R, desde libros y tutoriales en línea hasta cursos interactivos y canales de YouTube.

Libros

1. **R for Data Science** por Hadley Wickham y Garrett Grolemund
 - [Libro en línea gratuito](#) Un recurso excelente para principiantes y usuarios intermedios que cubre conceptos de manipulación, visualización y modelado de datos con R.
2. **Advanced R** por Hadley Wickham
 - [Libro en línea gratuito](#) Ideal para aquellos que ya tienen conocimientos básicos de R y desean profundizar en aspectos más avanzados del lenguaje.
3. **El arte de programar en R** por Julio Santana y Efraín Mateos
 - [Enlace en CRAN](#) Cubre una amplia gama de temas, desde los fundamentos hasta técnicas avanzadas de programación en R.

Tutoriales en Línea

1. **Swirl - Learn R, in R**
 - [Swirl](#) Un paquete de R que ofrece cursos interactivos directamente en la consola de R.
2. **Coursera: R Programming**

- [Curso en Coursera](#) Un curso ofrecido por la Universidad Johns Hopkins que es parte de la especialización en Ciencia de Datos.

3. **Datacamp: Introduction to R**

- [Curso en Datacamp](#) Curso gratuito introductorio que cubre los conceptos básicos de R.

Canales de YouTube

1. **Data School**

- [Data School](#) Ofrece tutoriales detallados sobre R y técnicas de ciencia de datos.

2. **R Programming for Data Science**

- [R Programming for Data Science](#) Enfocado en la programación en R para ciencia de datos, con numerosos ejemplos prácticos.

3. **StatQuest with Josh Starmer**

- [StatQuest](#) Explicaciones claras y simples sobre estadísticas y R.

Documentación Oficial

1. **The Comprehensive R Archive Network (CRAN)**

- [CRAN](#) La fuente oficial de la documentación de R, incluyendo el manual del usuario y otros recursos útiles.

2. **RStudio Documentation**

- [RStudio Documentation](#) Documentación y guías sobre cómo usar RStudio, el IDE más popular para R.

Blogs y Comunidades

1. **R-Bloggers**

- [R-Bloggers](#) Una agregación de blogs sobre R que cubren una amplia gama de temas, desde tutoriales hasta análisis avanzados.

2. **Stack Overflow**

- [Stack Overflow](#) Una comunidad de preguntas y respuestas donde puedes encontrar soluciones a problemas comunes y preguntar sobre tus propios desafíos en R.

Recursos Adicionales

1. Cheat Sheets

- [RStudio Cheat Sheets](#) Hojas de referencia rápidas para diversos paquetes y funciones de R, proporcionadas por RStudio.

2. Tidyverse

- [Tidyverse](#) Una colección de paquetes de R diseñados para la ciencia de datos, que incluyen `ggplot2`, `dplyr`, `tidyr`, y más.