Curso básico de análisis de datos con R

Con ejemplos aplicados a las Ciencias de la Salud





Indice de contenidos

Pr	efaci		e este manual	
			e este manual	
1	Intr	oducció	naR 5	
•	1.1		ación de R	
	1.2		nos de desarrollo	
2	Tipo	os de da	atos simples 7	,
	2.1		rsión de tipos)
	2.2		ciones con números)
		2.2.1	Operadores aritméticos)
		2.2.2	Operadores relacionales)
		2.2.3	Funciones y constantes numéricas)
	2.3	Operac	ciones con cadenas	3
		2.3.1	Funciones de cadenas	3
		2.3.2	Operaciones de comparación de cadenas	Į
	2.4	Operac	ciones con datos lógicos o booleanos	,)
		2.4.1	Operadores lógicos	,)
	2.5	Variab	les	;
		2.5.1	Prioridad de los operadores	3
	2.6	Ejercio	cios)
3	Tipo	os de da	atos estructurados 21	L
	3.1	Vector	es	L
		3.1.1	Creación de vectores	L
		3.1.2	Tamaño de un vector	3
		3.1.3	Coerción de elementos	3
		3.1.4	Acceso a los elementos de un vector	Į
		3.1.5	Pertenencia a un vector	;
		3.1.6	Modificación de los elementos de un vector	7
		3.1.7	Añadir elementos a un vector	7
		3.1.8	Eliminar elementos de un vector	3
		3.1.9	Eliminación de un vector	3
		3.1.10	Operaciones aritméticas con vectores)

	3.2	Factor	es
		3.2.1	Operaciones con factores
		3.2.2	Acceso a los elementos de un factor
		3.2.3	Modificación de los elementos de un factor
	3.3	Listas	33
		3.3.1	Creación de listas
		3.3.2	Tamaño de una lista
		3.3.3	Acceso a los elementos de una lista
		3.3.4	Modificación de los elementos de una lista
		3.3.5	Añadir elementos a una lista
		3.3.6	Conversión de una lista en un vector
	3.4	Matrio	es
		3.4.1	Creación de matrices
		3.4.2	Tamaño y dimensiones de una matriz
		3.4.3	Acceso a los elementos de una matriz
		3.4.4	Pertenencia a una matriz
		3.4.5	Modificación de los elementos de una matriz
		3.4.6	Añadir elementos a una matriz
		3.4.7	Trasponer una matriz
		3.4.8	Operaciones aritméticas con matrices
		3.4.9	Determinante de una matriz
		3.4.10	Inversa de una matriz
		3.4.11	Autovalores y autovectores de una matriz
	3.5	Data f	rames
		3.5.1	Creación de un data frame
		3.5.2	Coerción de otras estructuras de datos a data frames
		3.5.3	Acceso a los elementos de un data frame
		3.5.4	Modificación de los elementos de un data frame
		3.5.5	Añadir elementos a un data frame
		3.5.6	Eliminar filas y columnas de un data frame 60
	3.6	Ejercio	cios
4	Fstr	ucturas	de control 64
•	4.1		turas condicionales
		4.1.1	Condicionales (if)
		4.1.2	La función switch()
	4.2	Bucles	
		4.2.1	Bucles iterativos (for)
		4.2.2	Bucles condicionales (while)
		4.2.3	La instrucción break
		4.2.4	La instrucción next
	4.3	Ejercio	

5	Fund	ciones	78
	5.1	Definición y llamada a funciones	78
	5.2	Parámetros y argumentos de una función	79
		5.2.1 Paso de argumentos a una función	79
		5.2.2 Argumentos por defecto	80
	5.3	Retorno de una función	80
	5.4	Entorno y ámbito de las variables	82
	5.5	Componentes de una función	84
	5.6	Funciones recursivas	85
	5.7	Paquetes	86
		5.7.1 Instalación de paquetes	86
		5.7.2 Carga de un paquete	88
		5.7.3 Paquetes habituales	88
	5.8	Ejercicios	89
6	Prep	procesamiento de datos	91
	6.1	La colección de paquetes tidyverse	91
	6.2	Tibbles	92
	6.3	Conjuntos de datos ordenados	93
	6.4		95
	6.5	Conteo del número de observaciones	95
	6.6	Selección de variables	96
	6.7	Filtrado de datos	98
	6.8	Reordenación de datos	00
	6.9	Renombrado de columnas	
	6.10	Creación de nuevas columnas o transformación de las existentes 1	03
	6.11	Resumen de datos	04
		Resúmenes por grupos	
		Composición de operaciones mediante tuberías	
	6.14	Ejercicios	06
7	Gráf	J	10
	7.1	Gramática de gráficos y el paquete ggplot2	10
	7.2	Inicialización de un gráfico	11
	7.3	Diagramas de puntos	12
	7.4	Diagramas de líneas	17
	7.5	Diagramas de barras	20
	7.6	Histogramas	27
	7.7	Diagramas de densidad	31
	7.8	Diagramas de cajas	33
	7.9	Diagrama de sectores	35
	7.10	Interpolación y ajustes de regresión	36
	7.11	Facetas	39

	7.12	Personalización de gráficos	
		7.12.1 Títulos	
		7.12.2 Escalas	
		7.12.3 Temas	
	7.13	Ejercicios	
8	Anál	sis Estadísticos 152	
	8.1	Una variable cuantitativa	
		8.1.1 Estudios descriptivos	
		8.1.2 Estudios inferenciales	
	8.2	Una variable cualitativa	
		8.2.1 Estudios descriptivos	
		8.2.2 Estudios inferenciales	
	8.3	Dos variables: Variable dependiente cuantitativa y variable independiente	
		culitativa con dos categorías o grupos	
		8.3.1 Estudios descriptivos	
		8.3.2 Estudios inferenciales	
	8.4	Dos variables: Variable dependiente cuantitativa y variable independiente	
		culitativa con dos categorías o grupos pareados	
		8.4.1 Estudios descriptivos	
		8.4.2 Estudios inferenciales	
	8.5	Dos variables: Variable dependiente cuantitativa y variable independiente	
		culitativa con más de dos categorías o grupos	
		8.5.1 Estudios descriptivos	
		8.5.2 Estudios inferenciales	
	8.6	Dos variables: Variable dependiente cuantitativa y variable independiente	
		cuantitativa	
		8.6.1 Estudios descriptivos	
		8.6.2 Estudios inferenciales	
	8.7	Ejercicios	

Prefacio

Propósito de este manual

Este manual proporciona una introducción amigable al lenguaje de programación R para aquellas personas interesadas en utilizar este lenguaje para el análisis de datos.

El manual empieza con los conceptos básicos del lenguaje de programación R pero enseguida aborda su uso para la visualización y el análisis estadístico de datos, haciendo un recorrido por los test estadísticos más comunes.

Lo más interesante de este manual es la multitud de ejemplos que ilustran el uso de las técnicas estadísticas presentadas, así como los problemas propuestos.

El manual no aborda los fundamentos matemáticos de los análisis estadísticos presentados, aunque si explica brevemente cuándo deben usarse y cuándo no, así como las interpretaciones de los resultados obtenidos en los ejemplos. Si alguien está interesado en profundizar en los detalles matemáticos, puede visitar esta página.

No es un curso de programación en R, sino de uso de sus funciones predefinidas y de los paquetes más habituales para el análisis de datos.

Para cualquier comentario o sugerencia sobre este manual escriba al autor (asalber@ceu.es).

Licencia

Esta obra está bajo una licencia Reconocimiento – No comercial – Compartir bajo la misma licencia 3.0 España de Creative Commons. Para ver una copia de esta licencia, visite https://creativecommons.org/licenses/by-nc-sa/3.0/es/.

Con esta licencia eres libre de:

- Copiar, distribuir y mostrar este trabajo.
- Realizar modificaciones de este trabajo.

Bajo las siguientes condiciones:

- Reconocimiento. Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).
- No comercial. No puede utilizar esta obra para fines comerciales.
- Compartir bajo la misma licencia. Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.

Estas condiciones pueden no aplicarse si se obtiene el permiso del titular de los derechos de autor.

Nada en esta licencia menoscaba o restringe los derechos morales del autor.

1 Introducción a R

La gran potencia de cómputo alcanzada por los ordenadores ha convertido a los mismos en poderosas herramientas al servicio de todas aquellas disciplinas que, como la Estadística, requieren manejar un gran volumen de datos. Actualmente, prácticamente nadie se plantea hacer un estudio estadístico serio sin la ayuda de un buen programa de análisis de datos.

R es un potente lenguaje de programación que incluye multitud de funciones para la representación y el análisis de datos. Fue desarrollado en 1993 por Robert Gentleman y Ross Ihaka en la Universidad de Auckland en Nueva Zelanda, aunque actualmente es mantenido por una enorme comunidad científica en todo el mundo.



Figura 1.1: Logotipo de R

Las ventajas de R frente a otros programas habituales de análisis de datos, como pueden ser SPSS, SAS o Matlab, son múltiples:

- Es software libre y por tanto gratuito. Puede descargarse desde la web http://www.r-project.org/.
- Es multiplataforma. Existen versiones para Windows, Macintosh, Linux y otras plataformas.
- Es un lenguaje maduro.
- Está avalado y en constante desarrollo por una amplia comunidad científica distribuida por todo el mundo que lo utiliza como estándar para el análisis de datos.
- Cuenta con multitud de paquetes para todo tipo de análisis estadísticos y representaciones gráficas, desde los más habituales, hasta los más novedosos y sofisticados que no incluyen otros programas. Los paquetes están organizados y documentados en un repositorio CRAN (Comprehensive R Archive Network) desde donde pueden descargarse libremente.

• Diseñado para el análisis la representación de datos.

Existen multitud de libros, manuales y tutoriales libres que permiten su aprendizaje e ilustran el análisis estadístico de datos en distintas disciplinas científicas como las Matemáticas, la Física, la Biología, la Psicología, la Medicina, etc.

1.1 Instalación de R

R puede descargarse desde el sitio web oficial de R o desde el repositorio principal de paquetes de R CRAN. Basta con descargar el archivo de instalación correspondiente al sistema operativo de nuestro ordenador y realizar la instalación como cualquier otro programa.

El intérprete de R se arranca desde la terminal, aunque en Windows incorpora su propia aplicación, pero es muy básica. En general, para trabajos serios, conviene utilizar un entorno de desarrollo para R.

1.2 Entornos de desarrollo

Por defecto el entorno de trabajo de R es en línea de comandos, lo que significa que los cálculos y los análisis se realizan mediante comandos o instrucciones que el usuario teclea en una ventana de texto. No obstante, existen distintas interfaces gráficas de usuario que facilitan su uso, sobre todo para usuarios noveles. Algunas de ellas, como las que se enumeran a continuación, son completos entornos de desarrollo que facilitan la gestión de cualquier proyecto:

- RStudio. Probablemente el entorno de desarrollo más extendido para programar con R ya que incorpora multitud de utilidades para facilitar la programación con R.
- RKWard. Es otra otro de los entornos de desarrollo más completos que además incluye a posibilidad de añadir nuevos menús y cuadros de diálogo personalizados.
- Visual Studio Code. Es un entorno de desarrollo de propósito general ampliamente extendido. Aunque no es un entorno de desarrollo específico para R, incluye una extensión con utilidades que facilitan mucho el desarrollo con R.

2 Tipos de datos simples

En R existen distintos tipos de datos simples predefinidos:

• numeric: Es el tipo de los números. Secuencia de dígitos (pueden incluir el - para negativos y el punto como separador de decimales) que representan números. Por ejemplo, 1, -2.0, 3.1415 o 4.5e3.

Por defecto, cualquier número que se teclee tomará este tipo.

- integer: Es el tipo de los números enteros. Secuencia de dígitos sin separador de decimales que representan un número entero. Por ejemplo 1 o -2. Son una subclase del tipo de datos numérico.
- double: Es el tipo de los números reales. Secuencia de dígitos que incluyen decimales separados por punto. Por ejemplo 3.1415 o -2.0. Son una subclase del tipo de datos numérico.
- character: Es el tipo de las cadenas de caracteres. Secuencia de caracteres alfanuméricos que representan texto. Se escriben entre comillas simples o dobles. Por ejemplo "Hola" o 'Hola'.
- logical: Es el tipo de los booleanos. Puede tomar cualquiera de los dos valores lógicos TRUE (verdadero) o FALSE (falso). También se pueden abreviar como T o F.
- NA: Se utiliza para representar datos desconocidos o perdidos. Aunque en realidad es un dato lógico, puede considerarse con un tipo de dato especial.
- NULL: Se utiliza para representar la ausencia de datos. La principal diferencia con NA es que NULL aparece cuando se intenta acceder a un dato que no existe, mientras que NA se utiliza para representar explícitamente datos perdidos en un estudio.

Para averiguar el tipo de un dato se puede utilizar la siguiente función:

• class(x): Devuelve el tipo del dato x.

Ejemplo 2.1. A continuación se muestran los tipos de algunos datos.

class(3.1415)

[1] "numeric"

```
class(-1)
[1] "numeric"

class("Hola")

[1] "character"

class(TRUE)

[1] "logical"

class(NA)

[1] "logical"

class(NULL)
```

También pueden utilizarse las siguientes funciones que devuelven un booleano:

- is.numeric(x): Devuelve el booleano TRUE si x es del tipo numeric.
- is.double(x): Devuelve el booleano TRUE si x es del tipo double.
- is.integer(x): Devuelve el booleano TRUE si x es del tipo integer.
- is.character(x): Devuelve el booleano TRUE si x es del tipo character.
- is.logical(x): Devuelve el booleano TRUE si x es del tipo logical.
- is.na(x): Devuelve el booleano TRUE si x es del tipo NA.
- is.null(x): Devuelve el booleano TRUE si x es del tipo NULL.

Ejemplo 2.2.

```
is.double(3.1415)

[1] TRUE

is.character(TRUE)
```

[1] FALSE

2.1 Conversión de tipos

En muchas ocasiones es posible convertir un dato de un tipo a otro distinto. Para ello se usan las siguientes funciones:

- as.numeric(x): Convierte el dato de x al tipo numeric siempre que sea posible o tenga sentido la conversión. Para convertir una cadena en un número, la cadena tiene que representar un número. El valor lógico TRUE se convierte en 1 y el FALSE en 0.
- as.integer(x): Convierte el dato de x al tipo integer siempre que sea posible o tenga sentido la conversión. Para convertir una cadena en un número entero, la cadena tiene que representar un número entero. El valor lógico TRUE se convierte en 1 y el FALSE en 0.
- as.character(x): Convierte el tipo de dato de x al tipo character simplemente añadiendo comillas.
- as.logical(x): Convierte el tipo de dato de x al tipo lógico. Para datos numéricos, el 0 se convierte en FALSE y cualquier otro número en TRUE. Para cadenas se obtiene NA excepto para las cadenas "TRUE" y "true" que se convierten a TRUE y las cadenas "FALSE" y "false" que se convierten a FALSE.

El tipo NA no se puede convertir a ningún otro tipo pues representa la ausencia del dato. Lo mismo ocurre con NULL.

2.2 Operaciones con números

2.2.1 Operadores aritméticos

Los siguientes operadores permiten realizar las clásicas operaciones aritméticas entre datos numéricos:

- x + y: Devuelve la suma de x e y.
- x y: Devuelve la resta de x e y.
- x * y: Devuelve el producto de x e y.
- x / y: Devuelve el cociente de x e y.
- x %% y: Devuelve el resto de la división entera de x e y.
- x ^ y: Devuelve la potencia x elevado a y.

Ejemplo 2.3.

```
2 + 3

[1] 5

5 * -2

[1] -10

5 / 2

[1] 2.5

1 / 0

[1] Inf

5 % 2

[1] 1

2 ^ 3

[1] 8
```

2.2.2 Operadores relacionales

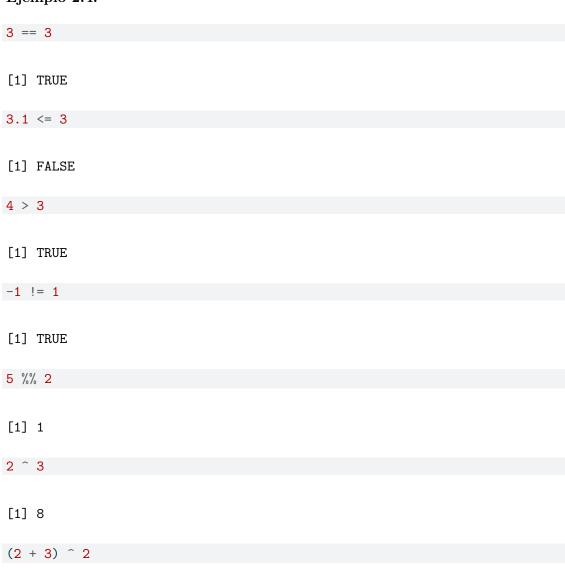
Comparan dos números y devuelven un valor lógico.

- $\bullet\,$ x == y: Devuelve TRUE si el número x es igual que el número y, y FALSE en caso contrario.
- $\bullet\,$ x > y: Devuelve TRUE si el número x es mayor que el número y, y FALSE en caso contrario.
- x < y: Devuelve TRUE si el número x es menor que el número y, y FALSE en caso contrario.
- $x \ge y$: Devuelve TRUE si el número x es mayor o igual que el número y, y FALSE en caso contrario.
- $x \le y$: Devuelve TRUE si el número x es menor o igual a que el número y, y FALSE en caso contrario.

 $\bullet\,$ x $\,!=\,$ y: Devuelve TRUE si el número x es distinto del número y, y FALSE en caso contrario.

Ejemplo 2.4.

[1] 25



2.2.3 Funciones y constantes numéricas

Las siguientes constantes y funciones matemáticas también están ya predefinidas.

- pi: Devuelve el número π .
- sqrt(x): Devuelve la raíz cuadrada de x.
- abs(x): Devuelve el valor absoluto de x.
- round(x, n): Devuelve el redondeo de x a n decimales.
- $\exp(x)$: Devuelve la exponencial de $x(e^x)$.
- log(x): Devuelve el logaritmo neperiano de x.
- sin(x): Devuelve el seno del ángulo x en radianes.
- cos(x): Devuelve el coseno del ángulo x en radianes.
- tan(x): Devuelve la tangente del ángulo x en radianes.
- asin(x): Devuelve el arcoseno de x.
- acos(x): Devuelve el arcocoseno de x.
- atan(x): Devuelve el arcotangente de x.

Ejemplo 2.5.

```
sqrt(9)
[1] 3
abs(-1)
[1] 1
round(1.7)
[1] 2
exp(1)
```

[1] 1

log(exp(1))

```
cos(pi)
[1] -1
asin(1)
```

[1] 1.570796

2.3 Operaciones con cadenas

2.3.1 Funciones de cadenas

Existen muchas funciones para cadenas de texto pero las más comunes son:

- nchar(c): Devuelve el número de caracteres de la cadena.
- paste(x, y, ..., sep=s): Concatena las cadenas x, y, etc. separándolas por la cadena s. Por defecto la cadena de separación es un espacio en blanco.
- substr(c, start=i, stop=j): Devuelve la subcadena de la cadena c desde la posición i hasta la posición j. El primer carácter de una cadena ocupa la posición 1.
- tolower(c): Devuelve la cadena que resulta de convertir la cadena c a minúsculas.
- toupper(c): Devuelve la cadena que resulta de convertir la cadena c a mayúsculas.

Ejemplo 2.6.

```
nchar("Me gusta R")

[1] 10

paste("Me", "gusta", "R")

[1] "Me gusta R"

paste("Me", "gusta", "R", sep = "-")

[1] "Me-gusta-R"
```

```
paste("Me", "gusta", "R", sep = "")

[1] "MegustaR"

substr("Me gusta R", 4, 8)

[1] "gusta"

tolower("Me gusta R")

[1] "me gusta r"

toupper("Me gusta R")
```

[1] "ME GUSTA R"

2.3.2 Operaciones de comparación de cadenas

- x == y: Devuelve TRUE si la cadena x es igual que la cadena y, y FALSE en caso contrario.
- \bullet x > y : Devuelve TRUE si la cadena x sucede a la cadena y, y FALSE en caso contrario.
- x < y: Devuelve TRUE si la cadena x antecede a la cadena y, y FALSE en caso contrario.
- $x \ge y$: Devuelve TRUE si la cadena x sucede o es igual a la cadena y, y FALSE en caso contrario.
- $x \le y$: Devuelve TRUE si la cadena x antecede o es igual a la cadena y, y FALSE en caso contrario.
- x != y : Devuelve TRUE si la cadena x es distinta de la cadena y, y FALSE en caso contrario.

Utilizan el orden alfabético, las minúsculas van antes que las mayúsculas, y los números antes que las letras.

Ejemplo 2.7.

```
"R" == "R"
```

[1] TRUE

```
"R" == "r"

[1] FALSE

"uno" < "dos"

[1] FALSE

"A" > "a"

[1] TRUE

"" < "R"
```

2.4 Operaciones con datos lógicos o booleanos

2.4.1 Operadores lógicos

[1] TRUE

A la hora de comparar valores lógicos R asocia a TRUE el valor 1 y a FALSE el valor 0.

- x == y: Devuelve TRUE si los booleanos x y y son iguales, y FALSE en caso contrario.
- x < y: Devuelve TRUE si el booleano x es menor que el booleano y, y FALSE en caso contrario.
- $x \le y$: Devuelve TRUE si el booleano x es menor o igual que el booleano y, y FALSE en caso contrario.
- x > y: Devuelve TRUE si el booleano x es mayor que el booleano y, y FALSE en caso contrario.
- $x \ge y$: Devuelve TRUE si el booleano x es mayor o igual que el booleano y, y FALSE en caso contrario.
- x != y: Devuelve TRUE si el booleano x es distinto que el booleano y, y FALSE en caso contrario.
- Negación !b: Devuelve TRUE si el booleano b es FALSE, y FALSE si es TRUE.
- Conjunción x & y: Devuelve TRUE si los booleanos x, y y son TRUE y FALSE en caso contrario.
- Disyunción $x \mid y$: Devuelve TRUE si alguno de los booleanos x o y son TRUE, y FALSE en caso contrario.

Tabla de verdad

Х	у	!x	х & у	хІу
FALSE	FALSE	TRUE	FALSE	FALSE
FALSE	TRUE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	FALSE	TRUE
TRUE	TRUE	FALSE	TRUE	TRUE

Ejemplo 2.8.

!TRUE

[1] FALSE

FALSE | TRUE

[1] TRUE

FALSE | FALSE

[1] FALSE

TRUE & FALSE

[1] FALSE

TRUE & TRUE

[1] TRUE

2.5 Variables

Una variable es un identificador ligado a un valor.

Reglas para nombrar variables

- Comienzan siempre por una letra o punto, seguida de otras letras, números, puntos o guiones bajos. Si empieza por punto no puede seguirle un número.
- No se pueden utilizarse palabras reservadas del lenguaje.

A diferencia de otros lenguajes de programación, las variables no tienen asociado un tipo de dato y no es necesario declararlas antes de usarlas (tipado dinámico).

Para asignar un valor a una variable se utiliza el operador de asignación <-:

• x <- y: Asigna el valor y a la variable x.

Aunque es menos común también se puede utilizar el operador =.

Se puede crear una variable sin ningún valor asociado asignándole el valor NULL.

Una vez definida una variable, puede utilizarse en cualquier expresión donde tenga sentido el valor que tiene asociado.

Si una variable ya no va a usarse, es posible eliminarla y liberar el espacio que ocupan sus datos asociados con la siguiente función:

• rm(x): Elimina la variable x.

Ejemplo 2.9.

```
x <- 3
x

[1] 3

y <- x + 2
y

[1] 5

# Valor no definido
x <- NULL</pre>
```

NULL

```
# Eliminar y
rm(y)
# A partir de aquí, una llamada a y produce un error.
```

2.5.1 Prioridad de los operadores

Al evaluar una expresiones R utiliza el siguiente orden de prioridad de evaluación:

1	Funciones predefinidas
2	Potencias (^)
3	Productos y cocientes (*, /, %%)
4	Sumas y restas (+, -)
5	Operadores relacionales (==, >, <, >=, <=, !=)
6	Negación (!)
7	Conjunción (&)
8	Disyunción (1)
9	Asignación (<-)

Se puede saltar el orden de evaluación utilizando paréntesis ().

Ejemplo 2.10.

```
4 + 8 / 2 ^ 2
```

[1] 6

[1] 20

[1] 3

[1] 64

```
x <- 2
y <- 3
z <- ! x + 1 > y & y * 2 < x ^ 3
z
```

[1] TRUE

2.6 Ejercicios

Ejercicio 2.1. Se dispone de los siguientes datos de una persona:

Variable	Valor
edad	20
estatura	165
peso	60
sexo	mujer

a. Declarar las variables anteriores y asignarles los valores correspondientes.

```
i Solución

# Declaración de variables
edad <- 20
estatura <- 165
peso <- 60
sexo <- "mujer"
```

b. Definir la variable numérica imc con el índice de masa corporal aplicando la siguiente fórmula a las variables anteriores:

$$imc = \frac{peso (kg)}{estatura (m)^2}$$

```
i Solución

# Cálculo del índice de masa corporal
imc <- peso / (estatura / 100) ^ 2
imc

[1] 22.03857
```

c. Mostrar por pantalla el indice de masa corporal calculado en el apartado anterior redondeado a dos decimales y con sus unidades en mayúsculas.

```
i Solución

# Salida por pantalla
paste("Índice de masa corporal: ", round(imc,2), toupper("KG/M2"))

[1] "Índice de masa corporal: 22.04 KG/M2"
```

d. Definir la variable booleana obesa con el valor correspondiente a la siguiente condición: ser mujer y no tener una edad superior a 60 y tener un índice de masa corporal mayor o igual que 30. ¿Es esta persona obesa?

```
# Cálculo de la obesidad
obesa <- sexo == "mujer" & ! edad > 60 & imc >= 30
obesa
[1] FALSE
```

3 Tipos de datos estructurados

Los tipos estructurados de datos, a diferencia de los simples, son colecciones de datos con una determinada estructura. En R existen varios tipos tipos estructurados de datos que pueden clasificarse de acuerdo a su dimensión y a si son homogéneos (todos sus elementos son del mismo tipo) o heterogéneos.

Dimensiones	Homogéneos	Heterogéneos
1	Vector	Lista
2	Matriz	Data frame
n	Array	

Para averiguar la estructura de un dato estructurado se puede utilizar la función siguiente:

• str(x): Devuelve una cadena de texto con la estructura de x en un formato amigable para las personas.

3.1 Vectores

El vector es el tipo de dato estructurado más básicos en R. Un vector es una colección ordenada de elementos del mismo tipo.

3.1.1 Creación de vectores

Para construir un vector se utiliza la función de combinación c():

• c(x1, x2, ...): Devuelve el vector formado por los elementos x1, x2, etc.

También es posible utilizar el operador : para generar un vector de números enteros consecutivos:

• x:y: Devuelve el vector de números enteros consecutivos desde x hasta y.

Ejemplo 3.1.

```
c(1, 2, 3)

[1] 1 2 3

c("uno", "dos", "tres")

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

# Vector vacío
c()

NULL

# Vector con elementos perdidos
c(1, NA, 3)

[1] 1 NA 3
```

[1] 2 3 4 5 6

2:6

3.1.1.1 Vectores con nombres

Es posible asignar un nombre a cada elemento de un vector. Los nombres son etiquetas de texto que se asocian a cada elemento. Para asociar un nombre a un elemento se utiliza la sintaxis nombre = valor, donde nombre es una cadena de caracteres y valor es el elemento del vector.

Vector de números enteros consecutivos del 2 al 6

Ejemplo 3.2.

```
c("Matemáticas" = 8.2, "Física" = 6.5, "Economía" = 4.5)
```

Matemáticas Física Economía 8.2 6.5 4.5 Para acceder a los nombres de un vector se utiliza la siguiente función:

• names(x): Devuelve un vector de cadenas de caracteres con los nombres de los elementos del vector x.

Ejemplo 3.3.

```
notas <- c("Matemáticas" = 8.2, "Física" = 6.5, "Economía" = 4.5)
names(notas)
```

[1] "Matemáticas" "Física" "Economía"

3.1.2 Tamaño de un vector

El número de elementos de un vector es su tamaño y puede averiguarse con la siguiente función.

• lenght(x): Devuelve el número de elementos del vector x.

Ejemplo 3.4.

```
length(c(1, 2, 3))
```

[1] 3

```
length(c())
```

[1] 0

3.1.3 Coerción de elementos

Puesto que los elementos de un vector tienen que ser del mismo tipo, cuando se crea un vector con datos de distintos tipos, la función c() los convertirá al mismo tipo, lo que se conoce como *coerción* de tipos. La coerción se produce de los tipos menos flexibles a los más flexibles: logical < integer < double < character.

Ejemplo 3.5.

```
c(1, 2.5)
```

[1] 1.0 2.5

```
c(FALSE, TRUE, 2)
[1] 0 1 2
c(FALSE, TRUE, 2, "tres")
```

"tres"

3.1.4 Acceso a los elementos de un vector

Para acceder a los elementos de un vector se utiliza un índice. Como veremos a continuación, este índice puede ser entero, lógico o de cadena de caracteres y se indica siempre entre corchetes [] a continuación del vector.

3.1.4.1 Acceso mediante un índice entero

Los elementos de un vector están ordenados y el acceso más simple a ellos es mediante su número de orden, es decir, indicando entre corchetes [] el entero que corresponde a su número de orden. Se puede acceder simultáneamente a varios elementos mediante un vector con sus números de orden.



Advertencia

[1] "FALSE" "TRUE"

En R los índices enteros para acceder a los elementos de un vector comienzan en 1, a diferencia de otros lenguajes de programación que empiezan en 0.

Ejemplo 3.6.

```
x \leftarrow c(2,4,6,8,10)
# Acceso al elemento que está en la tercera posición
x[3]
```

[1] 6

```
# Acceso a los elementos de las posiciones 2 y 4
x[c(2, 4)]
```

[1] 48

```
# Acceso a los elementos de la posición 2 a la 4 x[2:4]
```

[1] 4 6 8

```
# Acceso a todos los elementos excepto el primero y el cuarto x[c(-1, -4)]
```

[1] 4 6 10

3.1.4.2 Acceso mediante un índice lógico

Cuando se utiliza un índice lógico, se obtienen los elementos correspondientes a las posiciones donde está el valor booleano TRUE.

Ejemplo 3.7.

```
x \leftarrow c(2,4,6,8,10)
# Acceso al elemento que está en la tercera posición
x[c(F,F,T,F,F)]
```

[1] 6

```
# Acceso a los elementos de las posiciones 2 y 4 x[c(F,T,F,T,F)]
```

[1] 4 8

Esta forma de acceder es útil cuando se genera el vector de índices mediante operadores relacionales. Cuando se aplica un operador relacional a un vector se obtiene otro vector lógico que resulta de aplicar el operador relacional a cada uno de los elementos del vector. De esta manera se puede realizar filtros para obtener los elementos de un vector que cumplen una determinada condición.

Ejemplo 3.8.

```
x <- 1:6

x \% 2 == 0
```

[1] FALSE TRUE FALSE TRUE FALSE TRUE

```
# Filtrado de los valores pares x[x \% 2 == 0]
```

[1] 2 4 6

```
# Filtrado de los valores pares menores que 5 x[x \% 2 == 0 \& x < 5]
```

[1] 2 4

3.1.4.3 Acceso mediante un índice de cadena

Si los elementos de un vector tienen nombre, es posible acceder a ellos usando sus nombres como índices.

Ejemplo 3.9.

```
notas <- c("Matemáticas" = 8.2, "Física" = 6.5, "Economía" = 4.5)
notas["Física"]
```

Física

6.5

```
notas[c("Matemáticas", "Economía")]
```

```
Matemáticas Economía 8.2 4.5
```

3.1.5 Pertenencia a un vector

Para comprobar si un valor en particular es un elemento de un vector se puede utilizar el operador %in%:

• x %in% y: Devuelve el booleano TRUE si x es un elemento del vector y, y FALSE en caso contrario.

Ejemplo 3.10.

```
x <- 1:3
2 %in% x
```

[1] TRUE

```
4 %in% x
```

[1] FALSE

3.1.6 Modificación de los elementos de un vector

Para modificar uno o varios elementos de un vector basta con acceder a esos elementos y usar el operador de asignación para asignar nuevos valores.

- v[i] <- x: Asigna el dato x a la posición i del vector v.
- v[c(i,j,...)] <- x: Asigna el dato x a las posiciones i, j, etc. del vector v.

Ejemplo 3.11.

```
x \leftarrow c(1, 2, 3)

x[2] \leftarrow 0
```

[1] 1 0 3

```
x[c(1, 3)] <- 1
x
```

[1] 1 0 1

3.1.7 Añadir elementos a un vector

Para añadir nuevos elementos a un vector pueden usarse las siguientes funciones:

- c(x, y): Devuelve el vector que resulta de añadir al vector x los elementos del vector y.
- append(x, y, pos): Devuelve el vector que resulta de añadir al vector x los elementos del vector y, a continuación de la posición pos. El parámetro pos es opcional y si no se indica, los elementos de y se añaden al final de los de x.

Ejemplo 3.12.

```
x \leftarrow c(1, 2, 3)

c(x, c(4, 5))

[1] 1 2 3 4 5

append(x, c(4, 5), 2)
```

[1] 1 2 4 5 3

3.1.8 Eliminar elementos de un vector

Para eliminar los elementos que ocupan una determinada posición se utiliza el operador de acceso, es decir, los corchetes [] pero con los índices correspondientes a las posiciones a eliminar, en negativo.

Ejemplo 3.13.

```
x <- c("a", "b", "c", "d", "e")
x[-3]

[1] "a" "b" "d" "e"

x[-c(2,4)]

[1] "a" "c" "e"</pre>
```

3.1.9 Eliminación de un vector

Para eliminar los elementos de un vector basta con asignar NULL a la variable que lo contiene, pero si se quiere liberar la memoria que ocupa la variable se utiliza la función rm().

3.1.10 Operaciones aritméticas con vectores

3.1.10.1 Operaciones aritméticas elemento a elemento

Para vectores numéricos las operaciones aritméticas habituales se aplican elemento a elemento. Si los vectores tienen distinto tamaño, el tamaño del vector más pequeño se equipara al tamaño del mayor, reutilizando sus elementos, empezando por el primero.

Ejemplo 3.14.

```
x \leftarrow c(1, 2, 3)

y \leftarrow c(0, 1, -1)

x + y
```

[1] 1 3 2

```
x * y
```

```
[1] 0 2 -3
```

```
x / y
```

```
[1] Inf 2 -3
```

```
x ^ y
```

[1] 1.0000000 2.0000000 0.3333333

3.1.10.2 Producto escalar de vectores

Para calcular el producto escalar de dos vectores numéricos se utiliza el operador %*%. Si los vectores tienen distinto tamaño se produce un error.

Ejemplo 3.15.

```
x <- c(1, 2, 3)
y <- c(0, 1, -1)
x %*% y
```

3.2 Factores

3.2.1 Operaciones con factores

Un factor es una estructura de datos especial que se utiliza para representar categorías de variables cualitativas y por tanto puede tomar un conjunto finito de valores predefinidos conocido como *niveles* del factor.

Para definir un factor se utiliza la siguiente función:

• factor(x, levels = niveles): Crea un dato de tipo factor con los elementos del vector x. Los niveles del factor pueden indicarse mediante el parámetro levels, pasándole un vector con los valores posibles. Si no se indica el parámetro levels los niveles del factor se obtienen automáticamente a partir de los elementos del vector x (tantos niveles con valores distintos tenga).

Los factores son en realidad vectores de números enteros a los que se le añade un atributo especial para indicar los niveles del factor.

Ejemplo 3.16.

```
sexo <- factor(c("mujer", "hombre", "mujer"))
sexo

[1] mujer hombre mujer
Levels: hombre mujer

class(sexo)

[1] "factor"

str(sexo)

Factor w/ 2 levels "hombre", "mujer": 2 1 2

grupo.sanguineo <- factor(c("B", "A", "A"), levels = c("A", "B", "AB", "O"), )
grupo.sanguineo</pre>
[1] B A A
Levels: A B AB O
```

Es posible establecer un orden entre los niveles de un factor añadiendo el parámetro ordered = TRUE a la función anterior. Esto es útil para representar categorías ordinales entre las que existe un orden natural.

Ejemplo 3.17.

```
nivel.estudio <- factor(c("Secundarios", "Graduado", "Bachiller"), levels = c("Sin estudio")</pre>
```

[1] Secundarios Graduado Bachiller

Levels: Sin estudios < Primarios < Secundarios < Bachiller < Graduado

Para comprobar si una estructura es del tipo factor se utiliza la siguiente función:

• is.factor(x): Devuelve el booleano TRUE si x es del tipo factor, y FALSE en caso contrario.

3.2.2 Acceso a los elementos de un factor

Se puede acceder a los elementos de un factor de la misma manera que se accede a los elementos de un vector. Y para obtener sus niveles se utiliza la siguiente función:

• levels(x): Devuelve un vector con los niveles del factor x.

Ejemplo 3.18.

```
sexo <- factor(c("mujer", "hombre", "mujer"))
sexo[2]</pre>
```

[1] hombre

Levels: hombre mujer

```
sexo[c(1, 2)]
```

[1] mujer hombre Levels: hombre mujer

```
sexo[-2]
```

[1] mujer mujer

Levels: hombre mujer

```
levels(sexo)
```

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

3.2.3 Modificación de los elementos de un factor

Se puede modificar los elementos de un factor de manera similar a como se modifican los elementos de un vector, es decir accediendo al elemento que se quiere modificar y asignándole un nuevo valor. La única diferencia con los vectores es que si el nuevo valor que se quiere asignar no está entre los niveles del factor, se obtiene el valor NA.

Ejemplo 3.19. A continuación se muestran varios de modificación de los elementos de un factor.

```
grupo.sanguineo <- factor(c("B", "A", "A"), levels = c("A", "B", "AB", "0"))
grupo.sanguineo

[1] B A A
Levels: A B AB 0

grupo.sanguineo[2] <- "AB"
grupo.sanguineo

[1] B AB A
Levels: A B AB 0</pre>
grupo.sanguineo[1] <- "C"
```

Warning in `[<-.factor`(`*tmp*`, 1, value = "C"): invalid factor level, NA generated

```
grupo.sanguineo
```

```
[1] <NA> AB A
Levels: A B AB 0
```

Obsérvese en el ejemplo anterior que cuando se intenta asignar un valor a un factor que no está entre sus niveles, se produce un error.

3.3 Listas

Las listas son colecciones ordenadas de elementos que pueden ser de distintos tipos. Los elementos de una lista también pueden ser de tipos estructurados (vectores o listas), lo que las convierte en el tipo de dato más versátil de R. Como veremos más adelante, otras estructuras de datos como los *data frames* o los propios modelos estadísticos se construyen usando listas.

3.3.1 Creación de listas

Para construir una lista se utiliza la función list():

• list(x1, x2, ...): Devuelve la lista con los elementos x1, x2, etc.

Ejemplo 3.20.

```
list(1, "dos", TRUE)
[[1]]
[1] 1
[[2]]
[1] "dos"
[[3]]
[1] TRUE
# Lista con vectores y listas
x <- list(1, c("dos", "tres"), list(4, "cinco"))
[[1]]
[1] 1
[[2]]
[1] "dos" "tres"
[[3]]
[[3]][[1]]
[1] 4
[[3]][[2]]
[1] "cinco"
```

List of 3 \$: num 1 \$: chr [1:2] "dos" "tres" \$:List of 2 ..\$: num 4 ..\$: chr "cinco" # Lista vacía list()

3.3.1.1 Listas con nombres

Al igual que con los vectores, es posible asignar un nombre a cada uno de los elementos de una lista.

Ejemplo 3.21.

[1] "Madrid"

list()

```
list("nombre" = "María", "edad" = 21, "dirección" = list("calle" = "Delicias", "número" :

$nombre
[1] "María"

$edad
[1] 21

$dirección
$dirección$calle
[1] "Delicias"

$dirección$número
[1] 24

$dirección$municipio
```

Para obtener los nombres de una lista se utiliza la siguiente función:

• names(x): Devuelve un vector de cadenas de caracteres con los nombres de los elementos de la lista x.

Ejemplo 3.22.

```
persona <- list("nombre" = "María", "edad" = 21, "dirección" = list("calle" = "Delicias"
names(persona)

[1] "nombre" "edad" "dirección"</pre>
```

3.3.2 Tamaño de una lista

El número de elementos de una lista es su $tama\~no$ y puede averiguarse con la siguiente función:

• lenght(x): Devuelve el número de elementos de la lista x.

Ejemplo 3.23.

```
length(list(1, "dos", TRUE))

[1] 3
length(list(1, c("dos", "tres"), list(4, "cinco")))

[1] 3
length(list())
```

[1] 0

3.3.3 Acceso a los elementos de una lista

Se accede a los elementos de una lista de forma similar a los vectores, mediante índices enteros, lógicos o de cadena, entre corchetes [].

3.3.3.1 Acceso mediante un índice entero

Al igual que los vectores, los elementos de una lista están ordenados y se puede utilizar un índice entero para acceder a los elementos que ocupan una determinada posición.

Ejemplo 3.24.

```
x <- list(1, "dos", TRUE, 4.5)
# Acceso al elemento que está en la segunda posición
x[2]
[[1]]
[1] "dos"
# Acceso a los elementos de las posiciones 1 y 3
x[c(1, 3)]
[[1]]
[1] 1
[[2]]
[1] TRUE
# Acceso a todos los elementos excepto el primero y el cuarto
x[c(-1, -4)]
[[1]]
[1] "dos"
[[2]]
[1] TRUE
```

3.3.3.2 Acceso mediante un índice lógico

Cuando se utiliza un índice lógico, se obtienen los elementos correspondientes a las posiciones donde está el valor booleano TRUE.

Ejemplo 3.25.

```
x <- list(1, "dos", TRUE, 4.5)
x[c(T,F,F,T)]

[[1]]
[1] 1

[[2]]
[1] 4.5

x < 2

Warning: NAs introducidos por coerción</pre>
```

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

```
# Filtrado de valores menores que 2 x[x < 2]
```

Warning: NAs introducidos por coerción

[[1]]

[1] 1

[[2]]

NULL

[[3]]

[1] TRUE

Obsérvese que para los elementos que no tiene sentido la comparación se obtiene NA, y que el acceso mediante este índice devuelve NULL.

3.3.3.3 Acceso mediante nombres

Si los elementos de una lista tienen nombre, se puede acceder a ellos utilizando sus nombres como índices. La única diferencia con el acceso mediante cadenas de vectores es que se obtiene siempre una lista, incluso cuando sólo se quiere acceder a un elemento. Para obtener un elemento, y no una lista con ese único elemento, se utilizan dobles corchetes [[]].

Ejemplo 3.26.

```
persona <- list("nombre" = "María", "edad" = 21, "dirección" = list("calle" = "Delicias"</pre>
persona[c("edad", "nombre")]
$edad
[1] 21
$nombre
[1] "María"
persona["nombre"]
$nombre
[1] "María"
typeof(persona["nombre"])
[1] "list"
# Acceso a un único elemento
persona[["nombre"]]
[1] "María"
# Acceso a una lista anidada
persona[["dirección"]][["municipio"]]
[1] "Madrid"
```

Una alternativa a los dobles corchetes es el operador de acceso a listas \$. Este operador además permite utilizar coincidencias parciales en los nombres de los elementos para acceder a ellos.

```
persona <- list("nombre" = "María", "edad" = 21, "dirección" = list("calle" = "Delicias"
# Acceso a un único elemento
persona$nombre</pre>
```

[1] "María"

```
# Acceso mediante coincidencia parcial
persona$nom

[1] "María"

# Acceso a una lista anidada
persona$dirección$municipio
```

[1] "Madrid"

3.3.4 Modificación de los elementos de una lista

Para modificar uno o varios elementos de una lista basta con acceder a esos elementos y reasignarles valores con el operador de asignación.

Ejemplo 3.27.

```
persona <- list("nombre" = "María", "edad" = 21)
persona$edad <- 22
persona
$nombre
[1] "María"
$edad
[1] 22</pre>
```

3.3.5 Añadir elementos a una lista

La forma más sencilla de añadir un elemento con nombre a una lista es indicando el nombre con el operador \$ y asignándole un valor con el operador de asignación <-:

• x\$nombre <- y: Añade el elemento y a la lista x con el nombre nombre.

El nuevo elemento se añade siempre al final de la lista.

Para añadir elementos sin nombre o en una posición determinada se puede utilizar la función append():

• append(x, y, pos): Devuelve la lista vector que resulta de añadir a x los elementos de la lista y, a continuación de la posición pos. El parámetro pos es opcional y si no se indica, los elementos de y se añaden al final de los de x.

Ejemplo 3.28.

```
persona <- list("nombre" = "María", "edad" = 21)</pre>
persona$email <- "maria@ceu.es"
persona
$nombre
[1] "María"
$edad
[1] 21
$email
[1] "maria@ceu.es"
append(persona, list("sexo" = "Mujer"), 2)
$nombre
[1] "María"
$edad
[1] 21
$sexo
[1] "Mujer"
$email
[1] "maria@ceu.es"
```

3.3.6 Conversión de una lista en un vector

Es posible convertir una lista en un vector con la siguiente función:

• unlist(x): Devuelve el vector que resulta de aplanar recursivamente la lista x y convertir todos los elementos al mismo tipo mediante coerción de tipos.

Ejemplo 3.29.

```
persona <- list("nombre" = "María", "edad" = 21, "dirección" = list("calle" = "Delicias"
unlist(persona)</pre>
```

```
dirección.calle
                                                                   dirección.número
             nombre
                                     edad
             "María"
                                     "21"
                                                    "Delicias"
                                                                                "24"
dirección.municipio
           "Madrid"
```

```
typeof(unlist(persona))
```

[1] "character"



🛕 Advertencia

Obsérvese que cuando se convierte una lista en un vector, los elementos de la lista se convierten al tipo más general mediante coerción.

3.4 Matrices

Una matriz es una estructura de datos bidimensional de elementos del mismo tipo organizados en filas y columnas. Una matriz es similar a un vector pero contiene una atributo adicional con sus dimensiones (número de filas y número de columnas).

3.4.1 Creación de matrices

Para crear una matriz se utiliza la siguiente función:

• matrix(x, nrow = m, ncol = n): Devuelve la matriz con los elementos del vector x organizados en n filas y m columnas. Habitualmente basta con especificar el número de filas o el número de columnas.

Ejemplo 3.30.

```
matrix(1:6, nrow = 2, ncol = 3)
      [,1] [,2] [,3]
[1,]
         1
              3
                    5
[2,]
              4
         2
                    6
matrix(1:6, nrow = 2)
```

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

```
matrix(1:6, ncol = 3)
```

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

```
# La matriz de 1 x 1
matrix()
```

Como se puede observar en el ejemplo anterior, los elementos se disponen por columnas, pero se pueden disponer los elementos por filas pasando el parámetro byrow = TRUE a la función matrix.

Ejemplo 3.31.

```
matrix(1:6, nrow = 2)
```

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

```
matrix(1:6, nrow = 2, byrow = TRUE)
```

3.4.1.1 Matrices con nombres de filas y columnas

Es posible poner nombres a las filas y a las columnas de una matriz añadiendo el parámetro dimnames y pasándole una lista de dos vectores de cadenas con los nombres de las filas y las columnas respectivamente.

Ejemplo 3.32.

```
matrix(1:6, nrow = 2, ncol = 3, dimnames = list(c("fila1", "fila2"), c("columna1", "columna1")
columna1 columna2 columna3
```

fila1 1 3 5 fila2 2 4 6

Para obtener los nombres de las filas y las columnas de una matriz se utilizan las siguientes funciones:

- rownames(x): Devuelve un vector de cadenas de caracteres con los nombres de las filas de la matriz x.
- colnames(x): Devuelve un vector de cadenas de caracteres con los nombres de las columnas de la matriz x.

Ejemplo 3.33.

```
x <- matrix(1:6, nrow = 2, ncol = 3, dimnames = list(c("fila1", "fila2"), c("columna1",
rownames(x)

[1] "fila1" "fila2"

colnames(x)</pre>
```

3.4.2 Tamaño y dimensiones de una matriz

[1] "columna1" "columna2" "columna3"

Para obtener el número de elementos y las dimensiones de una matriz se pueden utilizar las siguientes funciones:

- length(x): Devuelve un entero con el número de elementos de la matriz x.
- nrow(x): Devuelve un entero con el número de filas de la matriz x.
- ncol(x): Devuelve un entero con el número de columnas de la matriz x.

• dim(x): Devuelve un vector de dos enteros con el número de filas y el número de columnas de la matriz x.

Ejemplo 3.34.

```
x <- matrix(1:6, nrow = 2)
length(x)

[1] 6

nrow(x)

[1] 2

ncol(x)

[1] 3

dim(x)</pre>
```

[1] 2 3

Usando esta última función se pueden modificar las dimensiones de una matriz asignando un vector de dos enteros con las nuevas dimensiones. Esto también permite crear una matriz a partir de un vector.

Ejemplo 3.35.

```
x <- 1:6
dim(x) <- c(2, 3)
     [,1] [,2] [,3]
[1,]
              3
        1
[2,]
              4
                   6
        2
dim(x) <- c(3, 2)
     [,1] [,2]
[1,]
              4
        1
[2,]
        2
              5
[3,]
        3
              6
```

3.4.3 Acceso a los elementos de una matriz

Para acceder a los elementos de una matriz se utilizan dos índices (uno para las filas y otro para las columnas), separados por comas y entre corchetes [] a continuación de la matriz. Al igual que para los vectores, los índices pueden ser enteros, lógicos o de cadenas de caracteres.

3.4.3.1 Acceso mediante índices enteros

Para acceder a los elementos de una matriz mediante índices enteros se indica el número de fila y el número de columna del elemento entre corchetes:

• x[i,j]: Devuelve el elemento de la matriz x que está en la fila i y la columna j.

Se puede acceder a más de un elemento indicando un vector de enteros para las filas y otro para las columnas. De esta manera se obtiene una submatriz. Si no se indica la fila o la columna se obtienen todos los elementos de todas las filas o columnas. Al igual que para vectores, se pueden utilizar enteros negativos para descartar filas o columnas

Ejemplo 3.36.

```
x <- matrix(1:9, nrow = 3)
x</pre>
```

```
[,1] [,2] [,3]
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
```

```
# Acceso al elemento de la segunda fila y tercera columna x[2,3]
```

[1] 8

Acceso a la submatriz de la primera y tercera filas, y tercera y segunda columnas x[c(1, 3), c(3, 2)]

```
# Acceso a la primera fila x[1, ]
```

[1] 1 4 7

```
# Acceso a la segunda columna x[, 2]
```

[1] 4 5 6

Acceso a la submatriz con todos los elementos salvo la tercera fila y la segunda column x[-3, -2]

```
[,1] [,2]
[1,] 1 7
[2,] 2 8
```

3.4.3.2 Acceso mediante índices lógicos

Cuando se utilizan índices lógicos, se obtienen los elementos correspondientes a las filas y columnas donde está el valor booleano TRUE.

Ejemplo 3.37.

```
x <- matrix(1:9, nrow = 3)
x</pre>
```

```
[,1] [,2] [,3]
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
```

```
# Acceso al elemento de la segunda fila y tercera columna x[c(F, T, F), c(F, F, T)]
```

[1] 8

```
# Acceso a la submatriz de la primera y tercera filas, y segunda y tercera columnas x[c(T, F, T), c(F, T, T)]
```

```
[,1] [,2]
[1,] 4 7
[2,] 6 9
```

```
# Acceso a la primera fila
x[c(T, F, F), ]
```

[1] 1 4 7

```
# Acceso a la segunda columna
x[, c(F, T, F)]
```

[1] 4 5 6

3.4.3.3 Acceso mediante índices de cadena

Si las filas y las columnas de una matriz tienen nombre, es posible acceder a sus elementos usando los nombres de las filas y columnas como índices.

Ejemplo 3.38.

[1] 8

```
# Acceso a la submatriz de la primera y tercera filas, y tercera y segunda columnas x[c("f1", "f3"), c("c3", "c2")]
```

```
c3 c2
f1 7 4
f3 9 6
```

Finalmente, es posible combinar distintos tipos de índices (enteros, lógicos o de cadena) para indicar las filas y las columnas a las que acceder.

3.4.4 Pertenencia a una matriz

Para comprobar si un valor en particular es un elemento de una matriz se puede utilizar el operador %in%:

• x %in% y: Devuelve el booleano TRUE si x es un elemento de la matriz y, y FALSE en caso contrario.

Ejemplo 3.39.

```
x <- matrix(1:9, nrow = 3)
2 %in% x</pre>
```

[1] TRUE

```
-1 %in% x
```

[1] FALSE

3.4.5 Modificación de los elementos de una matriz

Para modificar uno o varios elementos de una matriz basta con acceder a esos elementos y usar el operador de asignación para asignar nuevos valores.

Ejemplo 3.40.

```
x <- matrix(1:9, nrow = 3)
x
```

```
[,1] [,2] [,3]
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
```

3.4.6 Añadir elementos a una matriz

Para añadir nuevas filas o columnas a una matriz se utilizan las siguientes funciones:

- rbind(x, y): Devuelve la matriz que resulta de añadir nuevas filas a la matriz x con los elementos del vector y.
- cbind(x, y): Devuelve la matriz que resulta de añadir nuevas columnas a la matriz x con los elementos del vector y.

Ejemplo 3.41.

```
x <- matrix(1:6, nrow = 2)
x</pre>
```

```
# Añadir una nueva fila
rbind(x, c(7, 8, 9))
```

```
[,1] [,2] [,3]
               3
[1,]
[2,]
         2
               4
                     6
[3,]
         7
               8
                     9
```

```
# Añadir una nueva columna
cbind(x, c(7, 8))
```

```
[,1] [,2] [,3] [,4]
[1,]
              3
        1
                    5
[2,]
        2
              4
                    6
                          8
```

Advertencia

Obsérvese que si el número de elementos proporcionados en el vector es menor del necesario para completar la fila o columna, se reutilizan los elementos del vector empezando desde el principio.

3.4.7 Trasponer una matriz

Para trasponer una matriz se utiliza la función siguiente:

• t(x): Devuelve la matriz traspuesta de la matriz x.

Ejemplo 3.42. A continuación se muestran un ejemplo de la trasposición de una matriz.

```
x <- matrix(1:6, nrow=2)</pre>
t(x)
```

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

3.4.8 Operaciones aritméticas con matrices

3.4.8.1 Operaciones aritméticas elemento a elemento

Para matrices numéricas las operaciones aritméticas habituales se aplican elemento a elemento. Si las dimensiones de las matrices son distintas se produce un error.

Ejemplo 3.43.

[2,]

2 -4

6

```
x \leftarrow matrix(1:6, nrow = 2)
     [,1] [,2] [,3]
[1,]
      1 3
[2,]
       2
            4
y \leftarrow matrix(c(0, 1, 0, -1, 0, 1), nrow = 2)
     [,1] [,2] [,3]
[1,]
       0 0 0
[2,]
      1 -1
x + y
     [,1] [,2] [,3]
[1,]
        1
            3
[2,]
        3
            3
                 7
x * y
     [,1] [,2] [,3]
[1,]
        0 0
[2,]
        2
          -4
x / y
     [,1] [,2] [,3]
[1,] Inf Inf
               Inf
```

x ^ y

```
[,1] [,2] [,3]
[1,]
        1 1.00
[2,]
        2 0.25
                    6
```

Advertencia

Obsérvese en el ejemplo anterior que la división por 0 produce el valor Inf que representa infinito.

3.4.8.2 Multiplicación de matrices

Para multiplicar dos matrices numéricas se utiliza el operador %*%.

Ejemplo 3.44.

```
x \leftarrow matrix(1:6, ncol = 3)
y <- matrix(1:6, nrow = 3)
x %*% y
```

y %*% x

```
[,1] [,2] [,3]
[1,]
         9
              19
                   29
[2,]
        12
              26
                   40
[3,]
                   51
        15
             33
```

🛕 Advertencia

Para poder multiplicar dos matrices deben tener dimensiones compatibles. Si el número de columnas de la primera matriz no es igual que el número de filas de la segunda se produce un error.

3.4.9 Determinante de una matriz

Para calcular el determinante de una matriz numérica cuadrada se utiliza la siguiente función:

• det(x): Devuelve el determinante de la matriz x. Si x no es una matriz numérica cuadrada produce un error.

Ejemplo 3.45.

```
x <- matrix(1:4, ncol = 2)
det(x)</pre>
```

[1] -2

3.4.10 Inversa de una matriz

Para calcular la matriz inversa de una matriz numérica cuadrada se utiliza la siguiente función:

• solve(x): Devuelve la matriz inversa de la matriz x. Si x no es una matriz numérica cuadrada produce un error. Si la matriz no es invertible por tener determinante nulo también se obtiene un error.

Ejemplo 3.46.

```
x <- matrix(1:4, nrow = 2)
solve(x)</pre>
```

```
[,1] [,2]
[1,] -2 1.5
[2,] 1 -0.5
```

```
# El producto de una matriz por su inversa es la matriz identidad. x \%*\% solve(x)
```

```
[,1] [,2]
[1,] 1 0
[2,] 0 1
```

3.4.11 Autovalores y autovectores de una matriz

Para calcular los autovalores y los autovectores de una matriz numérica cuadrada se utiliza la siguiente función:

• eigen(x): Devuelve una lista con los autovalores y los autovectores de la matriz x. Para acceder a los autovalores se utiliza el nombre values y para acceder a los autovectores se utiliza el nombre vectors. Si x no es una matriz numérica cuadrada produce un error.

Ejemplo 3.47.

[1]

```
x \leftarrow matrix(1:4, nrow = 2)
      [,1] [,2]
[1,]
         1
[2,]
               4
         2
# Autovalores
eigen(x)$values
```

```
5.3722813 -0.3722813
```

```
# Autovectores
eigen(x)$vectors
```

```
[,1]
                       [,2]
[1,] -0.5657675 -0.9093767
[2,] -0.8245648  0.4159736
```

3.5 Data frames

Un data frame es una estructura bidimensional cuyos elementos se organizan por filas y columnas de manera similar a una matriz. La principal diferencia con las matrices es que sus columnas están formadas por vectores, pero pueden tener tipos de datos distintos. Un data frame es un caso particular de lista formada por vectores del mismo tamaño con nombre.

Los data frames son las estructuras de datos más utilizadas en R para almacenar los datos en los análisis estadísticos.

3.5.1 Creación de un data frame

Para crear un data frame se utiliza la siguiente función:

• data.frame(nombrex = x, nombrey = y, ...): Devuelve el data frame con columnas los vectores x, y, etc. y nombres de columna nombrex, nombrey, etc.

Ejemplo 3.48.

```
df <- data.frame(asignatura = c("Matemáticas", "Física", "Economía"), nota = c(8.5, 7, 4
df
  asignatura nota
1 Matemáticas 8.5
2
       Física 7.0
    Economía 4.5
3
str(df)
'data.frame':
                3 obs. of 2 variables:
$ asignatura: chr
                   "Matemáticas" "Física" "Economía"
             : num 8.5 7 4.5
# Data frame vacío
data.frame()
```

data frame with 0 columns and 0 rows

Para grandes conjuntos de datos es más común crear un data frame a partir de un fichero en formato csv mediante la siguiente función:

- read.csv(f): Devuelve el data frame que se genera a partir de los datos del fichero csv f. Cada fila del fichero csv se corresponde con una fila del data frame y por defecto utiliza la coma , parara separar los datos de las columnas y punto . como separador de decimales de los datos numéricos. Los nombres de las columnas se obtienen automáticamente a partir de la primera fila del fichero.
- read.csv2(f): Funciona igual que la función anterior pero utiliza como separador de columnas el punto y coma ; y como separador de decimales la coma ,.

Ejemplo 3.49.

	nombre	edad	sexo	peso	altura	colesterol
1	José Luis Martínez Izquierdo	18	Н	85	1.79	182
2	Rosa Díaz Díaz	32	М	65	1.73	232
3	Javier García Sánchez	24	Н	NA	1.81	191
4	Carmen López Pinzón	35	М	65	1.70	200
5	Marisa López Collado	46	М	51	1.58	148
6	Antonio Ruiz Cruz	68	Н	66	1.74	249
7	Antonio Fernández Ocaña	51	Н	62	1.72	276
8	Pilar Martín González	22	М	60	1.66	NA
9	Pedro Gálvez Tenorio	35	Н	90	1.94	241
10	Santiago Reillo Manzano	46	Н	75	1.85	280
11	Macarena Álvarez Luna	53	М	55	1.62	262
12	José María de la Guía Sanz	58	Н	78	1.87	198
13	Miguel Angel Cuadrado Gutiérrez	27	Н	109	1.98	210
14	Carolina Rubio Moreno	20	M	61	1.77	194

3.5.2 Coerción de otras estructuras de datos a data frames

Para convertir otras estructuras de datos en data frames, se utiliza la siguiente función:

- as.data.frame(x): Devuelve el data frame que se obtiene a partir la estructura de datos x a plicanco las siguientes reglas de coerción:
 - Si x es un vector se obtiene un data frame con una sola columna.
 - Si x es una lista se obtiene un data frame con tantas columnas como elementos tenga la lista. Si los elementos de la lista tienen tamaños distintos se obtiene un error.
 - Si x es una matriz se obtiene un data frame con el mismo número de columnas y filas que la matriz.

3.5.3 Acceso a los elementos de un data frame

Puesto que un data frame es una lista, se puede acceder a sus elementos como se accede a los elementos de una lista utilizando índices. Con corchetes simples [] se obtiene siempre un data frame, mientras que con corchetes dobles [[]] o \$ se obtiene un vector. Pero también se puede acceder a los elementos de un data frame como si fuese una matriz, indicando un par de índices para las filas y las columnas respectivamente.

Ejemplo 3.50.

```
df <- data.frame(asignatura = c("Matemáticas", "Física", "Economía"), nota = c(8.5, 7, 4</pre>
   asignatura nota
1 Matemáticas 8.5
2
       Física 7.0
3
     Economía 4.5
# Acceso como lista
df["asignatura"]
   asignatura
1 Matemáticas
2
       Física
3
     Economía
df$asignatura
[1] "Matemáticas" "Física"
                                 "Economía"
# Acceso como matriz
df[2:3, "nota"]
[1] 7.0 4.5
df[df$nota >= 5, ]
   asignatura nota
1 Matemáticas 8.5
       Física 7.0
```

Obsérvese en el último ejemplo anterior cómo se pueden utilizar condiciones lógicas para filtrar un data frame.

Para acceder a las primeras o últimas filas de un data frame se pueden utilizar las siguientes funciones:

• head(df, n): Devuelve un data frame con las n primeras filas del data frame df.

• tail(df, n): Devuelve un data frame con las n últimas filas del data frame df.

Estas funciones son útiles para darse una idea del contenido de un data frame con muchas filas.

Ejemplo 3.51.

```
df <- data.frame(x = 1:26, y = letters) # letters es un vector predefinido con las letra:
head(df, 3)

x y
1 1 a
2 2 b
3 3 c

tail(df, 2)

x y
25 25 y
26 26 z</pre>
```

3.5.4 Modificación de los elementos de un data frame

Para modificar uno o varios elementos de un data frame basta con acceder a esos elementos y usar el operador de asignación para asignar nuevos valores.

Ejemplo 3.52.

```
df <- data.frame(asignatura = c("Matemáticas", "Física", "Economía"), nota = c(8.5, 7, 4
df

    asignatura nota
1 Matemáticas 8.5
2    Física 7.0
3    Economía 4.5

df[3, "nota"] <- 5
df

    asignatura nota
1 Matemáticas 8.5
2    Física 7.0
3    Economía 5.0</pre>
```

3.5.5 Añadir elementos a un data frame

Para añadir nuevas filas o columnas a una data frame se utilizan las mismas funciones que para matrices:

- rbind(df, x): Devuelve el data frame que resulta de añadir nuevas filas al data frame df con los elementos de la lista x.
- cbind(df, nombrex = x): Devuelve el data frame que resulta de añadir nuevas columnas al data frame df con los elementos del vector x con nombre nombrex.

Ejemplo 3.53.

```
df <- data.frame(asignatura = c("Matemáticas", "Física", "Economía"), nota = c(8.5, 7, 4
df
   asignatura nota
1 Matemáticas 8.5
2
       Física 7.0
3
     Economía 4.5
# Añadir una nueva fila
rbind(df, list("Programación", 10))
    asignatura nota
1 Matemáticas 8.5
2
        Física 7.0
      Economía 4.5
4 Programación 10.0
# Añadir una nueva columna
cbind(df, créditos = c(6, 4, 3))
   asignatura nota créditos
1 Matemáticas 8.5
                          4
2
       Física 7.0
3
    Economía 4.5
                          3
```

3.5.6 Eliminar filas y columnas de un data frame

Para eliminar una columna de un data frame basta con acceder a la columna y asignarle el valor NULL, mientras que para eliminar una fila basta con acceder a la fila con índice negativo.

Ejemplo 3.54.

```
df <- data.frame(asignatura = c("Matemáticas", "Física", "Economía"), nota = c(8.5, 7, 4
  asignatura nota créditos
1 Matemáticas 8.5
2
      Física 7.0
                         4
    Economía 4.5
# Eliminar una columna
df$nota <- NULL
df
   asignatura créditos
1 Matemáticas
2
      Física
                    4
3
    Economía
                    3
# Eliminar una fila
df[-2,]
  asignatura créditos
1 Matemáticas
                    6
    Economía
                    3
```

3.6 Ejercicios

Ejercicio 3.1. La siguiente tabla recoge las notas de los alumnos de un curso con dos asignaturas.

Alumno	Sexo	Física	Química
Carlos	H	6.7	8.1

Alumno	Sexo	Física	Química
María	Μ	7.2	9.5
Carmen	\mathbf{M}	5.5	5
Pedro	${ m H}$		4.5
Luis	${ m H}$	3.5	5
Sara	M	6.2	4

a. Definir cuatro vectores con el nombre, el sexo y las notas de Física y Química.

```
i Solución

nombre <- c("Carlos", "María", "Carmen", "Pedro", "Luis", "Sara")

sexo <- c("H", "M", "M", "H", "H", "M")

fisica <- c(6.7, 7.2, 5.5, NA, 3.5, 6.2)

quimica <- c(8.1, 9.5, 5, 4.5, 5, 4)
```

b. Convertir el sexo en un factor y mostrar sus niveles.

```
i Solución

sexo <- factor(sexo)
levels(sexo)

[1] "H" "M"
```

c. Crear un nuevo vector con la nota media de Física y Química.

```
i Solución

media <- (fisica + quimica) / 2

media

[1] 7.40 8.35 5.25 NA 4.25 5.10
```

d. Crear la variable booleana aprobado que tenga el valor TRUE si la media es mayor o igual que 5 y FALSE en caso contrario.

```
i Solución

aprobado <- media >= 5
aprobado

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

e. Aplicar un filtro al vector de nombres para quedarse con los nombres de los alumnos que han aprobado.

```
i Solución

nombre[aprobado & !is.na(aprobado)]

[1] "Carlos" "María" "Carmen" "Sara"
```

f. Crear un data frame con el nombre, sexo y las notas de Física y Química.

```
i Solución
df <- data.frame(nombre, sexo, fisica, quimica)</pre>
 nombre sexo fisica quimica
1 Carlos
            Η
                  6.7
2 María
                  7.2
                          9.5
            Μ
3 Carmen
            Μ
                  5.5
                          5.0
  Pedro
            Η
                  NA
                          4.5
5
    Luis
            Η
                  3.5
                          5.0
                  6.2
                          4.0
    Sara
```

g. Añadir el vector con la media al data frame.

```
i Solución
df$media <- media
  nombre sexo fisica quimica media
1 Carlos
            Η
                 6.7
                         8.1
                              7.40
2 María
            Μ
                 7.2
                         9.5 8.35
3 Carmen
            М
                 5.5
                         5.0 5.25
                  NA
                         4.5
  Pedro
            Η
                                NA
                 3.5
                         5.0 4.25
    Luis
            Η
6
            М
                 6.2
                         4.0 5.10
    Sara
```

h. Filtrar el data frame para quedarse con el nombre y la media de las mujeres que han aprobado.

```
i Solución

df[sexo == "M" & media >= 5, c("nombre", "media")]

nombre media
2 María 8.35
3 Carmen 5.25
6 Sara 5.10
```

4 Estructuras de control

Como en otros lenguajes de programación, en R existen instrucciones para controlar el flujo de ejecución de un programa. Básicamente existen dos tipos:

- Condicionales: Son instrucciones que bifurcan el flujo del programa en función de si se cumple o no una condición.
- Bucles: Son instrucciones que repiten un bloque de código un numero determinado de veces o hasta que se cumple una condición.

4.1 Estructuras condicionales

Las estructuras condicionales permiten evaluar el estado del programa y tomar decisiones sobre qué código ejecutar en función del mismo.

4.1.1 Condicionales (if)

La principal estructura condicional comienza con la palabra reservada if, lleva asociada expresión de tipo lógico o booleano y permite ejecutar un bloque de código dependiendo de si la evaluación de esa expresión es TRUE o FALSE.

```
if (<exp>) {
     <código>
}
```

Si el resultado de evaluar la expresión <exp> es TRUE entonces se ejecuta el código <código>, mientras que si es FALSE no.

Estructura Condicional simple

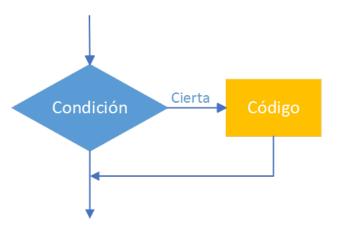


Figura 4.1: Diagrama de flujo de la estructura condicional simple

Estructura Condicional simple

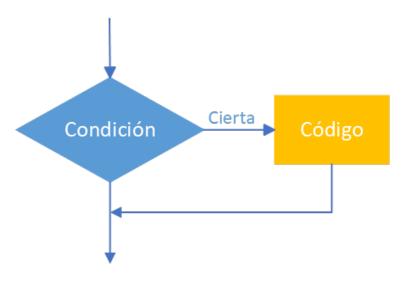


Figura 4.2: Diagrama de flujo de la estructura condicional simple

Ejemplo 4.1.

```
x <- 1
y <- 0
if (y != 0){
```

```
print(x / y)
}
```

Si se desea ejecutar un bloque de código alternativo cuando no se cumpla la condición se puede añadir a continuación con la palabra reservada else.

```
if (<exp>) {
    <código 1>
} else {
    <código 2>
}
```

En este caso, si la evaluación de la condición es TRUE se ejecuta el código <código 1> y si es FALSE se ejecuta el código <código 2>.

Estructura Condicional doble

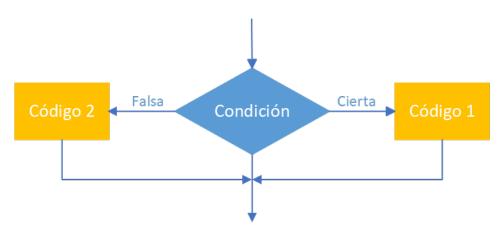


Figura 4.3: Diagrama de flujo de la estructura condicional doble

Ejemplo 4.2.

```
nota <- 8.5
if (nota < 5){
  print("Suspenso")
} else {
  print("Aprobado")
}</pre>
```

[1] "Aprobado"

Se puede comprobar más de una condición encadenando otra instrucción if tras las instrucción else.

```
if (<exp 1>) {
     <código 1>
} else if (<exp 2>) {
     <código 2>
...
} else {
     <código n>
}
```

Cuando se encadenan múltiples condiciones de esta forma, solamente se ejecuta el bloque de código asociado a la primera condición cuya evaluación sea TRUE. El último bloque de código solamente se ejecuta si todas las condiciones son falsas.

Estructura Condicional múltiple Cierta Condición 1 Falsa Cierta Condición 2 Falsa Cierta Condición n Falsa

Figura 4.4: Diagrama de flujo de la estructura condicional múltiple

Ejemplo 4.3.

```
nota <- 8.5
if (nota < 5){
  print("Suspenso")
} else if (nota < 7) {
  print("Aprobado")
} else if (nota < 9) {
  print("Notable")
} else {
  print("Sobresaliente")
}</pre>
```

[1] "Notable"

4.1.2 La función switch()

Otra forma de tomar decisiones sobre el código a ejecutar es la función switch.

• switch(x, 1): Ejecuta el código del valor de la lista 1 cuyo nombre asociado coincide con el resultado de evaluar la expresión x. Si el resultado de evaluar x no es ningún nombre de los elementos de la lista devuelve NULL.

Ejemplo 4.4.

```
tipo.iva <- "reducido"
precio <- 1000
iva <- precio * switch(tipo.iva, "superreducido" = 4, "reducido" = 10, "normal" = 21) /
iva</pre>
```

4.2 Bucles

[1] 100

Un bucle es una estructura que permite la repetición de un bloque de código. En R existen dos tipos de bucles, los bucles iterativos y los bucles condicionales.

4.2.1 Bucles iterativos (for)

Lo bucles iterativos repiten un bloque de código un número determinado de veces. Comienzan por la palabra reservada for y llevan asociado un *iterador*, que es una variable que recorre una secuencia de un tipo de datos compuesto, normalmente un vector o una lista. El bloque de código se ejecuta tantas veces como elementos tenga la secuencia, y en cada repetición el iterador toma como valor un elemento distinto de la secuencia.

```
for (i in <secuencia>) {
    <código>
}
```

Inicializar iterador Fin secuencia? Si Código

Figura 4.5: Diagrama de flujo de un bucle iterativo

Ejemplo 4.5. A continuación se muestra varios ejemplos de uso del bucle for.

```
asignaturas <- c("Matemáticas", "Física", "Programación")
for (i in asignaturas) {</pre>
```

```
print(i)
[1] "Matemáticas"
[1] "Física"
[1] "Programación"
for (i in 1:5) {
  print(paste("El cuadrado de ", i, " es ", i^2))
}
[1] "El cuadrado de
                    1
                       es
[1] "El cuadrado de
                     2
[1] "El cuadrado de
                            9"
                    3
[1] "El cuadrado de
                    4
                            16"
                        es
[1] "El cuadrado de
                     5
                            25"
```

También es posible recorrer los elementos de la secuencia por posición ayudándonos de la siguiente función:

• seq_along(x): que devuelve un vector con los enteros desde 1 hasta el número de elementos de la secuencia x.

Ejemplo 4.6.

```
asignaturas <- c("Matemáticas", "Física", "Programación")
for (i in seq_along(asignaturas)){
  print(paste("Asignatura ", i, ":", asignaturas[i]))
}</pre>
```

```
[1] "Asignatura 1 : Matemáticas"
[1] "Asignatura 2 : Física"
[1] "Asignatura 3 : Programación"
```

Los bucles iterativos se utilizan habitualmente para recorrer estructuras de una dimensión como los vectores y las listas, donde se sabe de antemano el número de elementos que contiene y, por tanto, el número de iteraciones del bucle. No obstante, también se pueden recorrer estructuras de más de una dimensión, como por ejemplo matrices, utilizando varios bucles for anidados.

Ejemplo 4.7. A continuación se muestra un ejemplo de dos bucles for anidados para recorrer los elementos de una matriz.

```
x <- matrix(1:6, 2, 3)
for (i in 1:nrow(x)) {
  for (j in 1:ncol(x)){
    print(x[i,j])
  }
}</pre>
```

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

4.2.2 Bucles condicionales (while)

Los bucles condicionales repiten un bloque de código mientras se cumpla una condición. Comienzan con la palabra reservada while y llevan asociada una expresión lógica, de manera que mientras la evaluación de la expresión lógica sea TRUE se repite la ejecución del bloque de código que contiene.

```
while (<condición>) {
    <código>
}
```

La expresión lógica <condición> se evalúa antes de ejecutar el bloque de código y solo se ejecuta el <código> si el resultado de la evaluación es TRUE. Obsérvese que cuando el flujo de ejecución del programa llega al bucle while si la condición no es cierta, el código no se ejecuta ni tan siquiera una vez.

Bucle condicional

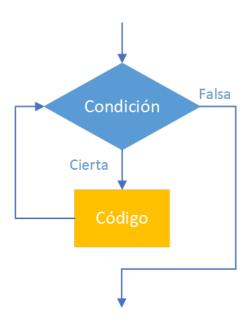


Figura 4.6: Diagrama de flujo de un bucle condicional

Ejemplo 4.8.

```
i <- 5
while (i >= 0) {
  print(i)
  i <- i - 1
}</pre>
```

- [1] 5
- [1] 4
- [1] 3
- [1] 2
- [1] 1
- [1] 0

4.2.3 La instrucción break

La instrucción break se utiliza para detener un bucle y salir de él, tanto en bucles iterativos como en bucles condicionales. Normalmente se suele utilizar esta instrucción

cuando se cumple una determinada condición en bloque de código del bucle y se decide parar su ejecución y salir del bucle.

Ejemplo 4.9.

```
# Bucle que recorre los números enteros del -2 al 2 pero termina al llegar al 0.
for (i in -2:2) {
   if (i == 0) {
      break
   }
   print(i)
}
```

- [1] -2
- [1] -1

4.2.4 La instrucción next

La instrucción **next** se utiliza para interrumpir la ejecución del bloque de código de un bucle, pero en lugar de salir del bucle pasa a la siguiente iteración. Si se trata de un bucle iterativo el iterador pasa al siguiente elemento de la secuencia de iteración y si se trata de un bucle condicional se pasa evaluar de nuevo la condición de repetición.

Ejemplo 4.10.

```
# Bucle que recorre los enteros del 1 al 10 pero solo imprime los números
for (i in 1:10) {
   if (i %% 2) {
       next
   }
   print(i)
}
```

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

4.3 Ejercicios

Ejercicio 4.1. Considérese el data frame creado en el Ejercicio 3.1.

```
nombre <- c("Carlos", "María", "Carmen", "Pedro", "Luis", "Sara")
sexo <- c("H", "M", "H", "H", "M")
fisica <- c(6.7, 7.2, 5.5, NA, 3.5, 6.2)
quimica <- c(8.1, 9.5, 5, 4.5, 5, 4)
df <- data.frame(nombre, sexo, fisica, quimica)
```

a. Calcular la media de las notas de Química.

```
i Solución

suma <- 0
for (i in quimica){
   suma <- suma + i
}

media <- suma / length(quimica)
media

[1] 6.016667</pre>
```

b. Calcular la media de las notas de Física.

```
i Solución

suma <- 0
n <- 0
for (i in fisica){
   if (!is.na(i)){
      suma <- suma + i
      n <- n + 1
    }
}
media <- suma / n
media

[1] 5.82</pre>
```

c. Construir un vector con las medias de las columnas numéricas del data frame.

```
i Solución
medias <- NULL
for (j in 3:4){
  suma <- 0
  n < 0
  for (i in df[[j]]){
    if (!is.na(i)){
      suma <- suma + i
      n \leftarrow n + 1
  medias <- c(medias, suma / n)</pre>
medias
[1] 5.820000 6.016667
```

d. Añadir al data frame nueva columnas con la calificaciones (suspenso, aprobado, notable, sobresaliente) de las notas de Física y Química.

```
i Solución
for (j in 3:4){
  calificaciones <- NULL
  for (i in df[[j]]) {
    if (is.na(i)){
      calificacion <- NA
    } else if (i < 5) {</pre>
      calificacion <- "SS"
    } else if (i < 7) {</pre>
      calificacion <- "AP"
    } else if (i < 9) {</pre>
      calificacion <- "NT"
    } else {
      calificacion <- "SB"
    calificaciones <- c(calificaciones, calificacion)</pre>
  df[[paste("calificacion", colnames(df)[j])]] <- calificaciones</pre>
}
df
  nombre sexo fisica quimica calificacion fisica calificacion quimica
```

-							
	1	Carlos	Н	6.7	8.1	AP	NT
	2	María	M	7.2	9.5	NT	SB
	3	Carmen	M	5.5	5.0	AP	AP
	4	Pedro	Н	NA	4.5	<na></na>	SS
	5	Luis	Н	3.5	5.0	SS	AP
	6	Sara	М	6.2	4.0	AP	SS
ı							

5 Funciones

Una función es un bloque de código que tiene asociado un nombre, de manera que cada vez que se quiera ejecutar el bloque de código basta con invocar el nombre de la función. Las funciones permite dividir el código en unidades lógicas que resultan más fáciles de manejar y mantener.

En R las funciones son objetos en sí mimas y pueden usarse como cualquier otro dato. El tipo de dato de las funciones es function.

5.1 Definición y llamada a funciones

Para definir una función se utiliza la siguiente estructura de código:

```
nombre.funcion <- function (parámetros) {
    <código>
}
```

El código que va entre llaves se conoce como cuerpo de la función.

Para llamar a la función y que se ejecute el código de su cuerpo hay que utilizar el nombre de la función y a continuación los valores pasados a sus parámetros entre paréntesis.

Ejemplo 5.1.

```
# Definición de la función
saludo <- function() {
   print("¡Hola!")
}
class(saludo)</pre>
```

[1] "function"

```
# Llamada a la función saludo()
```

```
[1] "¡Hola!"
```

5.2 Parámetros y argumentos de una función

Una función puede recibir valores cuando se invoca a través de unas variables conocidas como *parámetros* que se definen entre paréntesis en la declaración de la función. En el cuerpo de la función se pueden usar estos parámetros como si fuesen variables.

Los valores que se pasan a la función en una llamada o invocación concreta de ella se conocen como argumentos y se asocian a los parámetros de la declaración de la función.

Ejemplo 5.2.

```
# Función con un parámetro
saludo <- function(nombre) {
   print(paste("¡Hola ", nombre, "!", sep = ""))
}
# Llamada a la función con un argumento
saludo("Alf")</pre>
```

```
[1] "¡Hola Alf!"
```

En este ejemplo la función saludo tiene un parámetro nombre. En la llamada a la función se pasa la cadena Alf como argumento que se asocia al parámetro nombre en el cuerpo de la función.

5.2.1 Paso de argumentos a una función

Los argumentos de una función pueden pasarse de dos formas:

- Argumentos posicionales: Se asocian a los parámetros de la función en el mismo orden que aparecen en la definición de la función.
- Argumentos nominales: Se indica explícitamente el nombre del parámetro al que se asocia un argumento de la forma parametro = argumento. En este caso el orden de los argumentos no importa.

Ejemplo 5.3.

```
# Función con un argumento por defecto
area.triangulo <- function(base, altura) {
  base * altura / 2
}
# Cálculo del área de un triángulo de base 4 y altura 3
# Paso de argumentos por posición.
area.triangulo(4, 3)</pre>
```

```
# Paso de argumentos por nombre
area.triangulo(altura = 3, base = 4)
```

[1] 6

5.2.2 Argumentos por defecto

En la definición de una función se puede asignar a cada parámetro un argumento por defecto, de manera que si se invoca la función sin proporcionar ningún argumento para ese parámetro, se utiliza el argumento por defecto.

Ejemplo 5.4.

```
saludo <- function(nombre, lenguaje = "R") {
  print(paste("¡Hola ", nombre, "! ¡Bienvenido a ", lenguaje, "!", sep = ""))
}
# Llamada a la función con un argumento
saludo("Alf")</pre>
```

[1] "¡Hola Alf! ¡Bienvenido a R!"

5.3 Retorno de una función

Una función puede devolver un objeto de cualquier tipo tras su invocación. Para ello se utiliza la función return(), indicando entre paréntesis el valor que devuelve la función. El retorno suele realizarse al final del cuerpo de la función, porque con él finaliza la ejecución de la función y se devuelve el control de la ejecución al punto desde donde se llamó a la función, de manera que cualquier instrucción de cuerpo que vaya después no se ejecutará. Si no se indica ningún objeto, la función devolverá el valor de la última expresión calculada en el cuerpo de la función.

Ejemplo 5.5.

```
# Función que devuelve el area de un triángulo
area.triangulo <- function(base, altura) {
  return(base * altura / 2)
}
area.triangulo(4, 3)</pre>
```

```
[1] 6
```

```
# Función que devuelve el valor absoluto de un número
valor.absoluto <- function(x) {
  if (x < 0)
    return(x * -1)
  else
    return(x)
}
valor.absoluto(-1)</pre>
```

[1] 1

```
valor.absoluto(2)
```

[1] 2

Para devolver más de un valor se pueden utilizar estructuras de datos como vectores, listas, matrices o data frames.

Ejemplo 5.6.

```
circulo <- function(radio) {
  return(list(perimetro = 2 * pi * radio, area = pi * radio ^ 2))
}
circulo(5)</pre>
```

\$perimetro
[1] 31.41593

\$area

[1] 78.53982

```
circulo(5)$perimetro
```

[1] 31.41593

```
circulo(5)$area
```

[1] 78.53982

5.4 Entorno y ámbito de las variables

El entorno de un programa en R es el conjunto de todos los objetos (funciones, variables, etc.) creados durante la ejecución del programa. Cuando se ejecuta el interprete de R siempre se crea un primer entorno R_GlobalEnv conocido como entorno global. Es posible referirse a él en cualquier momento con la constante .GlobalEnv.

Para ver el entorno activo en cada momento de la ejecución y el contenido del mismo se utiliza la siguiente función:

- environment(): Devuelve el nombre del entorno actual.
- ls(): Devuelve un vector con los nombres de las objetos (variables, funciones, etc.) que contiene el entorno global.

Ejemplo 5.7.

```
x <- 4
y <- 3
area.triangulo <- function(base, altura) {
  base * altura / 2
}
environment()</pre>
```

<environment: R_GlobalEnv>

```
ls()
```

```
[1] "area.triangulo" "x" "y"
```

Como se puede observar en el ejemplo anterior, los parámetros de la función base y altura no aparecen en el entorno global. En R, cuando se ejecuta una función se crea un nuevo entorno hijo dentro del entorno al que pertenece la función. Durante la ejecución de la función este pasa a ser el entorno activo y cuando termina la ejecución de la función deja de serlo y vuelve a activarse el entorno padre desde donde se llamó a la función.

Ejemplo 5.8.

```
x <- 4
y <- 3
area.triangulo <- function(base, altura) {
   print("Entorno de la función area.triangulo")
   print(environment())</pre>
```

```
print(ls())
  return(base * altura / 2)
}
print("Entorno fuera de la función")

[1] "Entorno fuera de la función"
environment()

<environment: R_GlobalEnv>

ls()

[1] "area.triangulo" "x" "y"

area.triangulo(x, y)

[1] "Entorno de la función area.triangulo"
<environment: 0x562e9fd1f6d8>
[1] "altura" "base"

[1] 6
```

Los parámetros y los objetos (funciones, variables, etc.) definidos dentro de una función son de $\acute{a}mbito\ local$, mientras que los objetos definidos fuera de ella en alguno de los entornos ancestros son de $\acute{a}mbito\ global$.

Tanto los parámetros como las variables del ámbito local de una función sólo están accesibles durante la ejecución de la función, es decir, cuando termina la ejecución de la función estas variables desaparecen y no son accesibles desde fuera de la función.

Cuando una función declara un objeto (función, variable, etc.) que ya existe en alguno de los entornos ancestros con ámbito global, durante la ejecución de la función el objeto global queda eclipsado por el local y no es accesible hasta que finaliza la ejecución de la función.

Ejemplo 5.9.

```
lenguaje = "Python"
saludo <- function(lenguaje) {
  print(paste("Bienvenido a", lenguaje))
}
saludo("R")</pre>
```

[1] "Bienvenido a R"

Obsérvese cómo al ejecutar la función anterior, la variable lenguaje queda inaccesible al tener la función un parámetro con el mismo nombre.

Las variables globales están accesibles siempre que no sean eclipsadas por otras con el mismo nombre de ámbito local. Si embargo, cuando se intenta asignar un valor a una variable global en el ámbito local, se crea una nueva variable local. Para asignar valores a variables globales en el ámbito local se tiene que utilizar el operador de superasignación <<-. Cuando se utiliza este operador para asignar un valor a una variable, R busca la variable entorno padre, y si no existe continua con la búsqueda en los entornos ancestros hasta llegar a entorno global. Si la búsqueda tiene éxito, asigna el nuevo valor a la variable global, mientras que si no tiene éxito se crea una nueva variable de ámbito local y se le asigna el valor.

Ejemplo 5.10.

```
saludo <- function() {
  lenguaje <<- "R"
  return(paste("Bienvenido a", lenguaje))
}
lenguaje</pre>
```

[1] "Python"

5.5 Componentes de una función

Los tres componentes de una función son:

- Cuerpo: Es el código dentro de la función.
- Parámetros: Es la lista de parámetros que requiere la función.
- Entorno: Es donde se ubican las variables de la función.

Para acceder a estos componentes se pueden utilizar las siguientes funciones:

- body(f): Devuelve el cuerpo de la función f.
- formals(f): Devuelve la lista de parámetros de la función f.
- environment(f): Devuelve el entorno de la función f.

Ejemplo 5.11.

```
# Definición de la función
area.triangulo <- function(base, altura) {
  base * altura / 2
}
body(area.triangulo)

{
  base * altura/2
}

formals(area.triangulo)</pre>
```

\$altura

```
environment(saludo)
```

<environment: R_GlobalEnv>

5.6 Funciones recursivas

Una función recursiva es una función que en su cuerpo contiene una llama a sí misma.

La recursión es una práctica común en la mayoría de los lenguajes de programación ya que permite resolver las tareas recursivas de manera más natural.

Para garantizar el final de una función recursiva, las sucesivas llamadas tienen que reducir el grado de complejidad del problema, hasta que este pueda resolverse directamente sin necesidad de volver a llamar a la función. De lo contrario la recursión no tendría fin y nunca terminaría la ejecución de la función.

Ejemplo 5.12.

```
factorial <- function(n) {
  if (n <= 1) return(n)
  else return(n * factorial(n - 1))
}
factorial(4)</pre>
```

[1] 24

5.7 Paquetes

Para facilitar la reutilización código y datos R permite la creación de paquetes que pueden importarse desde otros programas. Un paquete es una colección de código, funciones y datos que se almacenan en un fichero dentro de un directorio llamado library en el entorno de R. Para ver la ubicación de este directorio dentro del sistema de archivos local se puede utilizar la función .libPaths().

Ejemplo 5.13.

```
.libPaths()
```

```
[1] "/home/alf/R/x86_64-pc-linux-gnu-library/4.2"
[2] "/usr/lib/R/library"
```

Durante la instalación de R también se instalan varios paquetes básicos que están disponibles en cualquier sesión de trabajo con R. Pero añadir nuevas funciones o procedimientos es necesario instalar el paquete que los contiene y después cargarlo en la sesión de trabajo.

Para ver los paquetes instalados en un ordenador se utiliza la función library().

5.7.1 Instalación de paquetes

La mayor parte de los paquetes para R están disponibles en el repositorio oficial CRAN (Comprehensive R Archive Network), aunque cualquier persona puede desarrollar un paquete y ponerlo a disposición de la comunidad en cualquier otro repositorio.

Existen distintas formas de instalar un paquete en R:

- Directamente desde el repositorio oficial CRAN
- Desde otros repositorios no oficiales (por ejemplo Github)
- Descargando el paquete e instalándolo manualmente.

5.7.1.1 Instalación de paquetes desde el repositorio CRAN

Para instalar un paquete desde el repositorio oficial CRAN se utiliza la siguiente función:

• install.packages(x): Obtiene el paquete con el nombre x desde un servidor con el repositorio CRAN y lo instala localmente en el directorio library del entorno de R. Se puede instalar más de un paquete a la vez pasando un vector con los nombres de los paquetes.

Ejemplo 5.14.

```
install.packages("remotes")
```

5.7.1.2 Instalación desde otros repositorios (GitHub, GitLab, etc.)

El paquete remotes incorpora funciones para instalar paquetes alojados en otros repositorios habituales para el desarrollo de software como GitHub, GitLab, etc.

Ejemplo 5.15.

```
install.packages("remotes")
remotes::install_github("rkward-community/rk.Teaching")
```

5.7.1.3 Instalación desde Bioconductor

Bioconductor es un repositorio de paquetes especializados en Bioinformática.

Bioconductor utiliza sus propio gestor de paquetes BiocManager, pero la instalación de paquetes es igualmente sencilla.

Ejemplo 5.16.

```
install.packages("BiocManager")
BiocManager::install("edgeR")
```

5.7.1.4 Instalación manual

Finalmente es posible instalar un paquete manualmente a partir de su código fuente. Para ello hay previamente hay que descargar el código fuente del paquete en un fichero comprimido en formato zip y después utilizar la siguiente función:

• install.packages(x, repos = NULL, type = "source"): Instala el paquete ubicado en la ruta x del sistema de archivos local en la librería library.

Una vez instalado un paquete ya está disponible para cargarlo en cualquier sesión de trabajo de R y no es necesario volver a instalarlo.

5.7.2 Carga de un paquete

Una vez instalado un paquete, para poder ejecutar su contenido es necesario cargarlo en el entorno de trabajo de R. Para ello se utiliza la siguiente función:

• library(x): Ejecuta el código del paquete x en la sesión de trabajo activa.

Ejemplo 5.17.

library("remotes")

5.7.3 Paquetes habituales

A continuación se presenta una lista ordenada alfabéticamente (no por importancia) de los paquetes más populares para el análisis de datos:

- caret es un paquete para la creación de modelos de clasificación y regresión mediante aprendizaje automático.
- data.table es un paquete para la manipulación de grandes conjuntos de datos (de hasta 100GB) de manera rápida y eficiente.
- devtools es un paquete con herramientas para el desarrollo de paquetes en R.
- knitr es un paquete que proporciona un motor para la generación de informes dinámicos que permite la integración de código en R con los lenguajes de procesamiento de textos LaTeX, HTML, Markdown, AsciiDoc o reStructuredText.
- mlr3 es un paquete que proporciona funciones para las principales técnicas de aprendizaje automático.
- plotly es un paquete para la creación de gráficos interactivos.
- rmarkdown es un paquete que facilita el uso del paquete knitr para la elaboración de documentos en múltiples formatos (HTML, pdf, Word y otros) permitiendo la integración de código R en el lenguaje Markdown.
- shiny es un paquete para la construcción de aplicaciones web interactivas.

- tidymodels es una colección de paquetes para la construcción y evalucación de modelos con técnicas de aprendizaje automático.
- tidyverse es una colección de paquetes para la Ciencia de Datos que incluye paquetes para la carga, limpieza, manipulación y representación gráfica de datos.

5.8 Ejercicios

Ejercicio 5.1. Crear una función para calcular la media de un vector numérico y usarla para calcular la media del vector (1, 2, NA, 3, 4).

```
i Solución

media <- function(x){
    suma <- 0
    n <- 0
    for (i in x){
        if (!is.na(i)){
            suma <- suma + i
            n <- n + 1
        }
    }
    return(suma / n)
}

media(c(1, 2, NA, 3, 4))</pre>
[1] 2.5
```

Ejercicio 5.2. Usar la función anterior para crear otra función para calcular las medias de las columnas de un data frame numérico. La función debe devolver un vector con las medias de las columnas. Usarla para calcular el data frame formado por los vectores (1, 2, NA, 3, 4) y (-1, 0, -2, 0, NA).

```
i Solución

medias <- function(df){
    medias <- NULL
    for (i in colnames(df)){
        medias <- c(medias, media(df[[i]]))
    }
    return(medias)
}

df <- data.frame(x = c(1, 2, NA, 3, 4), y = c(-1, 0, -2, 0, NA))
    medias(df)

[1] 2.50 -0.75</pre>
```

6 Preprocesamiento de datos

Cualquier análisis de datos comienza con la carga de datos en un *data frame*. Normalmente los datos brutos deben limpiarse y prepararse para su análisis. Este proceso se conoce como preprocesamiento de datos y suele incluir las siguientes tareas:

- Reestructuración del data frame.
- Selección de las variables (columnas) de interés.
- Filtrado de los casos (filas) de interés.
- Cálculo de nuevas variables a partir de las existentes.
- Ordenación de datos.
- Agrupación de datos.
- Tratamiento de datos no disponibles (NA, NaN).

Estas tareas pueden realizarse con las funciones básicas de R pero actualmente existen paquetes que facilitan mucho su realización como por ejemplo la colección de paquetes tidyverse.

6.1 La colección de paquetes tidyverse

tidyverse es una colección de paquetes para la Ciencia de Datos. Incluye los siguientes paquetes:

- tibble: Define la estructura de datos tibble que es una versión mejorada de los data frames.
- readr: Proporciona funciones para la lectura y escritura de tablas de datos en formato plano csv y tsv.
- tidyr: Proporciona funciones para la limpieza y preparación de los datos de manera consistente.
- dplyr: Proporciona una gramática de funciones para la manipulación de datos y las tareas más habituales de preprocesamiento.
- stringr: Proporciona funciones especializadas en la manipulación de cadenas.
- forcats: Proporciona funciones especializadas en la manipulación de factores.
- purr: Proporciona funciones para la programación funcional que mejoran las ya existentes en R.
- ggplot2: Proporciona una gramática de funciones para la realización de gráficos.

Estos paquetes están diseñados bajo una misma filosofía por lo interactúan y se complementan a la perfección.

6.2 Tibbles

El paquete tibble define la estructura de datos tibble que es similiar a los data frames, pero optimizada, ya que realiza una carga en memoria y evaluación perezosa, lo que hace más eficiente el manejo de grandes volúmenes de datos estructurados en forma de tabla. Los tibbles, además, suelen dar más información sobre el contenido y la estructura de los datos, así como de incoherencias en los datos.

Los tibbles nunca cambian el tipo de los datos (por ejemplo de cadenas a factores), nunca cambian los nombres de las variables, ni crean nombres de filas, como suelen hacer los data frames.

Aunque los paquetes de tidyverse trabajan perfectamente con data frames, están optimizados para trabajar con tibbles.

Para convertir un data frame en un tibble se utiliza la función

• as_tibble(df): Convierte el data frame df en un tibble.

Ejemplo 6.1.

```
library(tibble)
df <- data.frame(asignatura = c("Matemáticas", "Física", "Economía"), nota = c(8.5, 7, 4)
df

    asignatura nota
1 Matemáticas 8.5
2 Física 7.0
3 Economía 4.5

as_tibble(df)</pre>
```

Al igual que los data frames, las columnas de los tibbles son vectores cuyos elementos son del mismo tipo, de manera que suelen representar variables en los estudios estadísticos, mientras que las filas representan individuos, aunque no siempre es así.

6.3 Conjuntos de datos ordenados

Existen dos formas habituales de disponer los datos de un estudio en un data frame o un tibble: formato ancho y formato largo.

	Formato ancho									
Economía	Matemáticas	Programación								
5.0	3.5	9.0								
6.5	7.0	4.0								
8.0	8.5	6.5								
	5.0 6.5	5.0 3.5 6.5 7.0								

Formato largo

Nombre	Asignatura	Nota
Carmen	Economía	5.0
Luis	Economía	6.5
María	Economía	8.0
Carmen	Matemáticas	3.5
Luis	Matemáticas	7.0
María	Matemáticas	8.5
Carmen	Programación	9.0
Luis	Programación	4.0
María	Programación	6.5

Figura 6.1: Formatos de un data frame

La mayoría de los paquetes de tidyverse asumen que los datos del data frame o tibble están en formado largo, lo que significa que las columnas del data frame representan variables y las filas observaciones, de manera que cada dato pertenece a una variable y una observación única. Las variables (columnas) contienen valores que miden la misma característica o atributo (edad, estatura, etc.) en cada unidad experimental. Una observación (fila) contiene los valores medidos en la misma unidad experimental (una persona, un día, etc.) en todos atributos estudiados. En resumen, un data frame o tibble esta ordenado (tidy) si

- Cada columna es una variable
- Cada fila es una observación
- Cada casilla es un valor

A menudo los conjuntos de datos no están ordenados y violan alguna de de estas condiciones. Lo más común es encontrarnos

• Encabezados de columnas que son valores en lugar de variables.

- Una misma columna contiene varias variables.
- Variables que están almacenadas tanto en filas como en columnas.

Para facilitar el preprocesamiento y posterior análisis de los datos es recomendable ordenar el data frame. Para ello el paquete tidyr proporciona dos funciones que permiten pivotar un data frame:

- pivot_longer(df, columnas, names_to = columna-nombres, values_to = columna-valores): Devuelve el tibble que resulta de convertir las columnas indicadas en el parámetro columnas del data frame df a formato largo, es decir, las columnas se reestructuran en dos nuevas columnas con nombres columna-nombres y columna-valores que contienen los nombres de las columnas originales y sus valores, respectivamente.
- pivot_wider(df, names_from = columna-nombres, values_from = columna-valores):

 Devuelve el tibble que resulta de convertir el data frame df a formato ancho,
 es decir, se crean tantas columnas como nombres distintos haya en la columna
 columna-nombres, usando estos nombres como los nombres de las columnas,
 mientras que los valores se toman de la columna columna-valores.

Ejemplo 6.2.

```
library(tidyr)
df <- data.frame(nombre = c('María', 'Luis', 'Carmen'),
edad = c(18, 22, 20),
Matemáticas = c(8.5, 7, 3.5),
Economía = c(8, 6.5, 5),
Programación = c(6.5, 4, 9))
df</pre>
```

```
      nombre
      edad
      Matemáticas
      Economía
      Programación

      1
      María
      18
      8.5
      8.0
      6.5

      2
      Luis
      22
      7.0
      6.5
      4.0

      3
      Carmen
      20
      3.5
      5.0
      9.0
```

```
# Pivotar de formato ancho a formato largo
df_largo <- pivot_longer(df, Matemáticas:Programación, names_to = "Asignatura", values_to
df_largo</pre>
```

```
2 María
            18 Economía
                               8
3 María
            18 Programación
                               6.5
4 Luis
            22 Matemáticas
                               7
5 Luis
            22 Economía
                               6.5
6 Luis
            22 Programación
7 Carmen
            20 Matemáticas
                               3.5
8 Carmen
            20 Economía
                               5
9 Carmen
            20 Programación
                               9
```

```
# Pivotar de formato largo a formato ancho
df_ancho <- pivot_wider(df_largo, names_from = Asignatura, values_from = Nota)
df_ancho</pre>
```

```
# A tibble: 3 x 5
```

	nombre	edad	Matemáticas	Economía	Programación
	<chr></chr>	<dbl></dbl>	<dbl></dbl>	<dbl></dbl>	<dbl></dbl>
1	María	18	8.5	8	6.5
2	Luis	22	7	6.5	4
3	Carmen	20	3.5	5	9

6.4 El paquete dplyr

El paquete dplyr proporciona una gramática para el preprocesamiento de data frames o tibbles, de manera que cada acción sobre data frame se corresponde con un verbo y las funciones que realizan esa acción tienen como nombre el verbo correspondiente. Las funciones de preprocesamiento más habituales incluidas en el paquete dplyr son:

- count: Cuenta el número de observaciones de un data frame.
- select: Selecciona un subconjunto de columnas de un data frame.
- filter: Selecciona un subconjunto de filas de un data frame.
- arrange: Reordena las filas de un data frame.
- rename: Renombra las columnas de un data frame.
- mutate: Añade nuevas columnas a un data frame o transforma las existentes.
- summarise: Genera resúmenes estadísticos de las columnas de un data frame.
- group_by: Divide las filas de un data frame en grupos de acuerdo a una columna categórica.

6.5 Conteo del número de observaciones

Para contar el número de observaciones (filas) de un data frame se utiliza la función

• count(df, columnas): Devuelve el número de filas del data frame df para cada posible combinación de los valores de las columnas indicadas en el parámetro columnas.

Esta función se utiliza habitualmente para calcular tamaños muestrales.

Ejemplo 6.3.

```
library(dplyr)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/colestere
# Contar las filas del data frame
count(df)</pre>
```

n 1 14

Ejemplo 6.4.

```
library(dplyr)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/colestere
# Contar las filas del data frame
count(df, sexo)</pre>
```

sexo n 1 H 8 2 M 6

6.6 Selección de variables

Para seleccionar un subconjunto de variables de un data frame se utiliza la función

• select(df, columnas): Devuelve un tibble con las columnas indicadas en el parámetro columnas del data frame df.

Ejemplo 6.5.

```
library(dplyr)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/colestere
# Seleccionar las columnas nombre, sexo y edad
select(df, nombre, sexo, edad)</pre>
```

```
nombre sexo edad
1
      José Luis Martínez Izquierdo
                                       Η
                                            18
2
                    Rosa Díaz Díaz
                                       Μ
                                            32
3
             Javier García Sánchez
                                            24
                                       Η
4
               Carmen López Pinzón
                                            35
5
              Marisa López Collado
                                           46
                 Antonio Ruiz Cruz
6
                                       Η
                                            68
7
           Antonio Fernández Ocaña
                                       Η
                                            51
8
             Pilar Martín González
                                           22
                                       Μ
9
              Pedro Gálvez Tenorio
                                           35
10
           Santiago Reillo Manzano
                                           46
             Macarena Álvarez Luna
                                            53
11
12
        José María de la Guía Sanz
                                            58
13 Miguel Angel Cuadrado Gutiérrez
                                            27
                                       Η
             Carolina Rubio Moreno
                                            20
```

Selectionar la primera y tercera columnas
select(df, 1, 3)

```
nombre sexo
1
      José Luis Martínez Izquierdo
2
                    Rosa Díaz Díaz
                                       Μ
3
             Javier García Sánchez
                                       Η
4
               Carmen López Pinzón
5
              Marisa López Collado
6
                 Antonio Ruiz Cruz
                                       Η
7
           Antonio Fernández Ocaña
                                       Н
8
             Pilar Martín González
                                       Μ
9
              Pedro Gálvez Tenorio
                                       Η
10
           Santiago Reillo Manzano
             Macarena Álvarez Luna
11
12
        José María de la Guía Sanz
                                       Η
13 Miguel Angel Cuadrado Gutiérrez
                                       Η
             Carolina Rubio Moreno
```

Seleccionar las columnas desde el peso hasta el colesterol
select(df, peso:colesterol)

```
peso altura colesterol
1 85 1.79 182
2 65 1.73 232
3 NA 1.81 191
```

```
4
     65
           1.70
                        200
5
     51
           1.58
                        148
6
     66
           1.74
                        249
7
     62
           1.72
                        276
8
     60
           1.66
                         NA
9
     90
           1.94
                        241
10
     75
           1.85
                        280
11
     55
           1.62
                        262
12
     78
           1.87
                        198
13
    109
           1.98
                        210
14
     61
           1.77
                        194
```

Selectionar todas las columnas menos la edad
select(df, -edad)

	nombre	sexo	peso	${\tt altura}$	colesterol
1	José Luis Martínez Izquierdo	Н	85	1.79	182
2	Rosa Díaz Díaz	M	65	1.73	232
3	Javier García Sánchez	Н	NA	1.81	191
4	Carmen López Pinzón	M	65	1.70	200
5	Marisa López Collado	M	51	1.58	148
6	Antonio Ruiz Cruz	Н	66	1.74	249
7	Antonio Fernández Ocaña	Н	62	1.72	276
8	Pilar Martín González	М	60	1.66	NA
9	Pedro Gálvez Tenorio	Н	90	1.94	241
10	Santiago Reillo Manzano	Н	75	1.85	280
11	Macarena Álvarez Luna	М	55	1.62	262
12	José María de la Guía Sanz	Н	78	1.87	198
13	Miguel Angel Cuadrado Gutiérrez	Н	109	1.98	210
14	Carolina Rubio Moreno	М	61	1.77	194

6.7 Filtrado de datos

Para filtrar un data frame y quedarse con las filas que cumplen una condición se usa la función

• filter(df, condicion): Devuelve el tibble con las filas del data frame df que cumplen la condición indicada en el parámetro condición.

Ejemplo 6.6.

```
library(dplyr)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/colesterent)</pre>
# Filtrar las mujeres
filter(df, sexo == "M")
                 nombre edad sexo peso altura colesterol
         Rosa Díaz Díaz
                           32
                                     65
                                          1.73
                                                      232
1
2
    Carmen López Pinzón
                          35
                                     65
                                          1.70
                                                      200
3 Marisa López Collado
                          46
                                Μ
                                    51
                                         1.58
                                                      148
4 Pilar Martín González
                          22
                                Μ
                                     60
                                          1.66
                                                       NA
5 Macarena Álvarez Luna
                          53
                                M
                                     55
                                          1.62
                                                      262
6 Carolina Rubio Moreno
                          20
                                     61
                                          1.77
                                                      194
# Filtrar los hombres mayores de 30
filter(df, sexo == "H" & edad > 30)
                      nombre edad sexo peso altura colesterol
1
           Antonio Ruiz Cruz
                                68
                                      Η
                                          66
                                               1.74
                                                           249
2
     Antonio Fernández Ocaña 51
                                          62
                                               1.72
                                                           276
3
        Pedro Gálvez Tenorio
                                35
                                      Η
                                          90
                                               1.94
                                                           241
     Santiago Reillo Manzano
                               46
                                      Η
                                          75
                                               1.85
                                                           280
5 José María de la Guía Sanz
                                      Η
                                          78
                                               1.87
                                                           198
                                58
# Filtrar las filas con valores de colesterol
filter(df, !is.na(colesterol))
```

	nombre	edad	sexo	peso	altura	colesterol
1	José Luis Martínez Izquierdo	18	Н	85	1.79	182
2	Rosa Díaz Díaz	32	М	65	1.73	232
3	Javier García Sánchez	24	Н	NA	1.81	191
4	Carmen López Pinzón	35	М	65	1.70	200
5	Marisa López Collado	46	M	51	1.58	148
6	Antonio Ruiz Cruz	68	Н	66	1.74	249
7	Antonio Fernández Ocaña	51	Н	62	1.72	276
8	Pedro Gálvez Tenorio	35	Н	90	1.94	241
9	Santiago Reillo Manzano	46	Н	75	1.85	280
10	Macarena Álvarez Luna	53	М	55	1.62	262
11	José María de la Guía Sanz	58	Н	78	1.87	198
12	Miguel Angel Cuadrado Gutiérrez	27	Н	109	1.98	210
13	Carolina Rubio Moreno	20	М	61	1.77	194

Existe un filtro bastante habitual que consiste en eliminar las filas de un data frame que contienen algún dado no disponible (NA). Para ello dplyr dispone de la función

• na.omit(df): Devuelve el tibble que resulta de eliminar las filas del data frame df con algún valor NA.

Ejemplo 6.7.

na.omit(df)

	nombre	edad	sexo	peso	altura	colesterol
1	José Luis Martínez Izquierdo	18	Н	85	1.79	182
2	Rosa Díaz Díaz	32	М	65	1.73	232
4	Carmen López Pinzón	35	М	65	1.70	200
5	Marisa López Collado	46	M	51	1.58	148
6	Antonio Ruiz Cruz	68	Н	66	1.74	249
7	Antonio Fernández Ocaña	51	Н	62	1.72	276
9	Pedro Gálvez Tenorio	35	Н	90	1.94	241
10	Santiago Reillo Manzano	46	Н	75	1.85	280
11	Macarena Álvarez Luna	53	М	55	1.62	262
12	José María de la Guía Sanz	58	Н	78	1.87	198
13	Miguel Angel Cuadrado Gutiérrez	27	Н	109	1.98	210
14	Carolina Rubio Moreno	20	M	61	1.77	194

6.8 Reordenación de datos

Para reordenar las filas de un data frame se utiliza la función

• arrange(df, columnas): Devuelve un tibble con las mismas filas del data frame df pero ordenadas de acuerdo a los valores de las columnas indicadas en el parámetro columnas. Por defecto, la ordenación es ascendente, para hacerla descendente, hay que aplicar la función desc() a la columna con respecto se quiere ordenar descendentemente.

Ejemplo 6.8.

```
library(dplyr)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/colesterc
# Ordenar según alfabéticamente por nombre
arrange(df, nombre)</pre>
```

	nombre	edad	sexo	peso	altura	colesterol
1	Antonio Fernández Ocaña	51	Н	62	1.72	276
2	Antonio Ruiz Cruz	68	Н	66	1.74	249
3	Carmen López Pinzón	35	M	65	1.70	200
4	Carolina Rubio Moreno	20	М	61	1.77	194
5	Javier García Sánchez	24	Н	NA	1.81	191
6	José Luis Martínez Izquierdo	18	Н	85	1.79	182
7	José María de la Guía Sanz	58	Н	78	1.87	198
8	Macarena Álvarez Luna	53	М	55	1.62	262
9	Marisa López Collado	46	М	51	1.58	148
10	Miguel Angel Cuadrado Gutiérrez	27	Н	109	1.98	210
11	Pedro Gálvez Tenorio	35	Н	90	1.94	241
12	Pilar Martín González	22	М	60	1.66	NA
13	Rosa Díaz Díaz	32	M	65	1.73	232
14	Santiago Reillo Manzano	46	Н	75	1.85	280

Ordenar según sexo y edad
arrange(df, sexo, edad)

	nombre	edad	sexo	peso	${\tt altura}$	colesterol
1	José Luis Martínez Izquierdo	18	Н	85	1.79	182
2	Javier García Sánchez	24	Н	NA	1.81	191
3	Miguel Angel Cuadrado Gutiérrez	27	Н	109	1.98	210
4	Pedro Gálvez Tenorio	35	Н	90	1.94	241
5	Santiago Reillo Manzano	46	Н	75	1.85	280
6	Antonio Fernández Ocaña	51	Н	62	1.72	276
7	José María de la Guía Sanz	58	Н	78	1.87	198
8	Antonio Ruiz Cruz	68	Н	66	1.74	249
9	Carolina Rubio Moreno	20	М	61	1.77	194
10	Pilar Martín González	22	М	60	1.66	NA
11	Rosa Díaz Díaz	32	М	65	1.73	232
12	Carmen López Pinzón	35	М	65	1.70	200
13	Marisa López Collado	46	М	51	1.58	148
14	Macarena Álvarez Luna	53	M	55	1.62	262

Ordenar ascendentemente por sexo y descendentemente por colesterol
arrange(df, sexo, desc(colesterol))

	nombre	edad	sexo	peso	altura	colesterol
1	Santiago Reillo Manzano	46	Н	75	1.85	280
2	Antonio Fernández Ocaña	51	Н	62	1.72	276
3	Antonio Ruiz Cruz	68	Н	66	1.74	249

4	Pedro Gálvez Tenorio	35	Н	90	1.94	241
5	Miguel Angel Cuadrado Gutiérrez	27	Н	109	1.98	210
6	José María de la Guía Sanz	58	Н	78	1.87	198
7	Javier García Sánchez	24	Н	NA	1.81	191
8	José Luis Martínez Izquierdo	18	Н	85	1.79	182
9	Macarena Álvarez Luna	53	M	55	1.62	262
10	Rosa Díaz Díaz	32	M	65	1.73	232
11	Carmen López Pinzón	35	M	65	1.70	200
12	Carolina Rubio Moreno	20	M	61	1.77	194
13	Marisa López Collado	46	M	51	1.58	148
14	Pilar Martín González	22	M	60	1.66	NA

6.9 Renombrado de columnas

Para cambiar el nombre de las columnas se utiliza la función

• rename(df, nuevo-nombre = columna): Devuelve un tibble con los mismos datos del data frame df pero cambiando el nombre de la columna de nombre columna por nuevo-nombre.

Ejemplo 6.9.

```
library(dplyr)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/colestere
# Ordenar según alfabéticamente por nombre
rename(df, estatura = altura)</pre>
```

	nombre	edad	sexo	peso	estatura	colesterol
1	José Luis Martínez Izquierdo	18	Н	85	1.79	182
2	Rosa Díaz Díaz	32	M	65	1.73	232
3	Javier García Sánchez	24	Н	NA	1.81	191
4	Carmen López Pinzón	35	M	65	1.70	200
5	Marisa López Collado	46	M	51	1.58	148
6	Antonio Ruiz Cruz	68	Н	66	1.74	249
7	Antonio Fernández Ocaña	51	Н	62	1.72	276
8	Pilar Martín González	22	M	60	1.66	NA
9	Pedro Gálvez Tenorio	35	Н	90	1.94	241
10	Santiago Reillo Manzano	46	Н	75	1.85	280
11	Macarena Álvarez Luna	53	M	55	1.62	262
12	José María de la Guía Sanz	58	Н	78	1.87	198
13	Miguel Angel Cuadrado Gutiérrez	27	Н	109	1.98	210
14	Carolina Rubio Moreno	20	M	61	1.77	194

6.10 Creación de nuevas columnas o transformación de las existentes

Para crear una nueva columna a partir de otras columnas del data frame se utiliza la función

• mutate(df, columna = formula): Devuelve el tibble que resulta de añadir una nueva columna al data frame df con el resultado de aplicar el procedimiento indicado por formula y con el nombre indicado en columna. Si columna es el nombre de una columna ya existente, entonces esa columna se reescribe con los resultados de aplicar la formula.

Ejemplo 6.10.

```
library(dplyr)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/colestere
# Cambiar las unidades de la altura a centímetros
mutate(df, altura = altura*100)</pre>
```

```
nombre edad sexo peso altura colesterol
                                                  85
1
      José Luis Martínez Izquierdo
                                       18
                                             Η
                                                         179
                                                                    182
2
                                                                    232
                     Rosa Díaz Díaz
                                       32
                                                  65
                                                        173
                                             Μ
3
              Javier García Sánchez
                                       24
                                             Η
                                                  NA
                                                        181
                                                                    191
4
                Carmen López Pinzón
                                       35
                                             Μ
                                                  65
                                                        170
                                                                    200
5
              Marisa López Collado
                                       46
                                             М
                                                  51
                                                        158
                                                                    148
6
                  Antonio Ruiz Cruz
                                       68
                                             Η
                                                  66
                                                        174
                                                                    249
7
           Antonio Fernández Ocaña
                                       51
                                                                    276
                                             Η
                                                  62
                                                        172
             Pilar Martín González
                                       22
8
                                             Μ
                                                  60
                                                        166
                                                                     NA
9
               Pedro Gálvez Tenorio
                                       35
                                             Η
                                                  90
                                                                    241
                                                        194
           Santiago Reillo Manzano
                                       46
                                                  75
10
                                             Η
                                                        185
                                                                    280
11
             Macarena Álvarez Luna
                                       53
                                             Μ
                                                  55
                                                        162
                                                                    262
        José María de la Guía Sanz
12
                                                  78
                                       58
                                              Η
                                                        187
                                                                    198
13 Miguel Angel Cuadrado Gutiérrez
                                       27
                                                 109
                                              Η
                                                        198
                                                                    210
             Carolina Rubio Moreno
                                       20
                                                  61
                                                        177
                                                                    194
                                              Μ
```

```
# Calcular el índice de masa corporal
mutate(df, imc = round(peso/altura^2))
```

	nombre	edad	sexo	peso	altura	colesterol	imc	
1	José Luis Martínez Izquierdo	18	Н	85	1.79	182	27	
2	Rosa Díaz Díaz	32	M	65	1.73	232	22	
3	Javier García Sánchez	24	Н	NA	1.81	191	NA	

4	Carmen López Pinzón	35	M	65	1.70	200	22
5	Marisa López Collado	46	M	51	1.58	148	20
6	Antonio Ruiz Cruz	68	H	66	1.74	249	22
7	Antonio Fernández Ocaña	51	H	62	1.72	276	21
8	Pilar Martín González	22	M	60	1.66	NA	22
9	Pedro Gálvez Tenorio	35	H	90	1.94	241	24
10	Santiago Reillo Manzano	46	H	75	1.85	280	22
11	Macarena Álvarez Luna	53	M	55	1.62	262	21
12	José María de la Guía Sanz	58	H	78	1.87	198	22
13	Miguel Angel Cuadrado Gutiérrez	27	H	109	1.98	210	28
14	Carolina Rubio Moreno	20	M	61	1.77	194	19

6.11 Resumen de datos

Para aplicar una función resumen a una o varias columnas de un data frame se utiliza la función

• sumarise(df, nombre-columna = funcion-resumen(columnas)): Devuelve el tibble con la columna de nombre nombre-columna y el valor que resulta de aplicar la función indicada en funcion-resumen a las columnas del data frame df indicadas en columnas.

Ejemplo 6.11.

```
library(dplyr)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/colestere
# Calcular la media de la edad
summarise(df, edad_media = mean(edad))

    edad_media
1    38.21429

# Calcular la media y la desviación típica del colesterol
summarise(df, media = mean(colesterol, na.rm=T), sd = sd(colesterol, na.rm=T))

    media    sd
1    220.2308    39.84795</pre>
```

6.12 Resúmenes por grupos

La función summarise suele combinarse con la siguiente función para obtener resúmenes estratificados por grupos.

• group_by(df, columnas): Devuelve un tibble estratificado de acuerdo a las categorías de las columnas indicadas en columnas. En combinación con la función summarise permite hacer resúmenes estadísticos por grupos.

Ejemplo 6.12.

```
library(dplyr)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/colester</pre>
# Estratificar por sexo
df.sexo <- group_by(df, sexo)</pre>
# Edades medias por sexo
summarise(df.sexo, edad_media = mean(edad))
# A tibble: 2 x 2
  sexo edad_media
  <chr>
            <dbl>
1 H
              40.9
2 M
              34.7
# Media y desviación típica del colesterol por sexo
summarise(df.sexo, media = mean(colesterol, na.rm=T), sd = sd(colesterol, na.rm=T))
# A tibble: 2 x 3
  sexo media
                 sd
  <chr> <dbl> <dbl>
         228. 38.4
1 H
2 M
         207. 42.9
```

6.13 Composición de operaciones mediante tuberías

dplyr permite componer varias operaciones sobre un data frame mediante el operador %>% (pipe), de manera que el data frame que resulta de aplicar una operación se convierte en el data frame de entrada para otra, siguiendo el esquema

```
df %>% operación-1 %>% operación-2 %>% ...
```

Cuando se utilizan tuberías para componer operaciones de esta forma, no es necesario indicar el data frame como parámetro de la función que define la acción ya que automáticamente se toma el data frame que resulta de la operación anterior.

Ejemplo 6.13.

```
library(dplyr)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/colester
# Inicio de la tubería
df %>%
    # Seleccionar sexo, edad y colesterol
    select(sexo, edad, colesterol) %>%
    # Filtrar mayores de 30 años
    filter(edad > 30) %>%
    # Estratificar por sexo
    group_by(sexo) %>%
    # Calcular la media del colesterol
    summarise(media_colesterol = mean(colesterol))
# A tibble: 2 x 2
  sexo media_colesterol
  <chr>
                   <dbl>
```

6.14 Ejercicios

1 H

2 M

Ejercicio 6.1. El fichero genetica, contiene información de la análitica fisiológica, microbiológica y bioquímica, de una muestra de ratas tratadas con distintos tratamientos.

a. Crear un tibble con los datos del fichero.

249.

210.

```
Solución
library(tidyverse)
df <- read_csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/ge
# A tibble: 21 x 20
   `código muestra` Tratamiento
                                  `mas cor (g)`
                                                   IHS
                                                           IES
                                                                   ITS
                                                                            IAS
                                          <dbl>
                                                 <dbl>
                                                                 <dbl>
                                                                          <dbl>
   <chr>
                    <chr>
                                                         <dbl>
 1 Ct-A1
                    Control
                                           236 0.0433 0.00198 0.00178 0.00043
 2 Ct-A2
                    Control
                                           202. 0.044 0.002
                                                               0.0017 0.0008
```

```
3 Ct-B1
                                         246. 0.042 0.002
                                                              0.0015 0.0005
                   Control
 4 Ct-B2
                   Control
                                         237. 0.0475 0.00238 0.00203 0.00041
 5 Ct-B3
                                         231. 0.0499 0.002
                                                              0.0011 0.0005
                   Control
                                                      0.0021 0.0033 0.0014
 6 Ct-C1
                   Control
                                         235. 0.05
                                         228. 0.05
                                                      0.0018 0.0031 0.00059
 7 Ct-C2
                   Control
                                         181. 0.061 0.00155 0.0005 0.00012
 8 Dx-1
                   Dexametasona
                                         198. 0.063 0.00191 0.0011 0.0002
 9 Dx-2
                   Dexametasona
10 Dx-3
                   Dexametasona
                                         201. 0.0577 0.0017 0.0001 0.0001
# i 11 more rows
# i 13 more variables: `glucog(mg/g)` <dbl>, `pbmc(cel/mL)` <dbl>,
    `CD4(cel/µL)` <dbl>, `monoc(cel/mL)` <dbl>, `IHQ: linfB(cel/mm2)` <dbl>,
    `pulp.blan(Tx1)` <dbl>, `microorg(log nº/g)` <dbl>, `AI2(absorb)` <dbl>,
    `gluc(mmol/L)` <dbl>, `TAG(mmol/L)` <dbl>, `col(mmol/L)` <dbl>,
    `HDL(mmol/L)` <dbl>, `ins(pmol/L)` <dbl>
```

b. Convertir el tratamiento en un factor.

```
i Solución

df <- mutate(df, Tratamiento = factor(Tratamiento))
```

c. Calcular el tamaño muestral de cada grupo de tratamiento

d. Filtrar las ratas de grupo control con una masa corporal (mas cor (g)) mayor de 230 g.

```
236 0.0433 0.00198 0.00178 0.00043
1 Ct-A1
                  Control
                                        246. 0.042 0.002 0.0015 0.0005
2 Ct-B1
                  Control
3 Ct-B2
                                        237. 0.0475 0.00238 0.00203 0.00041
                  Control
                                        231. 0.0499 0.002
4 Ct-B3
                  Control
                                                            0.0011 0.0005
                                        235. 0.05
5 Ct-C1
                  Control
                                                    0.0021 0.0033 0.0014
# i 13 more variables: `glucog(mg/g)` <dbl>, `pbmc(cel/mL)` <dbl>,
    `CD4(cel/\muL)` <dbl>, `monoc(cel/mL)` <dbl>, `IHQ: linfB(cel/mm2)` <dbl>,
    `pulp.blan(Tx1)` <dbl>, `microorg(log nº/g)` <dbl>, `AI2(absorb)` <dbl>,
    `gluc(mmol/L)` <dbl>, `TAG(mmol/L)` <dbl>, `col(mmol/L)` <dbl>,
    `HDL(mmol/L)` <dbl>, `ins(pmol/L)` <dbl>
```

e. Calcular la media y la desviación típica de la masa corporal (mas cor (g)) para cada tratamiento.

```
i Solución
df %>%
   group_by(Tratamiento) %>%
    summarise(media = mean(`mas cor (g)`), desv.est = sd(`mas cor (g)`))
# A tibble: 3 x 3
 Tratamiento media desv.est
  <fct>
               <dbl>
                        <dbl>
1 Control
                231.
                         13.6
2 Dexametasona 191.
                         14.2
3 Kanamicina
                229.
                         13.9
```

f. Calcular la media y la desviación típica de todas las variables para cada tratamiento.

```
i Solución
df %>%
   pivot_longer(-c(`código muestra`, Tratamiento), names_to = "Variable", values_
   group_by(Variable) %>%
   summarise(media = mean(Valor), desv.est = sd(Valor))
# A tibble: 18 x 3
  Variable
                        media desv.est
   <chr>
                         <dbl>
                               <dbl>
 1 AI2(absorb)
                      2.86e-1 1.63e-1
 2 CD4(cel/\mu L)
                     1.05e+3 1.50e+3
3 HDL(mmol/L)
                      1.49e+0 6.20e-1
 4 IAS
                      5.26e-4 3.92e-4
```

```
5 IES
                      1.96e-3 1.96e-4
6 IHQ: linfB(cel/mm2) 7.95e+1 7.62e+1
7 IHS
                      5.14e-2 7.12e-3
8 ITS
                      1.80e-3 8.63e-4
9 TAG(mmol/L)
                      3.98e+0 2.09e+0
10 col(mmol/L)
                      2.44e+0 1.29e+0
11 gluc(mmol/L)
                      3.57e+1 1.88e+1
12 glucog(mg/g)
                      7.83e+1 3.81e+1
13 ins(pmol/L)
                      1.91e+2 1.05e+2
14 mas cor (g)
                      2.17e+2 2.28e+1
15 microorg(log n^{\circ}/g) 2.35e+0 1.61e+0
16 monoc(cel/mL)
                      1.32e+4 4.68e+3
17 pbmc(cel/mL)
                      1.27e+7 2.10e+7
18 pulp.blan(Tx1)
                      2.75e-1 1.12e-1
```

7 Gráficos y visualización de datos

R incorpora funciones para realizar gráficos de distintos tipos, pero actualmente existen paquetes especializados para la visualización de datos como ggplot2 que permiten realizar multitud de gráficos de manera más estructurada.

7.1 Gramática de gráficos y el paquete ggplot2

El paquete ggplot2 forma parte de la colección de paquetes tidyverse que ya se introdujo en la sección anterior.

Este paquete implementa la gramática de gráficos descrita por Leland Wilkinson, que proporciona un sistema formal para representar distintas variables mediante distintos atributos gráficos (aesthetics) como la forma, el tamaño o el color de objetos geométricos como puntos, líneas o barras. Esto hace que la creación de gráficos con este paquete sea, en general, más intuitiva una vez se entiende la lógica de la gramática.

En general, para definir un gráfico con ggplot2 se suelen definir los siguientes elementos:

- **Datos**. Los datos deben estar contenidos en un data frame o tibble en formato ordenado (tidy).
- Atributos (aesthetics). Las variables que quieren representarse en el gráfico deben asociarse a atributos gráficos como los ejes x, y, z, el color, el tamaño, la forma de los objetos geométricos.
- Capas de objetos geométricos (*geoms*). Están formadas por elementos geométricos (puntos, líneas, barras, etc.) cuya posición, forma, tamaño y color, depende de los atributos.
- Escalas. Definen la escala para los ejes del diagrama así como las leyendas para el resto de atributos.
- Sistema de coordenadas. Describe el sistema de coordenadas utilizado para representar los objetos geométricos en el plano o en el espacio (normalmente el sistema cartesiano).
- Facetas. Permite descomponer un gráfico en múltiples gráficos para distintos subconjuntos del conjunto de datos.
- Tema. Permite cambiar elementos secundarios del gráfico como el tipo de letra de las etiquetas y leyendas, el tamaño de la fuente, el color de fondo, aspecto final del los gráficos.

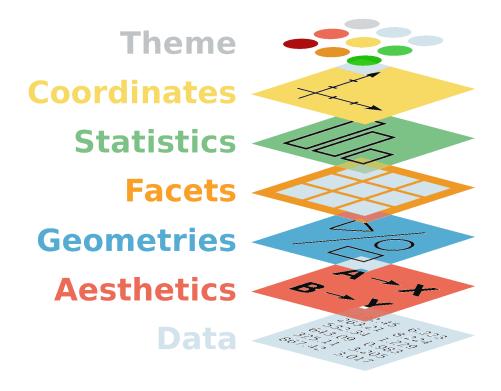


Figura 7.1: Gramática de gráficos de ggplot2

Estos elementos se suelen añadir por capas al gráfico mediante el operador +.

7.2 Inicialización de un gráfico

Para dibujar un gráfico con ggplot2 debemos comenzar por la siguiente función

- ggplot(df, aes(atributo1 = var1, atributo2 = var2, ...)): Inicializa un gráfico con las variables var1, var2, etc. del data frame df asociadas a los atributos atributo1, atributo2, etc. respectivamente. Los atributos pueden más comunes son:
 - x: Posición en el eje x del objeto geométrico.
 - y: Posición en el eje y del objeto geométrico.
 - z: Posición en el eje z del objeto geométrico.
 - shape: Forma del punto.
 - size: Tamaño del punto.
 - linetype: Forma de la línea.

- linewidth: Anchura de la línea.
- colour: Color del objeto geométrico.
- fill: Color de relleno del objeto geométrico.

Importante

Todos los atributos se indican dentro de la función aes().

7.3 Diagramas de puntos

Para dibujar un diagrama de puntos se utiliza la capa de objetos geométricos

• geom_point(): Dibuja un diagrama de líneas que unen los puntos con coordenadas dadas por los pares de valores de las variables asociadas a los atributos x e y.

Esta capa es ideal para representar diagramas de dispersión.

Ejemplo 7.1.

```
library(dplyr)
library(ggplot2)

df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/temperate
    mutate(Día = factor(Día, levels = c("L", "M", "X", "J", "V", "S", "D")))

# Filtrar los datos de Madrid

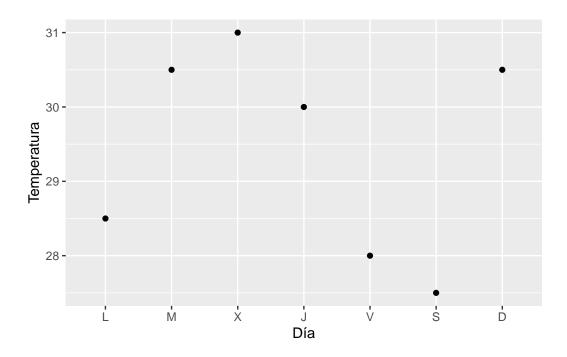
df.madrid = filter(df, Ciudad == "Madrid")

# Inicializar el gráfico con el día en el atributo x y la temperatura en el atributo y.

ggplot(df.madrid, aes(x = Día, y = Temperatura)) +

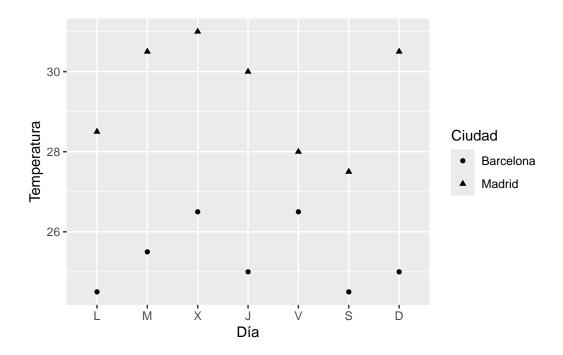
# Añadir la capa de los puntos.

geom_point()</pre>
```



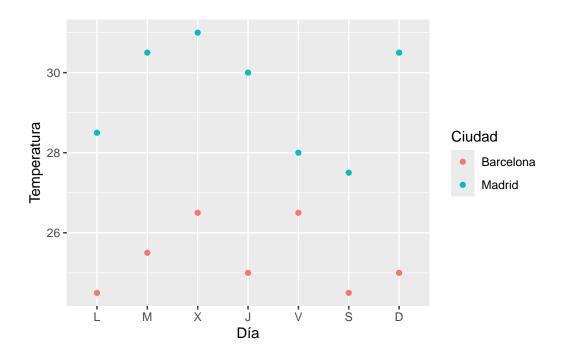
Ejemplo 7.2.

```
# Inicializar el gráfico con el día en el atributo x, la temperatura en el atributo y, y
ggplot(df, aes(x = Día, y = Temperatura, shape = Ciudad)) +
# Añadir la capa de los puntos.
geom_point()
```



Ejemplo 7.3.

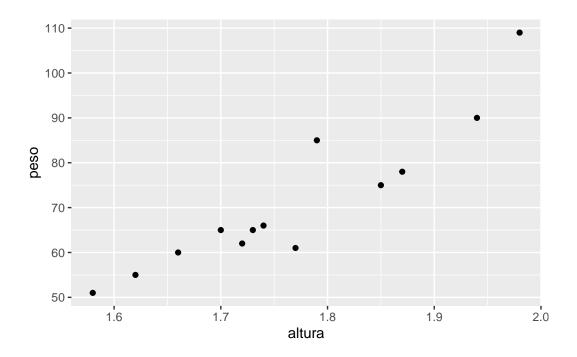
```
# Inicializar el gráfico con el día en el atributo x, la temperatura en el atributo y, y
ggplot(df, aes(x = Día, y = Temperatura, colour = Ciudad)) +
# Añadir la capa de los puntos.
    geom_point()
```



Ejemplo 7.4.

```
library(ggplot2)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/colestere
# Inicializar el gráfico con la altura en el atributo x y el peso en el atributo y.
ggplot(df, aes(x = altura, y = peso)) +
# Añadir la capa de los puntos.
    geom_point()</pre>
```

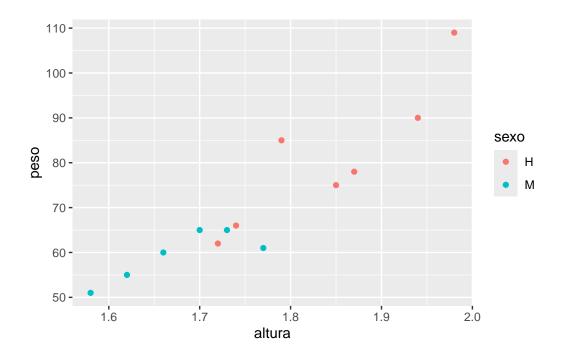
Warning: Removed 1 row containing missing values or values outside the scale range (`geom_point()`).



Ejemplo 7.5.

```
# Inicializar el gráfico con la altura en el atributo x, el peso en el atributo y, y el e
ggplot(df, aes(x = altura, y = peso, colour = sexo)) +
# Añadir la capa de los puntos.
    geom_point()
```

Warning: Removed 1 row containing missing values or values outside the scale range (`geom_point()`).



Más información sobre geom_point.

7.4 Diagramas de líneas

Para dibujar un diagrama de líneas se utiliza la capa de objetos geométricos

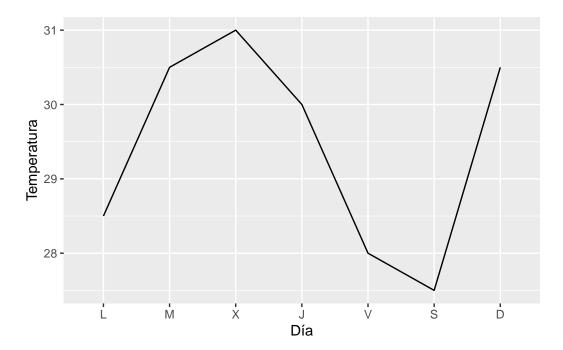
• geom_line(): Dibuja un diagrama de líneas que unen los puntos con coordenadas dadas por los pares de valores de las variables asociadas a los atributos x e y. Cuando la variable asociada al atributo x es un factor, debe asociarse también una variable al atributo group para determina los grupos que se unirán mediante líneas. Si solo hay un grupo debe indicarse group = 1.

Esta capa es ideal para representar series temporales.

Ejemplo 7.6.

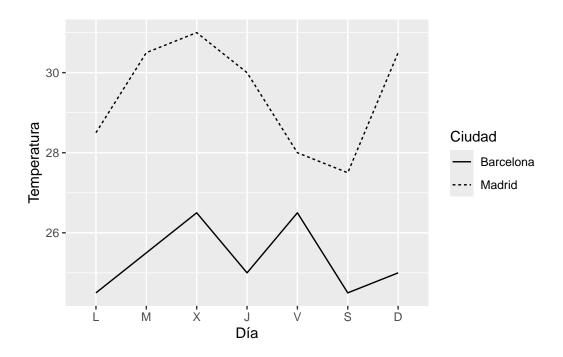
```
library(dplyr)
library(ggplot2)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/temperate
    mutate(Día = factor(Día, levels = c("L", "M", "X", "J", "V", "S", "D")))
# Filtrar los datos de Madrid
df.madrid = filter(df, Ciudad == "Madrid")
# Inicializar el gráfico con el día en el atributo x y la temperatura en el atributo y.</pre>
```

```
ggplot(df.madrid, aes(x = Día, y = Temperatura, group = 1)) +
# Añadir la capa de las lineas.
    geom_line()
```



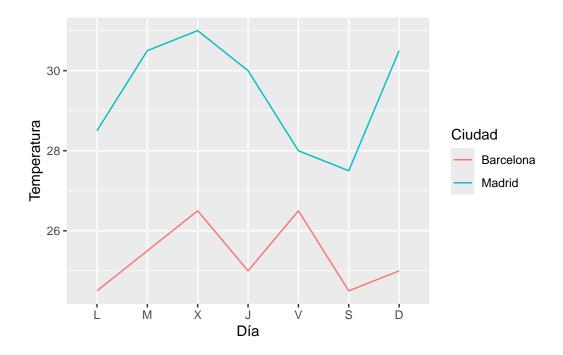
Ejemplo 7.7.

```
# Inicializar el gráfico con el día en el atributo x, la temperatura en el atributo y, y
ggplot(df, aes(x = Día, y = Temperatura, group = Ciudad, linetype = Ciudad)) +
# Añadir la capa de las lineas.
    geom_line()
```



Ejemplo 7.8.

```
# Inicializar el gráfico con el día en el atributo x, la temperatura en el atributo y, y
ggplot(df, aes(x = Día, y = Temperatura, group = Ciudad, colour = Ciudad)) +
# Añadir la capa de las lineas.
    geom_line()
```



Más información sobre geom_line.

7.5 Diagramas de barras

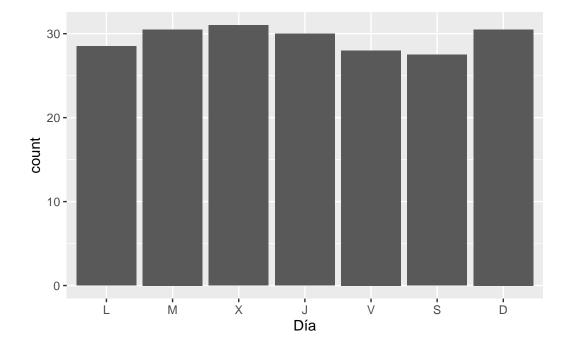
Para dibujar un diagrama de barras se utiliza la capa de objetos geométricos

• geom_bar(aes(weight = var)): Dibuja un diagrama de barras con la variable asociada al atributo x o y (si se usa el atributo x las barras son verticales y se usa y horizontales), donde la altura de las barras viene dada por la variable var. Si no se indica Por defecto, la altura de las barras representa la frecuencia absoluta de cada valor de la variable. Si no se indica el atributo weight la altura de las barras es la frecuencia absoluta de los valores de la variable asociada a los atributos x o y.

Ejemplo 7.9.

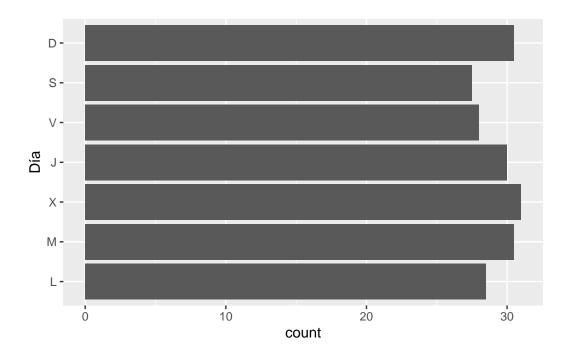
```
library(dplyr)
library(ggplot2)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/temperate
    mutate(Día = factor(Día, levels = c("L", "M", "X", "J", "V", "S", "D")))
df</pre>
```

```
Ciudad Día Temperatura
1
      Madrid
               L
                         28.5
2
      Madrid
                         30.5
               М
3
      Madrid
               Х
                         31.0
4
      Madrid
               J
                         30.0
5
      Madrid
               V
                         28.0
6
      Madrid
                         27.5
               S
7
      Madrid
                         30.5
               D
8 Barcelona
                         24.5
               L
9
  Barcelona
               Μ
                         25.5
10 Barcelona
                         26.5
               Х
11 Barcelona
                         25.0
               J
12 Barcelona
               V
                         26.5
13 Barcelona
                         24.5
               S
14 Barcelona
                         25.0
```

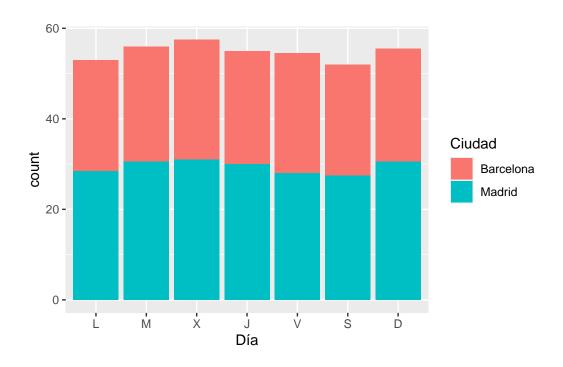


Ejemplo 7.10.

```
# Inicializar el gráfico con el día en el atributo y.
ggplot(df.madrid, aes(y = Día)) +
# Añadir la capa de las barras con altura la temperatura.
geom_bar(aes(weight = Temperatura))
```

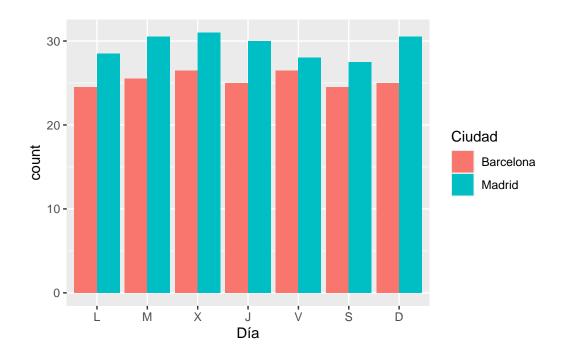


Ejemplo 7.11.

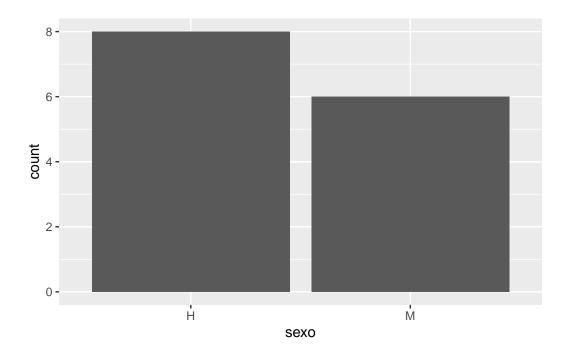


Ejemplo 7.12.

```
# Inicializar el gráfico con el día en el atributo x y la ciudad en el color de relleno.
ggplot(df, aes(x = Día, fill = Ciudad)) +
# Añadir la capa de las barras con altura la temperatura indicando barras separadas.
geom_bar(aes(weight = Temperatura), position = "dodge")
```

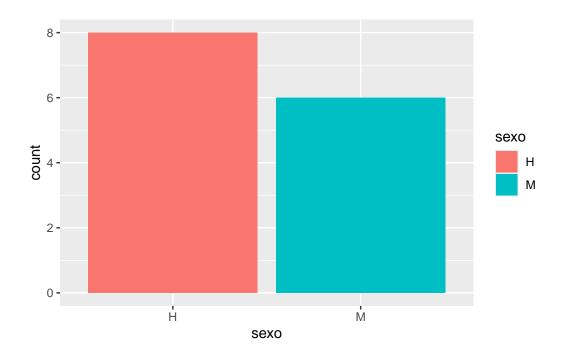


Ejemplo 7.13.

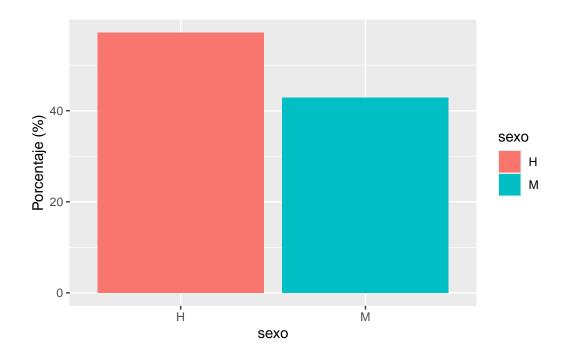


Ejemplo 7.14.

```
# Inicializar el gráfico con el sexo en los atributos x y fill.
ggplot(df, aes(x = sexo, fill = sexo)) +
# Añadir la capa de las barras.
geom_bar()
```



Ejemplo 7.15.



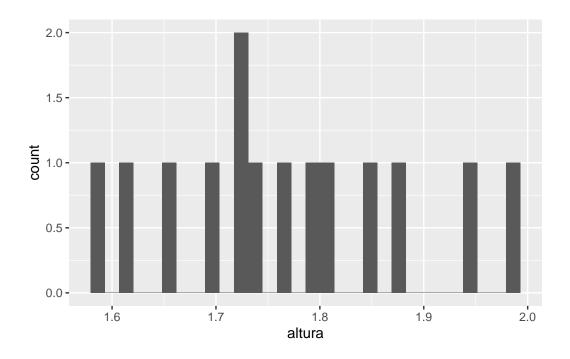
Más información sobre geom_bar.

7.6 Histogramas

Para dibujar un histograma se utiliza la capa de objetos geométricos

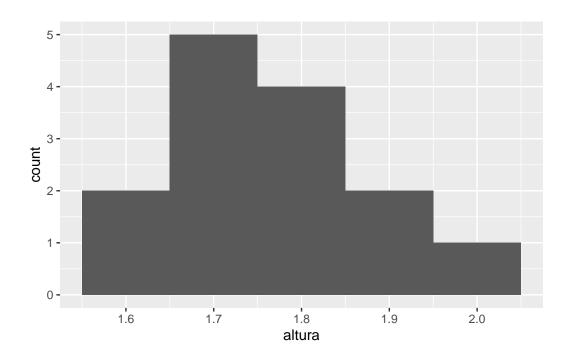
• geom_histogram(bins = clases, binwidth = anchura): Dibuja un histograma de la variable asociada al atributo x usando el número de clases indicado por clases o bien clases de amplitud indicada por anchura. Si no se indica el parámetro bins o bindwidth se toman 30 clases por defecto.

Ejemplo 7.16.



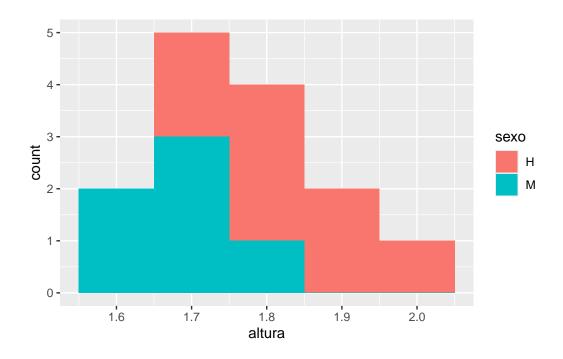
Ejemplo 7.17.

```
# Inicializar el gráfico con la edad en el atributo x.
ggplot(df, aes(x = altura)) +
# Añadir la capa del histograma con anchura de clases 5.
geom_histogram(binwidth = 0.1)
```

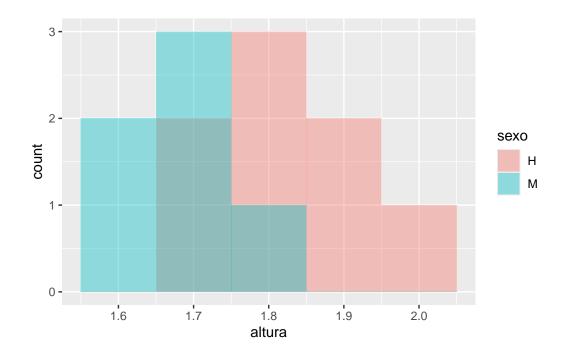


Ejemplo 7.18.

```
# Inicializar el gráfico con la edad en el atributo x y el sexo en el atributo fill.
ggplot(df, aes(x = altura, fill = sexo)) +
# Añadir la capa del histograma con anchura de clases 10.
geom_histogram(binwidth = 0.1)
```



Ejemplo 7.19.



Más información sobre geom_histogram.

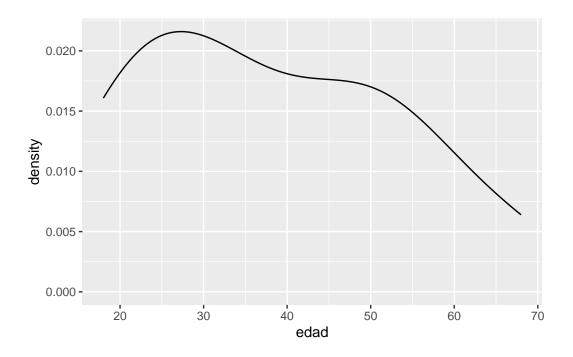
7.7 Diagramas de densidad

Para dibujar un diagrama de densidad se utiliza la capa de objetos geométricos

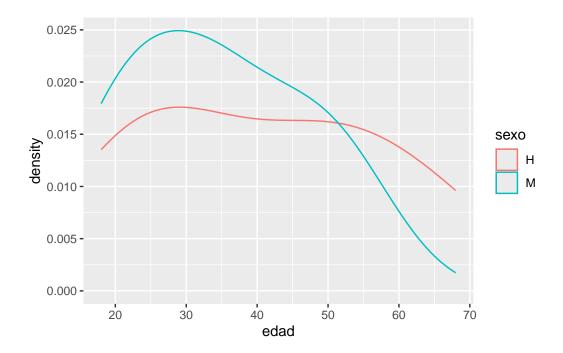
• geom_density(): Dibuja un diagrama de densidad de probabilidad estimada de los valores de la variable asociada al atributo x.

Este diagrama es una alternativa a los histogramas para representar la distribución de probabilidad de los valores de una variable.

Ejemplo 7.20.



Ejemplo 7.21.



Más información sobre geom_density.

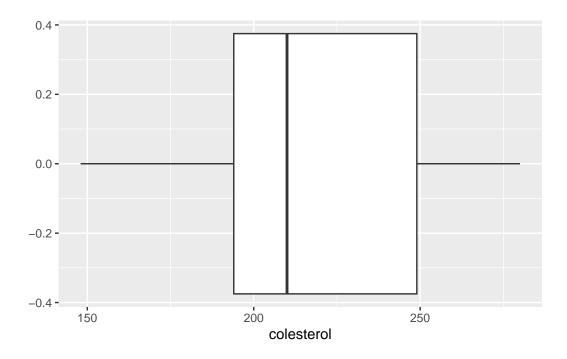
7.8 Diagramas de cajas

Para dibujar un diagrama de caja y bigotes se utiliza la capa de objetos geométricos

• geom_boxplot(): Dibuja un diagrama de caja y bigotes de la variable asociada al atributo x o y. Si se utiliza el atributo x la caja se representa horizontalmente, y si se utiliza el atributo y verticalmente.

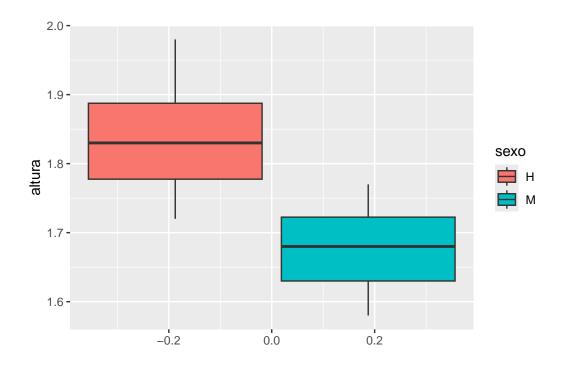
Ejemplo 7.22.

Warning: Removed 1 row containing non-finite outside the scale range (`stat_boxplot()`).



Ejemplo 7.23.

```
# Inicializar el gráfico con la altura en el atributo y y el sexo en el atributo fill.
ggplot(df, aes(y = altura, fill = sexo)) +
# Añadir la capa de la caja.
    geom_boxplot()
```



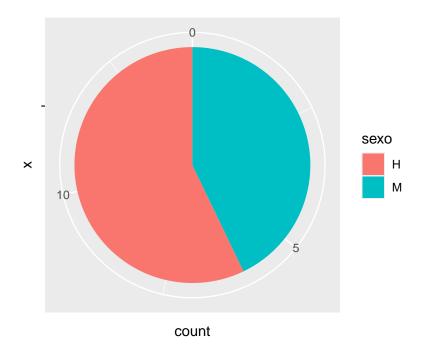
Más información sobre geom_boxplot.

7.9 Diagrama de sectores

Para dibujar un diagrama de sectores se utiliza la misma capa de objetos geométricos que para los diagramas de barras (geom_bar) pero añadiendo el sistema de coordenadas polares.

• coor_polar(theta = "x"|"y"): Cambia al sistema de coordenadas cartesianas polares, donde el ángulo viene dado por la variable asociada al atributo "x" o el atributo "y".

Ejemplo 7.24.



7.10 Interpolación y ajustes de regresión

Para dibujar una línea de interpolación o de ajuste de regresión se utiliza la capa de objetos geométricos

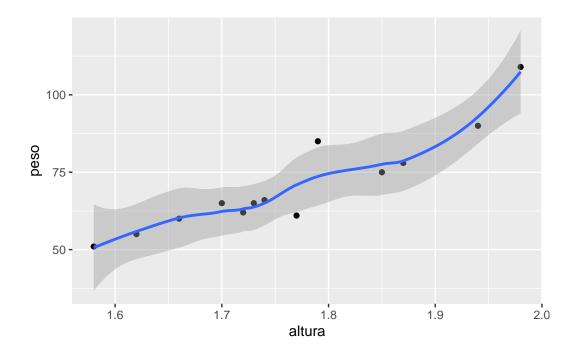
- geom_smooth(method = ajuste, formula = ecuación): Dibuja una línea de ajuste para los puntos con coordenadas dadas por los pares de valores de las variables asociadas a los atributos x e y, usando el método de ajuste dado por por ajuste y la fórmula dada por ecuación. Los métodos de ajuste más habituales son:
 - "loess": Ajuste de regresión polinomial local. Es la que se utiliza por defecto.
 - "lm": Ajuste de regresión de modelos lineal por mínimos cuadrados.
 - "glm": Ajuste de regresión modelos lineales generalizados por mínimos cuadrados. Por defecto se dibujan también las bandas con el error estándar del ajuste. Para desactivar estas bandas se debe indicar también el parámetro se
 = FALSE.

Esta capa suele usarse en combinación con la capa de puntos para diagramas de dispersión.

Ejemplo 7.25.

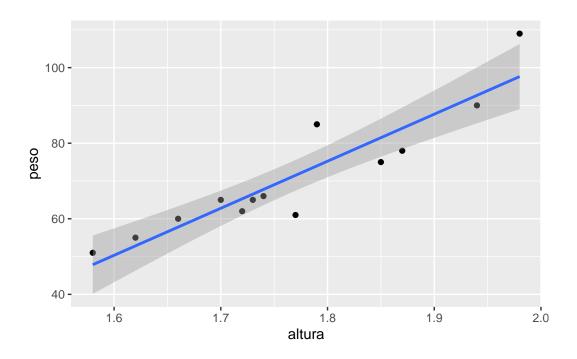
```
library(ggplot2)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/colestere
# Inicializar el gráfico con la altura en el atributo x y el peso en el atributo y.
ggplot(df, aes(x = altura, y = peso)) +
# Añadir la capa de puntos.</pre>
```

geom_point() +
Añadir la capa de ajuste de regresión polinomial local loess
geom_smooth()



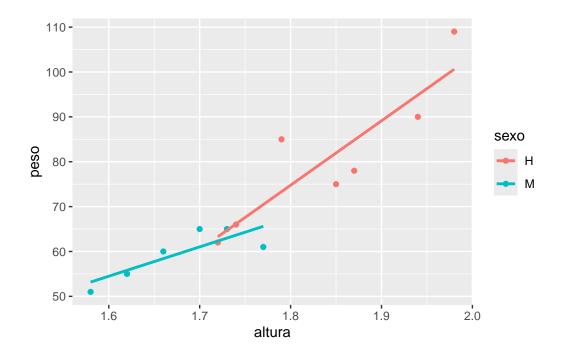
Ejemplo 7.26.

```
# Inicializar el gráfico con la altura en el atributo x y el peso en el atributo y.
ggplot(df, aes(x = altura, y = peso)) +
# Añadir la capa de puntos.
    geom_point() +
# Añadir la capa de ajuste de regresión lineal por mínimos cuadrados.
    geom_smooth(method = "lm")
```



Ejemplo 7.27.

```
# Inicializar el gráfico con la altura en el atributo x, el peso en el atributo y, y el s
ggplot(df, aes(x = altura, y = peso, colour = sexo)) +
# Añadir la capa de puntos.
    geom_point() +
# Añadir la capa de ajuste de regresión lineal por mínimos cuadrados sin las bandas de en
geom_smooth(method = "lm", se = FALSE)
```



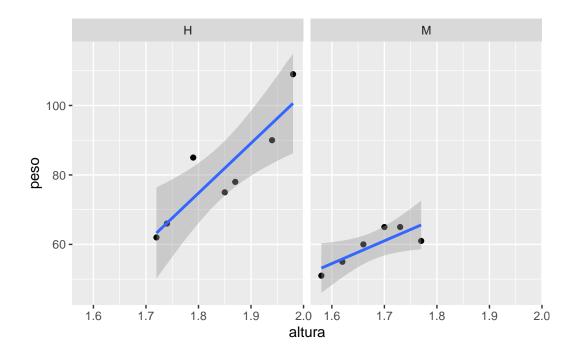
Más información sobre geom_smooth.

7.11 Facetas

Las facetas permiten desagregar un gráfico según los grupos de uno o varios factores del conjunto de datos. Para añadir facetas a un gráfico se añade la función

• facet_wrap(vars(var1, var2,...), nrow = n, ncol = m): Crea un un gráfico para cada combinación de valores de las variables var1, var2, etc. y los coloca en una tabla de n filas y m columnas.

Ejemplo 7.28.



Más información sobre facetas.

7.12 Personalización de gráficos

ggplot incluye multitud de posibilidades de personalización de gráficos. Aquí veremos solo las más habituales.

7.12.1 Títulos

Para poner un título al gráfico, a los ejes o a la leyenda se añade la función

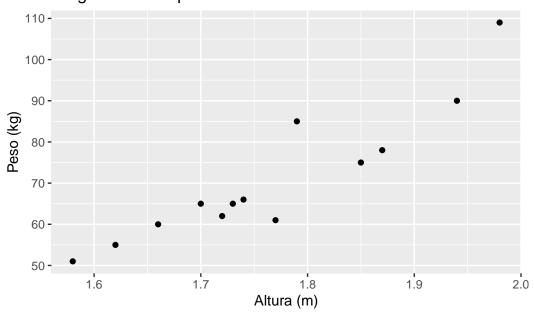
```
• labs(title = título, x = titulo-x, y = titulo-y, colour = leyenda-color, shape = leyenda-forma)
```

Ejemplo 7.29.

```
library(ggplot2)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/colestere
# Inicializar el gráfico con la altura en el atributo x y el peso en el atributo y.
ggplot(df, aes(x = altura, y = peso)) +
# Añadir la capa de puntos.
    geom_point() +</pre>
```

```
# Añadir un título al gráfico y a los ejes.
labs(title = "Diagrama de dispersión", x = "Altura (m)", y = "Peso (kg)")
```

Diagrama de dispersión



7.12.2 Escalas

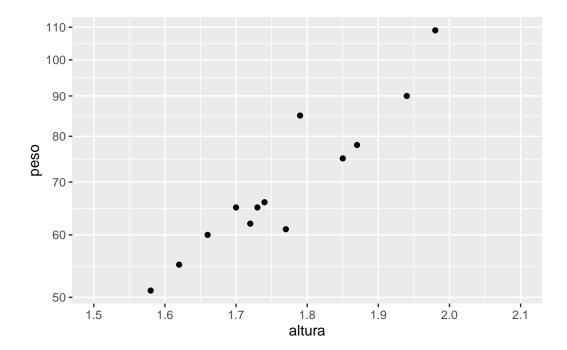
Para cambiar las escalas de los ejes cartesianos se añaden la funciones

- scale_x_continuos(limits = rango, breaks = cortes, labels = etiquetas, trans = transformación): Restringe escala del eje x al rango indicado por rango, incluye las marcas en eje indicadas por cortes con las etiquetas indicadas por etiquetas y aplica la transformación de escala indicada por transformación.

 Las transformaciones de escala más habituales son "log2" (logarítmo en base 2), "log10" (logarítmo en base 10) y sqrt (raíz cuadrada).
- scale_y_continuos(limits = rango, breaks = cortes, labels = etiquetas, trans = transformación): Igual que la función anterior pero para el eje y.

Ejemplo 7.30.

```
library(ggplot2)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/colestere
# Inicializar el gráfico con la altura en el atributo x y el peso en el atributo y.
ggplot(df, aes(x = altura, y = peso)) +</pre>
```



7.12.3 Temas

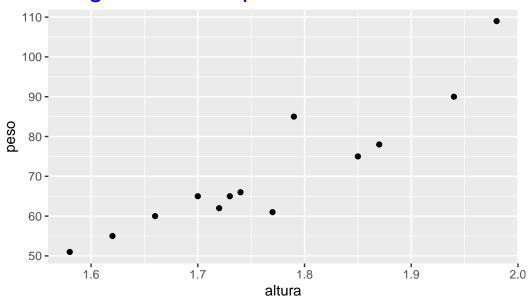
Finalmente se para cambiar otros aspectos del gráfico como las fuentes o los colores de fondos se utilizan temas. Para indicar o modificar un tema se añade la función

• theme(propiedades): Modifica las propiedades del tema indicadas. La lista es demasiado grande para cubrirla en este tutorial, por lo que se recomienda ver todas las opciones en la documentación de ggplot2.

Ejemplo 7.31.

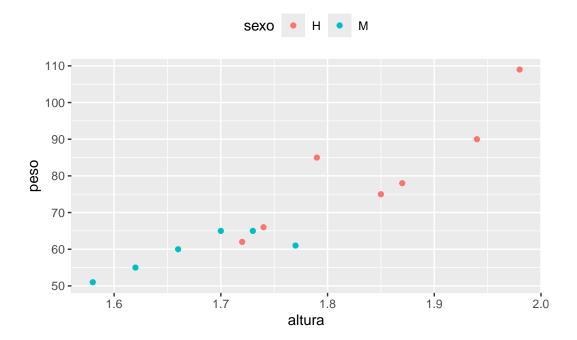
```
library(ggplot2)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/colestere
# Inicializar el gráfico con la altura en el atributo x y el peso en el atributo y.
ggplot(df, aes(x = altura, y = peso)) +</pre>
```

Diagrama de dispersión



Ejemplo 7.32.

```
# Inicializar el gráfico con la altura en el atributo x, el peso en el atributo y, y el a
ggplot(df, aes(x = altura, y = peso, colour = sexo)) +
# Añadir la capa de puntos.
    geom_point() +
# Cambiar el la posición de la leyenda
    theme(legend.position = "top")
```



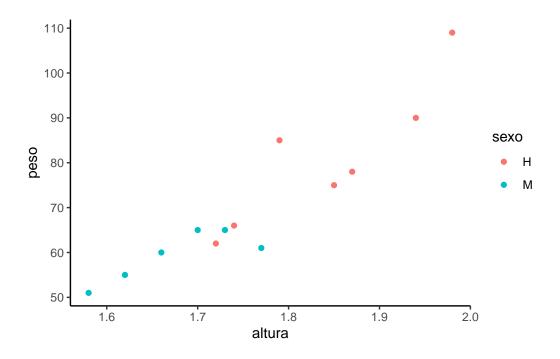
Más información sobre temas.

ggplot2 incorpora los siguientes temas predefinidos:

```
- `theme_gray()`: Tema en escala de grises. Es el tema por defecto.
- `theme_bw()`: Tema en blanco y negro.
- `theme_light()`: Tema con las líneas de los ejes y de la rejilla delgadas y atenuadas y
- `theme_dark()`: Similar al tema anterior pero con fondo oscuro.
- `theme_minimal()`: Tema sin fondo.
- `theme_classic()`: Tema sin rejilla.
- `theme_void()`: Tema vacío.
```

Ejemplo 7.33.

```
# Inicializar el gráfico con la altura en el atributo x, el peso en el atributo y, y el a
ggplot(df, aes(x = altura, y = peso, colour = sexo)) +
# Añadir la capa de puntos.
    geom_point() +
# Usar el tema sin rejilla
    theme_classic()
```



7.13 Ejercicios

Ejercicio 7.1. El fichero neonatos contiene información sobre los recién nacidos en un hospital durante un año.

a. Crear un tibble con los datos del fichero.

```
Solución
library(tidyverse)
df <- read_csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/ne
df
# A tibble: 320 x 8
    peso sexo
                edad
                             tabaco cigarros tabaco.antes apgar1 apgar5
                <chr>
                             <chr>
                                        <dbl> <chr>
                                                             <dbl>
                                                                    <dbl>
   <dbl> <chr>
                                                                 7
                                                                        7
   3.28 hombre mayor de 20 No
                                            0 No
 2
    3.49 hombre mayor de 20 No
                                            0 No
                                                                 7
                                                                        8
 3
   2.88 hombre mayor de 20 No
                                                                 5
                                            0 Si
                                                                        6
 4
    3.42 hombre mayor de 20 No
                                            0 No
                                                                 8
                                                                        8
 5
   3.31 hombre mayor de 20 No
                                                                 6
                                                                        7
                                            0 No
 6
   3.53 hombre mayor de 20 No
                                            0 No
                                                                 8
                                                                        8
    3.46 hombre mayor de 20 No
                                                                 7
                                                                        8
                                            0 No
```

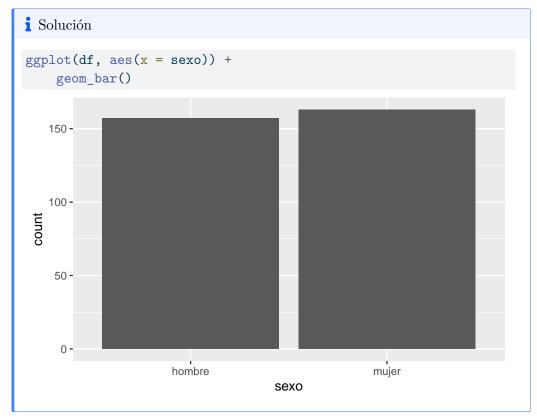
```
8 3.12 hombre mayor de 20 No 0 No 7 7
9 2.77 hombre mayor de 20 No 0 No 5
10 3.31 hombre mayor de 20 No 0 No 7 8
# i 310 more rows
```

b. Convertir las variables sexo, edad, tabaco y tabaco.antes en factores.

```
i Solución

df <- mutate(df, sexo = factor(sexo), edad = factor(edad), tabaco = factor(tabaco)
```

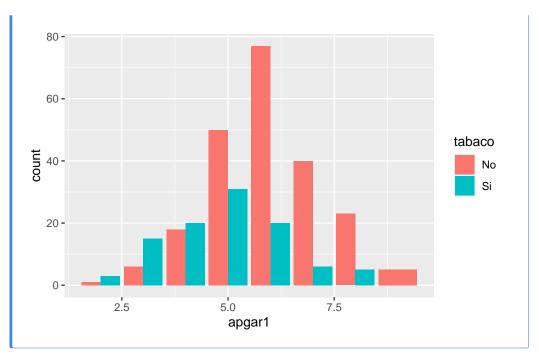
c. Dibujar un diagrama de barras con la frecuencia de niños y niñas.



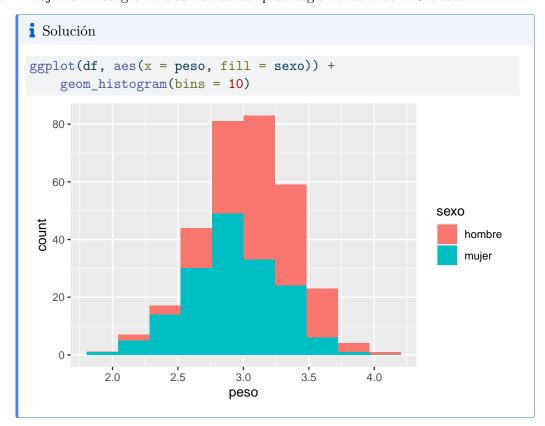
d. Dibujar un diagrama de barras del apgar1 de los neonatos de madres fumadoras y no fumadoras durante el embarazo.

```
i Solución

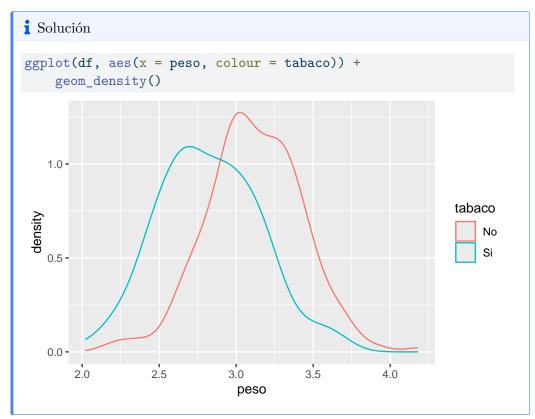
ggplot(df, aes(x = apgar1, fill = tabaco)) +
    geom_bar(position = "dodge")
```



e. Dibujar un histograma acumulado del peso según el sexo con 10 clases.



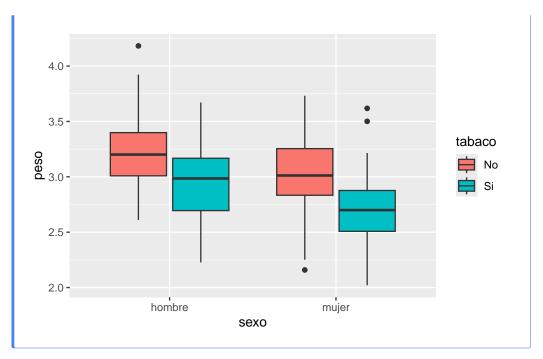
f. Dibujar un gráfico de densidad de probabilidad del peso de los neonatos de madres fumadoras y no fumadoras durante el embarazo.



g. Dibujar un diagrama de caja del peso de los recién nacidos según sexo y si la madre fumaba o no durante el embarazo.

```
i Solución

ggplot(df, aes(x = sexo, y = peso, fill = tabaco)) +
    geom_boxplot()
```

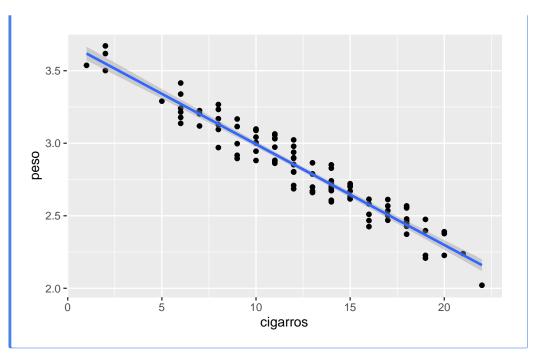


h. Dibujar un diagrama de dispersión del peso de los recién nacidos de madres fumadoras durante el embarazo, frente al número de cigarros que fumaban las madres. Incluir la recta de regresión.

```
i Solución

df %>%
    filter(tabaco == "Si") %>%
    ggplot(aes(x = cigarros, y = peso)) +
        geom_point() +
        geom_smooth(method = "lm")

`geom_smooth()` using formula = 'y ~ x'
```

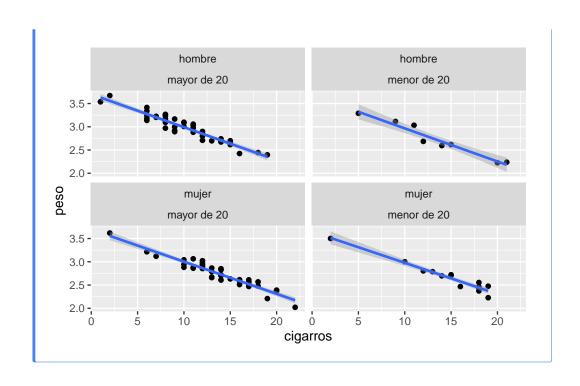


i. Dibujar un diagrama de dispersión del peso de los recién nacidos de madres fumadoras durante el embarazo, frente al número de cigarros que fumaban las madres, separando por facetas según el sexo y la edad de la madre.

```
i Solución

df %>%
    filter(tabaco == "Si") %>%
    ggplot(aes(x = cigarros, y = peso)) +
        geom_point() +
        geom_smooth(method = "lm") +
        facet_wrap(vars(sexo, edad))

`geom_smooth()` using formula = 'y ~ x'
```



8 Análisis Estadísticos

El tipo de estudio estadístico más apropiado en cada caso depende de varios factores:

- El objetivo del estudio.
- El número de variables que intervienen.
- El tipo de las variables dependientes e independientes.
- La naturaleza de las observaciones (independientes o emparejadas).

A continuación se presentan los estudios estadísticos más habituales en función de estos factores. La siguiente tabla de resumen de contrastes de hipótesis más habituales puede ayudar a identificar el más apropiado en cada caso.

Variables independientes

Variable dependiente

Objetivo

Ejemplo

Contraste

Ninguna (Una población)

Cuantitativa

Contrastar la normalidad de una variable

Comprobar si la nota de un examen tiene distribución normal (forma de campana de Gauss)

Komogorov-Smirnov (requiere muestras grandes)

Shapiro-Willks

Cuantitativa normal

Contrastar si la media poblacional de una variable tiene un valor determinado

Comprobar si la nota media de un examen es 5

Test T para la media de una población

Cuantitativa o cualitativa ordinal

Contrastar si la mediana poblacional de una variable tiene un valor determinado

Comprobar si la calificación mediana de un examen es Aprobado

Test para la mediana de una población

Cualitativa (2 categorías)

Contrastar si la proporción poblacional de una de las categorías tiene un valor determinado

Comprobar si la proporción de aprobados es de la mitad (o que el porcentaje es 50%)

Test Binomial

Cualitativa

Contrastar si las proporciones de cada una de las categorías tienen un valor determinado

Comprobar si las proporciones de alumnos matriculados en ciencias, letras o mixtas son $0.5,\,0.2$ y 0.3 respectivamente

Test Chi-cuadrado de bondad de ajuste

Una cualitativa con dos categorías independientes (Dos poblaciones independientes)

Cuantitativa normal

Contrastar si hay diferencias entre las medias la variable dependiente en dos poblaciones independientes

Comprobar si el grupo de mañana y el grupo de tarde han tenido notas medias diferentes

Test T para la comparación de medias de poblaciones independientes

Contrastar si hay diferencias entre las varianzas de la variable dependiente en dos poblaciones independientes

Comprobar si hay diferencias entre la variabilidad de las notas del grupo de mañana y el de tarde

Test F de Fisher

Contrastar si hay concordancia o acuerdo entre las dos variables

Comprobar si hay concordancia o acuerdo entre las notas que ponen dos profesores distintos para los mismos exámenes

Correlación intraclase

Cuantitativa o cualitativa ordinal

Contrastar si hay diferencias entre las distribuciones de la variable dependiente en dos poblaciones independientes

Comprobar si el grupo de mañana y el grupo de tarde han tenido calificaciones diferentes

Test de la U de Mann-Whitney

Contrastar si hay concordancia o acuerdo entre las dos variables

Comprobar si hay concordancia o acuerdo entre las calificaciones que ponen dos profesores distintos para los mismos exámenes

Kappa de Cohen

Cualitativa

Contrastar si hay relación entre las dos variables o bien si hay diferencias entre las proporciones de las categorías de la variable dependiente en las dos poblaciones definidas por las categorías de la variable independiente

Comprobar si existe relación entre los aprobados en una asignatura y el grupo al que pertenecen los alumnos, es decir, si la proporción de aprobados es diferente en dos grupos distintos.

Test Chi-cuadrado (si no ha más del 20% de frecuencias esperadas menores que 5) Test exacto de Fisher

Contrastar si hav concordancia o acuerdo entre las dos variables

Comprobar si hay concordancia o acuerdo entre la valoración (aprobado o suspenso) que hacen dos profesores distintos para los mismos exámenes

Kappa de Cohen

Una cualitativa con dos categorías relacionadas o pareadas (Dos poblaciones relacionadas o pareadas)

Cuantitativa normal

Contrastar si hay diferencias entre las medias de la variable dependiente en dos poblaciones relacionadas o pareadas

Comprobar si las notas medias de dos asignaturas cursadas por los mismos alumnos han sido diferentes o si las notas medias de un examen realizado al comienzo del curso (antes) y otro al final (después) de una misma asignatura han sido diferentes

Test T para la comparación de medias de poblaciones relacionadas o pareadas

Cuantitativa o cualitativa ordinal

Contrastar si hay diferencias entre las distribuciones de la variable dependiente en dos poblaciones relacionadas o pareadas

Comprobar si las calificaciones de dos asignaturas cursadas por los mismos alumnos han sido diferentes

Test de Wilcoxon

Cualitativa con dos categorías

Contrastar si hay diferencias entre las proporciones de las categorías de la variable dependiente en dos poblaciones relacionadas o pareadas

Comprobar si la proporción o el porcentaje de aprobados en un examen es distinta al comienzo y al final del curso

Test de McNemar

Una cualitativa con dos o más categorías

independientes (Dos o más poblaciones independientes)

Cuantitativa normal y homogeneidad de varianzas

Contrastar si hay diferencias entre las medias la variable dependiente en cada una de las poblaciones definidas por las categorías de la variable independiente

Comprobar si existen diferencias entre las notas medias de tres grupos distintos de clase.

Análisis de la Varianza de un factor (ANOVA) Si hay diferencias > Test de Tukey o Bonferroni para la diferencia por pares

Cuantitativa normal

Contrastar si hay diferencias entre las varianzas de la variable dependiente en cada una de las poblaciones definidas por las categorías de la variable independiente

Comprobar si la variabilidad de las notas de una asignatura es distinta en tres grupos diferentes de clase

Prueba de Levene para la homogeneidad de varianzas

Cuantitativa o cualitativa ordinal

Contrastar si hay diferencias entre las distribuciones de la variable dependiente en cada una de las poblaciones definidas por las categorías de la variable independiente

Comprobar si existen diferencias entre las calificaciones de tres grupos distintos de clase

Test de Kruskal Wallis

Cualitativa

Contrastar si hay relación entre las dos variables o bien si hay diferencias entre las proporciones de las categorías de la variable dependiente en cada una de las poblaciones definidas por las categorías de la variable independiente

Comprobar si existe relación entre los aprobados en una asignatura y el grupo al que pertenecen los alumnos, es decir, si la proporción de aprobados es diferente en los distintos grupos.

Test Chi-cuadrado (si no ha más del 20% de frecuencias esperadas menores que 5) Test exacto de Fisher

Una cualitativa con dos o más categorías relacionadas (medidas repetidas)

Cuantitativa normal

Contrastar si hay diferencias entre las medias repetidas de la variable dependiente

Comprobar si hay diferencias entre las notas que otorgan varios profesores a un mismo examen

Análisis de la Varianza (ANOVA) de medidas repetidas de un factor

Cuantitativa o cualitativa ordinal

Contrastar si hay diferencias entre las medidas repetidas de la variable dependiente

Comprobar si hay diferencias entre las calificaciones que otorgan varios profesores a un mismo examen

Test de Friedman

Cualitativa

Contrastar si hay diferencias entre las valoraciones repetidas de la variable dependiente

Comprobar si hay diferencias entre la valoración (aprobado o suspenso) que hacen varios profesores de un mismo examen

Regresión logística de medidas repetidas

Una cuantitativa normal

Cuantitativa normal

Contrastar si existe relación lineal entre las dos variables

Comprobar si existe relación entre las notas de dos asignaturas

Correlación de Pearson

Construir un modelo predictivo que explique la variable dependiente en función de la independiente

Construir el modelo (función de regresión) que mejor explique la relación entre la nota de un examen y las horas dedicadas a su estudio

Regresión simple (lineal o no lineal)

Cuantitativa o cualitativa ordinal

Contrastar si existe relación lineal entre las dos variables

Comprobar si existe relación entre las calificaciones de dos asignaturas

Correlación de Spearman

Cualitativa

Construir un modelo predictivo que explique la variable dependiente en función de la independiente

Construir el modelo (función logística) que mejor explique la relación entre el resultado de un examen (aprobado o suspenso) y las horas dedicadas a su estudio

Regresión logística simple

Los ejemplos de los distintos test que se presentan a continuación se han realizado a partir del siguiente conjunto de datos que contiene las notas y calificaciones de un curso. El fichero con los datos puede descargarse aquí para reproducir los estudios: notascurso.csv.

```
library(tidyverse)
df <- read.csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/notas-cur
head(df)</pre>
```

	sexo	turno	grupo	trabaja	notaA	notaB	notaC	notaD	notaE	calificacionA	
1	Mujer	Tarde	C	N	5.2	6.3	3.4	2.3	2.0	Aprobado	
2	Hombre	Mañana	Α	N	5.7	5.7	4.2	3.5	2.7	Aprobado	
3	Hombre	Mañana	В	N	8.3	8.8	8.8	8.0	5.5	Aprobado	
4	Hombre	Mañana	В	N	6.1	6.8	4.0	3.5	2.2	Aprobado	
5	Hombre	Mañana	Α	N	6.2	9.0	5.0	4.4	3.7	Aprobado	
6	Hombre	Mañana	Α	S	8.6	8.9	9.5	8.4	3.9	Aprobado	
	${\tt calificacionB}$		calif	icacionC	calif	icacio	nD cal:	ificac	ionE as	sinaturas.aprob	adas
1	Aprobado		Suspenso		Suspenso		30	Suspe	enso		2
2	Aprobado		Suspenso		Suspenso		30	Suspenso			2
3	Aprobado		Aprobado		Aprobado		do	Aprobado			5
4	Aprobado		Suspenso		Suspenso		30	Suspenso			2
5	Aprobado		1	Aprobado		Suspenso		Suspenso			3
6	Aprobado		1	Aprobado		Aprobado		Suspenso			4
	nota.me	edia									
1		3.8									
2		4.3									
3		7.9									
4		4.5									
5		5.7									
6		7.8									

8.1 Una variable cuantitativa

8.1.1 Estudios descriptivos

8.1.1.1 Estadísticos

- Tamaño muestral
- Media
- Desviación típica
- Mínimo, Máximo
- Cuartiles
- Coeficiente de asimetría
- Coeficiente de apuntamiento

```
# Tamaño muestral
nrow(df)

[1] 120

# Media
mean(df$notaA, na.rm = TRUE)

[1] 6.028333

# Desviación típica
sd(df$notaA, na.rm = TRUE)
[1] 1.340524
```

[1] 2.5

Min, max

min(df\$notaA, na.rm = TRUE)

```
max(df$notaA, na.rm = TRUE)
```

[1] 9.3

```
# Cuartiles
quantile(df$notaA, c(0.25, 0.5, 0.75), na.rm = TRUE)

25% 50% 75%
5.100 5.900 6.825

# Coef. asimetría
library(moments)
skewness(df$notaA, na.rm = TRUE)

[1] 0.1373915

# Coef. apuntamiento
kurtosis(df$notaA, na.rm = TRUE) - 3
```

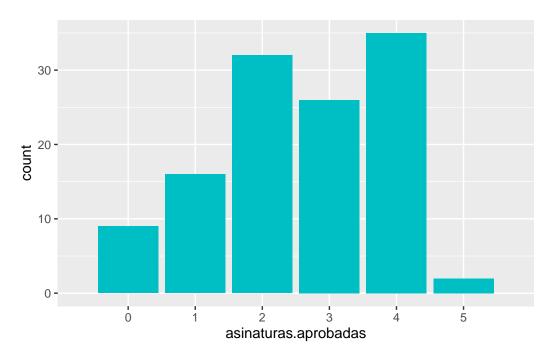
[1] -0.102287

8.1.1.2 Gráficos

• Diagrama de barras (variables discretas)

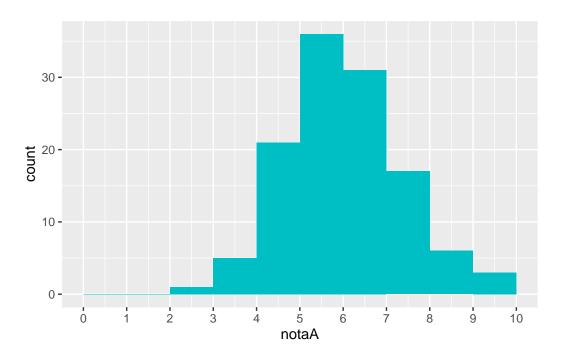
```
df %>% ggplot(aes(x = asinaturas.aprobadas)) +
  geom_bar(fill="#00BFC4") +
  # Cambio de escala del eje X
  scale_x_discrete(limits=0:5)
```

Warning: Continuous limits supplied to discrete scale.
i Did you mean `limits = factor(...)` or `scale_*_continuous()`?



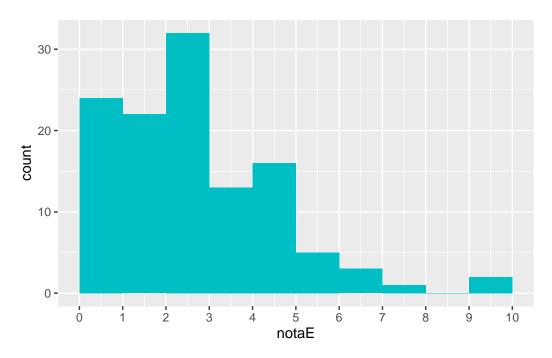
• Histograma

```
library(ggplot2)
# Limites de los intervalos
breaks = 0:10
# Histograma de las notasA
df %>% ggplot(aes(x = notaA)) +
  geom_histogram(breaks = breaks, fill="#00BFC4") +
  # Cambio de escala del eje X
  scale_x_continuous(limits=c(0, 10), breaks = 0:10)
```



```
# Histograma de notasE
df %>% ggplot(aes(x = notaE)) +
  geom_histogram(breaks = breaks, fill="#00BFC4") +
  # Cambio de escala del eje X
  scale_x_continuous(limits=c(0, 10), breaks = 0:10)
```

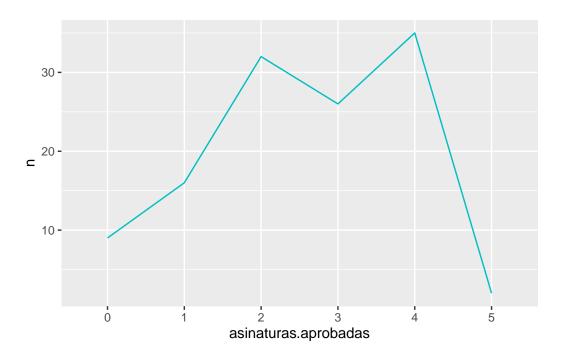
Warning: Removed 2 rows containing non-finite values (`stat_bin()`).



• Diagrama de líneas

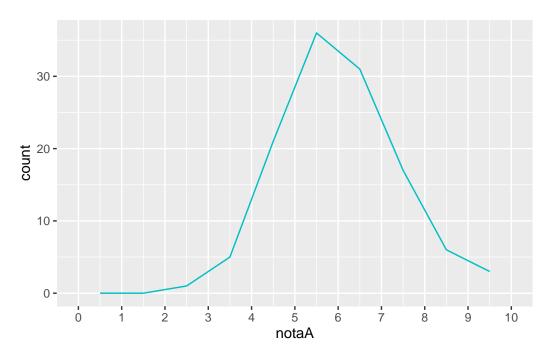
```
# Variables discretas
df %>% count(asinaturas.aprobadas) %>%
   ggplot(aes(x = asinaturas.aprobadas, y = n)) +
   geom_line(col="#00BFC4") +
   # Cambio de escala del eje X
   scale_x_discrete(limits=0:5)
```

Warning: Continuous limits supplied to discrete scale.
i Did you mean `limits = factor(...)` or `scale_*_continuous()`?



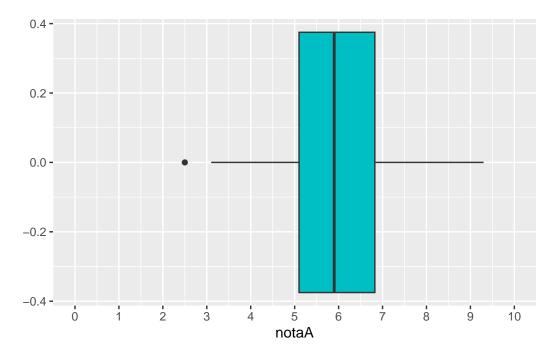
```
# Agrupación de datos en intervalos
df %>% ggplot(aes(x = notaA)) +
  geom_freqpoly(breaks = breaks, col="#00BFC4") +
  # Cambio de escala del eje X
  scale_x_continuous(limits=c(0, 10), breaks = 0:10)
```

Warning: Removed 2 rows containing missing values (`geom_path()`).



• Diagrama de caja y bigotes

```
df %>% ggplot(aes(x = notaA)) +
  geom_boxplot(fill="#00BFC4") +
  # Cambio de escala del eje X
  scale_x_continuous(limits=c(0, 10), breaks = 0:10)
```



8.1.2 Estudios inferenciales

8.1.2.1 Test de normalidad de Shapiro-Wilk

Objetivo: Comprobar la normalidad de la distribución.

Hipótesis nula: La distribución es normal.

```
shapiro.test(df$notaA)
```

Shapiro-Wilk normality test

data: df\$notaA W = 0.99424, p-value = 0.907

shapiro.test(df\$notaE)

Shapiro-Wilk normality test

data: df\$notaE W = 0.92264, p-value = 4.065e-06

8.1.2.2 Test t para la media de una población

Objetivo: Estimar la media de una variable o compararla con un valor dado 0.

Requisitos:

- Una variable cuantitativa.
- Distribución normal o tamaño muestral 30.

Hipótesis nula: La media de la población es igual a 0.

Ejemplo: Comprobar si la nota media de un examen es diferente de 5.

t.test(df\$notaA, mu = 5, alternative = "two.sided")

```
One Sample t-test
data: df$notaA
t = 8.4033, df = 119, p-value = 1.08e-13
alternative hypothesis: true mean is not equal to 5
95 percent confidence interval:
5.786023 6.270643
sample estimates:
mean of x
6.028333
```

8.2 Una variable cualitativa

8.2.1 Estudios descriptivos

8.2.1.1 Estadísticos

- Tamaños muestral
- Frecuencias muestrales
- Proporciones/porcentajes muestrales

```
# Tamaño muestral sin datos perdidos
length(na.omit(df$calificacionB))
[1] 115
# Frecuencias
table(df$calificacionB)
```

```
# Proporciones
table(df$calificacionB) / length(na.omit(df$calificacionB))
```

```
Aprobado Suspenso
0.8521739 0.1478261
```

Aprobado Suspenso 98

17

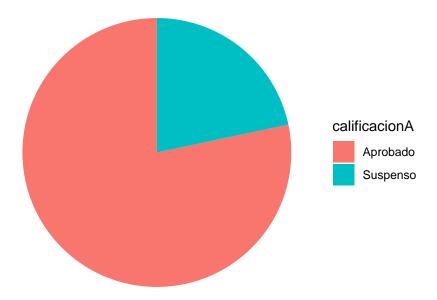
```
# Porcentajes
table(df$calificacionB) / length(na.omit(df$calificacionB)) * 100
```

Aprobado Suspenso 85.21739 14.78261

8.2.1.2 Gráficos

• Diagrama de sectores

```
df %>% ggplot(aes(x = "", fill = calificacionA)) +
  geom_bar() +
  # Cambiar a coordenadas polares
  coord_polar(theta = "y") +
  # Eliminar ejes
  theme_void()
```



8.2.2 Estudios inferenciales

8.2.2.1 Test binomial para una proporción de una población

Objetivo: Estimar la propoción de una categoría en una población o compararla con un valor p0.

Requisitos:

• One variable cualitativa

Hipótesis nula: La proporción poblacional es igual a p0.

Ejemplo: Comprobar si la proporción de aprobados es mayor de 0.5.

```
n <- nrow(df)
freq <- table(df$calificacionA)["Aprobado"]
binom.test(freq, n, p = 0.5, alternative = "greater")</pre>
```

Exact binomial test

8.2.2.2 Test Z para la proporción de una población

Objetivo: Estimar la propoción de una categoría en una población o compararla con un valor p0.

Requisitos:

- Una variable cualitativa
- Tamaño muestral >= 30

Observación: Utiliza la aproximación normal de la distribución Binomal.

Ejemplo: Comprobar si la proporción de aprobados es mayor de 0.5.

```
freq <- table(df$calificacionA)["Aprobado"]
prop.test(freq, n, p = 0.7, alternative = "greater")</pre>
```

1-sample proportions test with continuity correction

```
data: freq out of n, null probability 0.7
X-squared = 3.5813, df = 1, p-value = 0.02922
```

```
alternative hypothesis: true p is greater than 0.7 95 percent confidence interval: 0.7111099 1.0000000 sample estimates:

p
0.7833333
```

8.3 Dos variables: Variable dependiente cuantitativa y variable independiente culitativa con dos categorías o grupos

8.3.1 Estudios descriptivos

8.3.1.1 Estadísticos

- Tamaño muestral de cada grupo
- Media de cada grupo
- Desviación típica de cada grupo
- Mínimo, Máximo de cada grupo
- Cuartiles de cada grupo
- Coeficiente de asimetría de cada grupo
- Coeficiente de apuntamiento de cada grupo

```
# Tamaño muestral de notaA según el sexo
df %>% group_by(sexo) %>% group_size()
```

[1] 71 49

```
# Media, Desviación típica, Mín, Máx, Cuartiles, Coef. Asimetría y Coef. Apuntamiento
library(moments)
df %>% group_by(sexo) %>% summarize(Media = mean(notaA, na.rm=TRUE), Des.Tip = sd(notaA,
```

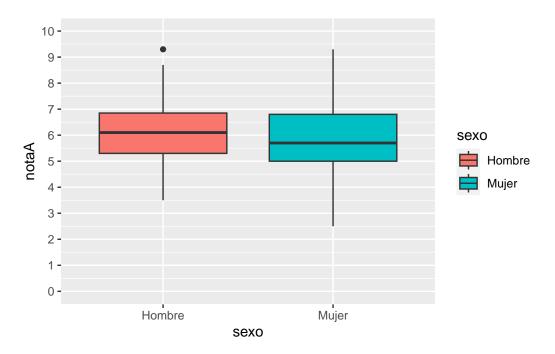
```
# A tibble: 2 x 10
      Media Des.Tip
                           C1
                               C2
                                    C3 Coef.Asimetría
 sexo
                 Mín
                      Máx
 <dbl>
1 Hombre 6.12
            1.23
                  3.5
                      9.3
                          5.3
                               6.1 6.85
                                            0.249
2 Mujer
       5.89
             1.49
                  2.5
                      9.3
                          5
                               5.7 6.8
                                            0.135
```

i 1 more variable: Coef.Apuntamiento <dbl>

8.3.1.2 Gráficos

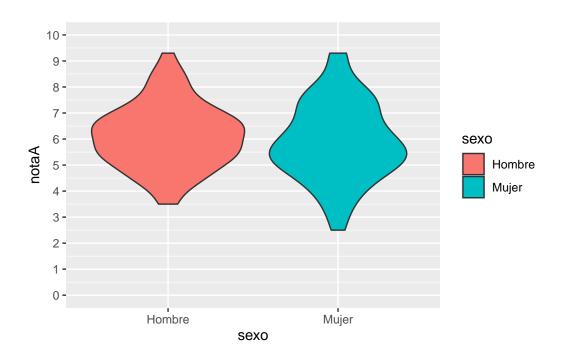
• Diagrama de cajas y bigotes

```
df %>% ggplot(aes(x = sexo, y = notaA, fill = sexo)) +
  geom_boxplot() +
  # Cambio de escala del eje y
  scale_y_continuous(limits=c(0, 10), breaks = 0:10)
```



• Diagrama de violín

```
df %>% ggplot(aes(x = sexo, y = notaA, fill = sexo)) +
  geom_violin() +
  # Cambio de escala del eje y
  scale_y_continuous(limits=c(0, 10), breaks = 0:10)
```



8.3.2 Estudios inferenciales

8.3.2.1 Test de normalidad de Shapiro-Wilks

Objetivo: Comprobar la normalidad de la distribución de cada población.

Hipótesis nula: La distribución es normal.

```
df %>% group_by(sexo) %>%
  summarise(`Estadístico W` = shapiro.test(notaA)$statistic, `p-valor` = shapiro.test(notaA)$statistic, `p-valor` = shapiro.test(notaA)$statistic
```

0.990

8.3.2.2 Test F de Fisher de comparación de varianzas de dos poblaciones independientes

0.942

Objetivo: Comparar las varianzas de dos poblaciones independientes.

Requisitos:

2 Mujer

- Variable dependiente cuantitativa.
- Una variable independiente cualitativa con dos categorías (poblaciones)
- Distribución normal de la variable dependiente en ambas poblaciones o tamaños de las muestras de cada población 30.

Hipótesis nula: La varianzas poblacionales son iguales (no existe una diferencia significativa entre las medias poblacionales).

Ejemplo: Comprobar si diferencias signicativas entre las notas medias de hombres y mujeres.

```
# Test de comparación de varianzas
var.test(notaA ~ sexo, data = df)
```

F test to compare two variances

```
data: notaA by sexo
F = 0.6769, num df = 70, denom df = 48, p-value = 0.1347
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
    0.3953421 1.1293155
sample estimates:
ratio of variances
    0.6769032
```

8.3.2.3 Test t de comparación de medias de dos poblaciones independientes

Objetivo: Estimar la diferencia de medias en las dos poblaciones o comprobar si hay diferencias significativas entre ellas.

Requisitos:

- Variable dependiente cuantitativa.
- Una variable independiente cualitativa con dos categorías (poblaciones)
- Distribución normal de la variable dependiente en ambas poblaciones o tamaños de las muestras de cada población 30.

Hipótesis nula: La medias poblacionales son iguales (no existe una diferencia significativa entre las medias poblacionales).

Observación: El resultado del test depende de si las varianzas poblacionales son iguales o no.

Ejemplo: Comprobar si diferencias signicativas entre las notas medias de hombres y mujeres.

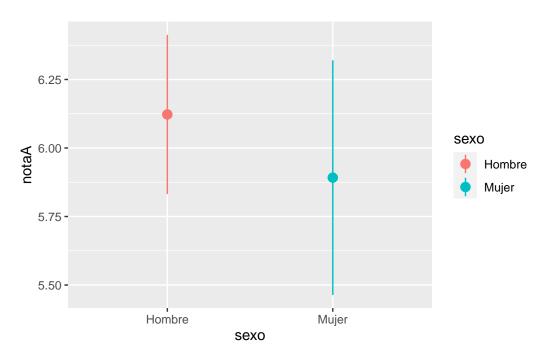
```
# Test de comparación de varianzas
var.test(notaA ~ sexo, data = df)
   F test to compare two variances
data: notaA by sexo
F = 0.6769, num df = 70, denom df = 48, p-value = 0.1347
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
0.3953421 1.1293155
sample estimates:
ratio of variances
        0.6769032
# Test de comparación de medias asumiendo varianzas iguales
t.test (notaA ~ sexo, data = df, alternative = "two.sided", var.equal = FALSE)
   Welch Two Sample t-test
data: notaA by sexo
t = 0.89364, df = 89.873, p-value = 0.3739
alternative hypothesis: true difference in means between group Hombre and group Mujer is
95 percent confidence interval:
-0.2821809 0.7435779
sample estimates:
mean in group Hombre mean in group Mujer
            6.122535
                                 5.891837
# Test de comparación de medias asumiendo varianzas iguales
t.test (notaA ~ sexo, data = df, alternative = "two.sided", var.equal = TRUE)
   Two Sample t-test
data: notaA by sexo
t = 0.92608, df = 118, p-value = 0.3563
alternative hypothesis: true difference in means between group Hombre and group Mujer is
95 percent confidence interval:
```

-0.262615 0.724012

```
sample estimates:
mean in group Hombre mean in group Mujer
6.122535 5.891837
```

• Diagrama de medias

```
df %>% ggplot(aes(x = sexo, y = notaA, colour = sexo)) +
    # Puntos de medias
    stat_summary(fun="mean", size=3, geom="point", position=position_dodge(width=0.25)) +
    # Intervalos de confianza para la media
    stat_summary(fun.data = function(x) mean_cl_normal(x, conf.int=0.95), geom = "pointrang")
```



8.3.2.4 Test U de Mann-Whitney de comparación de dos poblaciones independientes (no paramétrico)

Objetivo: Comprobar si hay diferencias significativas entre entre dos poblaciones independientes.

Requisitos:

- Variable dependiente cuantitativa.
- Una variable independiente cualitativa con dos categorías (poblaciones)

Hipótesis nula: La medianas poblacionales son iguales (no existe una diferencia significativa entre las medianas poblacionales).

Ejemplo: Comprobar si diferencias signicativas entre las notas de hombres y mujeres.

```
# Test de rangos U the Mann-Whitney
wilcox.test(notaA ~ sexo, data = df, alternative = "two.sided")
```

Wilcoxon rank sum test with continuity correction

```
data: notaA by sexo
W = 1917, p-value = 0.3445
alternative hypothesis: true location shift is not equal to 0
```

8.4 Dos variables: Variable dependiente cuantitativa y variable independiente culitativa con dos categorías o grupos pareados

Dos grupos o poblaciones están pareadas o emparejadas cuando los dos poblaciones contienen los mismos individuos, es decir, se trata en realidadad de una única población, pero la variable dependiente se mide dos veces en cada individuo (normalmente antes y después de la algún suceso) y por tanto cada individuo tiene asociado un par de valores.

Este estudio puede realizarse también creando una nueva variable a partir de la resta de las dos mediciones y planteando un estudio de una sola variable cuantitativa.

Ejemplo: Creación de la diferencia de notas de las asignaturas A y B.

```
# Creamos la variable diferencia = notaA - notaB
df <- df %>% mutate(diferencia = notaA - notaB)
```

8.4.1 Estudios descriptivos

8.4.1.1 Estadísticos

- Tamaño muestral del grupo
- Media de la diferencia
- Desviación típica de la diferencia
- Mínimo, Máximo de la diferencia
- Cuartiles de la diferencia
- Coeficiente de asimetría de la diferencia

• Coeficiente de apuntamiento de la diferencia

Ejemplo: Estadísticos descriptivos de la diferencia entre las notas de las asignaturas A y B de un mismo grupo de alumnos.

```
# Tamaño muestral de sin contar los datos perdidos
length(na.omit(df$diferencia))
```

[1] 115

```
# Media, Desviación típica, Mín, Máx, Cuartiles, Coef. Asimetría y Coef. Apuntamiento
library(moments)
df %>% summarize(Media = mean(diferencia, na.rm=TRUE), Des.Tip = sd(diferencia, na.rm = '')
```

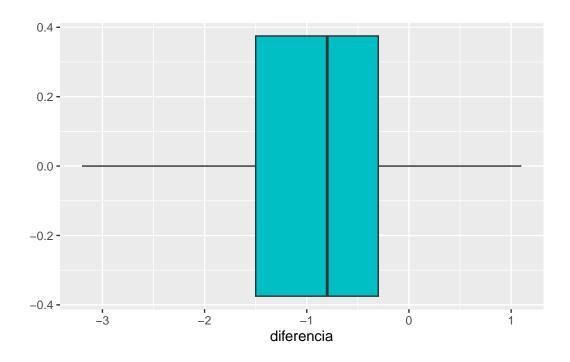
```
Media Des.Tip Mín Máx C1 C2 C3 Coef.Asimetría Coef.Apuntamiento 1 -0.8817391 0.9000568 -3.2 1.1 -1.5 -0.8 -0.3 -0.4304152 -0.1366944
```

8.4.1.2 Gráficos

• Diagrama de cajas y bigotes

```
df %>% ggplot(aes(x = diferencia)) +
  geom_boxplot(fill="#00BFC4")
```

Warning: Removed 5 rows containing non-finite values (`stat_boxplot()`).



8.4.2 Estudios inferenciales

8.4.2.1 Test de normalidad de Shapiro-Wilks

Objetivo: Comprobar la normalidad de la distribución de la diferencia.

Hipótesis nula: La distribución es normal.

Ejemplo: Comprobar la normalidad de la diferencia entre las notas de las asignaturas A y B de un mismo grupo de alumnos.

```
df %>% summarise(`Estadístico W` = shapiro.test(diferencia)$statistic, `p-valor` = shapiro.
```

```
Estadístico W p-valor
1 0.9794023 0.07367451
```

8.4.2.2 Test t de comparación de medias de dos poblaciones pareadas

Objetivo: Estimar la media de la diferencia o compararla con un valor dado 0.

Requisitos:

- Variable dependiente cuantitativa.
- Una variable independiente cualitativa con dos categorías (poblaciones) pareadas

• Distribución normal de la variable diferencia o tamaño muestral 30.

Hipótesis nula: La medias poblacionales son iguales (no existe una diferencia significativa entre las medias poblacionales).

Ejemplo: Comprobar si hay una diferencia signicativa entre las notas medias de las asinaturas A y B, o lo que es lo mismo, comprobar si la media de la diferencia de las notas de A y B es distinta de 0.

```
t.test (df$notaA, df$notaB, alternative = "two.sided", paired = TRUE)
```

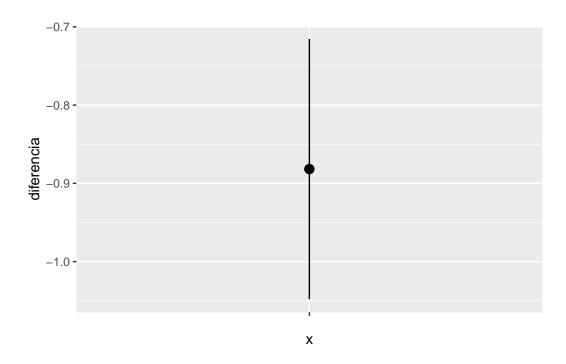
Paired t-test

```
data: df$notaA and df$notaB
t = -10.506, df = 114, p-value < 2.2e-16
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
   -1.048005 -0.715473
sample estimates:
mean difference
   -0.8817391</pre>
```

• Diagrama de medias

```
df %>% ggplot(aes(x="", y = diferencia)) +
    # Puntos de medias
    stat_summary(fun="mean", size=3, geom="point") +
    # Intervalos de confianza para la media
    stat_summary(fun.data = function(x) mean_cl_normal(x, conf.int=0.95), geom = "pointrang")
```

Warning: Removed 5 rows containing non-finite values (`stat_summary()`). Removed 5 rows containing non-finite values (`stat_summary()`).



8.4.2.3 Test Wilcoxon de comparación de dos poblaciones pareadas (no paramétrico)

Objetivo: Comparar las medianas de las dos poblaciones.

Requisitos:

- Variable dependiente cuantitativa.
- Una variable independiente cualitativa con dos categorías (poblaciones) pareadas.

Hipótesis nula: La medianas poblacionales son iguales (no existe una diferencia significativa entre las medianas poblacionales).

 $\bf Ejemplo:$ Comprobar si hay una diferencia signicativa entre las notas de las asignaturas A y B.

```
wilcox.test(df$notaA, df$notaB, alternative = "two.sided", paired = TRUE)
```

Wilcoxon signed rank test with continuity correction

data: df\$notaA and df\$notaB
V = 398, p-value = 2.442e-15

alternative hypothesis: true location shift is not equal to 0

8.5 Dos variables: Variable dependiente cuantitativa y variable independiente culitativa con más de dos categorías o grupos

8.5.1 Estudios descriptivos

8.5.1.1 Estadísticos

- Tamaño muestral de cada grupo
- Media de cada grupo
- Desviación típica de cada grupo
- Mínimo, Máximo de cada grupo
- Cuartiles de cada grupo
- Coeficiente de asimetría de cada grupo
- Coeficiente de apuntamiento de cada grupo

```
# Tamaño muestral de notaA según el grupo
df %>% group_by(grupo) %>% group_size()
```

```
[1] 38 35 47
```

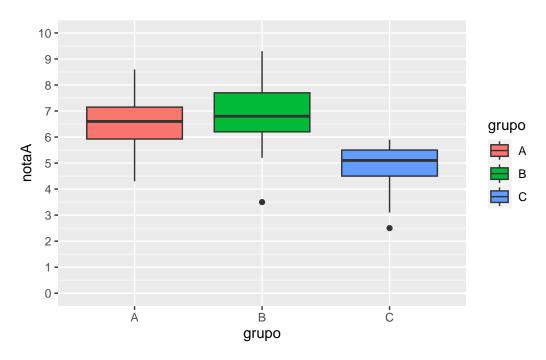
```
# Media, Desviación típica, Mín, Máx, Cuartiles, Coef. Asimetría y Coef. Apuntamiento
library(moments)
df %>% group_by(grupo) %>% summarize(Media = mean(notaA, na.rm=TRUE), Des.Tip = sd(notaA)
```

```
# A tibble: 3 x 10
                                                 C3 Coef.Asimetría
 grupo Media Des.Tip
                       Mín
                                     C1
                                           C2
                              Máx
 <chr> <dbl>
              <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <
                                                            <dbl>
1 A
         6.54
               0.998
                        4.3
                             8.6 5.93
                                          6.6 7.15
                                                            -0.250
2 B
        6.96
              1.23
                        3.5
                              9.3 6.2
                                          6.8 7.7
                                                            -0.141
                                                            -1.06
3 C
        4.92
               0.771
                        2.5
                              5.9 4.5
                                          5.1 5.5
# i 1 more variable: Coef.Apuntamiento <dbl>
```

8.5.1.2 Gráficos

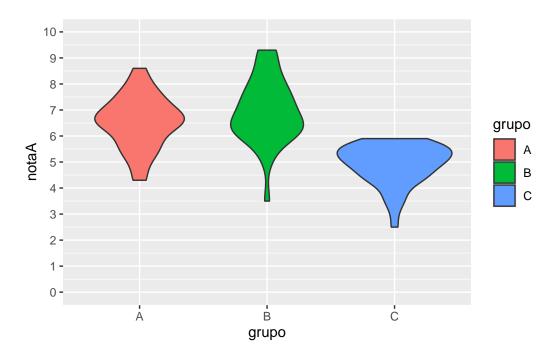
• Diagrama de cajas y bigotes

```
df %>% ggplot(aes(x = grupo, y = notaA, fill = grupo)) +
   geom_boxplot() +
   # Cambio de escala del eje X
   scale_y_continuous(limits=c(0, 10), breaks = 0:10)
```



• Diagrama de violín

```
df %>% ggplot(aes(x = grupo, y = notaA, fill = grupo)) +
   geom_violin() +
   # Cambio de escala del eje X
   scale_y_continuous(limits=c(0, 10), breaks = 0:10)
```



8.5.2 Estudios inferenciales

8.5.2.1 Test de normalidad de Shapiro-Wilks

Objetivo: Comprobar la normalidad de la distribución de cada población.

Hipótesis nula: La distribución es normal.

Ejemplo: Comprobar la normalidad de las distribuciones de la nota A en los grupos A, B y C.

```
df %>% group_by(grupo) %>%
  summarise(`Estadístico W` = shapiro.test(notaA)$statistic, `p-valor` = shapiro.test(notaA)$
```

Interpretación: La distribución de la nota A en los grupos A y B es normal (p-valores > 0.05) pero no en el grupo C (p-valor < 0.05)

Ejemplo: Comprobar la normalidad de las distribuciones de la nota C en los grupos A, B y C.

```
df %>% group_by(grupo) %>%
   summarise(`Estadístico W` = shapiro.test(notaC)$statistic, `p-valor` = shapi
```

Interpretación: La distribución de la nota C en los tres grupos es normal (p-valores > 0.05).

8.5.2.2 Test de Levene de comparación de varianzas de dos o más poblaciones independientes

Objetivo: Comparar las varianzas de dos o más poblaciones independientes.

Requisitos:

- Variable dependiente cuantitativa.
- Una variable independiente cualitativa con dos o más categorías (poblaciones)
- Distribución normal de la variable dependiente en todas las poblaciones o tamaños de las muestras de cada población 30.

Hipótesis nula: La varianzas poblacionales son iguales (no existe una diferencia significativa entre las medias poblacionales).

Ejemplo: Comprobar si diferencias signicativas entre las varianzas de las notas de la asignatura C de los grupos A, B y C.

```
# El test de Levene está disponible en el paquete car
library(car)
# Test de comparación de varianzas
leveneTest(notaC ~ grupo, data = df)
```

Warning in leveneTest.default(y = y, group = group, ...): group coerced to factor.

```
Levene's Test for Homogeneity of Variance (center = median)

Df F value Pr(>F)

group 2 0.3186 0.7278

116
```

Interpretación: No existe diferencia significativa entre las varianzas de la nota C en los grupos A, B y C (p-valor > 0.05).

8.5.2.3 ANOVA de un factor para la comparación medias de más de dos poblaciones independientes

Objetivo: Comprobar si hay diferencias significativas entre las medias de más de dos poblaciones independientes.

Requisitos:

- Variable dependiente cuantitativa.
- Una variable independiente cualitativa con más de dos categorías (poblaciones)-

- Distribución normal de la variable dependiente en todas las poblaciones o tamaños de las muestras de cada población 30.
- Homogeneidad de varianzas en las poblaciones.

Hipótesis nula: La medias poblacionales son iguales (no existe una diferencia significativa entre las medias poblacionales).

Ejemplo: Comprobar si diferencias signicativas entre las notas medias de la asignatura C de los grupos A, B y C.

```
# Análisis de la varianza de un factor
summary(aov(notaC ~ grupo, data = df))
```

```
Df Sum Sq Mean Sq F value Pr(>F)
grupo 2 80.69 40.34 20.05 3.32e-08 ***
Residuals 116 233.41 2.01
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
1 observation deleted due to missingness
```

Interpretación: Existen diferencias significativas entre las medias de la nota C entre al menos dos grupos (p-valor=3.32e-08 < 0.05).

Observación: Cuando se detectan diferencias significativas entre las medias de al menos dos grupos conviene realizar un test de comparación múltiple por pares para ver entre qué poblaciones hay diferencias y entre cuáles no. Los test más habituales de comparación por pares son el de Tukey y el de Bonferroni.

```
# Test de comparación múltiple de Tukey
TukeyHSD(aov(notaC ~ grupo, data = df))
```

```
Tukey multiple comparisons of means
95% family-wise confidence level

Fit: aov(formula = notaC ~ grupo, data = df)

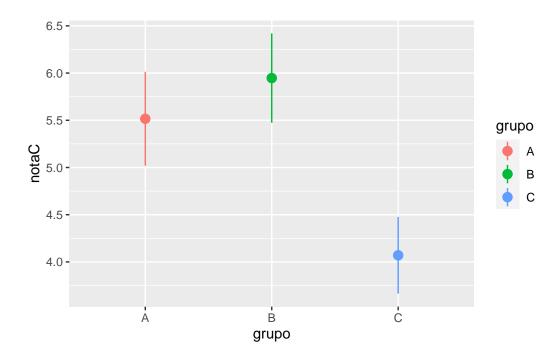
$grupo
diff lwr upr p adj
B-A 0.4312693 -0.3637573 1.2262960 0.4048482
C-A -1.4455767 -2.1802858 -0.7108676 0.0000241
C-B -1.8768461 -2.6350758 -1.1186163 0.0000001
```

Interpretación: No existe una diferencia significativa entre las notas medias de la asignatura C de los grupos A y B (p-valor=0.4048 > 0.05), pero si existe una diferencia significativa entre las notas medias de los grupos A y C (p-valor=0.00002 < 0.05) y también entre las notas medias de los grupos B y C (p-valor=0.0000001 < 0.05).

• Diagrama de medias

```
df %>% ggplot(aes(x = grupo, y = notaC, colour = grupo)) +
    # Puntos de medias
    stat_summary(fun="mean", size=3, geom="point", position=position_dodge(width=0.25)) +
    # Intervalos de confianza para la media
    stat_summary(fun.data = function(x) mean_cl_normal(x, conf.int=0.95), geom = "pointrang")
```

Warning: Removed 1 rows containing non-finite values (`stat_summary()`). Removed 1 rows containing non-finite values (`stat_summary()`).



8.5.2.4 Test Kruskal-Wallis de comparación de más de dos poblaciones independientes (no paramétrico)

Objetivo: Comprobar si hay diferencias significativas entre entre más de dos poblaciones independientes.

Requisitos:

- Variable dependiente cuantitativa.
- Una variable independiente cualitativa con más de dos categorías (poblaciones)

Hipótesis nula: La medianas poblacionales son iguales (no existe una diferencia significativa entre las medianas poblacionales).

Ejemplo: Comprobar si diferencias signicativas entre las notas de la asignatura A de los grupos A, B y C.

```
# Test de Kruskal-Wallis
kruskal.test(notaA ~ grupo, data = df)
```

Kruskal-Wallis rank sum test

```
data: notaA by grupo
Kruskal-Wallis chi-squared = 62.218, df = 2, p-value = 3.087e-14
```

Interpretación: Existen diferencias significativas entre las notas de la asignatura A de al menos dos de los grupos.

Observación: Cuando se detectan diferencias significativas entre al menos dos grupos conviene realizar un test de comparación múltiple por pares para ver entre qué poblaciones hay diferencias y entre cuáles no. El test más habitual es el de Wilcoxon.

```
# Test de comparación múltiple de Wilcoxon
pairwise.wilcox.test(df$notaA, df$grupo, p.adjust.method = "BH")
```

Warning in wilcox.test.default(xi, xj, paired = paired, \dots): cannot compute exact p-value with ties

Warning in wilcox.test.default(xi, xj, paired = paired, ...): cannot compute exact p-value with ties

Warning in wilcox.test.default(xi, xj, paired = paired, \dots): cannot compute exact p-value with ties

Pairwise comparisons using Wilcoxon rank sum test with continuity correction

data: df\$notaA and df\$grupo

A B

```
B 0.19 -
C 4.2e-10 1.3e-11
```

P value adjustment method: BH

Interpretación: No existe una diferencia significativa entre las notas de la asignatura A de los grupos A y B (p-valor=0.19 > 0.05), pero si existe una diferencia significativa entre las notas de los grupos A y C (p-valor=4.2e-10 < 0.05) y también entre las notas de los grupos B y C (p-valor=1.3e-11 < 0.05).

8.6 Dos variables: Variable dependiente cuantitativa y variable independiente cuantitativa

8.6.1 Estudios descriptivos

8.6.1.1 Estadísticos

8.6.1.2 Estadísticos

- Tamaño muestral de cada grupo
- Media de cada variable
- Desviación típica de cada variable
- Mínimo, Máximo de cada variable
- Cuartiles de cada variable
- Coeficiente de asimetría de cada variable
- Coeficiente de apuntamiento de cada variable

```
# Tamaño muestral de notaA según el grupo nrow(df)
```

[1] 120

```
# Media, Desviación típica, Mín, Máx, Cuartiles, Coef. Asimetría y Coef. Apuntamiento
library(moments)
df %>% summarize(Media = mean(notaA, na.rm=TRUE), Des.Tip = sd(notaA, na.rm = TRUE), Mín
```

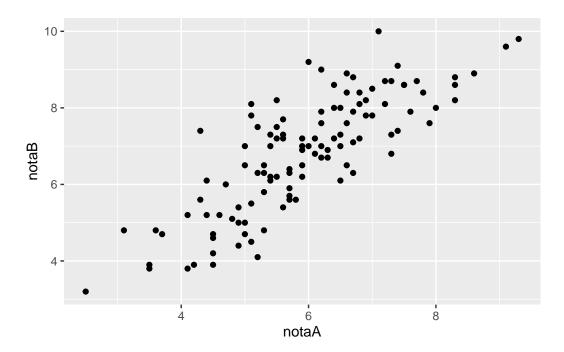
```
Media Des.Tip Mín Máx C1 C2 C3 Coef.Asimetría Coef.Apuntamiento
1 6.028333 1.340524 2.5 9.3 5.1 5.9 6.825 0.1373915 -0.102287
```

8.6.1.3 Gráficos

• Diagrama de dispersión

```
df %>% ggplot(aes(x = notaA, y = notaB)) +
  geom_point()
```

Warning: Removed 5 rows containing missing values (`geom_point()`).



8.6.2 Estudios inferenciales

8.6.2.1 Análisis de regresión

Objetivo: Construir un modelo matemático y = f(x) que explique lo mejor posible la variable dependiente en función de la independiente, para utilizarlo con fines predictivos.

Requisitos:

- Variable dependiente cuantitativa.
- Una variable independiente cuantitativa.

Ejemplo: Construir el modelo lineal (recta de regresión) que mejor explica la relación entre la nota B y la nota A.

```
# Análisis de la varianza de un factor
lm(notaB ~ notaA, data = df)
Call:
lm(formula = notaB ~ notaA, data = df)
Coefficients:
(Intercept)
                  notaA
     1.0224
                 0.9763
# Análisis de la varianza de un factor
summary(lm(notaB ~ notaA, data = df))
Call:
lm(formula = notaB ~ notaA, data = df)
Residuals:
    Min
               1Q
                   Median
                                3Q
                                        Max
-1.99907 -0.59551 -0.07534 0.61398 2.31991
Coefficients:
           Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.02243 0.40254
                                  2.54 0.0124 *
notaA
            0.97628
                       0.06637
                                 14.71
                                         <2e-16 ***
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 0.9035 on 113 degrees of freedom
  (5 observations deleted due to missingness)
                               Adjusted R-squared: 0.6539
Multiple R-squared: 0.6569,
F-statistic: 216.3 on 1 and 113 DF, p-value: < 2.2e-16
```

Interpretación: La recta de regresión explica el 65% de la variabilidad de la nota B.

8.6.2.2 Análisis de correlación

Objetivo: Estudiar el grado de relación entre las dos variables.

Requisitos:

- Variable dependiente cuantitativa.
- Una variable independiente cuantitativa.

Hipótesis nula: Las dos variables son independientes.

Ejemplo: Ver si las notas A y B están relacionadas y cuál es su grado de relación.

```
# Análisis de la varianza de un factor cor.test(df$notaA, df$notaB)
```

Pearson's product-moment correlation

```
data: df$notaA and df$notaB
t = 14.709, df = 113, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
    0.7367182  0.8651991
sample estimates:
        cor
0.8104926</pre>
```

Interpretación: Existe una correlación fuerte entre la nota A y la nota C, que además es estadísticamente muy significativa.

8.7 Ejercicios

Ejercicio 8.1. El fichero genetica, contiene información de la análitica fisiológica, microbiológica y bioquímica, de una muestra de ratas tratadas con distintos tratamientos.

a. Crear un tibble con los datos del fichero.

```
i Solución

library(tidyverse)

df <- read_csv('https://raw.githubusercontent.com/asalber/manual-r/master/datos/ged
```

```
# A tibble: 21 x 20
   `código muestra` Tratamiento
                                 `mas cor (g)`
                                                  IHS
                                                                   ITS
                                                                           IAS
                                                          IES
                    <chr>
                                                         <dbl>
   <chr>
                                         <dbl>
                                                <dbl>
                                                                 <dbl>
                                                                         <dbl>
 1 Ct-A1
                    Control
                                          236 0.0433 0.00198 0.00178 0.00043
 2 Ct-A2
                                          202. 0.044 0.002
                                                               0.0017
                                                                       0.0008
                    Control
                                                                       0.0005
 3 Ct-B1
                    Control
                                          246. 0.042 0.002
                                                               0.0015
                                          237. 0.0475 0.00238 0.00203 0.00041
 4 Ct-B2
                    Control
 5 Ct-B3
                    Control
                                          231. 0.0499 0.002
                                                               0.0011
                                                                      0.0005
                    Control
                                          235. 0.05
                                                      0.0021
                                                               0.0033
                                                                      0.0014
 6 Ct-C1
 7 Ct-C2
                    Control
                                          228. 0.05
                                                      0.0018
                                                              0.0031
                                                                      0.00059
                                          181. 0.061 0.00155 0.0005
                                                                     0.00012
 8 Dx-1
                    Dexametasona
                                          198. 0.063 0.00191 0.0011 0.0002
 9 Dx-2
                    Dexametasona
10 Dx-3
                    Dexametasona
                                          201. 0.0577 0.0017 0.0001 0.0001
# i 11 more rows
# i 13 more variables: `glucog(mg/g)` <dbl>, `pbmc(cel/mL)` <dbl>,
    `CD4(cel/µL)` <dbl>, `monoc(cel/mL)` <dbl>, `IHQ: linfB(cel/mm2)` <dbl>,
    `pulp.blan(Tx1)` <dbl>, `microorg(log nº/g)` <dbl>, `AI2(absorb)` <dbl>,
    `gluc(mmol/L)` <dbl>, `TAG(mmol/L)` <dbl>, `col(mmol/L)` <dbl>,
    `HDL(mmol/L)` <dbl>, `ins(pmol/L)` <dbl>
```

b. Convertir el tratamiento en un factor.

```
i Solución

df <- mutate(df, Tratamiento = factor(Tratamiento))
```

c. Calcular el tamaño muestral de cada grupo de tratamiento

d. Comprobar la normalidad de la variable IHS en cada grupo de tratamiento.

```
i Solución
df %>% group_by(Tratamiento) %>%
  summarise(`Estadístico W` = shapiro.test(IHS)$statistic, `p-valor` = shapiro.test
# A tibble: 3 x 3
 Tratamiento `Estadístico W` `p-valor`
  <fct>
                          <dbl>
                                    <dbl>
1 Control
                         0.837
                                   0.0928
2 Dexametasona
                         0.976
                                   0.939
                         0.908
                                   0.383
3 Kanamicina
```

e. Comprobar la homocedasticiad de varianzas de la variable IHS en los tres grupos de tratamiento.

f. Realizar un contraste de ANOVA para comparar las medias de la variable IHS entre los tres grupos de tratamiento.

```
i Solución

# Análisis de la varianza de un factor
summary(aov(IHS ~ Tratamiento, data = df))

Df Sum Sq Mean Sq F value Pr(>F)
Tratamiento 2 0.0007900 3.95e-04 31.78 1.24e-06 ***
Residuals 18 0.0002237 1.24e-05
---
Signif. codes: 0 '***' 0.001 '**' 0.05 '.' 0.1 ' ' 1
```

g. Realizar los contrastes de comparación de medias de la variable IHS por pares de trataimentos.

```
i Solución
# Test de comparación múltiple de Tukey
TukeyHSD(aov(IHS ~ Tratamiento, data = df))
 Tukey multiple comparisons of means
   95% family-wise confidence level
Fit: aov(formula = IHS ~ Tratamiento, data = df)
$Tratamiento
                              diff
                                          lwr
                                                      upr
                                                             p adj
Dexametasona-Control
                      Kanamicina-Control
                      0.0007857143 -0.004023559 0.005594988 0.9090945
Kanamicina-Dexametasona -0.0126000000 -0.017409273 -0.007790727 0.0000082
```

h. Realizar un contraste de Kruskal-Wallis para ver si hay diferencias significativas en la variable IHS entre los tres grupos de tratamiento.

```
i Solución

# Test de Kruskal-Wallis
kruskal.test(IHS ~ Tratamiento, data = df)

Kruskal-Wallis rank sum test

data: IHS by Tratamiento
Kruskal-Wallis chi-squared = 13.433, df = 2, p-value = 0.001211
```

i. Construir la recta de regresión del variable IHS sobre la variable IES.

```
summary(lm(IHS ~ IES, data = df))
Call:
lm(formula = IHS ~ IES, data = df)
Residuals:
      Min
                1Q
                        Median
                                    3Q
                                               Max
-0.0099081 -0.0040654 0.0006222 0.0031102 0.0142492
Coefficients:
           Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.08266 0.01470 5.625 2.01e-05 ***
           -15.97510
                       7.47347 -2.138 0.0458 *
IES
---
Signif. codes: 0 '***' 0.001 '**' 0.05 '.' 0.1 ' ' 1
Residual standard error: 0.006558 on 19 degrees of freedom
Multiple R-squared: 0.1939, Adjusted R-squared: 0.1514
F-statistic: 4.569 on 1 and 19 DF, p-value: 0.04577
```