

Freddy Hernández Barajas
Olga Cecilia Usuga Manco

Manual de R

Gracias a Dios por todo lo que me ha dado.

Índice general

Índice de cuadros	v
Índice de figuras	vii
Prefacio	ix
Sobre los autores	xiii
1. Introducción	1
1.1. Orígenes	1
1.2. Descarga e instalación	2
1.3. Apariencia del programa	3
1.4. Tipos de objetos	4
1.4.1. Vectores	5
1.4.2. Matrices	6
1.4.3. Arreglos	6
1.4.4. Marco de datos	7
1.4.5. Listas	8
1.5. Guía de estilo para la escritura en R	9
1.5.1. Nombres de los archivos	9
1.5.2. Nombres de los objetos	9
1.5.3. Longitud de una línea de código	10
1.5.4. Espacios	10
1.5.5. Asignación	12
1.5.6. Punto y coma	12
2. Funciones básicas de R	15
2.1. ¿Qué es una función R?	15
2.2. Operadores de asignación	16
2.3. Operaciones básicas	17
2.4. Pruebas lógicas	18
2.5. Operadores lógicos	20
2.6. Funciones sobre vectores	21
2.7. Funciones matemáticas	23
2.8. Función <code>seq</code>	24
2.9. Función <code>rep</code>	26
2.10. Funciones <code>round</code> , <code>ceiling</code> , <code>floor</code> y <code>trunc</code>	27

2.11. Funciones <code>sort</code> y <code>rank</code>	29
3. Lectura de bases de datos	33
3.1. ¿En qué formato almacenar una base de datos?	33
3.1.1. Almacenamiento de información en Excel	33
3.1.2. Almacenamiento de información en bloc de notas	34
3.2. Función <code>read.table</code>	35
4. Medidas de tendencia central	39
4.1. Media	40
4.2. Mediana	41
4.3. Moda	42
5. Medidas de posición	45
5.1. Cuantiles	46
6. Medidas de variabilidad	47
6.1. Rango	48
6.2. Desviación estándar muestral (S)	49
6.3. Varianza muestral (S^2)	51
6.4. Coeficiente de variación (CV)	53
7. Medidas de correlación	55
8. Creación de funciones en R	57
9. Distribuciones discretas	59
10. Distribuciones continuas	61
11. Pruebas de bondad de ajuste	63
12. Aproximación de integrales	65
12.1. Aproximación de Laplace unidimensional	65
Bibliografía	69
Índice alfabético	71

Índice de cuadros

3.1. Ejemplo de una base de datos simple.	33
---	----



Índice de figuras

1.1. Robert Gentleman (izquierda) y Ross Ihaka (derecha) creadores de R.	2
1.2. Página del Cran.	2
1.3. Página de instalación para la primera ocasión.	3
1.4. Página de descarga.	3
1.5. Apariencia del acceso directo para ingresar a R.	4
1.6. Apariencia de R.	4
2.1. Ilustración de una función, tomada de www.mathinsight.org	15
3.1. Forma de almacenar los datos en Excel.	34
3.2. Almacenamiento de los datos en bloc de notas usando la barra espaciadora	35
3.3. Almacenamiento de los datos en bloc de notas usando la barra tabuladora	35
6.1. Boxplot para el precio de los apartamentos dada la ubicación.	52
12.1. Perfil de la función $f(x)$	66



Prefacio

Este libro fue creado con la intención de apoyar el aprendizaje del programa R en estudiantes de pregrado, especialización, maestría e investigadores que necesiten realizar análisis estadísticos. En este libro se explica de una forma sencilla la utilidad de la principales funciones para realizar diversos análisis estadísticos, las cuestiones sobre creación de gráficos estadísticos no son abordadas en el presente libro, recomendamos consultar [Correa \(2018\)](#).

¿Por qué leer este libro?

Este libro es importante porque ...

Estructura del libro

El libro está estructurado de la siguiente manera.

En el capítulo [4](#) se muestra como obtener las diversas medidas de tendencial central para variables cuantitativas, el capítulo [6](#) muestra como calcular las medidas de variabilidad, en el capítulo [5](#) se ilustra cómo usar las funciones para obtener medidas de posición y en el capítulo [7](#) se muestra como obtener medidas de correlación entre pares de variables.

Información del software y convenciones

Para realizar este libro usamos los paquetes **knitr** ([Xie, 2015](#)) y **bookdown** ([Xie, 2016](#)).

Package names are in bold text (e.g., **rmarkdown**), and inline code and filenames are formatted in a typewriter font (e.g., `knitr::knit('foo.Rmd')`). Function names are followed by parentheses (e.g., `bookdown::render_book()`).

En todo el libro se presentarán códigos que el lector puede copiar y pegar en su consola de R para obtener los mismos resultados aquí presentados. Los códigos se destacan en una caja de color beis (o beige) similar a la mostrada a continuación.

```
4 + 6
a <- c(1, 5, 6)
5 * a
1:10
```

Los resultados o salidas obtenidos de cualquier código se destacan con dos símbolos de numeral (##) al inicio de cada línea o renglón, esto quiere decir que todo lo que inicie con ## son resultados obtenidos y **NO** los debe copiar. Abajo se muestran los resultados obtenidos luego de correr el código anterior.

```
## [1] 10
## [1]  5 25 30
## [1]  1  2  3  4  5  6  7  8  9 10
```

Bloques informativos

En varias partes del libro usaremos bloques informativos para resaltar algún aspecto importante. Abajo se encuentra un ejemplo de los bloques y su significado.



Este bloque sirve para una nota aclaratoria.



Este bloque sirve para una sugerencia.



Este bloque sirve para una advertencia.

Agradecimientos

Agradecemos enormemente a todos los estudiantes, profesores e investigadores que han leído este libro y nos han retroalimentado con comentarios valiosos para mejorar el documento.

Freddy Hernández Barajas

Olga Cecilia Usuga Manco



Sobre los autores

Freddy Hernández Barajas es profesor asistente de la Universidad Nacional de Colombia adscrito a la Escuela de Estadística de la Facultad de Ciencias.

Olga Cecilia Usuga Manco es profesora asociada de la Universidad de Antioquia adscrita al Departamento de Ingeniería Industrial de la Facultad de Ingeniería.



1

Introducción

1.1. Orígenes

R es un lenguaje de programación usado para realizar procedimientos estadísticos y gráficos de alto nivel, este lenguaje fue creado en 1993 por los profesores e investigadores Robert Gentleman y Ross Ihaka. Inicialmente el lenguaje se usó para apoyar los cursos que tenían a su cargo los profesores, pero luego de ver la utilidad de la herramienta desarrollada, decidieron colocar copias de R en StatLib. A partir de 1995 el código fuente de R está disponible bajo licencia GNU GPL para sistemas operativos Windows, Macintosh y distribuciones Unix/Linux. La comunidad de usuarios de R en el mundo es muy grande y los usuarios cuentan con diferentes espacios para interactuar, a continuación una lista no exhaustiva de los sitios más populares relacionados con R:

- Rbloggers¹.
- Comunidad hispana de R².
- Nabble³.
- Foro en portugués⁴.
- Stackoverflow⁵.
- Cross Validated⁶.
- R-Help Mailing List⁷.
- Revolutions⁸.
- R-statistics blog⁹.
- RDataMining¹⁰.

¹<https://www.r-bloggers.com/>

²<http://r-es.org/>

³<http://r.789695.n4.nabble.com/>

⁴<http://r-br.2285057.n4.nabble.com/>

⁵<http://stackoverflow.com/questions/tagged/r>

⁶<http://stats.stackexchange.com/questions/tagged/r>

⁷<https://stat.ethz.ch/mailman/listinfo/r-help>

⁸<http://blog.revolutionanalytics.com/>

⁹<https://www.r-statistics.com/>

¹⁰<https://rdatamining.wordpress.com/>



Figura 1.1: Robert Gentleman (izquierda) y Ross Ihaka (derecha) creadores de R.

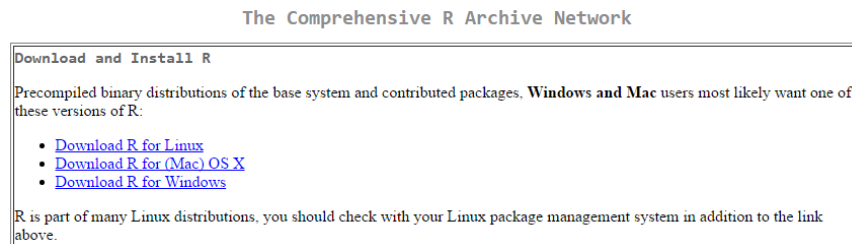


Figura 1.2: Página del Cran.

1.2. Descarga e instalación

Para realizar la instalación de R usted debe visitar la página del CRAN (*Comprehensive R Archive Network*) disponible en este enlace¹¹. Una vez ingrese a la página encontrará un cuadro similar al mostrado en la Figura 1.2 donde aparecen los enlaces de la instalación para los sistemas operativos Linux, Mac y Windows.

Supongamos que se desea instalar R en Windows, para esto se debe dar clic sobre el hipervínculo **Download R for Windows** de la Figura 1.2. Una vez hecho esto se abrirá una página con el contenido mostrado en la Figura 1.3. Una vez ingrese a esa nueva página usted debe dar clic sobre el hipervínculo **install R for the first time** como es señalado por la flecha roja en la Figura 1.3.

Luego de esto se abrirá otra página con un encabezado similar al mostrado en la Figura 1.4, al momento de capturar la figura la versión actual de R era 3.2.5 pero seguramente en este momento usted tendrá disponible una versión actua-

¹¹<https://cran.r-project.org/>

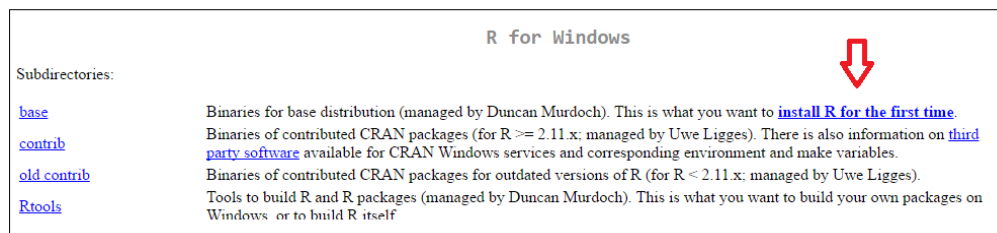


Figura 1.3: Página de instalación para la primera ocasión.



Figura 1.4: Página de descarga.

lizada. Una vez allí usted debe dar clic sobre **Download R 3.2.5 for Windows** como es señalado por la flecha verde. Luego de esto se descargará el instalador R en el computador el cual deberá ser instalado con las opciones que vienen por defecto.

Se recomienda observar el siguiente video didáctico de instalación de R disponible en este enlace¹² para facilitar la tarea de instalación.

1.3. Apariencia del programa

Una vez que esté instalado R en su computador, usted podrá acceder a él por la lista de programas o por medio del acceso directo que quedó en el escritorio, en la Figura 1.5 se muestra la apariencia del acceso directo para ingresar a R.

Al abrir R aparecerá en la pantalla de su computador algo similar a lo que está en la Figura 1.6. La ventana izquierda se llama consola y es donde se ingresan las instrucciones, una vez que se construye un gráfico se activa otra ventana llamada ventana gráfica. Cualquier usuario puede modificar la posición y tamaños de estas ventanas, puede cambiar el tipo y tamaño de las letras en la

¹²<http://tinyurl.com/jd7b9ks>



Figura 1.5: Apariencia del acceso directo para ingresar a R.

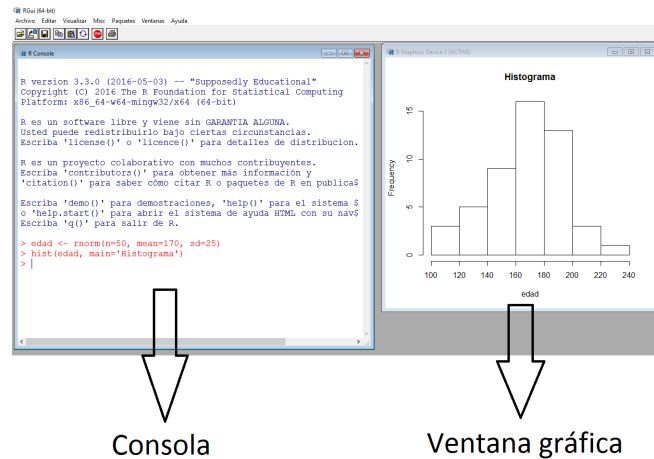


Figura 1.6: Apariencia de R.

consola, para hacer esto se deben explorar las opciones de *editar* en la barra de herramientas.

1.4. Tipos de objetos

En R existen varios tipos de objetos que permiten que el usuario pueda almacenar la información para realizar procedimientos estadísticos y gráficos. Los principales objetos en R son vectores, matrices, arreglos, marcos de datos y listas. A continuación se presentan las características de estos objetos y la forma para crearlos.

1.4.1. Vectores

Los vectores son arreglos ordenados en los cuales se puede almacenar información de tipo numérico (variable cuantitativa), alfanumérico (variable cualitativa) o lógico (TRUE o FALSE), pero no mezclas de éstos. La función de R para crear un vector es `c()` y que significa concatenar; dentro de los paréntesis de esta función se ubica la información a almacenar. Una vez construido el vector se acostumbra a etiquetarlo con un nombre corto y representativo de la información que almacena, la asignación se hace por medio del operador `<-` entre el nombre y el vector.

A continuación se presenta un ejemplo de cómo crear tres vectores que contienen las respuestas de cinco personas a tres preguntas que se les realizaron.

```
edad <- c(15, 19, 13, NA, 20)
deporte <- c(TRUE, TRUE, NA, FALSE, TRUE)
comic.fav <- c(NA, 'Superman', 'Batman', NA, 'Batman')
```

El vector `edad` es un vector cuantitativo y contiene las edades de las 5 personas. En la cuarta posición del vector se colocó el símbolo `NA` que significa *Not Available* debido a que no se registró la edad para esa persona. Al hacer una asignación se acostumbra a dejar un espacio antes y después del operador `<-` de asignación. El segundo vector es llamado `deporte` y es un vector lógico que almacena las respuestas a la pregunta de si la persona practica deporte, nuevamente aquí hay un `NA` para la tercera persona. El último vector `comic.fav` contiene la información del cómic favorito de cada persona, como esta variable es cualitativa es necesario usar las comillas ' ' para encerrar las respuestas.



Cuando se usa `NA` para representar una información *Not Available* no se deben usar comillas.



Es posible usar comillas sencillas `'foo'` o comillas dobles `"foo"` para ingresar valores de una variable cualitativa.

Si se desea ver lo que está almacenado en cada uno de estos vectores, se debe escribir en la consola de R el nombre de uno de los objetos y luego se presiona la tecla *enter* o *intro*, al realizar esto lo que se obtiene se muestra a continuación.

```
edad
## [1] 15 19 13 NA 20
```

```
deporte
```

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

```
comic.fav
```

```
## [1] NA "Superman" "Batman" NA
## [5] "Batman"
```

1.4.2. Matrices

Las matrices son arreglos rectangulares de filas y columnas con información numérica, alfanumérica o lógica. Para construir una matriz se usa la función `matrix()`. Por ejemplo, para crear una matriz de 4 filas y 5 columnas (de dimensión 4×5) con los primeros 20 números positivos se escribe el código siguiente en la consola.

```
mimatriz <- matrix(data=1:20, nrow=4, ncol=5, byrow=FALSE)
```

El argumento `data` de la función sirve para indicar los datos que se van a almacenar en la matriz, los argumentos `nrow` y `ncol` sirven para definir la dimensión de la matriz y por último el argumento `byrow` sirve para indicar si la información contenida en `data` se debe ingresar por filas o no. Para observar lo que quedó almacenado en el objeto `mimatriz` se escribe en la consola el nombre del objeto seguido de la tecla *enter* o *intro*.

```
mimatriz
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14   18
## [3,]    3    7   11   15   19
## [4,]    4    8   12   16   20
```

1.4.3. Arreglos

Un arreglo es una matriz de varias dimensiones con información numérica, alfanumérica o lógica. Para construir un arreglo se usa la función `array()`. Por ejemplo, para crear un arreglo de $3 \times 4 \times 2$ con las primeras 24 letras minúsculas del alfabeto se escribe el siguiente código.

```
miarray <- array(data=letters[1:24], dim=c(3, 4, 2))
```

El argumento `data` de la función sirve para indicar los datos que se van a almacenar en el arreglo y el argumento `dim` sirve para indicar las dimensiones del arreglo. Para observar lo que quedó almacenado en el objeto `miarray` se escribe en la consola lo siguiente.

```
miarray

## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,] "a"  "d"  "g"  "j"
## [2,] "b"  "e"  "h"  "k"
## [3,] "c"  "f"  "i"  "l"
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,] "m"  "p"  "s"  "v"
## [2,] "n"  "q"  "t"  "w"
## [3,] "o"  "r"  "u"  "x"
```

1.4.4. Marco de datos

El marco de datos o *data frame* es uno de los objetos más utilizados porque permite agrupar vectores con información de diferente tipo (numérica, alfanumérica o lógica) en un mismo objeto, la única restricción es que los vectores deben tener la misma longitud. Para crear un marco de datos se usa la función `data.frame()`, como ejemplo vamos a crear un marco de datos con los vectores `edad`, `deporte` y `comic.fav` definidos anteriormente.

```
mimarco <- data.frame(edad, deporte, comic.fav)
```

Una vez creado el objeto `mimarco` podemos ver el objeto escribiendo su nombre en la consola, a continuación se muestra lo que se obtiene.

```
mimarco

##   edad deporte comic.fav
## 1   15     TRUE    <NA>
## 2   19     TRUE  Superman
```

```
## 3    13      NA    Batman
## 4    NA FALSE    <NA>
## 5    20    TRUE    Batman
```

De la salida anterior vemos que el marco de datos tiene 3 variables (columnas) cuyos nombres coinciden con los nombres de los vectores creados anteriormente, los números consecutivos al lado izquierdo son sólo de referencia y permiten identificar la información para cada persona en la base de datos.

1.4.5. Listas

Las listas son otro tipo de objeto muy usado para almacenar objetos de diferente tipo. La instrucción para crear una lista es `list()`. A continuación vamos a crear una lista que contiene tres objetos: un vector con 5 números aleatorios llamado `mivector`, una matriz de dimensión 6×2 con los primeros doce números enteros positivos llamada `matriz2` y el tercer objeto será el marco de datos `mimarco` creado en el apartado anterior. Las instrucciones para crear la lista requerida se muestran a continuación.

```
set.seed(12345)
mivector <- runif(n=5)
matriz2 <- matrix(data=1:12, ncol=6)
milista <- list(E1=mivector, E2=matriz2, E3=mimarco)
```

La función `set.seed` de la línea número 1 sirve para fijar la semilla de tal manera que los números aleatorios generados en la segunda línea con la función `runif` sean siempre los mismos. En la última línea del código anterior se construye la lista, dentro de la función `list` se colocan los tres objetos `mivector`, `matriz2` y `mimarco`. Es posible colocarle un nombre especial a cada uno de los elementos de la lista, en este ejemplo se colocaron los nombres `E1`, `E2` y `E3` para cada uno de los tres elementos. Para observar lo que quedó almacenado en la lista se escribe `milista` en la consola y el resultado se muestra a continuación.

```
milista

## $E1
## [1] 0.7209 0.8758 0.7610 0.8861 0.4565
##
## $E2
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    3    5    7    9   11
## [2,]    2    4    6    8   10   12
##
```

```
## $E3
##   edad deporte comic.fav
## 1   15     TRUE      <NA>
## 2   19     TRUE Superman
## 3   13      NA    Batman
## 4   NA    FALSE      <NA>
## 5   20     TRUE    Batman
```

1.5. Guía de estilo para la escritura en R

Así como en el español existen reglas ortográficas, la escritura de códigos en R también tiene unas reglas que se recomienda seguir para evitar confusiones. Tener una buena guía de estilo es importante para que el código creado por usted sea fácilmente entendido por sus lectores [Wickham \(2015\)](#). No existe una única y mejor guía de estilo para escritura en R, sin embargo aquí vamos a mostrar unas sugerencias basadas en la guía llamada *Google's R style guide*¹³.

1.5.1. Nombres de los archivos

Se sugiere que el nombre usado para nombrar un archivo tenga sentido y que termine con extensión .R. A continuación dos ejemplos de como nombrar mal y bien un archivo.

- Mal: `hola.R`
- Bien: `analisis_icfes.R`

1.5.2. Nombres de los objetos

Se recomienda no usar los símbolos `_` y `-` dentro de los nombres de objetos. Para las variables es preferible usar letras minúsculas y separar las palabras con puntos (`peso.maiz`) o utilizar la notación camello iniciando en minúscula (`pesoMaiz`). Para las funciones se recomienda usar la notación camello iniciando todas la palabras en mayúscula (`PlotRes`). Para los nombres de las constantes se recomienda que inicien con la letra `k` (`kPrecioBus`). A continuación ejemplos de buenas y malas prácticas.

Para variables:

¹³<https://google.github.io/styleguide/Rguide.xml>

- Bien: `avg.clicks`
- Aceptable: `avgClicks`
- Mal: `avg_Clicks`

Para funciones:

- Bien: `CalculateAvgClicks`
- Mal: `calculate_avg_clicks` , `calculateAvgClicks`

1.5.3. Longitud de una línea de código

Se recomienda que cada línea tenga como máximo 80 caracteres. Si una línea es muy larga se debe cortar siempre por una coma.

1.5.4. Espacios

Use espacios alrededor de todos los operadores binarios (`=`, `+`, `-`, `<=`, etc.). Los espacios alrededor del símbolo “`=`” son opcionales cuando se usan para ingresar valores dentro de una función. Así como en español, nunca coloque espacio antes de una coma, pero siempre use espacio luego de una coma. A continuación ejemplos de buenas y malas prácticas.

```
tab <- table(df[df$days < 0, 2]) # Bien
tot <- sum(x[, 1])                # Bien
tot <- sum(x[1, ])               # Bien
tab <- table(df[df$days<0, 2])  # Faltan espacios alrededor '<'
tab <- table(df[df$days < 0,2]) # Falta espacio luego de coma
tab <- table(df[df$days < 0 , 2]) # Sobra espacio antes de coma
tab<- table(df[df$days < 0, 2]) # Falta espacio antes de '<-'
tab<-table(df[df$days < 0, 2]) # Falta espacio alrededor de '<-'
tot <- sum(x[,1])                # Falta espacio luego de coma
tot <- sum(x[1,])                # Falta espacio luego de coma
```

Otra buena práctica es colocar espacio antes de un paréntesis excepto cuando se llama una función.

```
if (debug)      # Correcto
if(debug)       # Funciona pero no se recomienda
colMeans (x)    # Funciona pero no se recomienda
```


Espacios extras pueden ser usados si con esto se mejora la apariencia del código, ver el ejemplo siguiente.

```
plot(x      = x.coord,
     y      = data.mat[, MakeColName(metric, ptilas[1], "roi0pt")],
     ylim = ylim,
     xlab = "dates",
     ylab = metric,
     main = (paste(metric, " for 3 samples ", sep = "")))
```

No coloque espacios alrededor del código que esté dentro de paréntesis () o corchetes [], la única excepción es luego de una coma, ver el ejemplo siguiente.

```
if (condicion)      # Correcto
x[1, ]              # Correcto
if ( condicion )    # Sobran espacios alrededor de condicion
x[1,]               # Se necesita espacio luego de coma
```

Los signos de agrupación llaves { } se utilizan para agrupar bloques de código y se recomienda que nunca una llave abierta { esté sola en una línea; una llave cerrada } si debe ir sola en su propia línea. Se pueden omitir las llaves cuando el bloque de instrucciones esté formado por una sola línea pero esa línea de código NO debe ir en la misma línea de la condición. A continuación dos ejemplos de lo que se recomienda.

```
if (is.null(ylim)) {                                # Correcto
  ylim <- c(0, 0.06)
}

if (is.null(ylim))                                  # Correcto
  ylim <- c(0, 0.06)

if (is.null(ylim)) ylim <- c(0, 0.06)              # Aceptable

if (is.null(ylim))                                  # No se recomienda
{
  ylim <- c(0, 0.06)
}

if (is.null(ylim)) {ylim <- c(0, 0.06)}
# Frente a la llave { no debe ir nada
# la llave de cierre } debe ir sola
```

La sentencia else debe ir siempre entre llaves } {, ver el siguiente ejemplo.

```
if (condition) {  
  one or more lines  
} else {           # Correcto  
  one or more lines  
}  
  
if (condition) {  
  one or more lines  
}  
else {           # Incorrecto  
  one or more lines  
}  
  
if (condition)  
  one line  
else           # Incorrecto  
  one line
```

1.5.5. Asignación

Para realizar asignaciones se recomienda usar el símbolo <-, el símbolo de igualdad = no se recomienda usarlo para asignaciones.

```
x <- 5 # Correcto  
x = 5 # No recomendado
```

Para una explicación más detallada sobre el símbolo de asignación se recomienda visitar este enlace¹⁴.

1.5.6. Punto y coma

No se recomienda colocar varias instrucciones separadas por ; en la misma línea, aunque funciona dificulta la revisión del código.

¹⁴<http://www.win-vector.com/blog/2016/12/the-case-for-using-in-r/>

```
n <- 100; y <- rnorm(n, mean=5); hist(y) # No se recomienda  
  
n <- 100                                # Correcto  
y <- rnorm(n, mean=5)  
hist(y)
```

A pesar de la anterior advertencia es posible que en este libro usemos el ; en algunas ocasiones, si lo hacemos es para ahorrar espacio en la presentación del código.



2

Funciones básicas de R

En este capítulo se presentará lo que es una función y se mostrarán varias funciones básicas que son útiles para realizar diversas tareas.

2.1. ¿Qué es una función R?

En la Figura 2.1 se muestra una ilustración de lo que es una función o máquina general. Hay unas entradas (*inputs*) que luego son procesadas dentro de la caja para generar unas salidas (*outputs*). Un ejemplo de una función o máquina muy común en nuestras casas es la licuadora. Si a una licuadora le ingresamos leche, fresas, azúcar y hielo, el resultado será un delicioso jugo de fresa.

Las funciones en R se caracterizan por un nombre corto y que dé una idea de lo que hace la función. Los elementos que pueden ingresar (*inputs*) a la función se llaman **parámetros** y se ubican dentro de paréntesis, el cuerpo de la función se ubica dentro de llaves y es ahí donde se procesan los *inputs*

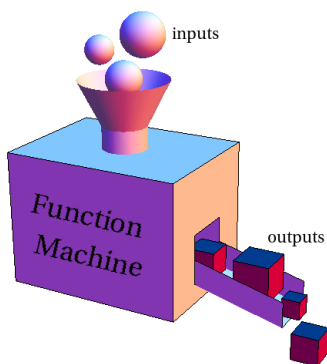


Figura 2.1: Ilustración de una función, tomada de www.mathinsight.org.

para convertirlos en *outputs*, a continuación se muestra la estructura general de una función.

```
nombre_de_funcion(parametro1, parametro2, ...) {  
  tareas internas  
  tareas internas  
  tareas internas  
  salida  
}
```

Cuando usamos una función sólo debemos escribir bien el nombre e ingresar correctamente los parámetros de la función, el cuerpo de la función ni lo vemos ni lo debemos modificar. A continuación se presenta un ejemplo de cómo usar la función `mean` para calcular un promedio.

```
notas <- c(4.0, 1.3, 3.8, 2.0) # Notas de un estudiante  
mean(notas)  
  
## [1] 2.775
```

2.2. Operadores de asignación

En R se pueden hacer asignación de varias formas, a continuación se presentan los operadores disponibles para tal fin.

- `<-` este es el operador de asignación a izquierda, es el más usado y recomendado.
- `->` este es el operador de asignación a derecha, no es frecuente su uso.
- `=` el símbolo igual sirve para hacer asignaciones pero **NO** se recomienda usarlo.
- `<<-` este es un operador de asignación global y sólo debe ser usado por usuarios avanzados.

Ejemplo

Almacene los valores 5.3, 4.6 y 25 en los objetos `a`, `b` y `age` respectivamente, use diferentes símbolos de asignación.

Para hacer lo solicitado se podría usar el siguiente código.

```
a <- 5.3 # Recomendado
4.6 -> b # It is not usual
age = 25 # Not recommended
```



Aunque una asignación se puede hacer de tres formas diferentes, se recomienda sólo usar el símbolo <-.

2.3. Operaciones básicas

En R se pueden hacer diversas operaciones usando operadores binarios. Este tipo de operadores se denomina binarios porque actúan entre dos objetos, a continuación el listado.

- + operador binario para sumar.
- - operador binario para restar.
- * operador binario para multiplicar.
- / operador binario para dividir.
- ^ operador binario para potencia.
- %/% operador binario para obtener el cociente en una división (número entero).
- %% operador binario para obtener el residuo en una división.

A continuación se presentan ejemplos de cómo usar las anteriores funciones.

```
6 + 4 # Para sumar dos números
```

```
## [1] 10
```

```
a <- c(1, 3, 2)
b <- c(2, 0, 1) # a y b de la misma dimensión
a + b # Para sumar los vectores a y b miembro a miembro
```

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

```
a - b # Para restar dos vectores a y b miembro a miembro
```

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

```
a * b # Para multiplicar
```

```
## [1] 2 0 2
```

```
a / b # Para dividir
```

```
## [1] 0.5 Inf 2.0
```

```
a ^ b # Para potencia
```

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

```
7 %/% 3 # Para saber las veces que cabe 3 en 7
```

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

```
7 %% 3 # Para saber el residuo al dividir 7 entre 3
```

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

2.4. Pruebas lógicas

En R se puede verificar si un objeto cumple una condición dada, a continuación el listado de las pruebas usuales.

- < para saber si un número es menor que otro.
- > para saber si un número es mayor que otro.
- == para saber si un número es igual que otro.
- <= para saber si un número es menor o igual que otro.
- >= para saber si un número es mayor o igual que otro.

A continuación se presentan ejemplos de cómo usar las anteriores funciones.

```
5 < 12 # ¿Será 5 menor que 12?
```

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



```
# Comparando objetos
x <- 5
y <- 20 / 4
x == y # ¿Será x igual a y?
```

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

```
# Usando vectores
a <- c(1, 3, 2)
b <- c(2, 0, 1)
a > b # Comparación término a término
```

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

```
a == b # Comparación de igualdad término a término
```

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

Ejemplo

Crear un vector con los números de 1 a 17 y extraer los números que son mayores o iguales a 12.

Primero se crea el vector `x` con los elementos del 1 al 17. La prueba lógica `x >= 12` se usa para evaluar la condición, el resultado es un vector de 17 posiciones con valores de `TRUE` o `FALSE` dependiendo de si la condición se cumple o no. Este vector lógico se coloca dentro de `x[]` para que al evaluar `x[x >= 12]` sólo aparezcan los valores del vector original que SI cumplen la condición. El código necesario se muestra a continuación.

```
x <- 1:17 # Se crea el vector
x[x >= 12] # Se solicitan los valores que cumplen la condición
```

```
## [1] 12 13 14 15 16 17
```

2.5. Operadores lógicos

En R están disponibles los operadores lógicos negación, conjunción y disyunción. A continuación el listado de los operadores entre los elementos `x` e `y`.

```
!x  # Negación de x
x & y  # Conjunción entre x e y
x && y
x | y  # Disyunción entre x e y
x || y
xor(x, y)
```

A continuación se presentan ejemplos de cómo usar el símbolo de negación `!`.

```
ans <- c(TRUE, FALSE, TRUE)
!ans  # Negando las respuestas almacenadas en ans
```

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

```
x <- c(5, 1.5, 2, 3, 2)
!(x < 2.5)  # Negando los resultados de una prueba
```

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

A continuación se presentan ejemplos de cómo aplicar la conjunción `&` y `&&`.

```
x <- c(5, 1.5, 2)  # Se construyen dos vectores para la prueba
y <- c(4, 6, 3)
```

```
x < 4  # ¿Serán los elementos de x menores que 4?
```

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

```
y > 5  # ¿Serán los elementos de y mayores que 5?
```

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

```
x < 4 & y > 5  # Conjunción entre las pruebas anteriores.
```

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

```
x < 4 && y > 5 # Conjunción vectorial
```

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

Note las diferencias entre los dos últimos ejemplos, cuando se usa `&` se hace una prueba término a término y el resultado es un vector, cuando se usa `&&` se aplica la conjunción al vector de resultados obtenido con `&`.

2.6. Funciones sobre vectores

En R podemos destacar las siguientes funciones básicas sobre vectores numéricos.

- `min`: para obtener el mínimo de un vector.
- `max`: para obtener el máximo de un vector.
- `length`: para determinar la longitud de un vector.
- `range`: para obtener el rango de valores de un vector, entrega el mínimo y máximo.
- `sum`: entrega la suma de todos los elementos del vector.
- `prod`: multiplica todos los elementos del vector.
- `which.min`: nos entrega la posición en donde está el valor mínimo del vector.
- `which.max`: nos da la posición del valor máximo del vector.
- `rev`: invierte un vector.

Ejemplo

Construir un vector llamado `myvec` con los siguientes elementos: 5, 3, 2, 1, 2, 0, NA, 0, 9, 6. Luego aplicar todas las funciones anteriores para verificar el funcionamiento de las mismas.

```
myvec <- c(5, 3, 2, 1, 2, 0, NA, 0, 9, 6)
myvec
```

```
## [1] 5 3 2 1 2 0 NA 0 9 6
```

```
min(myvec) # Opss, no aparece el mínimo que es Cero.
```

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

```
min(myvec, na.rm=TRUE) # Usamos na.rm = TRUE para remover el NA
```

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

```
max(myvec, na.rm=T) # Para obtener el valor máximo
```

```
## [1] 9
```

```
range(myvec, na.rm=T) # Genera min y max simultáneamente
```

```
## [1] 0 9
```

```
sum(myvec, na.rm=T) # La suma de los valores internos
```

```
## [1] 28
```

```
prod(myvec, na.rm=T) # El productor de los valores internos
```

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

```
which.min(myvec) # Posición del valor mínimo 0 en el vector
```

```
## [1] 6
```

```
which.max(myvec) # Posición del valor máximo 9 en el vector
```

```
## [1] 9
```

De las dos últimas líneas podemos destacar lo siguiente:

1. **NO es necesario** usar `na.rm = TRUE` para remover el NA dentro de las funciones `which.min` ni `which.max`.
2. El valor mínimo 0 aparece en las posiciones 6 y 8 pero la función `which.min` sólo entrega la posición del primer valor mínimo dentro del vector.

2.7. Funciones matemáticas

Otras funciones básicas muy utilizadas en estadística son: `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `atan2`, `log`, `logb`, `log10`, `exp`, `sqrt`, `abs`. A continuación algunos ejemplos de las anteriores funciones.

Ejemplos de medidas trigonométricas

```
angulos <- c(0, pi/2, pi)
sin(angulos)
```

```
## [1] 0.000e+00 1.000e+00 1.225e-16
```

```
tan(angulos)
```

```
## [1] 0.000e+00 1.633e+16 -1.225e-16
```

Ejemplos de logaritmos

```
log(100)
```

```
## [1] 4.605
```

```
log10(100)
```

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

```
logb(125, base=5)
```

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

Ejemplos de exponencial

```
exp(1)
```

```
## [1] 2.718
```

```
exp(2)
```

```
## [1] 7.389
```

```
exp(1:3)
```

```
## [1] 2.718 7.389 20.086
```

Ejemplos de raíces

```
sqrt(49) # Raiz cuadrada de 49
```

```
## [1] 7
```

```
27 ^ (1/3) # Raiz cúbica de 27
```

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

Ejemplos de valor absoluto

```
abs(2.5)
```

```
## [1] 2.5
```

```
abs(-3.6)
```

```
## [1] 3.6
```

2.8. Función seq

En R podemos crear secuencias de números de una forma sencilla usando la función `seq`, la estructura de esta función es:

```
seq(from=1, to=1, by, length.out)
```

Los argumentos de esta función son:

- `from`: valor de inicio de la secuencia.
- `to`: valor de fin de la secuencia, no siempre se alcanza.
- `by`: incremento de la secuencia.
- `length.out`: longitud deseada de la secuencia.

Ejemplo

Construya las siguientes tres secuencias usando la función `seq`.

- Once valores igualmente espaciados desde 0 hasta 1.
- Una secuencia de dos en dos comenzando en 1.
- Una secuencia desde 1 con un salto de π y sin pasar del número 9.

El código necesario para obtener las secuencias se muestra a continuación.

```
seq(from=0, to=1, length.out = 11)
```

```
## [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
seq(from=1, to=9, by=2) # matches 'end'
```

```
## [1] 1 3 5 7 9
```

```
seq(from=1, to=9, by=pi) # stays below 'end'
```

```
## [1] 1.000 4.142 7.283
```



En R existe el operador binario `:` que sirve para construir secuencias de uno en uno fácilmente.

Revise los siguientes ejemplos para entender el funcionamiento del operador `:`.

```
2:8
```

```
## [1] 2 3 4 5 6 7 8
```

```
3:-5
```

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

```
pi:6 # real sequence
```

```
## [1] 3.142 4.142 5.142
```

```
6:pi # integer sequence
```

```
## [1] 6 5 4
```

2.9. Función `rep`

En R podemos crear repeticiones usando la función `rep`, la estructura de esta función es:

```
rep(x, times=1, length.out=NA, each=1)
```

Los argumentos de esta función son:

- `x`: vector con los elementos a repetir.
- `times`: número de veces que el vector `x` se debe repetir.
- `length.out`: longitud deseada para el vector resultante.
- `each`: número de veces que cada elemento de `x` se debe repetir.

Ejemplo

Construya las siguientes repeticiones usando la función `rep`, no lo haga ingresando número por número.

- 1 2 3 4 1 2 3 4
- 1 1 2 2 3 3 4 4
- 1 1 2 3 3 4
- 1 1 2 2 3 3 4 4

La clave para construir una repetición es descubrir la semilla o elemento que se repite. Las instrucciones para obtener las repeticiones anteriores se muestra a continuación.

```
rep(x=1:4, times=2)
```

```
## [1] 1 2 3 4 1 2 3 4
```

```
rep(x=1:4, times=c(2,2,2,2))
```

```
## [1] 1 1 2 2 3 3 4 4
```

```
rep(x=1:4, times=c(2,1,2,1))
```

```
## [1] 1 1 2 3 3 4
```



```
rep(x=1:4, each=2)
```

```
## [1] 1 1 2 2 3 3 4 4
```

Ejemplo

La función `rep` es muy versátil, observe los siguientes 4 ejemplos y saque una conclusión de cada uno de ellos.

```
rep(x=1:4, each=2)
```

```
## [1] 1 1 2 2 3 3 4 4
```

```
rep(x=1:4, each=2, len=4)    # first 4 only.
```

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

```
rep(x=1:4, each=2, len=10)  # 8 integers plus two recycled 1's.
```

```
## [1] 1 1 2 2 3 3 4 4 1 1
```

```
rep(x=1:4, each=2, times=3) # length 24, 3 complete replications
```

```
## [1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
```

2.10. Funciones *round*, *ceiling*, *floor* y *trunc*

Existen 4 funciones útiles para modificar u obtener información de un número, estas funciones son *round*, *ceiling*, *floor* y *trunc*.

- `round(x, digits)`: sirve para redondear un número según los dígitos indicados.
- `ceiling(x)`: entrega el mínimo entero mayor o igual que `x`.
- `floor(x)`: entrega el máximo entero menor o igual que `x`.
- `trunc(x)`: entrega la parte entera de un número `x`.

Ejemplo

Aplique las funciones `round`, `ceiling`, `floor` y `trunc` a un valor positivo y a un valor negativo para inspeccionar los resultados.

A continuación el código de prueba para un número positivo cualquiera.

```
x <- 5.34896 # Número positivo elegido  
round(x, digits=3)
```

```
## [1] 5.349
```

```
ceiling(x)
```

```
## [1] 6
```

```
floor(x)
```

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

```
trunc(x)
```

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

A continuación las pruebas con un número negativo cualquiera.

```
x <- -4.26589 # Número negativo elegido  
round(x, digits=3)
```

```
## [1] -4.266
```

```
ceiling(x)
```

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

```
floor(x)
```

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

```
trunc(x)
```

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

2.11. Funciones *sort* y *rank*

Las funciones *sort* y *rank* son útiles para ordenar los elementos de un vector o para saber las posiciones que ocuparían los elementos de un vector al ser ordenado. La estructura de las dos funciones es la siguiente.

```
sort(x, decreasing = FALSE)
rank(x)
```

En el parámetro *x* se ingresa el vector y el parámetro *decreasing* sirva para indicar si el ordenamiento es de menor a mayor (por defecto es este) o de mayor a menor.

Ejemplo

Considere el vector *x* que tiene los siguientes elementos: 2, 3, 6, 4, 9 y 5. Ordene el vector de menor a mayor, de mayor a menor y por último encuentre la posición que ocupan los elementos de *x* si se ordenaran de menor a mayor.

```
x <- c(2, 3, 6, 4, 9, 5)
sort(x)
```

```
## [1] 2 3 4 5 6 9
```

```
sort(x, decreasing=TRUE)
```

```
## [1] 9 6 5 4 3 2
```

```
rank(x)
```

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

EJERCICIOS

Use funciones o procedimientos (varias líneas) de R para responder cada una de las siguientes preguntas.

1. ¿Qué cantidad de dinero sobra al repartir 10000\$ entre 3 personas?
2. ¿Es el número 4560 divisible por 3?
3. Construya un vector con los números enteros del 2 al 87. ¿Cuáles de esos números son divisibles por 7?
4. Construya dos vectores, el primero con los números enteros desde 7 hasta 3, el segundo vector con los primeros cinco números positivos divisibles por 5. Sea A la condición de ser par en el primer vector. Sea B la condición de ser mayor que 10 en el segundo vector. ¿En cuál de las 5 posiciones se cumple A y B simultáneamente?
5. Consulte el siguiente enlace sobre una anéctoda de Gauss cuando tenía 10 años de edad <http://tinyurl.com/hk2l8h2>. Use R para obtener el resultado de la suma solicitada por el profesor del niño Gauss.
6. Construya un vector con los siguientes elementos: 1, -4, 5, 9, -4. Escriba un procedimiento para extraer **las posiciones** donde está el valor mínimo en el vector.
7. Calcular 8!
8. Evaluar la siguiente suma $\sum_{i=3}^{i=7} e^i$
9. Evaluar la siguiente productoria $\prod_{i=1}^{i=10} \log \sqrt{i}$
10. Construya un vector cualquiera e inviértalo, es decir, que el primer elemento quede de último, el segundo de penúltimo y así sucesivamente. Compare su resultado con el de la función `rev`.
11. Create the vector: 1, 2, 3, ..., 19, 20.
12. Create the vector: 20, 19, ..., 2, 1.
13. Create the vector: 1, -2, 3, -4, 5, -6, ..., 19, -20.
14. Create the vector: $0.1^3, 0.2^1, 0.1^6, 0.2^4, \dots, 0.1^{36}, 0.2^{34}$.
15. Calculate the following: $\sum_{i=10}^{100} (i^3 + 4i^2)$ and $\sum_{i=1}^{25} \left(\frac{2^i}{i} + \frac{3^i}{i^2} \right)$.
16. Read the data set available in: <http://tinyurl.com/hcusrdc>
17. Use a code to obtain the number of variables of the data set.
18. Use a code to obtain the number of countries in the data set.
19. Which is the country with the higher population?
20. Which is the country with the lowest literacy rate?
21. ¿Qué valor de verdad tiene la siguiente afirmación? “Los resultados de la función `floor` y `trunc` son siempre los mismos”.



3

Lectura de bases de datos

En este capítulo se mostrará cómo leer una base de datos externa hacia R.

3.1. ¿En qué formato almacenar una base de datos?

Los archivos con la información para ser leídos por R se pueden almacenar en formato:

- plano con extensión **.txt** o,
- Excel con extensión **.csv**.

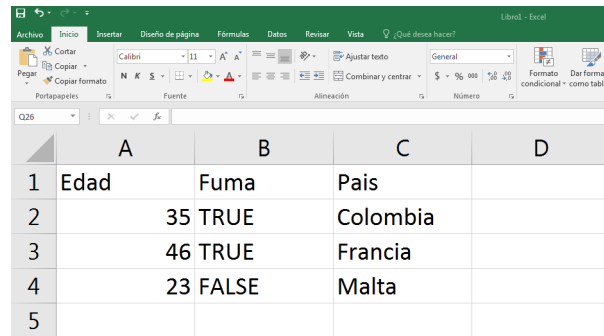
En las secciones siguientes se mostrará cómo almacenar datos en los dos formatos para ser leídos en R. En el Cuadro 3.1 se presenta una base de datos pequeña, tres observaciones y tres variables, que nos servirá como ejemplo para mostrar cómo se debe almacenar la información.

3.1.1. Almacenamiento de información en Excel

Para almacenar la información del Cuadro 3.1 en Excel, abrimos un archivo nuevo de Excel y copiamos la información tal como se muestra en la Figura 3.1. Se debe iniciar en la parte superior izquierda, no se deben dejar filas vacías, no se debe colorear, no se deben colocar bordes ni nada, se ingresa la información sin embellecer el contenido. Por último se guarda el archivo en

Cuadro 3.1: Ejemplo de una base de datos simple.

Edad	Fuma	Pais
35	TRUE	Colombia
46	TRUE	Francia
23	FALSE	Malta



	A	B	C	D
1	Edad	Fuma	Pais	
2	35	TRUE	Colombia	
3	46	TRUE	Francia	
4	23	FALSE	Malta	
5				

Figura 3.1: Forma de almacenar los datos en Excel.

la carpeta deseada y al momento de nombrar el archivo se debe modificar la opción tipo de archivo a **csv (delimitado por comas)**.



Recuerde que el archivo de Excel se debe guardar con extensión **.csv**.

3.1.2. Almacenamiento de información en bloc de notas

Para almacenar la información del Cuadro 3.1 en bloc de notas, abrimos un archivo nuevo de bloc de notas y copiamos la información tal como se muestra en la Figura 3.2. Se copian los nombres de las variables o los datos separados por un espacio obtenido con la tecla tabuladora, cada línea se finaliza con un *enter*. Se recomienda al guardar el archivo que el cursor al inicio de una línea vacía, en la Figura 3.2 se señala la posición del cursor con la flecha roja, a pesar de que no existe línea número 5, el curso debe quedar al inicio de esa línea número 5.

Es posible mejorar la apariencia de la información almacenada en el bloc de notas si, en lugar de usar espacios con la barra espaciadora, se colocan los espacios con la barra tabuladora, así la información se ve más organizada y se puede chequear fácilmente la información ingresada. En la Figura 3.3 se muestra la información para el ejemplo, claramente se nota la organización de la información.



Una buena práctica es usar la barra tabuladora para separar, eso permite que la información se vea ordenada.

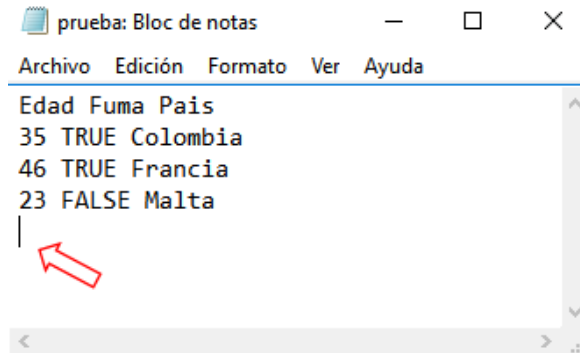


Figura 3.2: Almacenamiento de los datos en bloc de notas usando la barra espaciadora

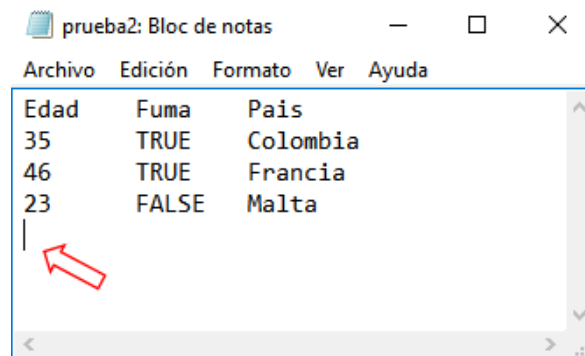


Figura 3.3: Almacenamiento de los datos en bloc de notas usando la barra tabuladora

3.2. Función `read.table`

La función `read.table` se puede usar para leer bases de datos hacia R. La estructura de la función con los parámetros más comunes de uso es la siguiente.

```
read.table(file, header, sep, dec)
```

Los argumentos de la función `read.table` son:

- **file:** nombre o ruta donde están alojados los datos. Puede ser un url o una dirección del computador. Es también posible usar `file.choose()` para que se abra un ventana y adjuntar el archivo deseado manualmente.

- **header**: valor lógico, se usa **TRUE** si la primera línea de la base de datos tiene los nombres de las variables, caso contrario se usa **FALSE**.
- **sep**: tipo de separación interna para los datos dentro del archivo. Los valores usuales para este parámetros son:
 - **sep=';'** si el archivo tiene extensión **.csv**.
 - **sep=' '** si el archivo es bloc de notas con espacios por la barra **espa-ciadora**.
 - **sep='\t'** si el archivo es bloc de notas con espacios por la barra **ta-buladora**.
- **dec**: símbolo con el cual están indicados los decimales.

Ejemplo

Crear la base de datos del Cuadro 3.1 en Excel y bloc de notas para luego leerla desde R.

Lo primero que se debe hacer es construir tres archivos (uno de Excel y dos bloc de notas) igual a los mostrados en las figuras 3.1, 3.2 y 3.3, vamos a suponer que los nombres para cada uno de ellos son **base1.csv**, **base2.txt** y **base3.txt** respectivamente.

Para Excel

Para leer el archivo de Excel llamado **base1.csv** podemos usar el siguiente código.

```
datos <- read.table(file='C:/Users/Hernandez/Desktop/base1.csv',
                    header=TRUE, sep=';')
datos
```

La dirección **file='C:/Users/Hernandez/Desktop/base1.csv'** le indica a R dónde debe buscar el archivo, note que se debe usar el símbolo **/** para que sea un dirección válida.

Si no se conoce la ubicación del archivo a leer o si la dirección es muy extensa se puede usar **file.choose()** para que se abra una ventana y así adjuntar manualmente el archivo. A continuación se muestra el código para hacerlo de esta manera.

```
datos <- read.table(file.choose(), header=TRUE, sep=';')
datos
```

Para bloc de notas con barra espaciadora

Para leer el archivo de Excel llamado `base2.txt` podemos usar el siguiente código.

```
datos <- read.table(file='C:/Users/Hernandez/Desktop/base2.txt',  
                    header=TRUE, sep=' ')  
datos
```

Para bloc de notas con barra tabuladora

Para leer el archivo de Excel llamado `base3.txt` podemos usar el siguiente código.

```
datos <- read.table(file='C:/Users/Hernandez/Desktop/base3.txt',  
                    header=TRUE, sep='\t')  
datos
```



El usuario puede usar indiferentemente `file='C:/Users/bla/bla'` o `file.choose()` para ingresar el archivo, con la práctica se aprende a decidir cuando conviene una u otra forma.



Un error frecuente es escribir la dirección o ubicación del archivo usando `\`, lo correcto es usar `/`.



4

Medidas de tendencia central

En este capítulo se mostrará cómo obtener las diferentes medidas de tendencia central con R.

Para ilustrar el uso de las funciones se utilizará una base de datos llamada **medidas del cuerpo**, esta base de datos cuenta con 6 variables registradas a un grupo de 36 estudiantes de la universidad. Las variables son:

1. **edad** del estudiante (años),
2. **peso** del estudiante (kilogramos),
3. **altura** del estudiante (centímetros),
4. **sexo** del estudiante (Hombre, Mujer),
5. **muneca**: perímetro de la muñeca derecha (centímetros),
6. **biceps**: perímetro del biceps derecho (centímetros).

A continuación se presenta el código para definir la url donde están los datos, para cargar la base de datos en R y para mostrar por pantalla un encabezado (usando `head`) de la base de datos.

```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/medidas_cuerpo'
datos <- read.table(file=url, header=T)
head(datos) # Para ver el encabezado de la base de datos
```

```
##  edad peso altura  sexo muneca biceps
## 1   43 87.3  188.0 Hombre   12.2   35.8
## 2   65 80.0  174.0 Hombre   12.0   35.0
## 3   45 82.3  176.5 Hombre   11.2   38.5
## 4   37 73.6  180.3 Hombre   11.2   32.2
## 5   55 74.1  167.6 Hombre   11.8   32.9
## 6   33 85.9  188.0 Hombre   12.4   38.5
```

4.1. Media

Para calcular la media de una variable cuantitativa se usa la función `mean`. Los argumentos básicos de la función `mean` son dos y se muestran a continuación.

```
mean(x, na.rm = FALSE)
```

En el parámetro `x` se indica la variable de interés para la cual se quiere calcular la media, el parámetro `na.rm` es un valor lógico que en caso de ser `TRUE`, significa que se deben remover las observaciones con `NA`, el valor por defecto para este parámetro es `FALSE`.

Ejemplo

Suponga que queremos obtener la altura media del grupo de estudiantes.

Para encontrar la media general se usa la función `mean` sobre el vector numérico `datos$altura`.

```
mean(x=datos$altura)
```

```
## [1] 171.6
```

Del anterior resultado podemos decir que la estatura media o promedio de los estudiantes es 171.5556 centímetros.

Ejemplo

Suponga que ahora queremos la altura media pero diferenciando por sexo.

Para hacer esto se debe primero dividir o partir el vector de altura según los niveles de la variable `sexo`, esto se consigue por medio de la función `split` y el resultado será una lista con tantos elementos como niveles tenga la variable `sexo`. Luego a cada uno de los elementos de la lista se le aplica la función `mean` con la ayuda de `sapply` o `tapply`. A continuación el código completo para obtener las alturas medias para hombres y mujeres.

```
sapply(split(x=datos$altura, f=datos$sexo), mean)
```

```
## Hombre  Mujer  
## 179.1    164.0
```

El resultado es un vector con dos elementos, vemos que la altura media para hombres es 179.0778 centímetros y que para las mujeres es de 164.0333 centímetros.

¿Qué sucede si se usa `tapply` en lugar de `sapply`? Substituya en el código anterior la función `sapply` por `tapply` y observe la diferencia entre los resultados.

Ejemplo

Suponga que se tiene el vector `edad` con las edades de siete personas y supóngase que para el individuo cinco no se tiene información de su edad, eso significa que el vector tendrá un `NA` en la quinta posición.

¿Cuál será la edad promedio del grupo de personas?

```
edad <- c(18, 23, 26, 32, NA, 32, 29)
mean(x=edad)
```

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

Al correr el código anterior se obtiene un error y es debido al símbolo `NA` en la quinta posición. Para calcular la media sólo con los datos de los cuales se tiene información, se incluye el argumento `na.rm = TRUE` para que R remueva los `NA`. El código correcto a usar en este caso es:

```
mean(x=edad, na.rm=TRUE)
```

```
## [1] 26.67
```

De este último resultado se obtiene que la edad promedio de los individuos es 26.67 años.

4.2. Mediana

Para calcular la mediana de una variable cuantitativa se usa la función `median`. Los argumentos básicos de la función `median` son dos y se muestran a continuación.

```
median(x, na.rm = FALSE)
```

En el parámetro `x` se indica la variable de interés para la cual se quiere calcular la mediana, el parámetro `na.rm` es un valor lógico que en caso de ser `TRUE`, significa que se deben remover las observaciones con `NA`, el valor por defecto para este parámetro es `FALSE`.

Ejemplo

Calcular la edad mediana para los estudiantes de la base de datos.

Para obtener la mediana usamos el siguiente código:

```
median(x=datos$edad)
```

```
## [1] 28
```

y obtenemos que la mitad de los estudiantes tienen edades mayores o iguales a 28 años.

El resultado anterior se pudo haber obtenido con la función `quantile` e indicando que se desea el cuantil 50 así:

```
quantile(x=datos$edad, probs=0.5)
```

```
## 50%
```

```
## 28
```

4.3. Moda

La moda de una variable cuantitativa corresponde a valor o valores que más se repiten, una forma sencilla de encontrar la moda es construir una tabla de frecuencias y observar los valores con mayor frecuencia.

Ejemplo

Calcular la moda para la variable edad de la base de datos de estudiantes.

Se construye la tabla con la función `table` y se crea el objeto `tabla` para almacenarla.


```
tabla <- table(datos$edad)
tabla
```

```
##
## 19 20 21 22 23 24 25 26 28 29 30 32 33 35 37 40 43 45
##  1  1  1  3  2  1  5  3  2  1  2  1  1  2  3  1  2  1
## 51 55 65
##  1  1  1
```

Al mirar con detalle la tabla anterior se observa que el valor que más se repite es la edad de 25 años en 5 ocasiones. Si la tabla hubiese sido mayor, la inspección visual nos podría tomar unos segundos o hasta minutos y podríamos equivocarnos, por esa razón es mejor ordenar los resultados de la tabla.

Para observar los valores con mayor frecuencia de la tabla se puede ordenar la tabla usando la función `sort` de la siguiente manera:

```
sort(tabla, decreasing=TRUE)
```

```
##
## 25 22 26 37 23 28 30 35 43 19 20 21 24 29 32 33 40 45
##  5  3  3  3  2  2  2  2  2  1  1  1  1  1  1  1  1  1
## 51 55 65
##  1  1  1
```

De esta manera se ve fácilmente que la variable edad es unimodal con valor de 25 años.



5

Medidas de posición

En este capítulo se mostrará cómo obtener las diferentes medidas de posición con R.

Para ilustrar el uso de las funciones se utilizará una base de datos llamada **medidas del cuerpo**, esta base de datos cuenta con 6 variables registradas a un grupo de 36 estudiantes de la universidad. Las variables son:

1. **edad** del estudiante (años),
2. **peso** del estudiante (kilogramos),
3. **altura** del estudiante (centímetros),
4. **sexo** del estudiante (Hombre, Mujer),
5. **muneca**: perímetro de la muñeca derecha (centímetros),
6. **biceps**: perímetro del biceps derecho (centímetros).

A continuación se presenta el código para definir la url donde están los datos, para cargar la base de datos en R y para mostrar por pantalla un encabezado (usando `head`) de la base de datos.

```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/medidas_cuerpo'
datos <- read.table(file=url, header=T)
head(datos) # Para ver el encabezado de la base de datos
```

```
##  edad peso altura  sexo muneca biceps
## 1   43 87.3  188.0 Hombre   12.2   35.8
## 2   65 80.0  174.0 Hombre   12.0   35.0
## 3   45 82.3  176.5 Hombre   11.2   38.5
## 4   37 73.6  180.3 Hombre   11.2   32.2
## 5   55 74.1  167.6 Hombre   11.8   32.9
## 6   33 85.9  188.0 Hombre   12.4   38.5
```

5.1. Cuantiles

Para obtener cualquier cuantil (cuartiles, deciles y percentiles) se usa la función `quantile`. Los argumentos básicos de la función `quantile` son tres y se muestran a continuación.

```
quantile(x, probs, na.rm = FALSE)
```

En el parámetro `x` se indica la variable de interés para la cual se quieren calcular los cuantiles, el parámetro `probs` sirve para definir los cuantiles de interés y el parámetro `na.rm` es un valor lógico que en caso de ser `TRUE`, significa que se deben remover las observaciones con `NA`, el valor por defecto para este parámetro es `FALSE`.

Ejemplo

Suponga que queremos obtener el percentil 5, la mediana y el decil 8 pa la altura del grupo de estudiantes.

Se solicita el percentil 5, la mediana que es el percentil 50 y el decil 8 que corresponde al percentil 80, por lo tanto es necesario indicarle a la función `quantile` que calcule los cuantiles para las ubicaciones 0.05, 0.5 y 0.8, el código para obtener las tres medidas solicitadas es el siguiente.

```
quantile(x=datos$altura, probs=c(0.05, 0.5, 0.8))
```

```
##      5%   50%   80%  
## 155.2 172.7 180.3
```

6

Medidas de variabilidad

En este capítulo se mostrará cómo obtener las diferentes medidas de variabilidad con R.

Para ilustrar el uso de las funciones se utilizará la base de datos llamada **aptos2015**, esta base de datos cuenta con 11 variables registradas a apartamentos usados en la ciudad de Medellín. Las variables de la base de datos son:

1. **precio**: precio de venta del apartamento (millones de pesos),
2. **mt2**: área del apartamento (m^2),
3. **ubicacion**: lugar de ubicación del apartamentos en la ciudad (cualitativa),
4. **estrato**: nivel socioeconómico donde está el apartamento (2 a 6),
5. **alcobas**: número de alcobas del apartamento,
6. **banos**: número de baños del apartamento,
7. **balcon**: si el apartamento tiene balcón (si o no),
8. **parqueadero**: si el apartamento tiene parqueadero (si o no),
9. **administracion**: valor mensual del servicio de administración (millones de pesos),
10. **avaluo**: valor del apartamento en escrituras (millones de pesos),
11. **terminado**: si el apartamento se encuentra terminado (si o no).

A continuación se presenta el código para definir la url donde están los datos, para cargar la base de datos en R y para mostrar por pantalla un encabezado (usando `head`) de la base de datos.

```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/aptos2015'
datos <- read.table(file=url, header=T)
head(datos) # Para ver el encabezado de la base de datos
```

```
##  precio  mt2 ubicacion estrato alcobas banos balcon
## 1     79 43.16   norte      3        3      1     si
## 2     93 56.92   norte      2        2      1     si
## 3    100 66.40   norte      3        2      2     no
## 4    123 61.85   norte      2        3      2     si
## 5    135 89.80   norte      4        3      2     si
```

```
## 6      140 71.00      norte      3      3      2      no
##  parqueadero administracion avaluo terminado
## 1          si          0.050  14.92          no
## 2          si          0.069  27.00          si
## 3          no          0.000  15.74          no
## 4          si          0.130  27.00          no
## 5          no          0.000  39.57          si
## 6          si          0.120  31.15          si
```

6.1. Rango

Para calcular el rango de una variable cuantitativa se usa la función `range`. Los argumentos básicos de la función `range` son dos y se muestran abajo.

```
range(x, na.rm = FALSE)
```

En el parámetro `x` se indica la variable de interés para la cual se quiere calcular el rango, el parámetro `na.rm` es un valor lógico que en caso de ser `TRUE`, significa que se deben remover las observaciones con `NA`, el valor por defecto para este parámetro es `FALSE`.

La función `range` entrega el valor mínimo y máximo de la variable ingresada y el valor de rango se puede obtener restando del valor máximo el valor mínimo.

Ejemplo

Suponga que queremos obtener el rango para la variable precio de los apartamentos.

Para obtener el rango usamos el siguiente código.

```
range(datos$precio)
```

```
## [1] 25 1700
```

```
max(datos$precio) - min(datos$precio)
```

```
## [1] 1675
```

Del resultado anterior podemos ver que los precios de todos los apartamentos van desde 25 hasta 1700 millones de pesos, es decir, el rango de la variable precio es 1675 millones de pesos.

Ejemplo

Suponga que queremos obtener nuevamente el rango para la variable precio de los apartamentos pero diferenciando por el estrato.

Primero vamos a crear una función auxiliar llamada `myrange` que calculará el rango directamente ($\max - \min$). Luego vamos a partir la información de los precios por cada estrato usando `split`, la partición se almacenará en la lista `precios`. Finalmente se aplicará la función `myrange` a la lista `precios` para obtener los rangos del precio por estrato socioeconómico. El código para realizar esto se muestra a continuación.

```
myrange <- function(x) max(x) - min(x)
precios <- split(datos$precio, f=datos$estrato)
sapply(precios, myrange)
```

```
##      2      3      4      5      6
## 103  225  610 1325 1560
```

De los resultados podemos ver claramente que a medida que aumenta de estrato el rango (variabilidad) del precio de los apartamentos aumenta. Apartamentos de estrato bajo tienden a tener precios similares mientras que los precios de venta para apartamentos de estratos altos tienden a ser muy diferentes entre sí.

6.2. Desviación estándar muestral (S)

Para calcular la desviación muestral de una variable cuantitativa se usa la función `sd`. Los argumentos básicos de la función `sd` son dos y se muestran abajo.

```
sd(x, na.rm = FALSE)
```

En el parámetro `x` se indica la variable de interés para la cual se quiere calcular la desviación estándar muestral, el parámetro `na.rm` es un valor lógico que en

caso de ser `TRUE`, significa que se deben remover las observaciones con `NA`, el valor por defecto para este parámetro es `FALSE`.

Ejemplo

Suponga que queremos obtener la desviación estándar muestral para la variable precio de los apartamentos.

Para obtener la desviación solicitada usamos el siguiente código:

```
sd(x=datos$precio)
```

```
## [1] 247.6
```

Ejemplo

Calcular la desviación estándar **poblacional** (σ) para el siguiente conjunto de 5 observaciones: 12, 25, 32, 15, 26.

Recordemos que las expresiones matemáticas para obtener S y σ son muy similares, la diferencia está en el denominador, para S el denominador es $n - 1$ mientras que para σ es n . Teniendo esto en cuenta podemos calcular la desviación poblacional apoyándonos en la función `sd`, para esto podemos construir una función llamada `Sigma` que calcule la desviación poblacional, a continuación el código necesario.

```
Sigma <- function(x) {  
  n <- length(x)  
  sd(x) * (n-1) / n  
}
```

Ahora para obtener la desviación estándar **poblacional** de los datos usamos el siguiente código.

```
y <- c(12, 25, 32, 15, 26)  
Sigma(y)
```

```
## [1] 6.621
```


6.3. Varianza muestral (S^2)

Para calcular la varianza muestral de una variable cuantitativa se usa la función `var`. Los argumentos básicos de la función `var` son dos y se muestran abajo.

```
var(x, na.rm = FALSE)
```

En el parámetro `x` se indica la variable de interés para la cual se quiere calcular la varianza muestral, el parámetro `na.rm` es un valor lógico que en caso de ser `TRUE`, significa que se deben remover las observaciones con `NA`, el valor por defecto para este parámetro es `FALSE`.

Ejemplo

Suponga que queremos determinar cuál región en la ciudad presenta mayor varianza en los precios de los apartamentos.

Para realizar esto debemos usar en conjunto la función `split`, `sapply` y `var` ya que se quiere la varianza de una variable (`precio`) dado los valores de otra variable (`ubicacion`). El código para obtener las varianzas es el siguiente.

```
precios <- split(datos$precio, f=datos$ubicacion)
sapply(precios, var)
```

##	aburra sur belen guayabal	centro
##	4169	2588
##	laureles	norte
##	25351	1009
##	poblado	occidente
##	84497	3596

De los resultados anteriores se nota que los apartamentos ubicados en el Poblado tienen la mayor variabilidad en el precio, este resultado se confirma al dibujar un boxplot para la variable precio dada la ubicación, en la Figura 6.1 se muestra el boxplot y se ve claramente la dispersión de los precios en el Poblado. El código usado para generar la Figura 6.1 se presenta a continuación.

```
with(datos, boxplot(precio ~ ubicacion, ylab='Precio (millones)'))
```

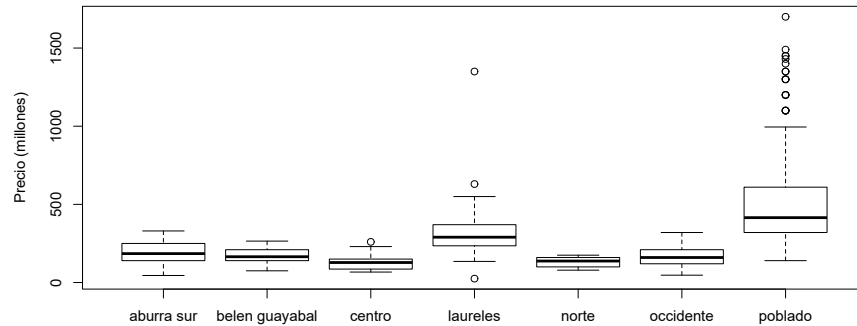


Figura 6.1: Boxplot para el precio de los apartamentos dada la ubicación.

Ejemplo

¿Son los resultados de la función `var` los mismos que los resultados de la función `sd` elevados al cuadrado?

La respuesta es **NO**. La función `sd` se aplica sólo a vectores mientras que la función `var` se puede aplicar tanto a vectores como a marcos de datos. Al ser aplicada a marcos de datos numéricos se obtiene una matriz en que la diagonal representa las varianzas de las de cada una de las variables mientras que arriba y abajo de la diagonal se encuentran las covarianzas entre pares de variables.

Por ejemplo, si aplicamos la función `var` al marco de datos sólo con las variables `precio`, `área` y `avaluo` se obtiene una matriz de dimensión 3×3 , a continuación el código usado.

```
var(datos[, c('precio', 'mt2', 'avaluo')])
```

```
##      precio  mt2 avaluo
## precio 61313 15874 33056
## mt2    15874  5579  9508
## avaluo 33056  9508 28589
```

Del anterior resultado se observa la matriz de varianzas y covarianzas de dimensión 3×3 .

6.4. Coeficiente de variación (CV)

El coeficiente de variación se define como $CV = s/\bar{x}$ y es muy sencillo de obtenerlo, la función CV mostrada abajo permite calcularlo.

```
CV <- function(x, na.rm = FALSE) {  
  sd(x, na.rm=na.rm) / mean(x, na.rm=na.rm)  
}
```

Ejemplo

Calcular el CV para el vector **w** definido a continuación.

```
w <- c(5, -3, NA, 8, 8, 7)
```

Vemos que el vector **w** tiene 6 observaciones y la tercera de ellas es un NA. Lo correcto aquí es usar la función CV definida antes pero indicándole que remueva los valores faltantes, para eso se usa el siguiente código.

```
CV(x=w, na.rm=T)
```

```
## [1] 0.9274
```



7

Medidas de correlación

En este capítulo se mostrará cómo obtener las diferentes medidas de



8

Creación de funciones en R

En este capítulo se



9

Distribuciones discretas

En este capítulo se



10

Distribuciones continuas

En este capítulo se



11

Pruebas de bondad de ajuste

En este capítulo se



12

Aproximación de integrales

En este capítulo se mostrará cómo aproximar integrales en una y varias dimensiones.

12.1. Aproximación de Laplace unidimensional

Esta aproximación es útil para obtener el valor de una integral usando la expansión de Taylor para una función $f(x)$ unimodal en \mathbb{R} , en otras palabras lo que interesa es:

$$I = \int_{-\infty}^{\infty} f(x) dx$$

Al hacer una expansión de Taylor de segundo orden para $\log(f(x))$ en su moda x_0 el resultado es:

$$\log(f(x)) \approx \log(f(x_0)) + \frac{\log(f)'(x_0)}{1!}(x - x_0) + \frac{\log(f)''(x_0)}{2!}(x - x_0)^2$$

El segundo término de la suma se anula porque $\log(f)'(x_0) = 0$ por ser x_0 el valor donde está el máximo de $\log(f(x))$. La expresión anterior se simplifica en:

$$\log(f(x)) \approx \log(f(x_0)) + \frac{\log(f)''(x_0)}{2!}(x - x_0)^2$$

al aislar $f(x)$ se tiene que

$$f(x) \approx f(x_0) \exp\left(-\frac{c}{2}(x - x_0)^2\right) \quad (12.1)$$

donde $c = -\frac{d^2}{dx^2} \log(f(x)) \Big|_{x=x_0}$.

La expresión 12.1 se puede reescribir de manera que aparezca el núcleo de la función de densidad de la distribución normal con media x_0 y varianza $1/c$, a continuación la expresión

$$f(x) \approx f(x_0) \frac{\sqrt{2\pi/c}}{\sqrt{2\pi/c}} \exp\left(-\frac{1}{2} \left(\frac{x - x_0}{1/\sqrt{c}}\right)^2\right)$$

Así al calcular la integral de $f(x)$ en \mathfrak{R} se tiene que:

$$I = \int_{-\infty}^{\infty} f(x) d(x) = f(x_0) \sqrt{2\pi/c} \quad (12.2)$$

Ejemplo

Calcular la integral de $f(x) = \exp(-(x - 1.5)^2)$ en \mathfrak{R} utilizando la aproximación de Laplace.

Primero vamos a dibujar la función $f(x)$ para ver en dónde está su moda x_0 .

```
fun <- function(x) exp(-(x-1.5)^2)
curve(fun, from=-5, to=5, ylab='f(x)', las=1)
```

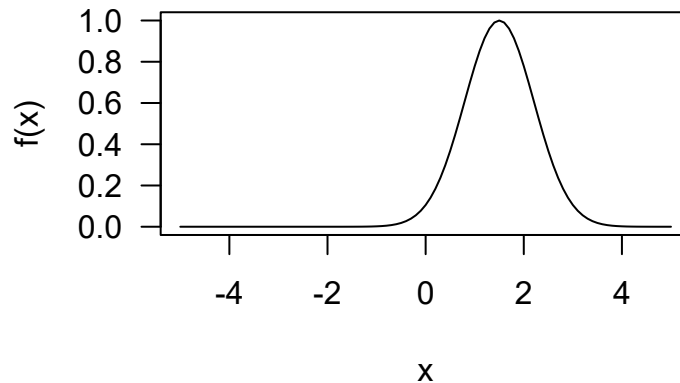


Figura 12.1: Perfil de la función $f(x)$.

Visualmente se nota que la moda está cerca del valor 1.5 y para determinar numéricamente el valor de la moda x_0 se usa la función `optimize`, los resultados se almacenan en el objeto `res`. El valor de la moda corresponde al elemento `maximum` del objeto `res`.

```
res <- optimize(fun, interval=c(-10, 10), maximum=TRUE)
res
```



```
## $maximum  
## [1] 1.5  
##  
## $objective  
## [1] 1
```

Para determinar el valor de c de la expresión 12.2 se utiliza el siguiente código.

```
require("numDeriv")  
constant <- - as.numeric(hessian(fun, res$maximum))
```

Para obtener la aproximación de la integral se usa la expresión 12.2 y para tener un punto de comparación se evalúa la integral usando la función `integrate`, a continuación el código.

```
fun(res$maximum) * sqrt(2*pi/constant)
```

```
## [1] 1.772
```

```
integrate(fun, -Inf, Inf) # Para comparar
```

```
## 1.772 with absolute error < 1.5e-06
```

De los anteriores resultados vemos que la aproximación es buena.



Bibliografía

Correa, J. & Hernández, F. (2018). *Gráficos con R*. Universidad Nacional de Colombia, Medellín, Colombia, primera edition. ISBN xxx-xxxxxxx.

Wickham, H. (2015). *R Packages*. O'Reilly Media, Inc.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2016). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.2.5.



Índice alfabético

.csv, 33
.txt, 33

arreglo, 6
asignación, 16

bloc de notas, 34

ceiling, 27
coeficiente de variación, 53
cuantiles, 46
cuartiles, 46

deciles, 46
desviación, 49

Excel, 33

floor, 27
función, 15
function, 15

guía de estilo, 9

lectura de bases de datos, 33
length, 21
lista, 8

marco de datos, 7
matrices, 6
max, 21
mean, 40
media, 40
median, 41
mediana, 41
min, 21
moda, 42

objetos, 4

operaciones básicas, 17
operadores lógicos, 20
ordenar, 29

percentiles, 46
posición, 29
prod, 21
pruebas lógicas, 18

quantile, 46

range, 21, 48
rango, 48
rank, 29
read.table, 35
rep, 26
repeticiones, 26
round, 27

sd, 49
secuencias, 24
seq, 24
sort, 29
sum, 21

trunc, 27

var, 51
varianza, 51
vectores, 5

which.max, 21
which.min, 21