INTRODUCCIÓN A LA ESTADÍSTICA DESCRIPTIVA CON R

Eduardo Aguilar Fernández
José Andrey Zamora Araya

INTRODUCCIÓN A LA ESTADÍSTICA DESCRIPTIVA CON R

Editorial Universidad Nacional
Heredia, Campus Omar Dengo
Costa Rica
Teléfono: 2562-6754 Fax: 2562-6761
Correo electrónico: euna@una.cr
Apartado postal: 86-3000 (Heredia, Costa Rica)

La Editorial Universidad Nacional (EUNA), es miembro del
Sistema Editorial Universitario Centroamericano (SEDUCA).

Producción editorial: Alexandra Meléndez C. amelende@una.cr
Diseño de portada: Mundo Creativo

519.53
A283-i
Aguilar Fernández, Eduardo
Introducción a la estadística descriptiva con R / Eduardo Aguilar Fernández, José Andrey Zamora Araya. - Primera edición. - Heredia, Costa Rica : EUNA, 2020.
1 recurso en línea (107 páginas) : PDF, gráficos
ISBN 978-9977-65-538-3
1. ESTADÍSTICA DESCRPTIVA 2. ANÁLISIS ESTADÍSTICO 3. R (LENGUAJE DE PROGRAMACIÓN) 4. MATEMÁTICAS I. Zamora Araya, José Andrey II. Título

A mi padre, madre, esposa e hija, por ser mis fuentes de inspiración.

Eduardo Aguilar Fernández

A mi esposa, mi hija y mi padre, por estar siempre a mi lado sin importar cuan fuerte sea la tormenta.

José Andrey Zamora Araya

Contenido

Índice de figuras

Prefacio

El presente documento tiene por objetivo ofrecer a todas aquellas personas que dan sus primeros pasos en el lenguaje R, un primer acercamiento al entorno mediante la ejecución de comandos básicos. Si bien es cierto, el público meta está representado en su mayoría por estudiantes de cursos donde se aborden temas de Estadística Descriptiva, también puede ser de utilidad para personas con mayor conocimiento como fuente de consulta, así como apoyo al personal docente que desee impartir sus clases utilizando esta herramienta.

La obra está constituida por cinco capítulos. El primero de ellos trata sobre las principales funciones para el manejo de paquetes y datos, de tal forma que se introduce poco a poco a la persona lectora en el manejo de comandos básicos para realizar operaciones habituales cuando se trabaja con vectores, matrices y bases de datos en general.

El segundo capítulo aprovecha el potencial de R como herramienta para realizar cálculos aritméticos y se exploran las funciones más utilizadas, así como las formas de redondeo y de operar con fracciones. El tercer capítulo enfatiza el manejo de archivos de datos, ya sea con las bases con que cuentan los paquetes de R, o bien, por medio de la importación al entorno de datos almacenados en diferentes formatos como .txt, csv o dta.

El cuarto capítulo explica la forma de calcular las principales medidas de tendencia central y variabilidad, así como la representación tabular de los datos. Finalmente, el quinto capítulo refiere a la representación gráfica de variables tanto numéricas como categóricas, explicando con detalle diferentes parámetros de las funciones utilizadas para su construcción.

El documento está estructurado de tal forma que sea comprensible y simple de seguir para estudiantes principiantes en el uso del software. Para ello, cada sección cuenta con suficientes ejemplos desarrollados con detalle para que el estudiantado tenga una idea clara del uso de las funciones y sus respectivos parámetros. Además de los ejemplos, se dejan a la persona lectora ejercicios que le permitirán reforzar su aprendizaje sobre el uso de comandos.

Asimismo, no pretendemos abarcar en estas pocas páginas la gran cantidad de opciones con que cuenta el entorno R para la representación y análisis de datos, pues su objetivo principal es brindar un apoyo para aquellas personas que de forma autodidacta o dirigida deseen aprender el uso básico de R como herramienta para el manejo de información.

Además, es importante recalcar que el documento no está diseñado para un curso en particular, por el contrario, tiene la intención de servir de apoyo a todas aquellas asignaturas relacionadas con la Estadística Descriptiva sin importar el área de conocimiento a la que pertenezcan, pues se da prioridad a la comprensión y uso apropiado de las funciones del entorno.

Finalmente, deseamos agradecer a todas las personas que hicieron posible la elaboración de este documento, en especial a estudiantes y docentes de la carrera de Bachillerato y Licenciatura en Enseñanza de la Matemática de la Universidad Nacional, pues sus observaciones han sido de gran ayuda para la retroalimentación del material.

Los autores

Julio, 2019

Heredia, Costa Rica

Introducción

R es un entorno y lenguaje de programación de uso libre (Licencia GNU) desarrollado inicialmente por Ross Ihaka y Robert Gentleman del departamento de Estadística de la Universidad de Auckland, que se inspira en la sintaxis del lenguaje S y tiene como fin llevar a cabo procesos de análisis estadístico (Contreras, Molina y Arteaga, 2010).

El programa está conformado por un entorno de programación en el que por medio de la introducción de líneas de código se solicita realizar acciones determinadas. Cuando la acción es ejecutada, se muestra o imprime en la pantalla el resultado de lo que se ha pedido o un mensaje en el que se anuncia la presencia de algún error en el código utilizado, propiciando de esta forma un ambiente interactivo entre el programa y el usuario.

El programa es de uso libre y puede instalarse en sistemas operativos Windows, GNU / Linux, Unix o Macintosh desde la dirección `http://www.r-project.org`.

Por otro lado, R tiene la particularidad de conformar un entorno en el que es posible llevar a cabo procesos de análisis estadístico como la realización de cálculos o conteos y la creación de gráficas, entre otros. El proceso se realiza mediante una serie de pasos en los que se va almacenando la información dentro de elementos llamados objetos que pueden observarse y analizarse en momentos posteriores.

Aunque inicialmente fue desarrollado por Gentleman e Ihaka, en la actualidad R es el fruto de la contribución de muchas personas y equipos de investigación alrededor del mundo, los cuales han proporcionado diversas herramientas orientadas a la realización de tareas cada vez más especializadas.

La conformación de R, desde un punto de vista conceptual (Santana y Mateos, 2014), puede concebirse en dos partes. La primera, el sistema base, el cual conforma la estructura que se descarga desde la página del proyecto y que a su vez contiene los elementos requeridos para su funcionamiento inicial. La segunda que está ligada a los aportes que la comunidad de personas usuarias ha realizado en el tiempo y que permite la ejecución de otra serie de tareas.

Este documento pretende ser una ayuda para todas aquellas personas que dan sus primeros pasos en el análisis estadístico de datos con R. Por ello, se hace referencia sobre todo al manejo de comandos básicos, pero útiles para el manejo de datos, de modo que le permita a la persona usuaria familiarizarse paulatinamente con la sintasis de este entorno de programación. Para su elaboración se empleó la versión 3.4.1 (R Core Team, 2017) y la interface gráfica RStudio, versión 1.1.383 (RStudio Team, 2015).

Capítulo 1

Objetos en R

El objeto es la entidad básica de R (Contreras y cols., 2010). Puede interpretarse como el elemento que almacena los resultados generados por la evaluación de una expresión particular dentro de R. Cada objeto pertenece a una clase y los más simples se clasifican en cinco clases, llamadas atómicas y son las siguientes:

- character (objeto conformado por cadenas de caracteres)
- numeric (considera números reales)
- integer (considera números enteros)
- complex (define números complejos)
- logical (considera valores lógicos que solo almacenan TRUE o FALSE)

Tal y como sucede en otros lenguajes de programación, en R es posible asignar nombres a los objetos. Generalmente, se consideran tres formas diferentes para realizar la asignación: "`x <- value`", "`value -> x`", o bien, "`x = value`", donde `x` representa el nombre que se asignará al objeto, `value` es el valor dado a `x` y `<-` , `=` y `->` son operadores de asignación.

Ejemplo 1.1 Asigne el valor de 5 a un objeto llamado `a`, un valor lógico `FALSE` a un objeto llamado `b` y un valor `O2` a un objeto llamado `oxígeno`.

```
> a <- 5
> FALSE -> b
> oxígeno = "O2"
```

Otra manera de asignar un valor a los objetos es mediante la operación de dos números, ya sean reales o complejos. La forma básica de escritura de un número complejo en R es `a + bi`.

Ejercicio 1.1 Defina un objeto de modo que sea el resultado de multiplicar 5 por 13.

Las instrucciones del Ejemplo 1.1 solo permiten crear los objetos. Para observar el contenido almacenado en un objeto, puede escribir en la línea de comando la expresión con la que se definió el objeto.

Ejemplo 1.2 Escriba las expresiones `a` y `oxígeno` para obtener los objetos definidos en el Ejemplo 1.1.

```
> a

[1] 5

> oxígeno

[1] "O2"
```

Es importante recalcar que el operador "`<-`" puede utilizarse en cualquier momento para definir un objeto, mientras que el operador "`=`" solo se permite en el nivel superior, es decir, el objeto debe estar definido antes de utilizarlo en cualquier instrucción que lo involucre.

Por otro lado, las clases más simples de objetos pueden combinarse para generar otros, entre ellos los vectores. Otras clases de objetos de R son las funciones, las listas, las matrices o las hojas de datos. A continuación se mencionan algunos de ellos.

1.1. Funciones en R

Las funciones son objetos encargados de ejecutar acciones en R. Estos objetos toman una serie de entradas llamadas argumentos, que corresponden a especificaciones suministradas para que una acción se ejecute correctamente. En general, las funciones toman la forma `f(argumento1, argumento2, ...)`, donde `f` corresponde al nombre de la función y los argumentos pueden ser vectores, hojas de datos o funciones. Algunas funciones pueden no tener argumentos.

En primer lugar, la función `getwd()` (sin argumento) permite verificar el directorio de trabajo, es decir, proporciona información sobre la ubicación del archivo de trabajo actual.

Ejemplo 1.3 Verifique su directorio de trabajo.

```
> getwd()

[1] "/Users/eduardoaguilarfernandez/Documents/Libro de R/Libro de R letra11"
```

Si desea cambiar de directorio puede ir a la opción del menú Archivo y en `Change directory` seleccionar la dirección en la que desea llevar a cabo sus trabajos. Luego verifique dónde está direccionado para copiar y leer archivos. Desde la línea de comando el cambio puede realizarse con la función `setwd()`.

Ejemplo 1.4 Cambie su directorio de trabajo a "`C:/Users/Documents/Manual`".

```
> setwd("C:/Users/Documents/Manual")
```

La función `help()` permite obtener información acerca de funciones determinadas. Su argumento puede escribirse entre comillas dobles y está constituido por el nombre de la función sobre la que desea hacerse la consulta.

Ejemplo 1.5 Obtenga información sobre la función `getwd`.

```
> help("getwd")
```

Ejemplo 1.6 Obtenga información sobre la función `setwd`.

```
> help("setwd")
```

Otra alternativa para solicitar información sobre una función es utilizar el operador " `?` " seguido por el nombre de la función. Por otro lado, el operador " `#` " permite realizar un **comentario** sobre una instrucción particular sin que esto afecte la ejecución de la orden.

Ejemplo 1.7 Utilice el operador " `?` " para obtener información sobre la función `getwd` y agregue el comentario: `El operador ? brinda ayuda sobre la función getwd.`

```
> ?getwd   # El operador ? brinda ayuda sobre la función getwd
```

1.2. Paquetes en R

Un paquete (package, por su nombre en inglés) es una colección de funciones, datos y código R que se almacenan en una carpeta conforme a una estructura bien definida, fácilmente accesible para R. Como se indicó antes, R posee un conjunto de paquetes dentro de su sistema base, así como un conjunto adicional que complementa su funcionalidad.

Algunos paquetes de R ubicados en el sistema base son `stats`, `datasets`, `graphics`, entre otros. Entre de los paquetes adicionales están `faraway`, `modeest` o `gdata`.

1.2.1. La función `install.packages()`

Esta función permite instalar un paquete adicional dentro de la memoria. Utiliza como argumento el nombre del paquete, el cual debe anotarse entre comillas dobles.

Ejemplo 1.8 Instale el paquete `faraway`.

```
> install.packages("faraway")
```

1.2.2. La función `library()`

Esta función realiza varias tareas. En primer lugar, permite cargar los paquetes, esto es, activarlos para que puedan utilizarse sus objetos. Para llevar a cabo esta acción basta indicar como argumento el nombre del paquete que desea utilizarse.

Ejemplo 1.9 Cargue el paquete `faraway`.

```
> library(faraway)
```

En segundo lugar, al escribir la expresión `library()` (sin argumento) es posible obtener una lista de los paquetes disponibles en la memoria de R.

Ejemplo 1.10 Visualice la lista de paquetes almacenados actualmente en la memoria de R.

```
> library()

Packages in library 'C:/Users/Ana/Documents/R/win-library/3.4':
 backports   Reimplementations of Functions Introduced Since R-3.0.0
 base64enc   Tools for base64 encoding
 BH          Boost C++ Header Files
 bitops      Bitwise Operations
 caTools     Tools: moving window statistics, GIF, Base64, ROC AUC, etc.
 cellranger  Translate Spreadsheet Cell Ranges to Rows and Columns
 colorspace  Color Space Manipulation
 combinat    combinatorics utilities
 covr        Test Coverage for Packages
 crayon      Colored Terminal Output
 curl        A Modern and Flexible Web Client for R
 dichromat   Color Schemes for Dichromats
 digest      Create Compact Hash Digests of R Objects
 ...
```

Si en la función se utiliza como argumento la expresión `help = nombre del paquete`, es posible conocer información del contenido del paquete que se indica.

Ejemplo 1.11 Obtenga información relacionada con el paquete `faraway`.

```
> library(help = faraway)

Information on package 'faraway'
 Description:

 Package:      faraway
 Version:      1.0.7
 Date:         2016-02-12
 Author:       Julian Faraway <jjf23@bath.ac.uk>
 Maintainer:   Julian Faraway <jjf23@bath.ac.uk>
 Title:        Functions and Datasets for Books by Julian Faraway
 Description:  Books are "Practical Regression and ANOVA in R.ºn CRAN,
               "Linear Models with R"
               published 1st Ed. August 2004, 2nd Ed. July 2014 by CRC
               press, ISBN 9781439887332, and .Extending the Linear Model
               with R"published by CRC press in 1st Ed. December 2005
               and 2nd Ed. March 2016, ISBN 9781584884248.
 Depends:      R (>= 2.10)
 License:      GPL
 URL:          http://people.bath.ac.uk/jjf23/
 LazyData:     yes
 ...
 ...
```

Por otra parte, la lista de objetos almacenados en un paquete también puede obtenerse con ayuda de la instrucción `help(package = nombre del paquete)`.

Ejemplo 1.12 Ejecute la instrucción `help(package = faraway)` y obtenga la lista de objetos almacenados en el paquete `faraway`.

```
> help(package = faraway)
```

El uso de esta función permite obtener más detalles sobre el contenido de cada objeto dentro de un paquete, pues en la ventana de ayuda que despliega es posible dar click en el objeto y en una nueva ventana aparece su descripción.

1.2.3. La función `class()`

Esta función permite identificar la clase de un objeto determinado. Su argumento es el nombre del objeto.

Además, el operador " ; " permite escribir varias instrucciones en la misma línea de comando.

Ejemplo 1.13 Determine la clase de los objetos definidos en el Ejemplo 1.1 digitando las instrucciones en la misma línea de comando.

```
> class(a); class(b); class(oxígeno)

[1] "numeric"

[1] "logical"

[1] "character"
```

Ejercicio 1.2 Determine la clase de los objetos `Cpplot` y `gala` ubicados dentro del paquete `faraway`.

A pesar de que el objeto `a` está definido por un número entero, R lo clasifica como numérico. Para definir un objeto como entero puede utilizarse la letra `L` seguido del valor asignado al objeto o utilizar la función `as.integer()` empleando como argumento el nombre asignado al objeto.

Ejemplo 1.14 Defina un objeto `d` a partir del objeto `a` del Ejemplo 1.1, de modo que su clase quede definida como `integer`.

```
> d <- 5L # también puede escribirse d <- as.integer(a)
> class(d)

[1] "integer"
```

1.2.4. Las funciones `ls()` y `rm()`

La función `ls()` permite visualizar la lista de objetos que el usuario ha almacenado en la memoria.

Ejemplo 1.15 Determine los objetos que han sido guardados en la memoria.

```
> ls()
```

Es posible utilizar la función `ls()` para obtener la lista de objetos cuyo nombre contiene un determinado carácter. Por ejemplo, al utilizarse la opción `ls(pattern =` "`a`") es posible obtener todos aquellos objetos almacenados en la memoria cuyo nombre contiene la letra " a ". Es común abreviar la expresión `pattern` por `pat`.

Ejemplo 1.16 Determine los objetos que han sido guardados en la memoria cuyo nombre contiene el carácter `p`.

```
> ls(pat = "p")
> ls(pat = "^p")
```

La primera opción le muestra al usuario todos los objetos cuyo nombre incluye el carácter `p`, mientras que la segunda le indica solo aquellos que inician con la letra `p`.

Por otro lado, la función `rm()` permite eliminar los objetos de la memoria y en ella pueden utilizarse las mismas opciones mencionadas para `ls()`.

1.3. Vectores

Los vectores son objetos de R que están conformados por números o caracteres (cadenas de caracteres). Un número o un carácter en R está considerado como un vector de una componente. Para crear un vector de cierta longitud se utiliza la función `c()`, la cual permite la concatenación de elementos del mismo tipo. Los vectores pueden contener valores numéricos, caracteres o valores lógicos (T o F); sin embargo, es importante que todos los elementos del vector sean del mismo tipo.

1.3.1. Vectores de caracteres

Los vectores pueden estar compuestos por caracteres o cadenas de ellos y para definirlos cada uno de sus elementos debe escribirse entre comillas dobles.

Ejemplo 1.17 Suponga que se tiene la zona de residencia de 5 personas según el siguiente orden: `rural, urbana, urbana, rural, urbana`. Defina un vector que almacene dicha información.

```
> zona <- c("rural", "urbana", "urbana", "rural", "urbana")
```

1.3.2. Vectores numéricos

Los vectores numéricos son aquellos cuyas componentes u objetos son números. Estos pueden ser reales o complejos.

Ejemplo 1.18 Defina el vector `x = (21,21,22,35,45)`.

```
> x<-c(21,21,22,35,45)
```

La función `print()` permite imprimir el objeto que se ha creado y observar así como está conformado. El argumento de esta función es la expresión que ha definido el objeto.

Ejemplo 1.19 Utilice la función `print()` para imprimir el vector `zona` del Ejemplo 1.17.

```
> print(zona)

[1] "rural"  "urbana" "urbana" "rural"  "urbana"
```

Ejercicio 1.3 Utilice la función `print()` e imprima el vector `x` del Ejemplo 1.18.

La impresión de un vector también puede llevarse a cabo escribiendo entre paréntesis toda la instrucción al momento de definir el vector.

Ejemplo 1.20 Defina el vector `x` del Ejemplo 1.18 encerrando entre paréntesis la instrucción.

```
> (x<-c(21,21,22,35,45))

[1] 21 21 22 35 45
```

Otra forma de imprimir un vector es digitar en la línea de comando la expresión con la que se ha definido, es decir, el nombre que se le ha asignado.

Ejemplo 1.21 Imprima el vector `x` del Ejemplo 1.18.

```
> x

[1] 21 21 22 35 45
```

Puede obtenerse uno o varios elementos del vector `x` por medio de la instrucción `x[i]`, donde " `i` " representa un vector que contiene las posiciones de los elementos que desean conocerse.

Ejemplo 1.22 Obtenga los elementos de las posiciones 2 y 5 en el vector `x`.

```
> zona[c(2,5)]

[1] "urbana" "urbana"
```

Ejercicio 1.4 Construya un vector con 20 elementos y obtenga luego los objetos ubicados en las posiciones 5 y 12.

Por otro lado, al escribir `x[-i]` se obtienen los elementos del vector `x` exceptuando los de las posiciones indicadas en el vector `i`.

Ejemplo 1.23 Obtenga los elementos del vector `zona` excepto los que se encuentran en las posiciones 2 y 5.

```
> zona[c(-2,-5)]

[1] "rural"  "urbana" "rural"
```

1.3.3. Vectores lógicos

Los vectores también pueden contener valores lógicos (`logical`) asociados a los valores de verdadero y falso. Estos valores se representan digitando la expresión `FALSE` o simplemente la letra mayúscula `F` en el caso del valor lógico falso. Para el valor lógico verdadero se escribe `TRUE` o `T`.

Ejemplo 1.24 Defina un vector que contenga la siguiente secuencia de objetos: `T, F, F, T`.

```
> v<-c(T,F,F,T)
```

1.3.4. Secuencias

Para crear secuencias en R existen varias formas.

En primer lugar, puede construirse un vector cuyas componentes sean una secuencia ascendente o descendente de números indicando los extremos del vector separados por el operador " : ".

Ejemplo 1.25 Defina un vector `k` de números enteros consecutivos que inicie en 1 y finalice en 5 usando el operador " : ".

```
> (k <- 1:5)

[1] 1 2 3 4 5
```

La función `seq()` permite crear un vector cuyas componentes presentan cierta separación. Sus argumentos son (`from`) para indicar donde inicia, (`to`) para señalar donde termina y `by` para definir la separación deseada entre cada elemento.

Ejemplo 1.26 Genere un vector que inicie en 1, finalice en 15 y la separación que origine la secuencia sea 2.

```
> seq(from = 1, to = 15, by = 2)

[1]  1  3  5  7  9 11 13 15
```

La instrucción `seq(by = 2, from = 1, to = 15)` obtiene el mismo resultado del Ejemplo 1.27. Además, la instrucción puede abreviarse escribiendo `seq(1, 15, 2)`. Si no se especifica el argumento `by`, entonces la separación por defecto es 1.

Ejercicio 1.5 Defina un vector que inicie en 5, finalice en 21 y la separación de sus elementos sea 2, utilizando la función `seq()` en su forma `(by, from, to)`.

Ejercicio 1.6 Construya un vector que inicie en 5, finalice en 21 y la separación de sus elementos sea 1.

Por otro lado, si se desea que la secuencia tenga una cantidad definida de elementos, la instrucción se modifica utilizando el argumento `length.out` en lugar de `by`.

Ejemplo 1.27 Genere un vector que inicie en 1, finalice en 100 y que contenga 11 elementos.

```
> seq(1, 100, length.out = 11)

 [1]   1.0  10.9  20.8  30.7  40.6  50.5  60.4  70.3  80.2  90.1 100.0
```

1.3.5. Operaciones con vectores

Los vectores numéricos creados en R pueden operarse entre sí. Al considerar el álgebra vectorial, las operaciones suma y diferencia pueden ejecutarse mediante los operadores " + " y " − ", respectivamente.

Ejemplo 1.28 Obtenga la suma y la diferencia de los vectores `x` y `k` definidos en los Ejemplos 1.18 y 1.25, respectivamente.

```
> x + k

[1] 22 23 25 39 50

> x - k

[1] 20 19 19 31 40
```

Para multiplicar un número y un vector se utiliza el operador "∗".

Ejemplo 1.29 Obtenga el producto de la constante −2 y el vector `x`.

```
> -2*x
```

Para el producto de dos vectores debe tenerse especial cuidado, pues si se escribe " `a * b` ", R genera un nuevo vector cuyas componentes son el resultado de multiplicar las componentes respectivas de los vectores `a` y `b`.

Ejemplo 1.30 Obtenga el producto de los vectores `x` y `k` definidos en los Ejemplos 1.18 y 1.25, respectivamente.

```
> x*k

[1]  21  42  66 140 225
```

Por el contrario, si lo que desea efectuarse es el producto punto de vectores, entonces se usa el operador " `%*%` ".

Ejemplo 1.31 Obtenga el producto punto de los vectores `x` y `k` definidos en los Ejemplos 1.18 y 1.25, respectivamente.

```
> x %*% k

     [,1]
[1,]  494
```

1.3.6. Otras funciones

La función `names()`, en su forma `names() <- c()`, permite asignar un nombre a cada uno de los objetos de un vector. Su argumento es el vector y la expresión `c()` es un vector en el que se especifican los nombres que se darán a los objetos.

Ejemplo 1.32 Asigne, respectivamente, los nombres A, B, C, D, E a los elementos del vector x del Ejemplo 1.18.

```
> names(x) <- c("A", "B", "C", "D", "E")
```

Al digitar `names(x)` es posible observar los nombres que le fueron asignados a cada uno de los elementos del vector `x`.

Ejemplo 1.33 Obtenga los nombres asignados a los elementos del vector `x`.

```
> names(x)

[1] "A" "B" "C" "D" "E"
```

La instrucción `x[i]` también permite acceder individualmente a cada uno de los elementos del vector `x` utilizando su nombre.

Ejemplo 1.34 Acceda al tercer elemento del vector `x` utilizando su nombre.

```
> x[c("C")]

 C
22
```

Para eliminar los nombres asignados a los elementos de un vector se utiliza la instrucción `names() <- NULL`, empleando como argumento el nombre del vector.

Ejemplo 1.35 Elimine los nombres asignados al vector `x`.

```
> names(x) <- NULL
```

La función `paste()` permite convertir varios objetos de R en un vector (cadenas) de caracteres. Sus argumentos son:

`...`: uno o más objetos de R que serán convertidos.

`sep`: permite asignar un carácter para separar los términos de los argumentos a convertir. Si la separación es el espacio se escribe `sep =` “ ”, en el caso de la coma se usa `sep =` “ , ” y para el punto y coma se utiliza `sep =` “ ; ”.

Ejemplo 1.36 Considere los vectores `x` y `zona` de los Ejemplos 1.18 y 1.25, respectivamente y cree un nuevo vector de modo que los elementos estén concatenados por la coma, el espacio o punto y coma.

```
> (nuevo = paste(x, zona, sep = ","))

[1] "21,rural"  "21,urbana" "22,urbana" "35,rural"  "45,urbana"

> (nuevo = paste(x, zona, sep = " "))

[1] "21 rural"  "21 urbana" "22 urbana" "35 rural"  "45 urbana"

> (nuevo = paste(x, zona, sep = ";"))

[1] "21;rural"  "21;urbana" "22;urbana" "35;rural"  "45;urbana"
```

La función `mode()` permite conocer el tipo de elementos que conforman un vector. Su argumento es la expresión con la que se ha definido el vector y puede tomar el valor logical, numeric, complex, character, entre otros.

Ejemplo 1.37 Determine el tipo de elemento que forman los vectores `x` y `v` definidos en los Ejemplos 1.18 y 1.24, respectivamente.

```
> mode(v); mode(x)

[1] "logical"

[1] "numeric"
```

También puede utilizarse para cambiar el tipo de elemento del vector, para ello la función adquiere la forma `mode() <- value`, donde `value` indica el tipo de objeto que se desea.

Ejemplo 1.38 Cambie a tipo complex los elementos que forma el vector `v` del Ejemplo 1.24.

```
> mode(v) <- "complex"
> v

[1] 1+0i 0+0i 0+0i 1+0i
```

La función `rep()` permite crear vectores que incluyen repeticiones de sus elementos o de otros vectores. Uno de sus argumentos es `times`, que indica el número de veces que debe repetirse el objeto.

Ejemplo 1.39 Obtenga un vector `h` que repita cuatro veces el número 1 y otro vector `z` que repita tres veces los elementos de `h`.

```
> h <- rep(1, 4) # crea un vector de 4 unos
> z <- rep(h, 3) # crea un vector que repite 3 veces los elementos de h
```

Ejercicio 1.7 Defina un nuevo vector de 10 elementos de modo que sus primeros cinco elementos estén representados por un 2 y los restantes por un 12.

La función `length()` permite obtener la cantidad de elementos o la longitud del vector.

Ejemplo 1.40 Determine la longitud del vector `x` del Ejemplo 1.18.

```
> length(x)

[1] 5
```

Esta función también permite definir la longitud del vector, esto es, ampliarla o disminuirla por medio de la instrucción `length() <- value`, donde la expresión `value` hace referencia a la nueva

cantidad de elementos que se desean en el vector. Cuando se aumenta el tamaño del vector, las nuevas componentes son consideradas como valores faltantes (`NA`).

Ejemplo 1.41 Modifique la cantidad de elementos de los vectores `h` y `z` definidos en el Ejemplo 1.39 de modo que su longitud sea 6.

```
> length(h) <- 6 # agrega dos elementos al vector h
> length(z) <- 6 # elimina elementos al vector z
```

Otra función en R que permite crear vectores es `vector()`, la cual requiere que se indique el tipo de dato u objeto (numérico, carácter o lógico) que conformará el vector, así como la longitud de este.

Ejemplo 1.42 Obtenga un nuevo vector numérico que tenga 5 elementos.

```
> vector("numeric", 5)

[1] 0 0 0 0 0
```

Por otra parte, la función `grep()` permite buscar coincidencias con el patrón del argumento dentro de cada elemento de un vector de caracteres. Los argumentos principales son `pattern` que representa una cadena de caracteres que contiene una expresión regular, `x` un vector de caracteres donde se buscan coincidencias y `value`, si es `FALSE` muestra un vector que contiene los índices de las coincidencias encontradas, si es `TRUE` devuelve un vector que contiene los elementos coincidentes.

Ejemplo 1.43 Obtenga todos los elementos que incluyen la subcadena "`na`" en el vector `zona` del Ejemplo 1.17.

```
> grep("na", zona, value=T)

[1] "urbana" "urbana" "urbana"
```

Ejercicio 1.8 Obtenga todos los elementos que incluyen la subcadena "`na`" en el vector `zona` del Ejemplo 1.17 utilizando la expresión `value = F`.

La función `which()` permite identificar la posición de los objetos de un vector que satisfacen una determinada condición lógica. Estas condiciones pueden ser planteadas utilizando los operadores `>, >=, <, <=, ==, !=`.

Ejemplo 1.44 Investigue si existen elementos en el vector `x` del Ejemplo 1.18 que son menores que 25.

```
> which(x < 25)

[1] 1 2 3
```

Si se desea conocer cuáles son los elementos que cumplen con la condición lógica especificada, entonces se escribe el nombre del vector y se indica la condición con ayuda del operador `[]`.

Ejemplo 1.45 Obtenga los elementos del vector `x` del Ejemplo 1.18 que son menores que 25.

```
> x[x < 25]

[1] 21 21 22
```

La función `sum()` proporciona la suma de los elementos de un vector.

Ejemplo 1.46 Asigne el nombre de `sec` al vector del Ejemplo 1.27 y determine la suma de sus elementos.

```
> sec=seq(1, 100, length.out = 11)
> sum(sec)

[1] 555.5
```

También es posible encontrar la suma de los elementos que satisfacen una condición lógica dada.

Ejemplo 1.47 Obtenga la suma de los elementos del vector `sec` del Ejemplo 1.46 que son mayores que 50.

```
> sum(sec[sec>50])

[1] 451.5
```

1.4. Listas

Como se mencionó anteriormente, los vectores en R están conformados por objetos de la misma clase. Para crear vectores con objetos de distintas clases pueden usarse las listas, las cuales se definen con el uso de la función `list()`.

Ejemplo 1.48 Genere una lista que contenga los siguientes objetos: `c(T,F,F,T)`, `0`, `1`, `m` y `0`.

```
> milista <- list(c(T,F,F,T), 0, 1, "m", 0)
> class(milista)

[1] "list"
```

La lista creada está conformada por 5 objetos, cuatro de los cuales son individuales y el otro es un vector que contiene 4 objetos.

Es posible tener acceso a los elementos individuales de la lista con ayuda de la instrucción `x[i]`, siendo `x` el nombre asignado a la lista.

Ejemplo 1.49 Obtenga el objeto de `milista` ubicado en la posición 5.

```
> milista[5]

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

A cada objeto de una lista también puede asignársele un nombre por medio de la función `names()`, o bien, indicando el nombre en el momento de definir la lista.

Ejemplo 1.50 Asigne, respectivamente, los nombres de `primero, segundo, tercero, cuarto` y `quinto` a los elementos de `milista`.

```
> names(milista)<-c("primero", "segundo", "tercero", "cuarto", "quinto")
```

También es posible acceder a los elementos de una lista utilizando su nombre con ayuda de los operadores `[[ ]]` y `$`. En el caso del operador `[[ ]]`, el nombre del objeto debe ir entre comillas dobles.

Ejemplo 1.51 Acceda al primer elemento de `milista` usando el operador `$` y al quinto elemento con ayuda del operador `[[ ]]`.

```
> milista$primero; milista[["quinto"]]

[1]  TRUE FALSE FALSE  TRUE

[1] 0
```

1.5. Matrices

Para crear una matriz se utiliza la función `matrix()`. Esta función tiene como sintaxis `matrix(data, nrow, ncol, byrow)`, donde

`data`: es un vector en el cual se definen las entradas de la matriz

`nrow`: indica el número de filas que componen la matriz

`ncol`: se refiere a la cantidad de columnas

`byrow`: valor lógico (T o F), donde `byrow = T` indica que la matriz será llenada por filas, en caso contrario, las entradas se completan por columna.

Ejemplo 1.52 Defina las matrices $A = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & -1 \end{pmatrix}$ y $B = \begin{pmatrix} -1 & 0 & 3 \\ 1 & 3 & -2 \\ 0 & 1 & -5 \end{pmatrix}$.

```
> A<-matrix(c(1,0,1, 1,1,1, 0,1,-1), nrow = 3, ncol = 3, byrow = T)
> B<-matrix(c(-1,0,3, 1,3,-2, 0,1,-5), nrow = 3, ncol = 3, byrow = T)
> A

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

En caso de omitir la instrucción `byrow = T`, la matriz se llena por columnas.

1.5.1. Operaciones con matrices

La suma, resta y multiplicación por un escalar están definidas por medio de los operadores aritméticos habituales " `+, -, *` ".

Ejemplo 1.53 Determine la suma de las matrices A y B del Ejemplo 1.52.

```
> A + B

     [,1] [,2] [,3]
[1,]    0    0    4
[2,]    2    4   -1
[3,]    0    2   -6
```

Ejercicio 1.9 Considerando las matrices del Ejemplo 1.52, determine la diferencia de A y B, así como, la matriz $3A$.

Para el producto matricial se utiliza el operador "`% *%`".

Ejemplo 1.54 Considerando las matrices del Ejemplo 1.52, determine el producto matricial de A por B.

```
> A %*% B

     [,1] [,2] [,3]
[1,]   -1    1   -2
[2,]    0    4   -4
[3,]    1    2    3
```

Ejercicio 1.10 Determine el producto matricial de B por A.

Nota: Si se utiliza el operador "`*`" en la multiplicación de A por B, el resultado que se obtiene es una matriz cuyas entradas son el resultado de multiplicar las entradas de A con las respectivas de B.

En el caso de las matrices cuadradas, su determinante puede obtenerse con la función `det()`, la diagonal principal con la función `diag()` y la traza puede calcularse por medio de la expresión `sum(diag())`. El argumento de estas funciones es una matriz.

Ejemplo 1.55 Obtenga el determinante, la diagonal principal y la traza de la matriz A del Ejemplo 1.52.

```
> det(A); diag(A); sum(diag(A))

[1] -1

[1]  1  1 -1

[1] 1
```

Ejercicio 1.11 Obtenga el determinante, la diagonal principal y la traza de la matriz B del Ejemplo 1.52.

Si la matriz es invertible, entonces su inversa se obtiene por medio de la función `solve()`. La transpuesta de una matriz se obtiene con la función `t()`.

Ejemplo 1.56 Obtenga la transpuesta y la matriz inversa de A del Ejemplo 1.52.

```
> (At=t(A))

     [,1] [,2] [,3]
[1,]    1    1    0
[2,]    0    1    1
[3,]    1    1   -1

> (invA=solve(A))

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

1.5.2. Otras funciones para crear matrices

También pueden crearse matrices con las funciones `cbind()`, `rbind()` o `as.matrix()`.

La función `cbind()` permite crear matrices pegando columnas y `rbind()` pegando las filas y presentan la forma `(argumento1, argumento2, ...)`. Estos argumentos pueden ser vectores o matrices con igual número de filas para el uso de `cbind()` o igual números de columnas para el caso de `rbind()`.

Ejemplo 1.57 Considerando los vectores `x` y `k` de los Ejemplos 1.18 y 1.25, respectivamente, defina una matriz con ayuda de las funciones `cbind` y `rbind()`.

```
> (C=cbind(x, k))

      x k
[1,] 21 1
[2,] 21 2
[3,] 22 3
[4,] 35 4
[5,] 45 5

> (D=rbind(x,k))

  [,1] [,2] [,3] [,4] [,5]
x   21   21   22   35   45
k    1    2    3    4    5
```

El procedimiento es similar si se utilizan matrices.

Ejemplo 1.58 Considerando las matrices A y B del Ejemplo 1.52, defina una matriz H con ayuda de la función `cbind()` y una matriz Y con ayuda de `rbind()`.

```
> (H=cbind(A, B))

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    0    1   -1    0    3
[2,]    1    1    1    1    3   -2
[3,]    0    1   -1    0    1   -5

> Y = rbind(A, B)
```

La función `rownames()` (`colnames()`) permite asignar nombre a cada una de las filas (columnas) de una matriz. En su formato `rownames(x) <- value` (`colnames(x) <- value`), `x` representa un objeto tipo matriz y `value` un vector en el que se detallan los nombres que se asignarán a las filas (columnas).

Ejemplo 1.59 Considere la matriz `H` del Ejemplo 1.58 y rotule sus filas como `f1, f2 y f3` y las columnas como `c1, c2, c3, c4, c5` y `c6`, respectivamente.

```
> rownames(H) <- c("f1", "f2", "f3")
> colnames(H) <- c("c1", "c2", "c3", "c4", "c5", "c6")
> H

   c1 c2 c3 c4 c5 c6
f1  1  0  1 -1  0  3
f2  1  1  1  1  3 -2
f3  0  1 -1  0  1 -5
```

Por otro lado, la función `rowSums(x)` (`colSums(x)`) permite obtener la suma de los elementos de las filas (columnas) de una matriz. El argumento `x` representa un arreglo de dos o más dimensiones que contiene valores numéricos, complejos, enteros o lógicos.

Ejemplo 1.60 Obtenga la suma de las filas de la matriz `H`.

```
> rowSums(H)

f1 f2 f3
 4  5 -4
```

Ejercicio 1.12 Obtenga la suma de las filas de la matriz `H` del Ejemplo 1.58.

Finalmente, la función `mode()` identifica el tipo de elementos que componen una matriz.

Ejemplo 1.61 Indique el tipo de dato de los elementos de la matriz `H`.

```
> mode(H)

[1] "numeric"
```

1.5.3. Sistemas de ecuaciones

El entorno R permite resolver un sistema cuadrado de ecuaciones lineales de la forma $MX = C$ que presenta la solución única a través de la función `solve()`. Sus argumentos principales son:

`a:` representa una matriz cuadrada, la cual contiene los coeficientes del sistema lineal.

`b:` representa un vector o matriz que contiene los términos independientes del sistema.

Ejemplo 1.62 Resuelva el sistema

$$\begin{cases} x + y + z = 4 \\ 2x + y + z = -2 \\ 3x - 2y + z = 1 \end{cases}$$

```
> M<-matrix(c(1,1,1, 2,1,1, 3,-2,1), nrow = 3, ncol = 3, byrow = T)
> C<-matrix(c(4,-2,1), nrow = 3, ncol = 1)
> (X<-solve(M,C))

     [,1]
[1,]   -6
[2,]   -3
[3,]   13
```

Ejercicio 1.13 Obtenga la solución del sistema

$$\begin{cases} 2x - 3y + z = 2 \\ 2x + y + z = -1 \\ x - 3y - z = 0 \end{cases}$$

Capítulo 2

Cálculos aritméticos en R

R puede usarse como una herramienta para realizar cálculos numéricos, por ejemplo, la suma, el producto, las potencias, entre otros, ya sea utilizando números reales o complejos. Cuando se utilizan números complejos la unidad imaginaria `i` debe ir acompañada de un coeficiente. En particular, para expresar el número complejo `1 + i` en R debe escribirse `1 + 1i`.

2.1. Suma y producto

Para llevar a cabo la suma de números reales o complejos puede utilizarse la función `sum()`, indicando como argumentos los números que deben sumarse.

Ejemplo 2.1 Realice la suma de los números 3, 7 y 4.

```
> sum(3,7,4)

[1] 14
```

Ejercicio 2.1 Utilice la función `sum()` para sumar los números $4 + 5i, 12 + 3i$ y $2 + i$.

Esta operación también puede realizarse simplemente digitando los elementos a sumar utilizando el operador " `+` ".

Ejemplo 2.2 Realice la suma de los números $4 + 5i, 12 + 3i$ y $2 + i$.

```
> 4 + 5i + 12 + 3i + 2 + 1i

[1] 18+9i
```

Para el producto se utiliza la función `prod()`, la cual tiene por argumentos los números que deben multiplicarse.

Ejemplo 2.3 Realice el producto de los números 3 y 7 utilizando la función `prod()`.

```
> prod(3,7)

[1] 21
```

Ejercicio 2.2 Realice el producto de los números $4+5i$ con $12+3i$ utilizando la función `prod()`.

El producto también puede llevarse a cabo escribiendo las expresiones que van a multiplicarse utilizando el operador " `*` ".

Ejemplo 2.4 Realice el producto de los números $4 + 5i$ y $12 + 3i$.

```
> (4 + 5i) * (12 + 3i)

[1] 33+72i
```

Ejercicio 2.3 Realice el producto de los números 4, 5, 12, y 3 utilizando el operador "`*`".

Para llevar a cabo la división se utiliza el operador " `/` ".

Ejemplo 2.5 Realice la división de 12 por 7.

```
> 12 / 7

[1] 1.714286
```

Si `a` y `b` son vectores de la misma longitud, la instrucción `a / b` realiza la división entre las componentes respectivas de los vectores `a` y `b`. Si `b` es un número, entonces cada componente del vector `a` se divide por `b`.

Ejemplo 2.6 Realice la división de 4 por 2, de 10 por 5 y de 15 por 3 considerando en un vector los dividendos y en otro los divisores.

```
> c(4,10,15)/c(2,5,3)

[1] 2 2 5
```

Por otro lado, los operadores "`%%`" y "`%/%`" permiten calcular el residuo y el cociente de la división, respectivamente.

Ejemplo 2.7 Obtenga el residuo y el cociente que resultan de la división de 12 por 7.

```
> 12 %% 7        # obtiene el residuo de la división

[1] 5

> 12 %/% 7       # obtiene el cociente de la división

[1] 1
```

El factorial de un número natural puede obtenerse con ayuda de la función `factorial()`.

Ejemplo 2.8 Obtenga el factorial de 0 y 6.

```
> factorial(0)

[1] 1

> factorial(6)

[1] 720
```

2.2. Cálculo de logaritmos

La función `log()` permite calcular el logaritmo de un número o un conjunto de números cualesquiera. Los argumentos de la función son:

`x`: representa un vector de números reales o complejos. Recuerde que R considera un número como un vector de una componente.

`base`: indica la base del logaritmo.

Ejemplo 2.9 Determine el logaritmo de 2 en base 4.

```
> log(2, 4)

[1] 0.5
```

Ejercicio 2.4 Calcule el logaritmo de 3, 7 y 10 en base 2.

El argumento `base` también puede ser un vector, esto permite calcular en una sola instrucción el logaritmo de varios números con distintas bases.

Ejemplo 2.10 Calcule el logaritmo de 3 en base 2, el logaritmo de 7 en base 4 y el logaritmo de 10 en base 6.

```
> log(c(3,7,10), c(2,4,6))

[1] 1.584963 1.403677 1.285097
```

Las funciones `log()` y `logb()` permiten obtener el logaritmo natural (base e) del argumento.

Ejemplo 2.11 Determine el logaritmo natural de 3, 7 y 10.

```
> log(c(3,7,10))

[1] 1.098612 1.945910 2.302585
```

Ejercicio 2.5 Calcule el logaritmo natural de 3, 7 y 10 utilizando la función `logb()`.

Finalmente, la función `log1p(x)` permite calcular el logaritmo natural de la expresión 1 + `x`.

2.3. Raíces y valor absoluto

La función `sqrt()` permite calcular la raíz cuadrada principal del argumento, el cual puede ser un vector de números reales o complejos.

Ejemplo 2.12 Calcule la raíz cuadrada de 4, 9 y 16.

```
> sqrt(c(4,9,16))

[1] 2 3 4
```

Ejercicio 2.6 Calcule la raíz cuadrada de los números $4 + 5i$ y $12 + 3i$.

En el caso de raíces de índice mayor a 2 puede utilizarse el símbolo "∧".

Ejemplo 2.13 Determine la raíz cúbica de 8, 27 y 30.

```
> c(8,27,30)^(1/3)

[1] 2.000000 3.000000 3.107233
```

La función `abs()` permite hallar el valor absoluto de un argumento real.

Ejemplo 2.14 Determine el valor absoluto de 4, -9 y -16.

```
> abs(c(4,-9,-16))

[1]  4  9 16
```

Si el argumento de la función es un número complejo, entonces `abs()` calcula el módulo de dicho número.

Ejemplo 2.15 Determine el módulo de los números complejos $3 + 2i$ y $4 + 4i$.

```
> abs(c(3 + 2i, 4 + 4i))

[1] 3.605551 5.656854
```

2.4. Redondeo en R

Para realizar redondeos pueden utilizarse las funciones `round()` o `signif()`. Sus argumentos están dados por:

`x`: es un vector numérico.

`digits`: indica el número de cifras decimales (función `round`) o dígitos significativos (función `signif`) que desean usarse.

Específicamente, la función `round()` realiza el redondeo usual considerando el número de cifras decimales especificado.

Ejemplo 2.16 Utilice la función `round()` para realizar el redondeo usual del número 3,456 y el redondeo usual del número 3,4327892345678901 considerando 6 cifras decimales.

```
> round(3.456)    # redondeo usual a un entero

[1] 3

> round(3.4327892345678901, digits = 6) # redondeo a 6 cifras decimales

[1] 3.432789
```

La función `signif()` realiza el redondeo usual de los valores del primer argumento especificando un número de dígitos significativos.

Ejemplo 2.17 Utilice la función `signif()` y redondee el número 3,4327892345678901 considerando 6 cifras significativas.

```
> signif(3.4327892345678901, digits = 6)

[1] 3.43279
```

Por otro lado, la función `ceiling()` permite redondear la expresión del argumento al entero superior y la función `floor()` redondea al entero inferior. Finalmente, `trunc()` realiza el redondeo por truncamiento.

Ejemplo 2.18 Utilice las funciones anteriores para realizar el redondeo de los números $-4,256$ y $3,7327892$ al entero superior e inferior.

```
> ceiling(c(3.7327892, -4.256)) # redondeo entero hacia entero superior
> floor(c(3.7327892, -4.256))   # redondeo entero hacia entero inferior
```

También pueden considerarse funciones para obtener exponenciales (`exp()`) o para realizar cálculos trigonométricos (`sin()`, `tan()`, entre otras) (Verzani, 2014).

2.5. Fracciones en R

En R existe la posibilidad de encontrar aproximaciones racionales a las componentes de un objeto numérico real. Para este procedimiento se utiliza la función `fractions()` que se encuentra dentro del paquete `MASS` (Venables y Ripley, 2002), la cual utiliza un método estándar de funciones continuas. Entre sus argumentos están:

`x:` un objeto en modo numérico.

`cycles:` indica el número máximo de pasos que se utilizarán en el proceso de aproximación continua de fracciones.

`max.denominator:` un criterio de finalización anticipada.

De esta forma, la función `fractions()` permite expresar en notación fraccionaria un número racional.

Ejemplo 2.19 Exprese en notación fraccionaria el número 0,5.

```
> library(MASS)
> fractions(0.5)

[1] 1/2
```

Si el argumento es un número irracional, `fractions()` permite hallar una aproximación racional de dicho número.

Ejemplo 2.20 Encuentre una aproximación racional al número e.

```
> fractions(exp(1))

[1] 2721/1001
```

Ejercicio 2.7 Obtenga una aproximación racional al número e utilizando el argumento `cycles` igual a 5.

También puede ser utilizado en objetos más complejos como en una matriz.

Ejemplo 2.21 Encuentre la matriz inversa de $L = \begin{pmatrix} 3 & 2 \\ 1 & 4 \end{pmatrix}$ expresando sus entradas en notación fraccionaria.

```
> L=matrix(c(3,2, 1,4), 2, 2)
> fractions(solve(L))

     [,1]  [,2]
[1,]  2/5 -1/10
[2,] -1/5  3/10
```

Capítulo 3

Hojas de datos

R constituye un entorno de programación creado con el fin de llevar a cabo procesos de análisis estadístico de datos. La palabra datos, según lo menciona Gómez (2003), equivale a información de tipo numérica sobre un determinado tema, en un determinado momento y cuya organización se realiza considerando aspectos relevantes y significativos que una vez recolectados pueden ser comparados, analizados o interpretados.

La organización de los datos en R requiere la creación de los llamados archivos de datos (data frames, por su nombre en inglés), los cuales permiten almacenar la información de tal forma que pueda ser resumida y estudiada a través del empleo de las técnicas estadísticas para el análisis de datos.

El archivo de datos puede crearse mediante líneas de código, esto es, introduciendo la información de cada objeto según la organización definida con anterioridad o importando el conjunto de datos ya almacenado en algún tipo de formato electrónico.

3.1. Introducción de los datos mediante línea de código

Para crear un data frame utilizamos la función `data.frame()`. Los argumentos principales de la función están constituidos por las variables (dadas como vectores o matrices) en que se ha organizado la información estadística que desea someterse a estudio.

Ejemplo 3.1 Cree un data frame llamado `mis_datos` que contenga los vectores `zona` y `x` de los Ejemplos 1.17 y 1.18, respectivamente.

```
> mis_datos<-data.frame(x,zona)
```

Se ha creado una hoja de datos con los vectores `x` y `zona`, los cuales pueden considerarse de ahora en adelante como variables. En R existe la posibilidad de modificar los nombres de las variables de modo que el usuario pueda conocer con exactitud la información almacenada en ellas. Para ejecutar este procedimiento se escribe en el argumento el nuevo nombre seguido por el operador " = " y el nombre que se quiere modificar.

Ejemplo 3.2 Suponiendo que `x` contiene información de las edades de un grupo de personas, construya la hoja de datos `mis_datos` con los vectores `x` y `zona` llamando `edad` al vector `x`.

```
> mis_datos<-data.frame(edad=x,zona)
```

Una vez creada la hoja de datos, puede observarse la información almacenada en una variable específica. Una forma de hacerlo es escribiendo el nombre de la hoja de datos seguido por el operador " $ " y el nombre de la variable.

Ejemplo 3.3 Obtenga la información contenida en la variable `edad`.

```
> mis_datos$edad

[1] 21 21 22 35 45
```

Por otro lado, si solo se escribe la palabra `edad` se imprime el mensaje: `objeto edad no encontrado` (puede hacerlo para comprobarlo), esto ocurre porque la hoja de datos no está activa. Si desea observarse la información referente a una variable y no escribir el nombre de la hoja de datos, el operador $ ni el nombre de la variable, es necesario activar la hoja. Esto se hace con la función `attach()`, la cual requiere como argumento el nombre de la hoja.

Ejemplo 3.4 Active la hoja de datos `mis_datos`.

```
> attach(mis_datos)
```

Una vez activa la hoja de datos, es posible visualizar la información que se almacena en una variable solamente digitando su nombre.

Ejemplo 3.5 Obtenga la información contenida en la variable `edad`.

```
> edad

[1] 21 21 22 35 45
```

Cuando las variables de la hoja de datos contienen mucha información y desea conocerse solo aspectos generales de los elementos que la componen puede observarse solo una parte de su contenido. Este proceso se realiza con la función `head()`, la cual requiere como argumentos el nombre de la hoja de datos y la cantidad de líneas que desean observarse. Por defecto la función `head()` muestra las primeras 6 filas de la hoja de datos.

Ejemplo 3.6 Obtenga la información contenida en las primeras tres líneas de la hoja `mis_datos`.

```
> head(mis_datos, 3)

  edad   zona
1   21  rural
2   21 urbana
3   22 urbana
```

3.2. Importando datos almacenados en otros formatos

Es muy común tener datos almacenados en otros formatos electrónicos como texto, Excel o Stata, entre otros. El programa R tiene la ventaja de poder leer los datos dentro de estos archivos.

3.2.1. Importando datos almacenados en formato txt

Un conjunto de datos almacenados en archivos tipo texto (`.txt`), normalmente presenta la característica de mostrar las variables organizadas por columnas y separadas por algún tipo de carácter, ya sea comas, tabulaciones, espacios, entre otros. Este aspecto tiene gran relevancia al momento de importar los datos.

Para leer los datos almacenados en este tipo de formato se usa la función `read.table()`, la cual tiene por argumentos principales los siguientes:

`file:` se refiere al nombre del archivo que desea importarse. Es importante aclarar que además de indicar el nombre también debe especificarse la dirección del archivo.

`header:` toma el valor de `TRUE` si la primera línea del archivo contiene el nombre de las variables, en caso contrario, adquiere el valor `FALSE`.

`sep:` se utiliza para identificar el carácter que se ha usado para separar las variables. Esta separación puede ser un espacio, una coma o una tabulación.

`dec:` para el separador que utiliza los decimales, el cual puede ser coma o punto.

Ejemplo 3.7 Suponga que el directorio `C:/ManualdeR/bases` contiene el archivo llamado `base1.txt`. Importe los datos de dicho archivo considerando que incluye el nombre de las variables en la primera fila, las cuales están separadas por tabulaciones, usando la coma como separador decimal y almacénelos en un data frame llamado `datos`.

```
> datos<-read.table("C:/ManualdeR/bases/base1.txt",
+ header = TRUE, sep = "\t", dec = ",")
```

3.2.2. Importando datos almacenados en formato Excel

Para leer datos almacenados en formato Excel se usa la función `read_excel()` del paquete `readxl` (Wickham y Bryan, 2018), para detectar en forma automática el formato de la extensión del archivo o las funciones `read_xls` o `read_xlsx` donde se especifica esta. Los argumentos principales de la función son los siguientes:

`path:` especifica la ruta del archivo.

`col_names:` toma el valor de `TRUE` si la primera línea del archivo contiene el nombre de las variables. Si es `FALSE` los nombres de las columnas serán generados automáticamente como `X1, X2, X3, etc`.

`col_types:` especifica el tipo de dato en cada columna, por defecto toma el valor `NULL`.

`sheet:` indica el nombre de la hoja del archivo de Excel donde se encuentran almacenados los datos. El valor por defecto corresponde a la primera hoja.

Ejemplo 3.8 Suponga que el directorio `C:/ManualdeR/bases` contiene el archivo llamado `base1.xlsx`. Importe los datos de dicho archivo considerando que incluye el nombre de las variables en la primera fila y almacénelos en un data frame llamado `base1`.

```
> library(readxl)
> base1 <- read_excel("C:/ManualdeR/bases/base1.xlsx", col_names = T)
```

3.2.3. Importando datos almacenados en formato csv

Para leer datos almacenados en formato csv se usa la función `read_csv()` del paquete `readr` (Wickham, Hester y Francois, 2017). Los principales argumentos de la función son los siguientes:

`file:` especifica la ruta del archivo.

`col_names:` toma el valor de `TRUE` si la primera línea del archivo contiene el nombre de las variables. Si es `FALSE` los nombres de las columnas serán generados automáticamente como `X1, X2, X3, etc`.

`col_types:` especifica el tipo de dato en cada columna, por defecto toma el valor `NULL`.

Ejemplo 3.9 Suponga que el directorio `C:/ManualdeR/bases` contiene el archivo llamado `base1.csv`. Importe los datos de dicho archivo considerando que incluye el nombre de las variables en la primera fila y almacénelos en un data frame llamado `base1`.

```
> library(readr)
> base1 <- read_csv("C:/ManualdeR/bases/base1.csv")
```

Si los datos se encuentran en archivos en formato dta (STATA), entonces se utiliza la función `read_stata` del paquete `haven` (Wickham y Miller, 2019).

Considere la siguiente información relacionada con el número de materias matriculadas, provincia de nacimiento, edad y peso de una muestra de 30 estudiantes de una institución educativa.

Estudiante	Materias	Provincia	Edad	Peso
1	2	San José	21	59
2	3	Alajuela	22	64
3	5	San José	23	70
4	2	San José	21	54
5	1	San José	22	60
6	5	San José	24	64
7	4	Heredia	25	72
8	3	Heredia	21	56
9	4	Heredia	22	60
10	1	Heredia	23	65
11	1	Heredia	24	73
12	3	Alajuela	24	56
13	4	San José	20	60
14	2	Alajuela	20	65
15	1	Alajuela	21	56
16	1	Alajuela	25	60
17	5	Alajuela	24	66
18	4	San José	23	56
19	3	San José	22	60
20	5	San José	24	66
21	4	San José	21	58
22	2	Heredia	21	61
23	4	Alajuela	20	66
24	2	Alajuela	19	59
25	1	Alajuela	25	61
26	2	Heredia	24	69
27	5	Heredia	23	59
28	3	Heredia	22	63
29	3	San José	21	69
30	4	Alajuela	20	59

Ejercicio 3.1 Digite la información anterior para crear una hoja de datos llamada base.

Capítulo 4

Tablas y medidas descriptivas en R

4.1. Medidas descriptivas

Como parte de los procesos del Análisis Estadístico se encuentra el cálculo de medidas. Entre las medidas descriptivas más comunes están el promedio, la mediana, la moda, los percentiles, la varianza y la desviación estándar.

Los comandos de R para la media y la mediana son `mean()` y `median()`, respectivamente. Los argumentos de estas funciones son:

`x`: representa el conjunto de datos que desean analizarse.

`na.rm`: valor lógico (T o F), que se refiere a los valores faltantes en el vector `x`, si `na.rm = T`, entonces los valores faltantes `NA` son eliminados en la realización de los cálculos.

Para ilustrar el uso de estas funciones se utilizó la hoja de datos `gala` del paquete `faraway` (Faraway, 2016), la cual contiene información sobre la variedad de especies en las Islas Galápagos. En los ejemplos se considera la variable `Elevation` que hace referencia a la elevación más alta de la isla (recuerde que para utilizar los objetos de un paquete primero debe cargarse y luego activar el objeto).

Ejemplo 4.1 Calcule el promedio y la mediana de la variable `Elevation`.

```
> mean(Elevation)

[1] 368.0333

> median(Elevation)

[1] 192
```

Para la moda puede utilizarse la función `mfv()` del paquete `modeest` (Poncet, 2012). Recuerde que los paquetes deben cargarse antes de usar alguno de sus objetos.

En el caso de los percentiles se utiliza la función `quantile()`, la cual incluye el argumento `probs` para indicar el percentil que desea calcularse.

Ejemplo 4.2 Calcule el percentil 75 de la variable `Elevation`.

```
> quantile(Elevation, probs=0.75)

   75%
435.25
```

Por otro lado, si desean obtenerse varios percentiles al mismo tiempo es posible hacerlo utilizando como argumento un vector que indique cada percentil.

Ejemplo 4.3 Calcule los percentiles 25, 50 y 75 de la variable `Elevation`.

```
> quantile(Elevation, c(0.25,0.5,0.75))

   25%    50%    75%
 97.75 192.00 435.25
```

Otras medidas que pueden calcularse son el mínimo (`min()`), el máximo (`max()`) y el rango intercuartílico (`IQR()`) de la variable en estudio.

Ejemplo 4.4 Calcule el mínimo, el máximo y el rango intercuartílico de la variable `Elevation`.

```
> c(min=min(gala$Elevation), max=max(gala$Elevation), IQR=IQR(gala$Elevation))

   min    max    IQR
  25.0 1707.0  337.5
```

Finalmente, la función `summary()` permite obtener un resumen de las principales medidas de posición de una variable.

Ejemplo 4.5 Obtenga las principales medidas de posición de la variable `Elevation`.

```
> summary(Elevation)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  25.00   97.75  192.00  368.03  435.25 1707.00
```

Para la varianza y la desviación estándar de una variable se utilizan las funciones `var()` y `sd()`, respectivamente.

Ejemplo 4.6 Obtenga la varianza y la desviación estándar de la variable `Elevation`.

```
> c(var=var(Elevation), sd=sd(Elevation))

        var          sd
177750.7230    421.6049
```

Ejercicio 4.1 Considere la hoja de datos `mtcars` del sistema base de R y calcule la media, mediana, moda, varianza y desviación estándar de la variable `mpg` (millas recorridas por galón).

La función `tapply()` permite aplicar una función a una variable según los factores de otra. Sus argumentos son:

`X:` normalmente un vector.

`INDEX:` una lista de factores, cada uno de la misma longitud de `X`.

`FUN:` la función que será aplicada.

`...:` argumentos opcionales para `FUN`.

`default:` (solo en el caso de simplificación de una matriz) es el valor con el que la matriz se inicializa como matriz.

`simplify:` valor lógico. Si es `FALSE`, siempre devuelve una matriz en modo `lista`.

Ejemplo 4.7 Considere la hoja de datos `base` del Ejercicio 3.1 y calcule la media y la varianza de la variable `edad` por `provincia de nacimiento`.

```
> tapply(base$edad1, base$provincia, mean)

Alajuela  Heredia San José
22.00000 22.77778 22.00000

> tapply(base$edad1, base$provincia, var)

Alajuela  Heredia San José
5.333333 1.944444 1.800000
```

Ejercicio 4.2 Utilice la hoja de datos `base` y calcule la mediana y la desviación estándar de la variable `edad` por `provincia de nacimiento`.

4.2. Frecuencias

Para obtener las frecuencias simples de una variable puede utilizarse la función `table()` en caso de frecuencias absolutas y `prop.table()` para las relativas. Es importante mencionar que la función `prop.table()` no se ejecuta en forma directa sobre la variable, sino sobre una tabla de frecuencias absolutas creada previamente mediante la función `table()`. En los ejemplos siguientes considere la hoja de datos `base` del Ejercicio 3.1.

Ejemplo 4.8 Obtenga las frecuencias simples para la variable `edad1`.

```
> table(base$edad1)

19 20 21 22 23 24 25
 1  4  7  5  4  6  3

> round(prop.table(table(base$edad1)), 2)

  19   20   21   22   23   24   25
0.03 0.13 0.23 0.17 0.13 0.20 0.10
```

Es posible determinar si la variable en estudio presenta valores faltantes cuando se emplean las funciones `table()` y `prop.table()` mediante el uso del argumento `useNA = "ifany"`.

4.3. Tablas de contingencias

Para la creación de tablas de contingencias también puede utilizarse la función `table()` y `prop.table()` indicando como argumentos las variables a analizar. En `prop.table()`, puede utilizarse el argumento `margin = 1` si desea obtener los marginales por fila, `margin = 2` si es por columna o `margin = NULL` (valor por defecto) si desea los valores globales.

Ejemplo 4.9 Obtenga la distribución absoluta y relativa de estudiantes según el número de materias matriculadas, clasificados por provincia de nacimiento.

```
> table(base$materias, base$provincia)

   Alajuela Heredia San José
1         2       2        1
2         2       2        2
3         3       2        2
4         2       2        3
5         1       1        3
```

Ejercicio 4.3 Obtenga la distribución relativa de los estudiantes según el número de materias matriculadas, clasificados por provincia de nacimiento.

Puede lograrse que aparezca en la tabla el nombre de cada una de las variables con ayuda de la función de concatenación `c()`, al definir un vector que incluya los nombres de las variables que van a presentarse.

Ejemplo 4.10 Obtenga la distribución de estudiantes según el número de materias matriculadas, clasificados por provincia de nacimiento indicando en la tabla el nombre de las variables.

```
> table(base[c("materias", "provincia")])

        provincia
materias Alajuela Heredia San José
       1        2       2        1
       2        2       2        2
       3        3       2        2
       4        2       2        3
       5        1       1        3
```

Ejemplo 4.11 Obtenga la distribución de estudiantes según la edad, clasificados por provincia de nacimiento indicando en la tabla el nombre de las variables.

```
> table(base[c("edad1", "provincia")])

     provincia
edad1 Alajuela Heredia San José
   19        1       0        0
   20        3       0        1
   21        1       2        4
   22        1       2        2
   23        0       2        2
   24        2       2        2
   25        2       1        0
```

Otra forma de construir las tablas de contingencias es utilizando la función `ftable()`. Sus argumentos son:

`x, ...`: objetos de R que pueden interpretarse como factores o listas cuyas componentes pueden considerarse como tablas de contingencia de la clase `table` o `ftable`.

`row.vars`: un vector de enteros que indica la cantidad de variables o un vector de caracteres que brinda los nombres de las variables a ser usadas en las filas de la tabla.

`col.vars:` un vector de enteros que indica la cantidad de variables o un vector de caracteres que brinda los nombres de las variables a ser usadas en las columnas de la tabla.

Si los argumentos `row.vars` y `col.vars` no son especificados, entonces las categorías de la última variable indicada son consideradas para las columnas.

Ejemplo 4.12 Obtenga la distribución de estudiantes según el número de materias matriculadas, clasificados por provincia de nacimiento utilizando la función `ftable()`.

```
> ftable(base$materias, base$provincia)

   Alajuela Heredia San José

1         2       2        1
2         2       2        2
3         3       2        2
4         2       2        3
5         1       1        3
```

Ejercicio 4.4 Obtenga la distribución de estudiantes según el número de materias matriculadas, clasificados por provincia de nacimiento utilizando la función `ftable()`, de modo que la distribución muestre el nombre de las variables.

La función `ftable()` es muy útil cuando deben construirse tablas de contingencias que involucran más de dos variables.

Ejercicio 4.5 Considere la hoja de datos `mtcars` y obtenga la distribución de vehículos considerando la cantidad de cilindros (`cyl`), el número de velocidades (`gear`) y el tipo de transmisión (`am`), utilizando la función `ftable()`.

4.4. Distribuciones de frecuencias de variables continuas

Para distribuciones de variables continuas puede utilizarse la función `fdt()` del paquete `fdth` (Faria, Jelihovschi y Allaman, 2017). Los argumentos principales son:

`x:` objeto tipo matriz, data frame o vector. Si `x` es un data frame o matriz debe contener al menos una columna numérica.

`k:` número de clases o intervalos.

`star:` límite inferior de la primera clase o intervalo de la distribución.

`end:` límite superior de la última clase o intervalo de la distribución.

`h:` ancho del intervalo de la distribución.

`breaks:` método utilizado para determinar el número de clases. Estos métodos son `Sturges`, `Scott` o `FD`.

`right:` define los intervalos de clase abiertos en el límite superior (por defecto: `right = FALSE`).

Ejemplo 4.13 Obtenga la distribución de estudiantes según peso que inicie en 54, finalice en 74 y el ancho de cada intervalo sea 4.

```
> library(fdth)
> dist <- fdt(base$peso, start = 54, end = 74, h = 4)
> dist

 Class limits  f   rf rf(%) cf  cf(%)
      [54,58)  5 0.17 16.67  5  16.67
      [58,62) 12 0.40 40.00 17  56.67
      [62,66)  5 0.17 16.67 22  73.33
      [66,70)  5 0.17 16.67 27  90.00
      [70,74)  3 0.10 10.00 30 100.00
```

Ejemplo 4.14 Obtenga la distribución de estudiantes según peso considerando 5 clases.

```
> print(fdt(base$peso, k = 5))

 Class limits  f   rf rf(%) cf  cf(%)
      [53,58)  5 0.17 16.67  5  16.67
      [58,62) 12 0.40 40.00 17  56.67
      [62,66)  5 0.17 16.67 22  73.33
      [66,70)  5 0.17 16.67 27  90.00
      [70,74)  3 0.10 10.00 30 100.00
```

Ejercicio 4.6 Obtenga la distribución de estudiantes según peso considerando el método de `Sturges` para determinar las clases.

Capítulo 5

Gráficas en R

Como parte de los procesos del Análisis Estadístico, R es un entorno que permite la creación de gráficas. Diferentes paquetes permiten la construcción de gráficas; sin embargo, en este documento se estudian aquellas más comunes mediante el empleo de funciones estándar.

El sistema de graficación base de R cuenta con tres tipos de funciones: de alto nivel, de bajo nivel y las interactivas.

5.1. Los parámetros de graficación

Los parámetros de graficación permiten realizar modificaciones a gráficas ya existentes. Existen 73 parámetros de graficación en R y la lista de ellos puede obtenerse mediante la expresión `help(par)`.

Entre los parámetros más relevantes se tienen:

`adj:` permite indicar la forma en que el texto puede justificarse. Toma valores entre [0, 1], donde 0 justifica el texto a la izquierda, 0.5 para centrar y 1 para justificar a la derecha.

`bg:` especifica el color de fondo de las gráficas.

`bty:` determina el tipo de caja que se dibuja alrededor de la gráfica.

`cex:` factor de expansión que permite modificar el tamaño de los símbolos.

`cex.main:` factor de expansión que permite modificar el tamaño del título principal.

`cex.sub:` factor de expansión que permite modificar el tamaño de los subtítulos.

`col:` permite asignar color a los símbolos utilizados en la gráfica.

`col.axis:` asigna color a las anotaciones sobre los ejes.

`col.lab:` indica el color que puede utilizarse en las etiquetas de los ejes.

`col.main:` para asignar color al título de la gráfica.

`fin:` las dimensiones de la figura (ancho y alto) en pulgadas.

`font:` valor para especificar el tipo de letra que se utilizará para el texto. 1 = plain (normal), 2 = bold (negrita), 3 = italic (cursiva) y 4 = bold italic (negrita y cursiva).

`font.axis`: tipo de fuente a utilizar para las anotaciones de los ejes.

`lty`: especifica el tipo de línea, 0 = blank (sin línea), 1 = solid (sólida), 2 = dashed (discontinua), 3 = dotted (punteada), 4 = dotdash (punto y guión), entre otras.

`lwd`: valor positivo para definir el ancho de la línea.

`mfrow`: un vector de la forma `c(a, b)` que indica que las gráficas posteriores serán dibujadas en un arreglo de `a` filas por `b` columnas y distribuidas por filas de izquierda a derecha y de arriba hacia abajo.

`mfcol`: aplicación similar a la de `mfrow` solo que las gráficas sucesivas son acomodadas por columnas de izquierda a derecha y de arriba hacia abajo.

`pch`: entero o carácter que especifica el tipo de símbolo que será utilizado para graficar los puntos.

5.1.1. La función `par()`

Esta función permite obtener los valores vigentes en los parámetros de graficación dentro del sistema base de R. La lista completa puede observarse al digitar en la línea de comando la expresión `par()`. Para observar un valor específico, se digita como argumento el nombre del parámetro entre comillas dobles.

Ejemplo 5.1 Identifique los valores en los parámetros tipo línea (`lty`) y color de fuente para el título (`col.main`) que están vigentes para realizar la construcción de gráficas.

```
> par("lty"); par("col.main")

[1] "solid"

[1] "black"
```

El resultado del Ejemplo 5.1 indica que se utiliza la línea sólida (continua) para el trazo y para la fuente del título de la gráfica se usa por defecto el color negro.

Ejercicio 5.1 Determine el tipo y color de fuente que están vigentes para las anotaciones del subtítulo de las gráficas.

Esta función también permite realizar cambios en los valores vigentes de dichos parámetros. Para llevarla a cabo, esta toma la forma `par(x = value)`, donde `x` representa el nombre del parámetro a modificar y `value` es el nuevo valor que será asignado, el cual puede estar especificado por un nombre o por un número.

Ejemplo 5.2 Modifique el tipo de línea para el trazo de gráficas de modo que estas muestren una línea discontinua (dashed).

```
> par(lty = "dashed")
```

O bien

```
> par(lty = 2)
```

Ejercicio 5.2 Cambie el color de fuente para las anotaciones del subtítulo y para el trazo de gráficas de modo que las nuevas gráficas utilicen el rojo.

5.2. La función `plot`

La función `plot()` es una de las principales funciones de alto nivel utilizada para la graficación en R. Sus argumentos principales son:

`x:` se refiere a las coordenadas del eje X dentro de la gráfica, o bien, una sola estructura que permita graficar como una función o un vector.

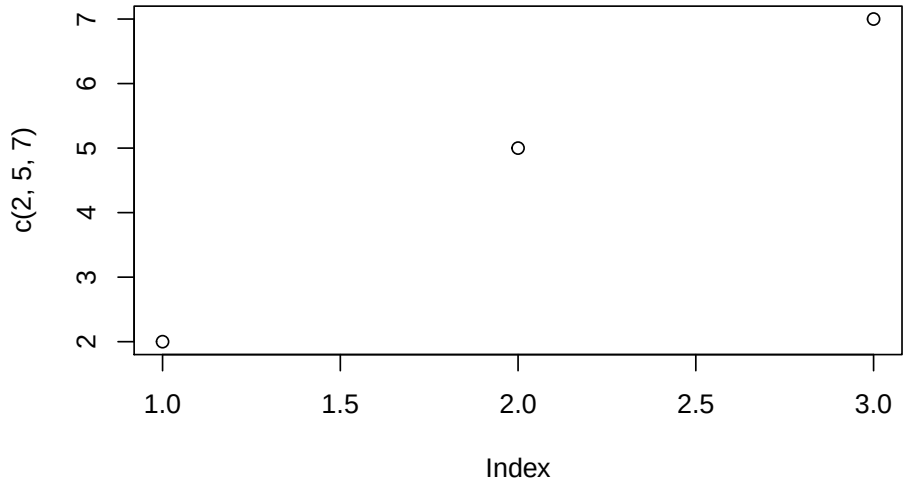
`y:` representa las coordenadas del eje Y. Es opcional y solo es requerido si `x` no es una estructura apropiada.

`type`: especifica el tipo de gráfica que desea trazarse. Se utiliza:

- "`p`" para puntos,
- "`l`" para líneas,
- "`b`" para ambos (puntos y líneas),
- "`c`" para líneas sin mostrar los puntos,
- "`o`" para líneas y puntos sobrepuestos,
- "`h`" para histograma como líneas verticales,
- "`s`" para escalonado,
- "`S`" para otro tipo de escalonamiento,
- "`n`" para no graficar.

Ejemplo 5.3 Grafique el vector `c(2,5,7)` digitando `plot(c(2,5,7))`.

```
> plot(c(2,5,7), fin = 2)
```



Gráfica 5.1: Gráfica generada por la función `plot`

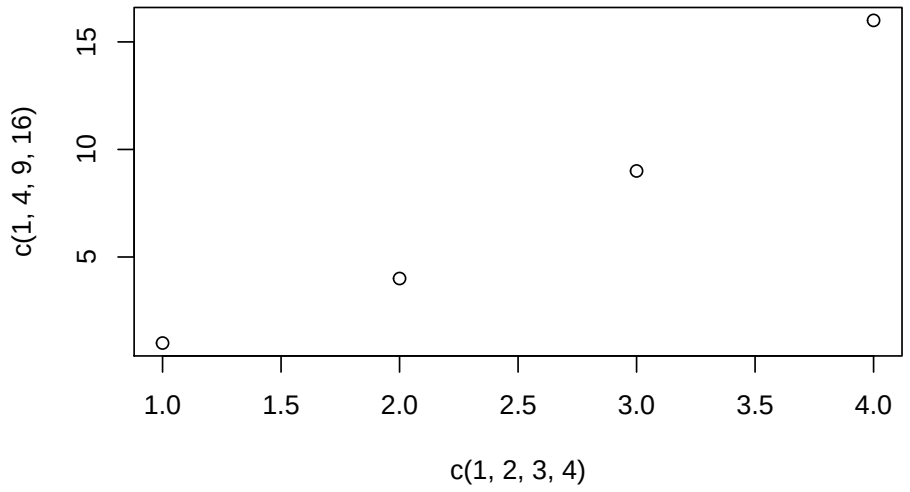
En el Ejemplo 5.3 se ha graficado cada elemento del vector (2,5,7) con respecto a la posición que ocupa en el vector por medio de puntos. En este caso el vector (2,5,7) es una estructura apropiada, por ello no fue necesario indicar el argumento `y`.

Ejercicio 5.3 Grafique los elementos del vector (4,3,2,1,2).

Para graficar pares ordenados, como por ejemplo los puntos (1,1), (2,4), (3,9) y (4,16), deben definirse dos estructuras adecuadas, una para enlistar los elementos del eje X y otra para las coordenadas en el eje Y. De esta manera, la función `plot()` requiere de los argumentos `x, y`.

Ejemplo 5.4 Grafique los puntos (1,1), (2,4), (3,9) y (4,16).

```
> plot(c(1,2,3,4), c(1,4,9,16))
```



Gráfica 5.2: Graficando pares ordenados con la función `plot`

Ejercicio 5.4 Grafique los puntos indicados en cada caso.

1. (4, 3), (2, 4), (6, 2) y (1, 2).
2. (14, 25), (12, 32), (10, 24) y (16, 25).
3. (100, 225), (125, 235) y (140, 260).

La función posee otros argumentos importantes que permiten modificar el aspecto de la gráfica y darle así el formato adecuado, de modo que su lectura e interpretación sean las más claras posibles. Algunos de estos son los siguientes:

`main`: para escribir el título de la gráfica.

`sub`: para escribir un subtítulo.

`xlab`: especifica un título al eje X.

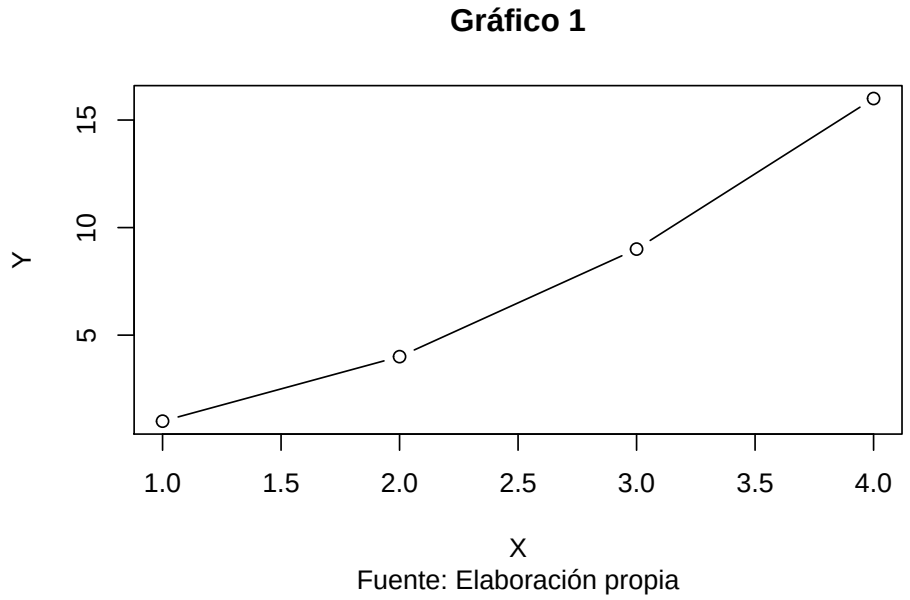
`ylab`: escribe un título al eje Y.

`col`: permite dar color al tipo de símbolo que se está usando.

`pch`: especifica el tipo de símbolo a utilizar en la graficación de puntos.

Ejemplo 5.5 Represente la información de la Gráfica 5.2 utilizando líneas y puntos, con título de `Gráfico 1`, subtítulo `Fuente: Elaboración propia` y rotulando los ejes como `X` e `Y`.

```
> plot(c(1,2,3,4), c(1,4,9,16), type = "b", main = "Gráfico 1",
+ sub = "Fuente: Elaboración propia", xlab = "X", ylab = "Y")
```



Gráfica 5.3: Gráfica de líneas y puntos con título

En cuanto al uso de los colores en el proceso de graficación, estos pueden ser llamados por el nombre (cadena de caracteres) o por medio de un código numérico. Exiten 657 colores en R y para conocerlos se utiliza la función `colors()`.

La función `grep()` puede ser utilizada para identificar los colores que muestran una subcadena específica.

Ejemplo 5.6 Obtenga todos los colores que incluye la subcadena "`blue`".

```
> grep("blue", colors(), value=F)
```

Además, en R existen 26 símbolos para puntos que pueden usarse en el argumento `pch`, cuyos códigos van del 0 al 25. Para conocerlos puede utilizarse la función `plot()`.

Ejercicio 5.5 Ejecute la instrucción `plot(1:26, pch = 0:25)`.

El entorno R permite que las gráficas existentes puedan modificarse incorporando nuevos elementos por medio de las funciones de bajo nivel.

5.3. Funciones de bajo nivel

Las funciones de bajo nivel permiten incorporar información adicional a una gráfica ya construida. A continuación se detallan algunas de ellas.

5.3.1. La función `points()`

`points()` es una función que permite agregar un conjunto de puntos a una gráfica ya existente. Sus argumentos principales son:

`x, y:` son las coordenadas o puntos que desean graficarse.

`type:` carácter que indica el tipo de gráfica que se quiere.

`...:` otros parámetros de graficación que pueden utilizarse como argumentos, entre los más comunes, se tienen `pch`, `col` y `bg`, este último para asignar color a los símbolos `pch = 21:25`.

5.3.2. La función `abline()`

Esta función permite agregar líneas a gráficas ya elaboradas. Sus argumentos son:

`a:` representa la intersección (con el eje Y) de la recta.

`b:` se refiere a la pendiente de la recta.

`v:` valor en el eje X para el trazo de una línea vertical que pasa por `v`.

`h:` valor en el eje Y para el trazo de una línea horizontal que pasa por `h`.

`coef:` un vector para especificar la intersección y la pendiente.

`...:` otros parámetros de graficación que pueden utilizarse como argumentos.

Algunos de los parámetros de graficación que pueden especificarse son `col`, `lty` y `lwd`.

Ejemplo 5.7 Considere la Gráfica 5.3 y agregue el punto `(1,5)` usando un cuadrado como símbolo con borde color `gris` y con relleno color `verde`, una línea vertical color gris en `x = 2`.

```
> plot(c(1,2,3,4), c(1,4,9,16), type = "b", bg = "black",
+ main = "Gráfico 1", xlab = "X", ylab = "Y")
> points(1,5, col = "grey", pch = 22, bg = "green", cex = 0.8)
> abline(v = 2, col = "green", lwd = 2)
```

5.3.3. La función `text()`

Esta función permite agregar texto a una gráfica que ha sido construida. Entre sus argumentos están:

`x, y:` vector de coordenadas numéricas que especifican donde será incorporado el texto.

`labels`: vector de caracteres o expresión que será agregada en la gráfica.

`...`: otros parámetros de graficación.

5.3.4. La función `polygon()`

Esta función permite construir un polígono indicando sus respectivos vértices. Sus argumentos son:

`x, y`: vectores que contienen las coordenadas de los vértices del polígono.

`col`: especifica el color con el que será llenado el polígono.

`border`: permite asignar color al borde.

`lty`: especifica el tipo de línea a utilizar como en `par`.

`...`: otros parámetros de graficación que pueden darse como argumentos.

Ejemplo 5.8 Considere la Gráfica 5.7 y agregue la etiqueta `x = 2` y una caja rectangular que contenga esta etiqueta.

```
> plot(c(1,2,3,4), c(1,4,9,16), type = "b", bg = "black",
+ main = "Gráfico 1", xlab = "X", ylab = "Y")
> points(1,5, col = "grey", pch = 22, bg = "green", cex = 0.8)
> abline(v = 2, col = "green", lwd = 2)
> text(2.2, 10, "x = 2")
> polygon(c(2.02, 2.4, 2.4, 2.02), c(9, 9, 11, 11))
```

5.3.5. La función `legend()`

Esta función permite agregar una leyenda a la gráfica en la posición que se especifique. Sus principales argumentos son:

`x, y`: para especificar las coordenadas de la gráfica en la que desea ubicarse la leyenda. Las coordenadas pueden indicarse por medio de un vector numérico o de caracteres, en el caso de este último se señala el lugar específico en el que desea ubicarse la leyenda, como por ejemplo, `top, topleft, center, topright, etc`.

`fill`: especifica los colores que se utilizan en las cajas para etiquetar las categorías de la variable.

`border`: se utiliza únicamente si `fill` es especificado y permite asignar color al borde de las cajas.

`text.col`: para asignar color al texto de la leyenda.

`text.font`: para indicar el tipo de fuente del texto de la leyenda.

`horiz`: posición en la que desea que aparezca la leyenda, si no se especifica aparece en posición vertical.

`title`: si se quiere indicar un título a la leyenda.

`inset`: representa la distancia de los márgenes de la gráfica a la que desea colocarse la leyenda.

5.4. Las funciones de alto nivel

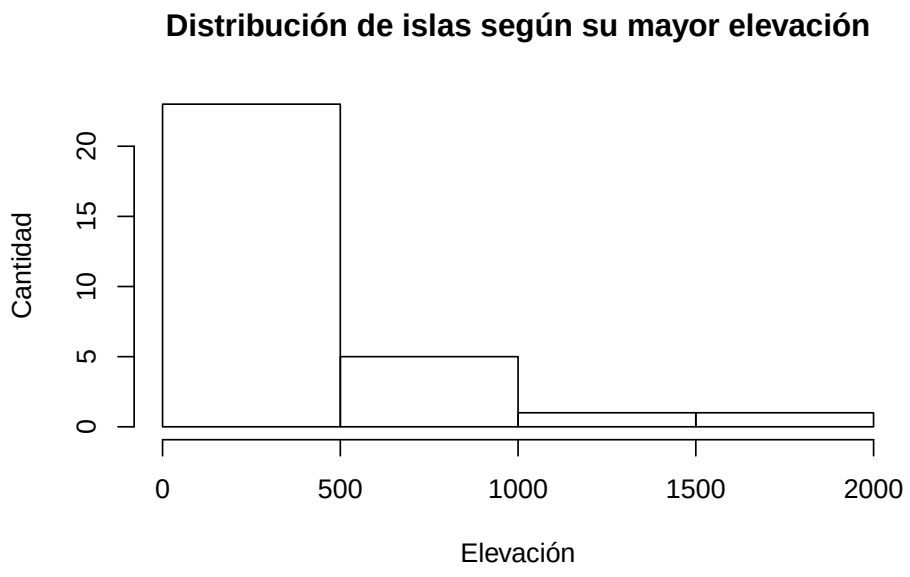
Las funciones de alto nivel permiten crear gráficas ya establecidas. Además de la función `plot()`, se tienen `hist()`, `barplot()`, entre otras.

5.4.1. Histograma

Para representar la distribución de frecuencias de variables continuas se usa el histograma, el cual puede construirse por medio de la función `hist()`. Entre sus argumentos principales están `main, sub, xlab, ylab, ylim`, `xlim` y `x`, donde este último constituye el vector de datos para el que se construye el histograma.

Ejemplo 5.9 Construya una Gráfica para representar la distribución de frecuencias de la variable `Elevation` de la hoja de datos `gala` cuyo título sea `Distribución de islas según su mayor elevación`, llamando el eje X como `Elevación` y el eje Y como `Cantidad`.

```
> library(faraway)
> hist(gala$Elevation,
+ main = "Distribución de islas según su mayor elevación",
+ xlab = "Elevación", ylab = "Cantidad")
```



Gráfica 5.4: Histograma para la distribución de islas según su mayor elevación

Otros argumentos importantes son:

`breaks`: valor para indicar el número aproximado de clases. También puede ser un vector cuyos valores indican los puntos límites de los intervalos o clases.

`freq`: valor lógico. Si es `TRUE` el histograma muestra las frecuencias absolutas de cada clase, si es `FALSE` entonces se grafican las densidades de probabilidad.

`probability`: es una representación de `!freq` para mostrar compatibilidad con S.

`right`: valor lógico, si es `TRUE` se generan intervalos abiertos a la izquierda y cerrados a la derecha, si es `FALSE` se tendrá el caso contrario.

`col`: permite asignar color a las barras.

`border`: permite definir un color al borde de las barras, por defecto se usa el negro.

`labels`: valor lógico, si es `TRUE` añade como etiqueta la frecuencia a cada barra, por defecto `labels = False`.

`nclass`: es equivalente al argumento `breaks`.

El parámetro `las` controla la orientación de las etiquetas de los ejes. Por defecto el sentido está dado por `las = 0`. En el caso de `las = 1`, las etiquetas en Y son perpendiculares al eje y las de X son paralelas, con `las = 2`, las etiquetas son perpendiculares al eje X y paralelas a Y y con `las = 3`, ambas etiquetas son perpendiculares a los ejes. Por otro lado, puede asignarse un color específico a las barras utilizando el argumento `col`.

Ejemplo 5.10 Construya la Gráfica 5.4 asignando a las barras el color verde y modificando la orientación de las anotaciones del eje Y.

```
> hist(gala$Elevation, las = 1,
+ main = "Distribución de islas según altura sobre el nivel del mar",
+ xlab = "Elevación", ylab = "Cantidad", col = "green")
```

Como se dijo anteriormente, el argumento `breaks` permite definir la cantidad de intervalos para representar el histograma y con ello modificar el efecto de la distribución. Este cambio puede hacerse digitando el número de clases que se desean o definiendo un vector que especifique los límites.

Ejemplo 5.11 Digite en la línea de comando `breaks = seq(0,2000, by = 500)` de modo que se obtenga una distribución cuya forma es similar a la generada en la Gráfica 5.4.

```
> hist(gala$Elevation,
+ main = "Distribución de islas según altura sobre el nivel del mar",
+ xlab = "Elevación", ylab = "Cantidad", col = "green",
+ breaks = seq(0,2000, by = 500), las = 1)
```

5.4.2. Gráficas de barras

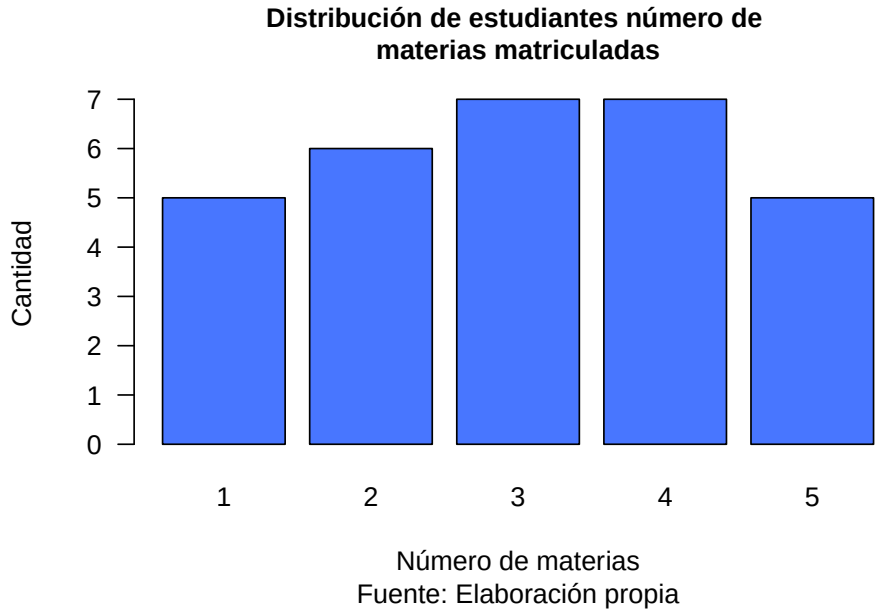
Las gráficas de barras son utilizadas para representar las series de datos cuantitativos discretos o cualitativos. En el caso de variables discretas, la distribución de frecuencias puede representarse en una gráfica de barras verticales. Para las variables cualitativas se usa la gráfica horizontal.

5.4.3. Gráfica de barras verticales

Para construir la gráfica de barras verticales se utiliza la función `barplot()`. Entre sus argumentos se encuentran `border`, `main`, `sub`, `xlab`, `ylab`, `xlim`, `ylim`, `las` y `height`, este último constituye un vector o matriz de valores que describen las barras que van a construirse.

Ejemplo 5.12 Considere la hoja de datos `base` y construya una gráfica de barras verticales que muestre la distribución absoluta de la variable `materias` cuyo título sea `Distribución de estudiantes según el número de materias matriculadas`, rotulando el eje X `Número de materias` y el eje Y `Cantidad` y la fuente `Elaboración propia`.

```
> barplot(table(base$materias), cex.main = 1, col = "royalblue1",
+ main = "Distribución de estudiantes según número de
+ materias matriculadas", xlab = "Número de materias", ylab = "Cantidad",
+ sub = "Fuente: Elaboración propia", las = 1)
```



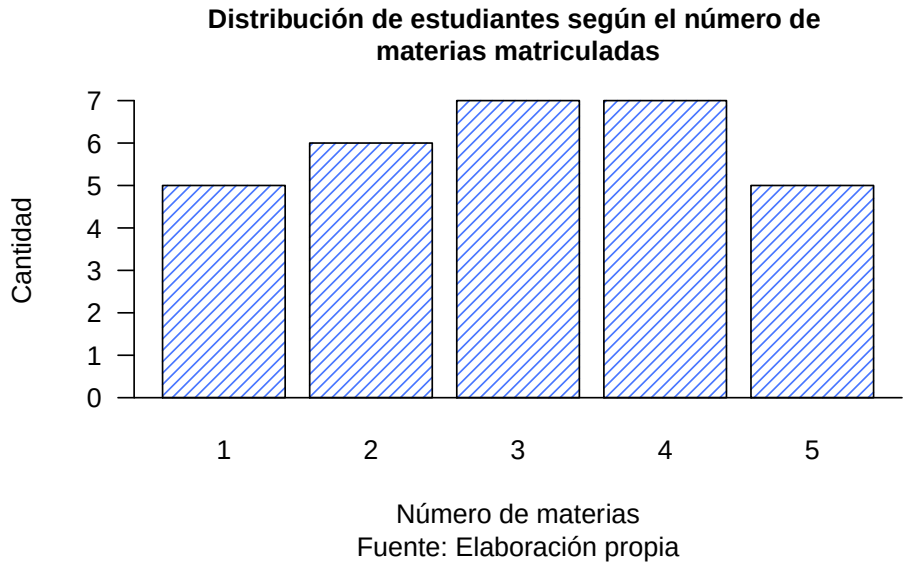
Gráfica 5.5: Gráfica de barras para la variable número de materias

En la Gráfica 5.5 se observa que el eje X no muestra una línea continua; sin embargo, esta situación podría modificarse trazando una línea horizontal en `y = 0` con ayuda de la función `abline()`.

Además, en las barras de la gráfica puede modificarse la densidad (en líneas por pulgada) de las líneas de sombreado, con ayuda del argumento `density`.

Ejemplo 5.13 Considerando la información de la Gráfica 5.5 utilice el argumento `density` para modificar la densidad del sombreado y dibuje una línea horizontal en `y = 0`.

```
> barplot(table(base$materias), density = 20, cex.main = 1,
+ main = "Distribución de estudiantes según el número de
+ materias matriculadas", xlab = "Número de materias", ylab = "Cantidad",
+ sub = "Fuente: Elaboración propia", las = 1, col ="royalblue1")
> abline(h = 0)
```



Gráfica 5.6: Gráfica de barras usando el argumento density

Otros argumentos importantes de esta función son los siguientes:

`width:` define el ancho de cada una de las barras.

`space:` define el ancho del espacio entre cada una de las barras que puede ser el mismo en todos los casos o distinto para cada barra.

`names.arg:` para indicar el nombre de las categorías de la variable que representa cada barra. Si se omite, entonces las barras son rotuladas utilizando los nombres que estén asignados en el objeto indicado en `height`.

`legend.text:` permite escribir una leyenda para la gráfica. Es útil si `height` es una matriz.

`beside:` es un valor lógico que permite dibujar gráficas compuestas si toma el valor `False` y comparativas si su valor es `True`.

`horiz:` valor lógico. Si es TRUE entonces las barras se dibujan en posición horizontal.

`col:` permite especificar el color de las barras.

`cex.axis:` factor de expansión para ajustar las etiquetas de los ejes numéricos.

`cex.names:` factor de expansión para ajustar los nombres o etiquetas de las barras.

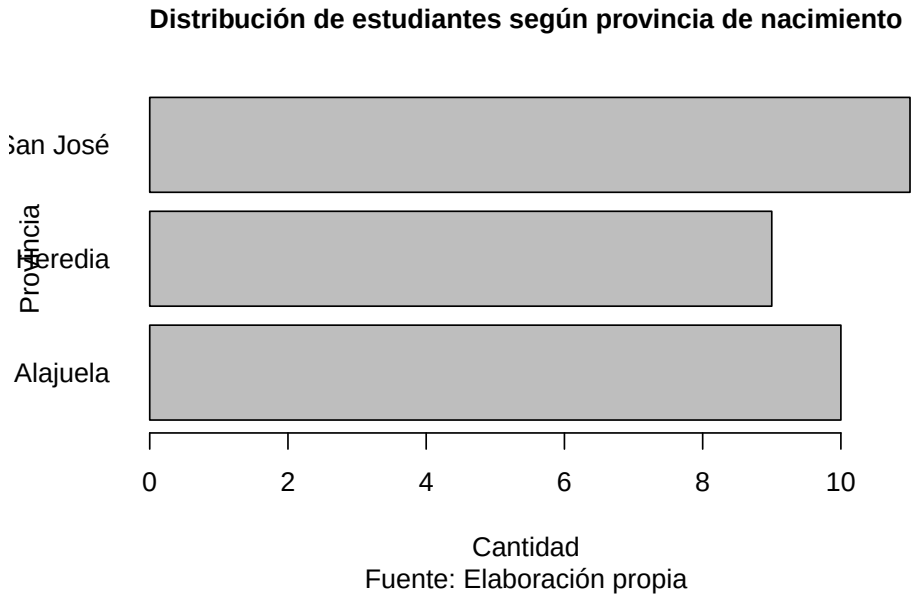
Ejercicio 5.6 Considere la información de la Gráfica 5.5 y utilice el argumento `density` para modificar la densidad de las líneas de sombreado, cambie el ancho y el color de las barras y ajuste el tamaño de las etiquetas de los ejes numéricos.

5.4.4. Gráfica de barras horizontales simples

La gráfica de barras horizontales permite representar la información de variables de tipo cualitativo. Para su construcción se considera el argumento `horiz = T` en la función `barplot()`.

Ejemplo 5.14 Considere la hoja de datos `base` y construya una gráfica de barras para la variable `provincia`, con título `Distribución de estudiantes según provincia de nacimiento`, llamando el eje Y `Provincia` y el eje X `Cantidad`.

```
> barplot(table(base$provincia),
+ main = "Distribución de estudiantes según provincia de nacimiento",
+ xlab = "Cantidad", ylab = "Provincia", cex.main = 1,
+ sub = "Fuente: Elaboración propia", las = 1, horiz = T)
```



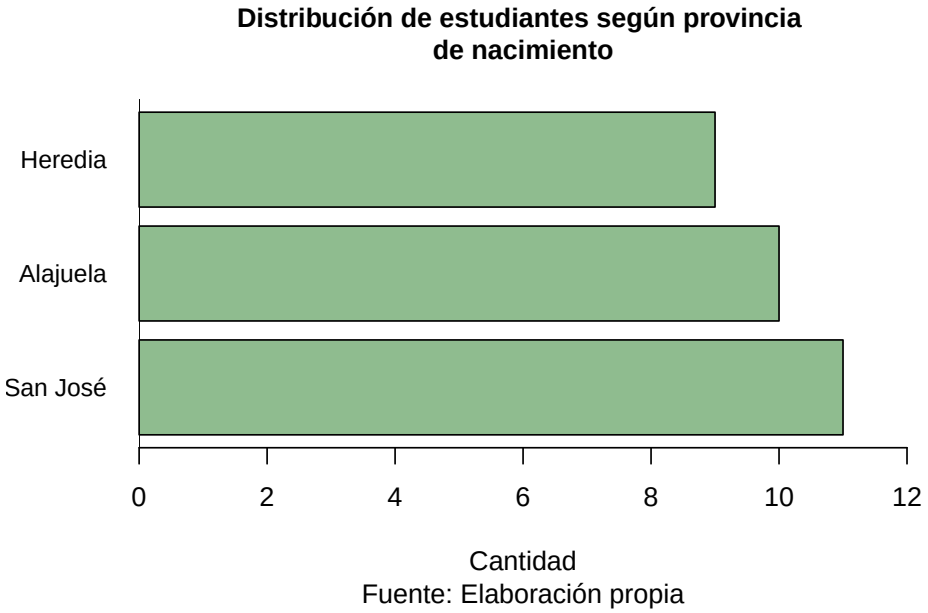
Gráfica 5.7: Gráfica de barras horizontales para la variable provincia

Pueden ajustarse las etiquetas asignadas al eje Y con el argumento `cex.names`, modificar la escala

del eje horizontal con el argumento `xlim`, ordenar la distribución de las barras con ayuda de la función `sort()` y dibujar el eje vertical con ayuda de la función `abline()`.

Ejemplo 5.15 Considere la Gráfica 5.7, ajuste el tamaño de las etiquetas de modo que pueda leerse adecuadamente, ordene las barras en forma decreciente, modifique la escala del eje horizontal, asigne un color a las barras y dibuje el eje vertical.

```
> barplot(sort(table(base$provincia), decreasing = T), xlim = c(0,12),
+ main = "Distribución de estudiantes según provincia
+ de nacimiento", xlab = "Cantidad", cex.main = 1, cex.names = 0.9,
+ sub = "Fuente: Elaboración propia", las = 1, horiz = T, col = "darkseagreen")
> abline(v = 0)
```



Gráfica 5.8: Gráfica de barras horizontales ajustando el nombre de las barras

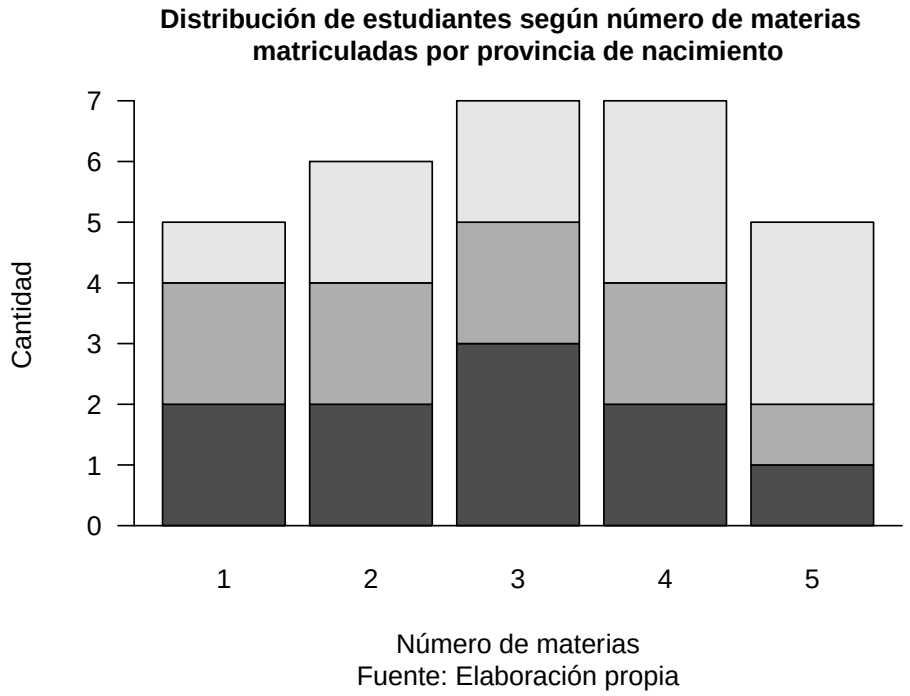
Ejercicio 5.7 Considere la Gráfica 5.7, ordene la distribución en forma creciente, asigne las etiquetas con el argumento `names.arg`, cambie el color del borde y del interior de las barras y modifique el espacio entre ellas.

5.4.5. Gráfica de barras compuestas

Para construir una gráfica de barras compuestas primero se construye una tabla de contingencias con las variables que desean compararse por medio de la función `table()` para el caso de frecuencias absolutas y `prop.table()` para las relativas. Luego, esta tabla se convierte en el argumento `height` en la función `barplot()`.

Ejemplo 5.16 Considere la hoja de datos `base` y construya una gráfica en la que pueda visualizarse la distribución de estudiantes por número de materias y provincia cuyo título sea `Distribución de estudiantes según número de materias matriculadas por provincia de nacimiento`, llamando `Cantidad` al eje Y, `Número de materias` al eje X y con subtítulo `Fuente: Elaboración propia`.

```
> l <- table(base$provincia, base$materias)
> barplot(l, xlab = "Número de materias", ylab = "Cantidad",
+ main = "Distribución de estudiantes según número de materias
+ matriculadas por provincia de nacimiento", cex.main = 1,
+ sub = "Fuente: Elaboración propia", las = 1)
> abline(h = 0)
```



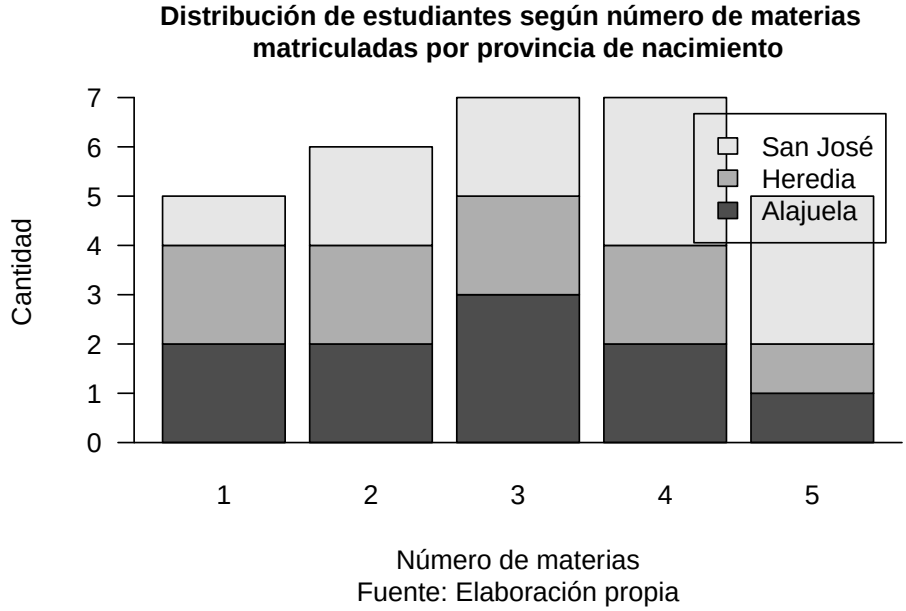
Gráfica 5.9: Gráfica compuesta para las variables número de materias y provincia

Ejercicio 5.8 Considere la Gráfica 5.9 y cambie el color para cada una de las categorías de la variable provincia.

En la Gráfica 5.9 puede utilizarse el argumento `legend` para identificar las categorías de la variable `provincia`.

Ejemplo 5.17 Incorpore la leyenda en la Gráfica 5.9 y disminuya el tamaño de la fuente del título.

```
> barplot(l, xlab = "Número de materias", ylab = "Cantidad",
+ main = "Distribución de estudiantes según número de materias
+ matriculadas por provincia de nacimiento", cex.main = 1, las = 1,
+ sub = "Fuente: Elaboración propia", legend = c("Alajuela", "Heredia",
+ "San José"))
> abline(h = 0)
```



Gráfica 5.10: Gráfica compuesta incluyendo la leyenda

Se observa que la leyenda está superpuesta sobre las barras, para corregir esta situación pueden modificarse los límites de los ejes con ayuda de los argumentos `ylim` y `xlim`.

Ejemplo 5.18 Modifique los límites de los ejes de la Gráfica 5.10 de modo que la leyenda no esté superpuesta en una de las barras.

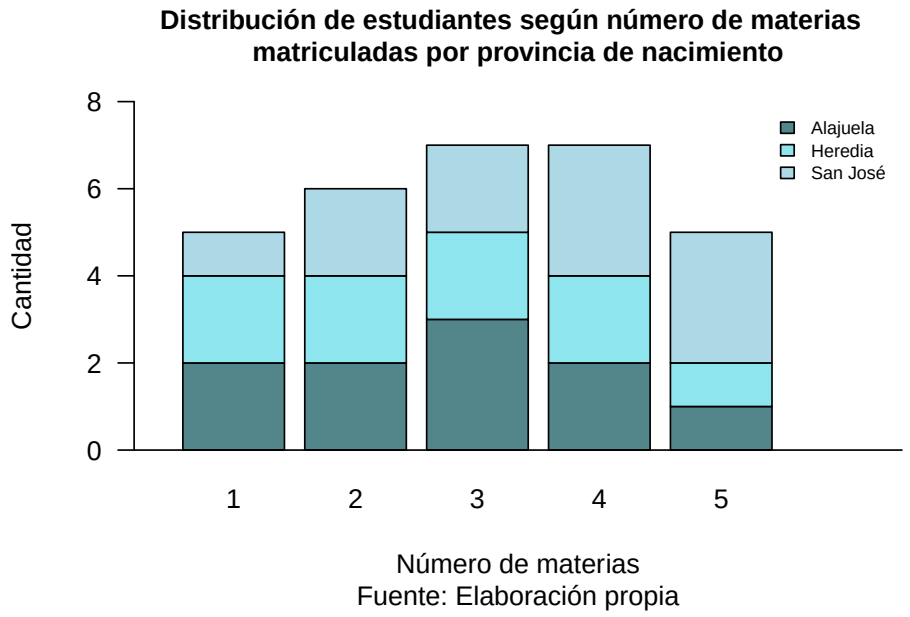
```
> barplot(l, xlab = "Número de materias", ylab = "Cantidad",
+ main = "Distribución de estudiantes según número de materias
+ matriculadas por provincia de nacimiento", las = 1, cex.main = 0.8,
+ sub = "Fuente: Elaboración propia", ylim = c(0, 8), xlim = c(0, 10),
+ legend.text = c("Alajuela", "Heredia", "San José"))
> abline(h = 0)
```

También puede incorporarse la leyenda con ayuda de la función `legend()` y utilizar sus argumentos

para modificar su aspecto. Por ejemplo, `inset` permite establecer con más precisión la posición de la leyenda.

Ejemplo 5.19 Utilice la función `legend()` para incorporar la leyenda de la Gráfica 5.10 y modifique su aspecto eliminando el borde con el argumento `bty`, cambiando el tamaño con el argumento `cex`, asignando colores distintos con el argumento `col` y ajustando su posición con ayuda de `inset`.

```
> barplot(l, xlab = "Número de materias", ylab = "Cantidad",
+ main = "Distribución de estudiantes según número de materias
+ matriculadas por provincia de nacimiento", las = 1, cex.main = 1,
+ sub = "Fuente: Elaboración propia", ylim = c(0, 8), xlim = c(0,7),
+ col = c("cadetblue4", "cadetblue2", "lightblue"))
> legend("topright", legend = c("Alajuela", "Heredia", "San José"),
+ bty = "n", cex = 0.7, fill = c("cadetblue4", "cadetblue2", "lightblue"),
+ inset = 0.01)
> abline(h = 0)
```



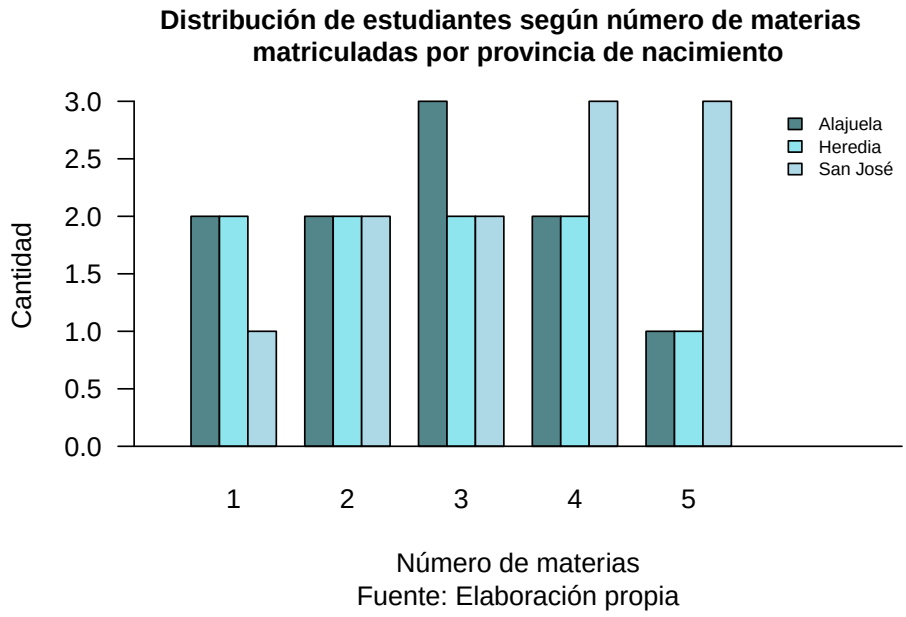
Gráfica 5.11: Gráfica compuesta modificando aspectos de la leyenda

5.4.6. Gráfica de barras comparativas

Para generar una gráfica de barras comparativas se sigue un procedimiento similar al utilizado en la construcción de gráficas compuestas, pero considerando el argumento `beside = TRUE`. Este argumento funciona tanto si se usan barras verticales como horizontales.

Ejemplo 5.20 Modifique la Gráfica 5.9 de modo que pueda compararse la distribución de estudiantes por número de materias y provincia.

```
> barplot(l, xlab = "Número de materias", ylab = "Cantidad",
+ main = "Distribución de estudiantes según número de materias
+ matriculadas por provincia de nacimiento", las = 1, cex.main = 1,
+ sub = "Fuente: Elaboración propia", beside = T, xlim = c(0, 25))
> legend("topright", legend = c("Alajuela", "Heredia", "San José"),
+ bty = "n", cex = 0.7, fill = c("cadetblue4", "cadetblue2", "lightblue"))
> abline(h = 0)
```



Gráfica 5.12: Gráfica de barras comparativas

Ejemplo 5.21 Modifique la Gráfica 5.12 asignando los colores gray4, gray2 y gray.

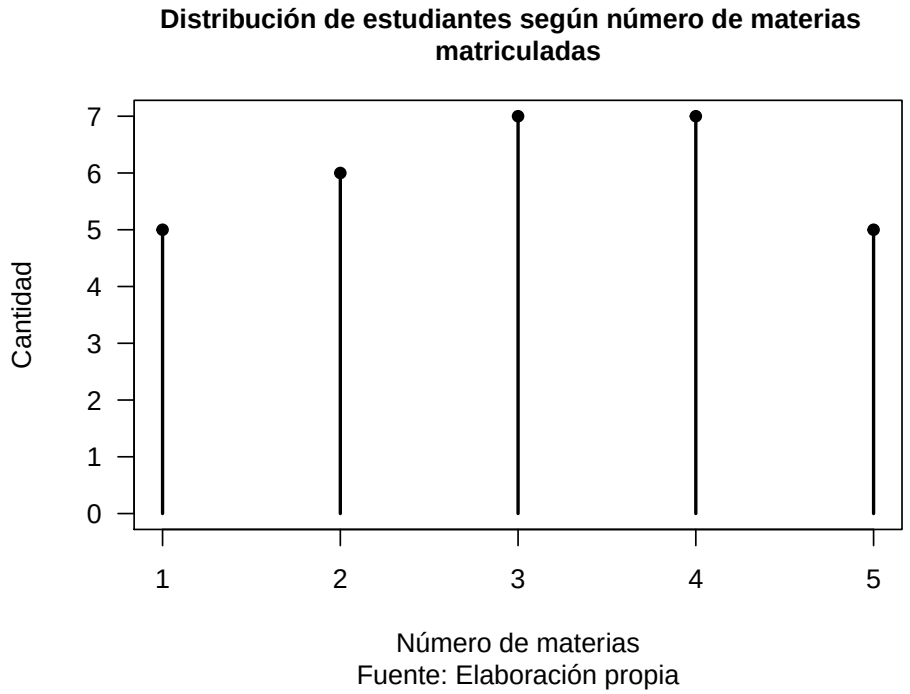
```
> barplot(l, xlab = "Número de materias", ylab = "Cantidad",
+ main = "Distribución de estudiantes según número de materias
+ matriculadas por provincia de nacimiento", las = 1, cex.main = 1,
+ sub = "Fuente: Elaboración propia", beside = T, ylim = c(0,5),
+ col = c("gray4", "gray2", "gray"))
> legend("topright", legend = c("Alajuela", "Heredia", "San José"),
+ bty = "n", fill = c("gray4", "gray2", "gray"),
+ cex = 0.7)
> abline(h = 0)
```

5.4.7. Gráfica de bastones

Para construir la gráfica de bastones de una variable discreta se utiliza la función `plot()` con el argumento `type =` "`h`".

Ejemplo 5.22 Represente la distribución de estudiantes según número de materias matriculadas de la hoja `base` por medio de una gráfica de bastones.

```
> plot(table(base$materias), type = "h", fin = 2,
+ main = "Distribución de estudiantes según número de materias
+ matriculadas", las = 1, cex.main = 1,
+ sub = "Fuente: Elaboración propia",
+ xlab = "Número de materias", ylab = "Cantidad")
> points(c(1, 2, 3, 4, 5), c(5, 6, 7, 7, 5),  pch = 21,
+ bg = "black", cex = 0.9)
```



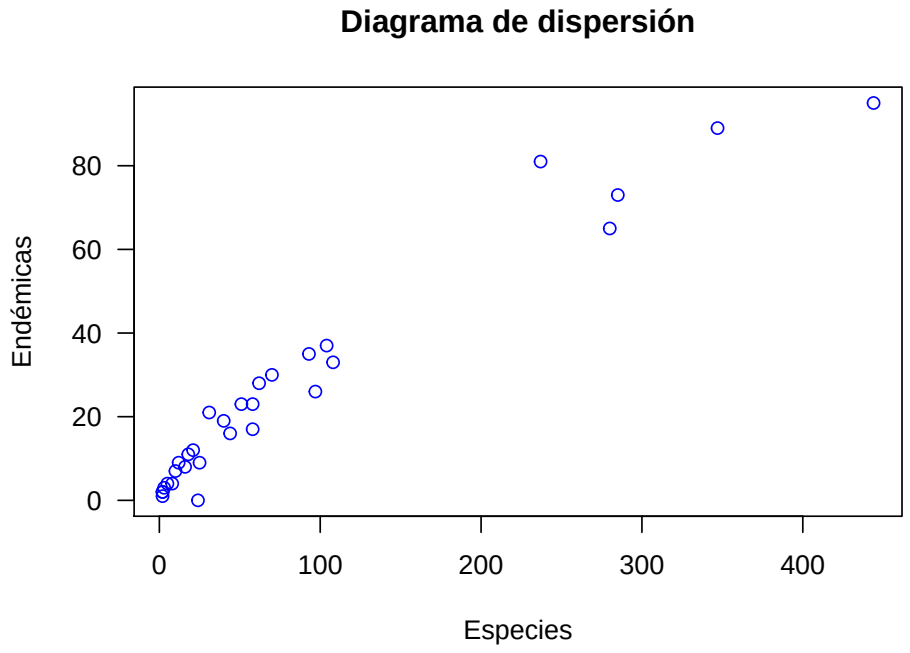
Gráfica 5.13: Gráfica de bastones

5.4.8. Diagramas de dispersión

Los diagramas de dispersión permiten determinar la existencia de relaciones entre dos variables numéricas. Pueden construirse con la función `plot()`.

Ejemplo 5.23 Considere la hoja de datos `gala` y muestre el comportamiento de las variables `Species` y `Endemics` en una gráfica cuyo título sea `Diagrama de dispersión`, definiendo el eje X como `Especies`, el Y como `Endémicas` y utilizando puntos.

```
> library(faraway)
> plot(gala$Species, gala$Endemics, main = "Diagrama de dispersión",
+ xlab = "Especies", ylab = "Endémicas", las = 1, col = "blue")
```



Gráfica 5.14: Gráfica de dispersión

Es posible modificar el tipo, color y tamaño del símbolo con los argumentos `pch`, `col` y `cex`, respectivamente.

Ejemplo 5.24 Represente la información de la Gráfica 5.14 utilizando puntos más pequeños y de color rojo.

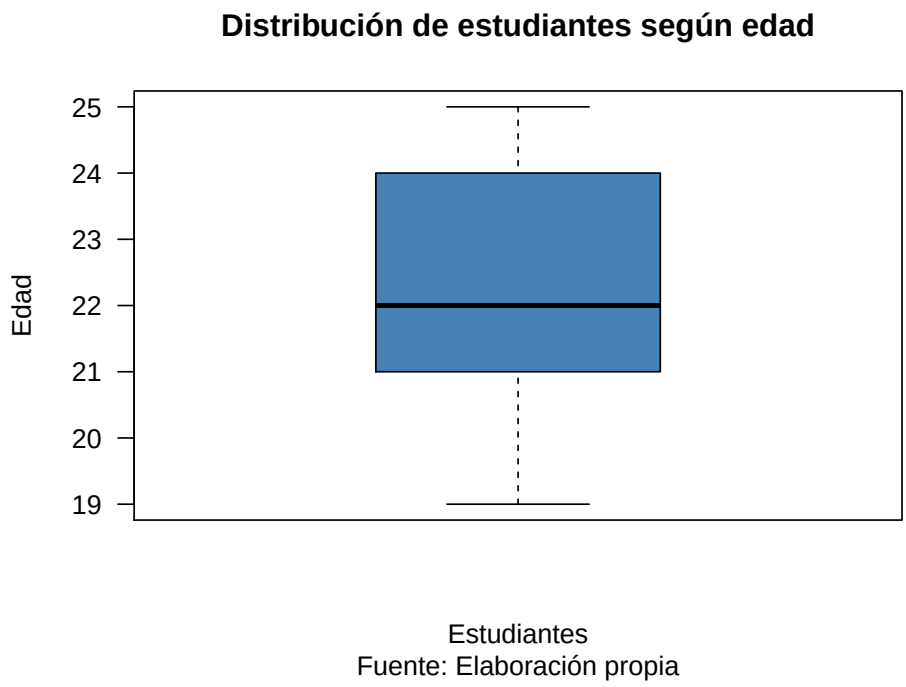
```
> plot(gala$Species, gala$Endemics, main = "Diagrama de dispersión",
+ xlab = "Especies", ylab = "Endémicas", pch = 16, col = "red",
+ cex = 0.9, las = 1)
```

5.4.9. Diagramas de cajas

El diagrama de cajas es muy útil para observar la distribución de una variable cuantitativa. Esta gráfica muestra el rango intercuartílico y da la posibilidad de detectar la presencia de valores extremos. Para su construcción se emplea la función `boxplot()`. Algunos de los argumentos que pueden utilizarse son, entre otros, `main, sub, xlab, ylab, col, border, horiz`.

Ejemplo 5.25 Obtenga el diagrama de cajas de la variable `Edad` de la hoja de datos `base` cuyo título sea `Distribución de estudiantes según edad`, llamando el eje X como `Estudiantes` y el eje Y como `Edad`, considerando algún tono de `azul` para el color de la caja, borde color `negro` y subtítulo `Fuente: Elaboración propia`.

```
> boxplot(base$edad1, xlab = "Estudiantes", ylab = "Edad",
+ main = "Distribución de estudiantes según edad", col = "steelblue",
+ border = "black", sub = "Fuente: Elaboración propia", las = 1)
```



Gráfica 5.15: Diagrama de cajas para una variable

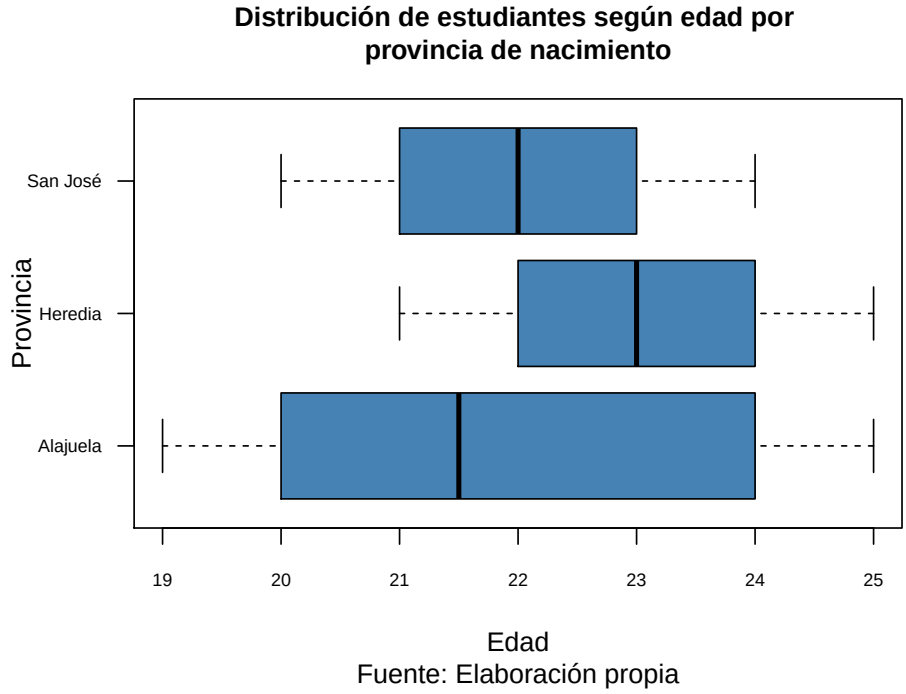
El argumento `horiz = T` permite dibujar las cajas en posición horizontal.

Ejercicio 5.9 Dibuje el diagrama anterior considerando la caja en posición horizontal, modificando el nombre de los ejes según corresponda.

Es importante recordar que el diagrama de cajas puede construirse para mostrar la distribución de una variable o para comparar por categorías.

Ejemplo 5.26 Compare la distribución de la edad de los estudiantes por provincia de nacimiento de la hoja de datos `base` cuyo título sea `Distribución de estudiantes según edad por provincia de nacimiento`, rotulando el eje Y como `Provincia`, el eje X como `Edad`, considerando algún tono de azul para el color de las cajas y subtítulo `Fuente: Elaboración propia`.

```
> boxplot(base$edad1~base$provincia, ylab = "Provincia",
+ main = "Distribución de estudiantes según edad por
+ provincia de nacimiento", xlab = "Edad", horizontal = T,
+ cex.axis = 0.70, sub = "Fuente: Elaboración propia",
+ las = 1, cex.main = 1, col = "steelblue")
```



Gráfica 5.16: Diagrama de cajas para la distribución de la edad por provincia

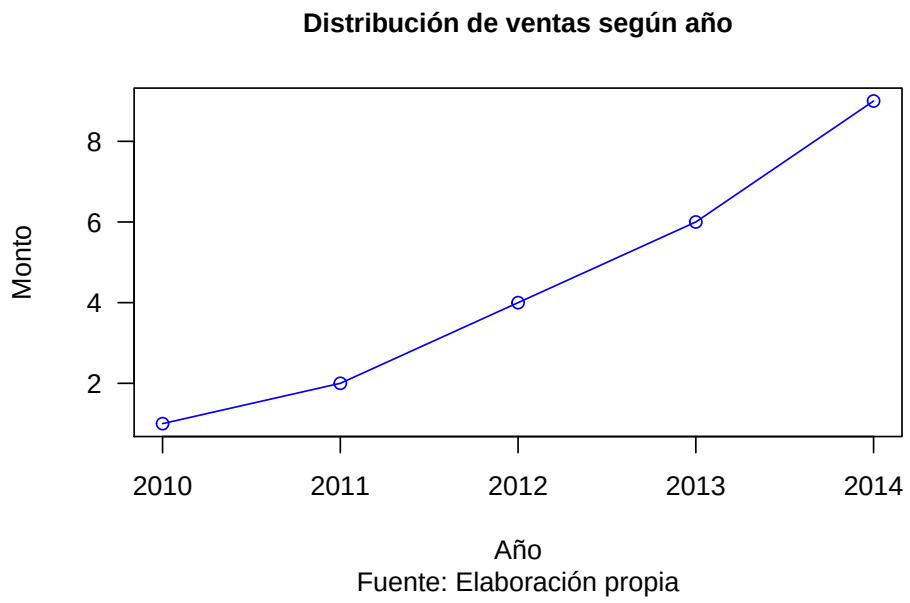
Ejercicio 5.10 Construya un diagrama de cajas que ayude a comparar la distribución del número de materias matriculadas por provincia de nacimiento, considerando todos aquellos elementos de graficación que permitan su adecuada presentación.

5.4.10. Gráficas de líneas

Las gráficas lineales pueden construirse por medio de la función `plot()` con el argumento `type =` "`l`". También puede considerarse `type =` "`o`" que dibuja puntos y líneas superpuestas.

Ejemplo 5.27 Suponga que las ventas de 5 años en millones de colones de una empresa determinada son de {1, 2, 4, 6, 9 }. Construya una gráfica de líneas que muestre dicha distribución cuyo título diga `Distribución de ventas por año`, los rótulos en el eje X y en el eje Y sean `Año` y `Monto`, respectivamente, las etiquetas del eje X sean `2010, 2011, 2012, 2013, 2014` y la línea sea de color `azul`.

```
> ventas=c(1, 2, 4, 6, 9)
> periodo=c(2010, 2011, 2012, 2013, 2014)
> plot(periodo, ventas, type = "o", xlab = "Año", ylab = "Monto",
+ main = "Distribución de ventas según año", las = 1, cex.main = 1,
+ sub = "Fuente: Elaboración propia", col = "blue")
```



Gráfica 5.17: Gráfica de líneas

Como se explicó en la Sección 5.3.3 puede incluirse texto en una gráfica previamente establecida.

Ejemplo 5.28 Considerando la Gráfica 5.17, escriba en la coordenada `(2011, 8)` la expresión `ventas en millones de ₡`.

```
> plot(periodo, ventas, type = "o", xlab = "Año", ylab = "Monto",
+ font.main = 6, main = "Distribución de ventas según año",
+ las = 1, cex.main = 1)
> text(2011, 8, labels = "ventas en millones de ¢ ", font = 6)
```

5.4.11. Gráficas circulares

Para construir las gráficas circulares se utiliza la función `pie()`. Entre sus argumentos están:

`x:` el cual constituye el vector de datos que desean mostrarse y representan las áreas de la gráfica.

`col:` vector de colores para asignar a cada sección de la gráfica.

`radius:` la región circular se dibuja centrada en una caja cuyos lados varían entre -1 y 1.

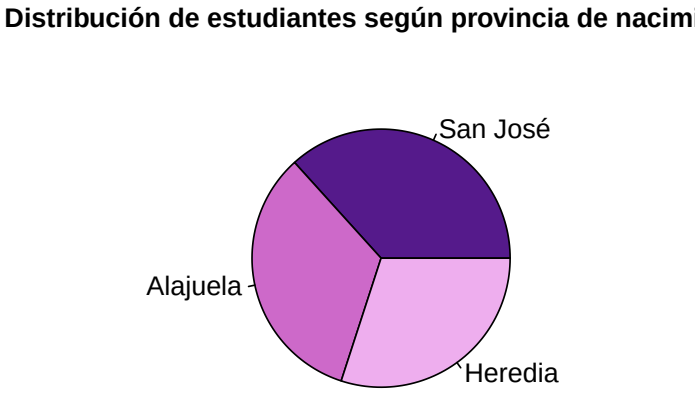
`clockwise:` valor lógico, si toma el valor `TRUE` permite la orientación de la gráfica considerando el sentido de las agujas del reloj.

`labels:` para mostrar las etiquetas de cada sección de la región circular.

También pueden considerarse `density`, `main`, `border`, `lty`, entre otros, cuya aplicación es similar a la de otras gráficas desarrolladas anteriormente.

Ejemplo 5.29 Considere la hoja de datos `base` y represente la distribución de estudiantes según la variable `provincia` cuyo título exprese `Distribución de estudiantes según provincia de nacimiento`.

```
> prov <- table(base$provincia)
> porcentaje <- sort(c(round(prov / sum(prov) * 100, 1)), decreasing = T)
> pie(porcentaje, col = c("purple4", "orchid3", "plum2"),
+ main = "Distribución de estudiantes según provincia de nacimiento",
+ sub = "Fuente: Elaboración propia", cex.main = 1)
```



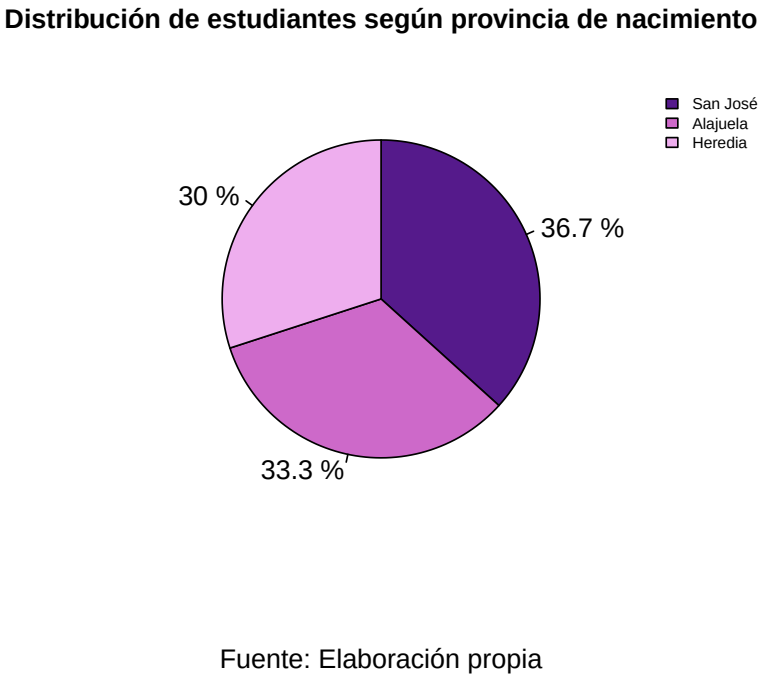
Gráfica 5.18: Gráfica circular

Ejercicio 5.11 Considere la distribución de la Gráfica 5.18 y asigne tonos de azul.

Para modificar las etiquetas de las categorías puede utilizarse la función `legend()`. Al igual que en otras gráficas, si desea modificarse el color del borde puede usarse el argumento `border`.

Ejemplo 5.30 En la Gráfica 5.18, haga que su orientación siga el sentido de las manecillas del reloj y cada sección circular muestre el valor porcentual que representa e incorpore la leyenda con ayuda de la función `legend()` (para que aparezca el símbolo de porcentajes en las etiquetas de la gráfica, puede agruparse el vector de valores con el símbolo % por medio de la función `paste()`).

```
> pie(porcentaje, col = c("purple4", "orchid3", "plum2"), cex.main = 1,
+ main = "Distribución de estudiantes según provincia de nacimiento",
+ sub = "Fuente: Elaboración propia",
+ clockwise = T, labels = paste(porcentaje, "%"))
> legend("topright", c("San José", "Alajuela", "Heredia"), bty = "n",
+ fill = c("purple4", "orchid3", "plum2"), horiz = F, cex = 0.6)
```



Gráfica 5.19: Gráfica circular modificando orientación de categorías

5.4.12. Gráfica de mosaico

La información contenida en una tabla de contingencias puede representarse en una gráfica de mosaico por medio de la función `mosaicplot()`. Entre sus principales argumentos se tienen:

`x:` especifica la tabla de contingencias.

`sort:` genera un ordenamiento vectorial de las variables.

`off:` vector de compensaciones para determinar el espaciado porcentual en cada nivel del mosaico.

`dir:` vector de direcciones divididas para cada nivel del mosaico.

`color:` valor lógico o vector de colores para utilizar en el sombreado. Por defecto las cajas del mosaico son pintadas de gris.

`cex.axis:` factor de expansión para las anotaciones de los ejes.

`formula:` expresión de la forma $y \sim x$.

`data:` un data.frame o lista o tabla de contingencias de la cual las variables van a ser tomadas.

Otros argumentos que pueden emplearse son `main, sub, xlab, ylab, las, border`.

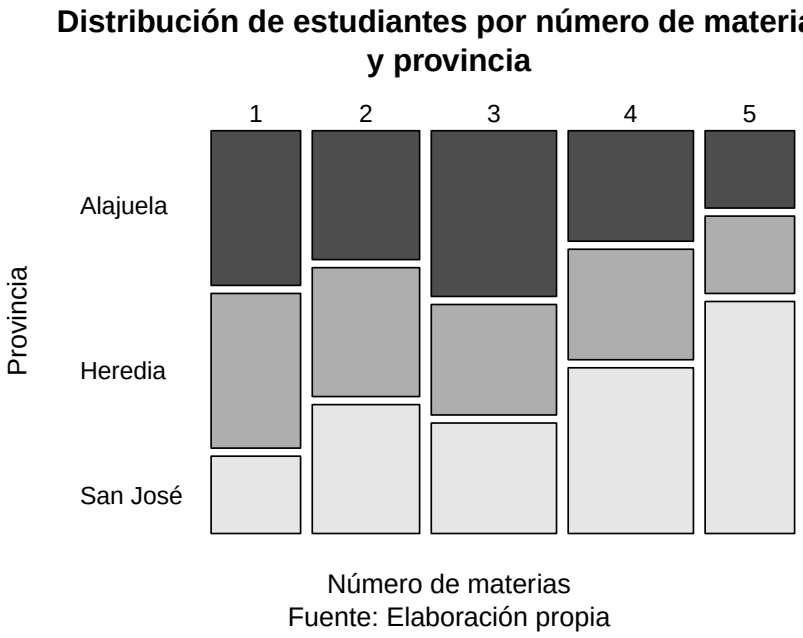
Ejemplo 5.31 Considere las variables `cantidad de materias` y `provincia de nacimiento` de la hoja de datos `base` y represente dicha distribución mediante una gráfica de mosaicos.

```
> mosaicplot(table(base$materias, base$provincia),
+ main = "Distribución de estudiantes por número de materias
+ y provincia", xlab = "Número de materias", ylab = "Provincia",
+ sub = "Fuente: Elaboración propia")
```

La instrucción del Ejemplo 5.31 también puede escribirse utilizando el argumento `formula`. Además, por defecto las cajas de la gráfica son pintadas de color gris; sin embargo, al indicar `color = T`, dichas cajas se colorean con tonos de grises.

Ejemplo 5.32 Utilice el argumento `formula` para representar gráficamente la información del Ejemplo 5.31.

```
> mosaicplot(~ materias + provincia, data = base, las = 1,
+ main = "Distribución de estudiantes por número de materias
+ y provincia", xlab = "Número de materias", ylab = "Provincia",
+ col = T, cex.axis = 0.9, sub = "Fuente: Elaboración propia")
```



Gráfica 5.20: Gráfica de mosaico utilizando el argumento fórmula

Ejemplo 5.33 Considere la información del Ejemplo 5.31 y represéntela mediante una gráfica de mosaico que incluya tonos de azul.

```
> mosaicplot(~ materias + provincia, data = base, las = 1,
+ main = "Distribución de estudiantes por número de materias
+ y provincia", xlab = "Número de materias", ylab = "Provincia",
+ col = c("cadetblue4", "cadetblue2", "lightblue"), cex.axis = 0.9,
+ sub = "Fuente: Elaboración propia")
```

Ejercicio 5.12 Considere la Gráfica 5.20 y asigne los colores que desee.

5.5. Funciones para asignar colores

Algunas funciones crean paletas de colores que permiten modificar la apariencia de una gráfica ya establecida. Estas funciones son `rainbow()`, `terrain.colors()`, `heat.colors()`, `topo.colors()` y `cm.colors()`. El principal argumento de estas funciones es `n` que se refiere a la cantidad de colores que se emplearán.

Ejemplo 5.34 Considere la función `rainbow()` y asigne colores a la Gráfica 5.19.

```
> pie(porcentaje, main = "Distribución de estudiantes según provincia de
+ nacimiento", col = topo.colors(3), radius = 0.9, clockwise = T,
+ labels = paste(porcentaje, "%"))
> legend("topright", c("San José", "Alajuela", "Heredia"),
+ fill = rainbow(3), horiz = F, cex=0.6, bty = "n")
```

Ejemplo 5.35 Considere la información de la Gráfica 5.19 y asigne colores por medio de la función `terrain.colors()`.

```
> pie(porcentaje, main = "Distribución de estudiantes según provincia
+ de nacimiento", col = terrain.colors(3), radius = 0.9, clockwise = T,
+ labels = paste(porcentaje, "%"), cex.main = 1)
> legend("topright", c("San José", "Alajuela", "Heredia"),
+ fill = terrain.colors(3), horiz = F, cex=0.6)
```

Ejercicio 5.13 Asigne colores a cada Gráfica con ayuda de la función indicada.

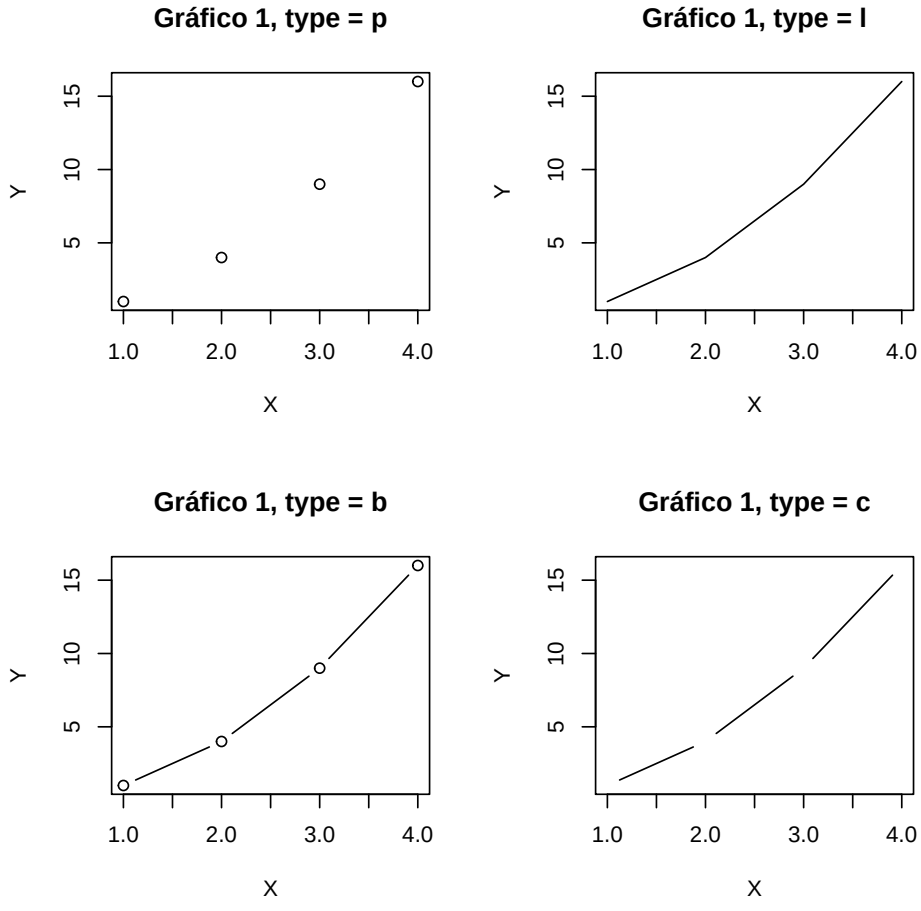
1. Considere la Gráfica 5.11 y asigne colores con ayuda de la función `rainbow()`.
2. Considere la Gráfica 5.19 y asigne colores por medio de la función `topo.colors()`.
3. Considere la Gráfica 5.19 y asigne colores por medio de la función `cm.colors()`.
4. Considere la Gráfica 5.20 y asigne colores usando la función `terrain.colors()`.

5.6. Mostrando varias gráficas en una misma ventana

Observe que al digitar `par`(“`mfrow`”) se tiene que la ventana gráfica está dividida en una matriz de `1 x 1`, esto indica que R solo muestra una gráfica a la vez. Sin embargo, algunas veces es necesario o resulta útil mostrar varias gráficas en una misma ventana. Esta acción es permitida en R simplemente modificando el valor del parámetro `mfrow` o `mfcol`, ya sea si se quiere que la distribución se realice por filas o columnas.

Ejemplo 5.36 Muestre en la misma página la Gráfica 5.3 utilizando 4 valores del argumento `type`, de modo que las figuras sean distribuidas por fila.

```
> par(mfrow = c(2,2))
> plot(c(1,2,3,4), c(1,4,9,16), type = "p", main = "Gráfico 1, type = p",
+ xlab = "X", ylab = "Y")
> plot(c(1,2,3,4), c(1,4,9,16), type = "l", main = "Gráfico 1, type = l",
+ xlab = "X", ylab = "Y")
> plot(c(1,2,3,4), c(1,4,9,16), type = "b", main = "Gráfico 1, type = b",
+ xlab = "X", ylab = "Y")
> plot(c(1,2,3,4), c(1,4,9,16), type = "c", main = "Gráfico 1, type = c",
+ xlab = "X", ylab = "Y")
```



Gráfica 5.21: Gráficas que pueden elaborarse con la función `plot()`

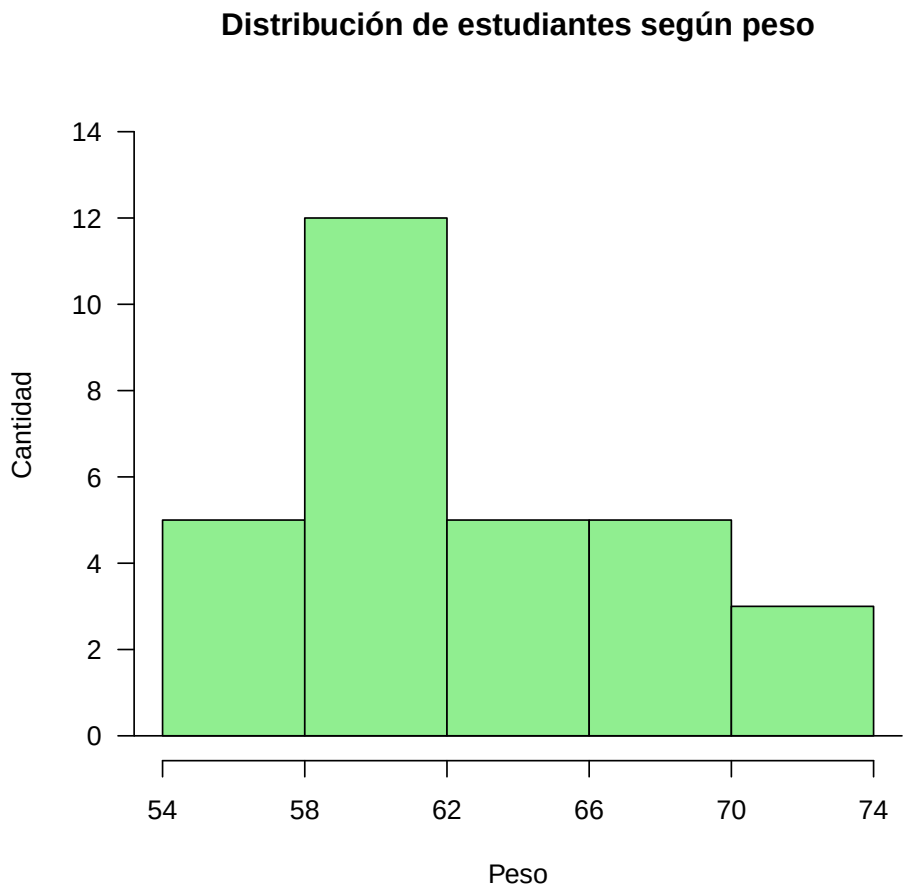
5.7. Graficación de distribuciones de frecuencias de variables continuas

Para representar gráficamente distribuciones de variables continuas puede utilizarse la función `plot()` y los argumentos `fh`, `rfh`, `fp`, `rfp`, `cfp`, `cfpp`, entre otros.

El argumento "`fh`" permite construir el histograma representando las frecuencias absolutas, mientras que "`rfh`" muestra el histograma de frecuencias relativas.

Ejemplo 5.37 Considere la distribución de la variable peso del Ejemplo 4.13 y construya un histograma que muestre las frecuencias absolutas.

```
> plot(dist, type = "fh", main = "Distribución de estudiantes según peso",
+ col = "palegreen2", las = 1, xlab = "Peso", ylab = "Cantidad")
> abline(h = 0)
```



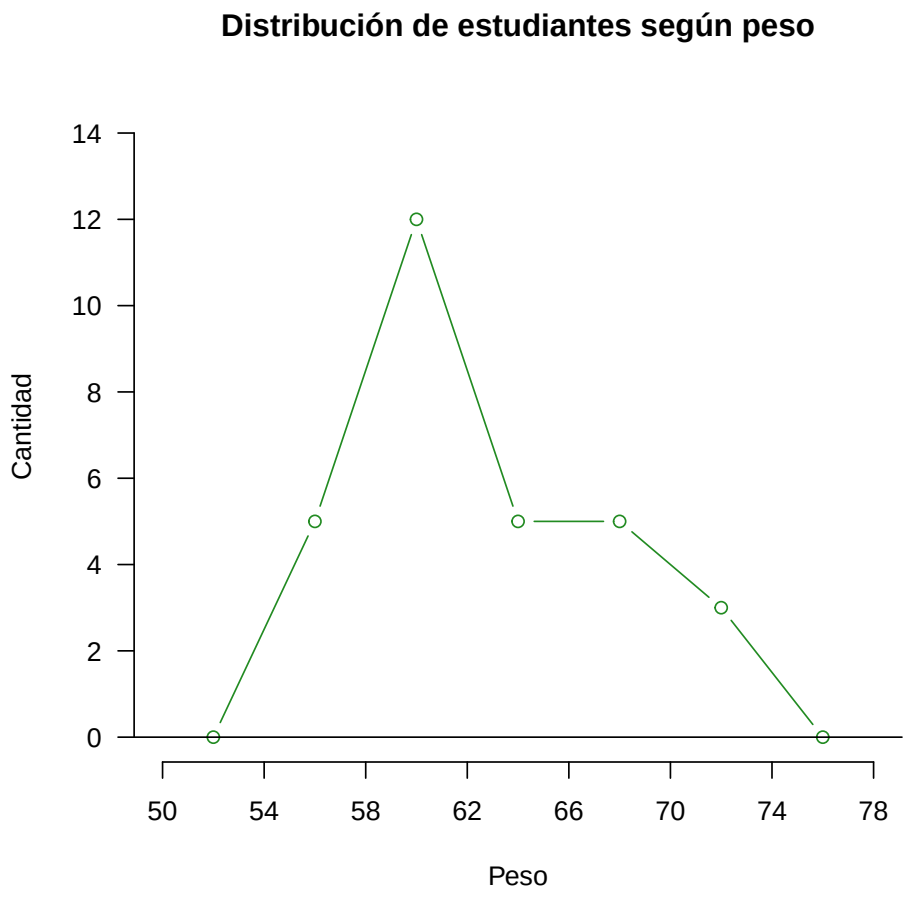
Gráfica 5.22: Histograma para la variable peso

Ejercicio 5.14 Considere la distribución de la variable peso del Ejemplo 4.13 y construya un histograma que muestre las frecuencias relativas simples.

El argumento "`fp`" permite construir el polígono de frecuencias representando las frecuencias absolutas simples, mientras que "`rfp`" muestra las frecuencias relativas.

Ejemplo 5.38 Considere la distribución de la variable peso del Ejemplo 4.13 y construya un polígono que muestre las frecuencias absolutas.

```
> dist1 <- fdt(base$peso, start = 50, end = 78, h = 4)
> plot(dist1, type = "fp", las = 1, col = "forestgreen",
+ main = "Distribución de estudiantes según peso",
+ xlab = "Peso", ylab = "Cantidad")
> abline(h = 0)
```



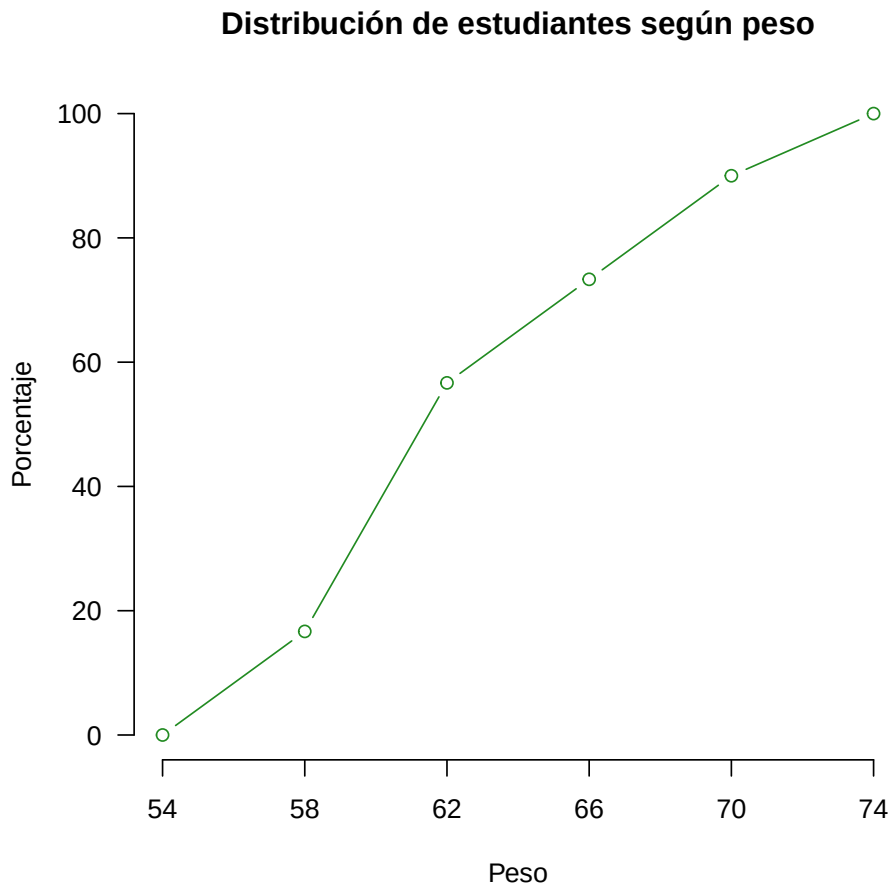
Gráfica 5.23: Polígono de frecuencias para la variable peso

Ejercicio 5.15 Considere la distribución de la variable peso del Ejemplo 4.13 y construya un polígono que muestre las frecuencias relativas simples.

Por otro lado, el argumento "`cfp`" permite construir el polígono de frecuencias acumuladas "menos de" (ojivas) representando las cantidades absolutas, mientras que "`cfpp`" muestra los porcentajes acumulados.

Ejemplo 5.39 Considere la distribución de la variable peso del Ejemplo 4.13 y construya un polígono (ojiva) que muestre los porcentajes acumulados.

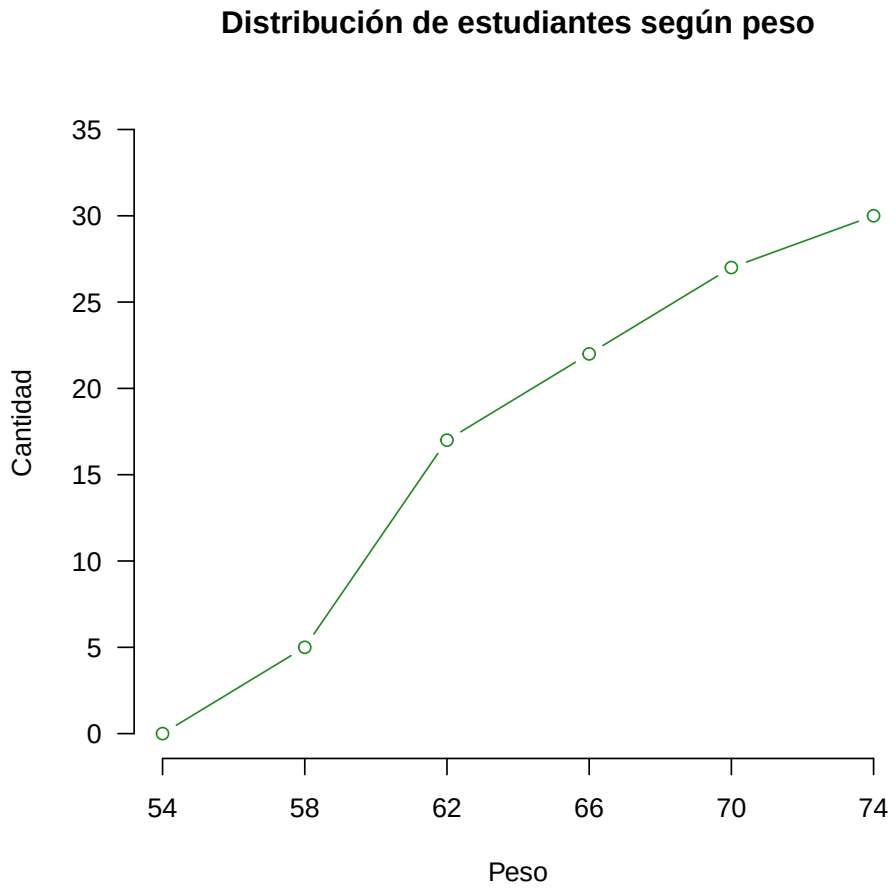
```
> plot(dist, type = "cfpp", las = 1, col = "forestgreen",
+ main = "Distribución de estudiantes según peso",
+ xlab = "Peso", ylab = "Porcentaje", ylim = c(0,100))
```



Gráfica 5.24: Polígono de porcentajes acumulados para la variable peso

Ejemplo 5.40 Considere la distribución de la variable peso del Ejemplo 4.13 y construya un polígono que muestre las frecuencias absolutas acumuladas.

```
> plot(dist, type = "cfp", las = 1, col = "forestgreen",
+ main = "Distribución de estudiantes según peso", col = "blue",
+ xlab = "Peso", ylab = "Cantidad")
```



Gráfica 5.25: Polígono de frecuencias acumulados para la variable peso

Referencias

Contreras, J. M., Molina, E. y Arteaga, P. (2010). *Introducción a la programación estadística con r para profesores*. Descargado de https://www.ugr.es/~batanero/pages/ARTICULOS/libroR.pdf

Faraway, J. (2016). faraway: Functions and datasets for books by julian faraway. Descargado de https://CRAN.R-project.org/package=faraway (R package version 1.0.7).

Faria, J. C., Jelihovschi, E. G. y Allaman, I. B. (2017). Frequency distribution tables, histograms and polygons. Ilheus, Bahia, Brasil.

Gómez, M. (2003). *Elementos de estadística descriptiva*. San José, Costa Rica. EUNED.

Poncet, P. (2012). modeest: Mode estimation. Descargado de https://CRAN.R-project.org/package=modeest (R package version 2.1).

R Core Team. (2017). R: A language and environment for statistical computing. Vienna, Austria. Descargado de https://www.R-project.org/

RStudio Team. (2015). RStudio: Integrated Development Environment for R. Boston, Ma. Descargado de http://www.rstudio.com/

Santana, J. y Mateos, E. (2014). *El arte de programar en r: un lenguaje para la estadística* Descargado de https://cran.r-project.org/doc/contrib/Santana_El_arte_de_programar_en_R.pdf

Venables, W. N. y Ripley, B. D. (2002). *Modern applied statistics with s*. Springer. Descargado de http://www.stats.ox.ac.uk/pub/MASS4

Verzani, J. (2014). *Using r for introductory statistics*. CRC Press.

Wickham, H. y Bryan, J. (2018). readxl: Read excel files. Descargado de https://CRAN.R-project.org/package=readxl (R package version 1.2.0).

Wickham, H., Hester, J. y Francois, R. (2017). readr: Read rectangular text data. Descargado de https://CRAN.R-project.org/package=readr (R package version 1.1.1).

Wickham, H. y Miller, E. (2019). haven: Import and export “spss”, “stata” and “sas” files. Descargado de https://CRAN.R-project.org/package=haven (R package version 2.0.0).