

*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	VII
Índice de figuras	IX
Prefacio	XI
Sobre los autores	XV
<b>1. Introducción</b>	<b>1</b>
1.1. Orígenes . . . . .	1
1.2. Descarga e instalación . . . . .	2
1.3. Apariencia del programa . . . . .	4
<b>2. Tipos de objetos</b>	<b>7</b>
2.1. Vectores . . . . .	7
2.1.1. ¿Cómo extraer elementos de un vector? . . . . .	8
2.2. Matrices . . . . .	9
2.2.1. ¿Cómo extraer elementos de una matriz? . . . . .	10
2.3. Arreglos . . . . .	11
2.3.1. ¿Cómo extraer elementos de un arreglo? . . . . .	12
2.4. Marco de datos . . . . .	13
2.4.1. ¿Cómo extraer elementos de un marco de datos? . . . . .	13
2.4.2. ¿Cómo extraer subconjuntos de un marco de datos? . . . . .	14
2.5. Listas . . . . .	16
2.5.1. ¿Cómo extraer elementos de una lista? . . . . .	17
<b>3. Guía de estilo</b>	<b>21</b>
3.1. Nombres de los archivos . . . . .	21
3.2. Nombres de los objetos . . . . .	21
3.3. Longitud de una línea de código . . . . .	22
3.4. Espacios . . . . .	22
3.5. Asignación . . . . .	24
3.6. Punto y coma . . . . .	25
<b>4. Funciones básicas de R</b>	<b>27</b>
4.1. ¿Qué es una función de R? . . . . .	27
4.2. Operadores de asignación . . . . .	29

4.3. Operaciones básicas	29
4.4. Pruebas lógicas	31
4.5. Operadores lógicos	33
4.6. Funciones sobre vectores	35
4.7. Funciones matemáticas	37
4.8. Función <code>seq</code>	38
4.9. Función <code>rep</code>	40
4.10. Funciones <code>round</code> , <code>ceiling</code> , <code>floor</code> y <code>trunc</code>	42
4.11. Funciones <code>sort</code> y <code>rank</code>	43
<b>5. Instrucciones de control</b>	<b>47</b>
5.1. Instrucción <code>if</code>	47
5.2. Instrucción <code>if else</code>	48
5.3. Instrucción <code>ifelse</code>	48
5.4. Instrucción <code>for</code>	49
5.5. Instrucción <code>while</code>	50
5.6. Instrucción <code>repeat</code>	51
<b>6. Creación de funciones en R</b>	<b>53</b>
6.1. Función en R	53
6.2. Partes de una función en R	53
<b>7. Lectura de bases de datos</b>	<b>63</b>
7.1. ¿En qué formato almacenar una base de datos?	63
7.1.1. Almacenamiento de información en Excel	63
7.1.2. Almacenamiento de información en bloc de notas	64
7.2. Función <code>read.table</code>	66
7.3. Lectura de bases de datos en Excel	68
<b>8. Tablas de frecuencia</b>	<b>71</b>
8.1. Tabla de contingencia con <code>table</code>	71
Ejemplo: tabla de frecuencia de una vía	71
Ejemplo: tabla de frecuencia de dos vías	73
8.2. Función <code>prop.table</code>	73
Ejemplo: tabla de frecuencia relativa de una vía	74
Ejemplo: tabla de frecuencia relativa de dos vías	74
8.3. Función <code>addmargins</code>	75
8.4. Función <code>hist</code>	76
<b>9. Medidas de tendencia central</b>	<b>81</b>
9.1. Media	82
9.2. Mediana	83
9.3. Moda	84
<b>10. Medidas de variabilidad</b>	<b>87</b>
10.1. Rango	88

<i>Contents</i>	v
10.2. Desviación estándar muestral ( $S$ ) . . . . .	89
10.3. Varianza muestral ( $S^2$ ) . . . . .	91
10.4. Coeficiente de variación ( $CV$ ) . . . . .	93
<b>11. Medidas de posición</b>	<b>95</b>
11.1. Cuantiles . . . . .	96
<b>12. Medidas de correlación</b>	<b>97</b>
12.1. Función cor . . . . .	97
<b>13. Distribuciones discretas</b>	<b>103</b>
13.1. Funciones disponibles para distribuciones discretas . . . . .	103
13.2. Distribuciones discretas generales . . . . .	109
<b>14. Distribuciones continuas</b>	<b>115</b>
14.1. Funciones disponibles para distribuciones continuas . . . . .	115
14.2. Distribuciones continuas generales . . . . .	121
<b>15. Verosimilitud</b>	<b>127</b>
15.1. Función de verosimilitud . . . . .	127
15.2. Función de log-verosimilitud . . . . .	127
15.3. Método de máxima verosimilitud . . . . .	128
<b>16. Estudiando la normalidad</b>	<b>143</b>
16.1. Consideraciones iniciales . . . . .	143
16.2. Histograma y densidad . . . . .	144
16.3. Gráficos cuantil cuantil . . . . .	147
16.4. Pruebas de normalidad . . . . .	154
<b>17. Aproximación de integrales</b>	<b>159</b>
17.1. Aproximación de Laplace unidimensional . . . . .	159
<b>Bibliografía</b>	<b>163</b>
<b>Índice alfabético</b>	<b>165</b>



---

## *Índice de cuadros*

---

7.1. Ejemplo de una base de datos simple. . . . .	63
7.2. Base de datos para practicar lectura. . . . .	70





---

## Índice de figuras

---

1.1. Robert Gentleman (izquierda) y Ross Ihaka (derecha) creadores de R. . . . .	2
1.2. Página del Cran. . . . .	3
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. . . . .	5
4.1. Ilustración de una función, tomada de <a href="http://www.mathinsight.org">www.mathinsight.org</a> . . . . .	28
6.1. Ilustración de una función, tomada de <a href="http://www.mathinsight.org">www.mathinsight.org</a> . . . . .	54
6.2. Ilustración del punto medio entre dos puntos, tomada de <a href="https://www.slideshare.net/bigpassy/midpoint-between-two-points">https://www.slideshare.net/bigpassy/midpoint-between-two-points</a> . . . . .	60
7.1. Forma de almacenar los datos en Excel. . . . .	64
7.2. Almacenamiento de los datos en bloc de notas usando la barra espaciadora . . . . .	65
7.3. Almacenamiento de los datos en bloc de notas usando la barra tabuladora . . . . .	65
8.1. Ubicación de los puntos del ejemplo con límites en color azul. . . . .	78
10.1. Boxplot para el precio de los apartamentos dada la ubicación. . . . .	92
12.1. Diagrama de dispersión para precio versus área de los apartamentos usados. . . . .	98
12.2. Matriz de coeficientes de correlación. . . . .	101
13.1. Función de masa de probabilidad para una $Binomial(n = 18, p = 0.1)$ . . . . .	105
13.2. Fotografía del cangrejo de herradura, tomada de <a href="http://sccoastalresources.com/home/2016/6/21/a-night-of-horseshoe-crab-tagging">http://sccoastalresources.com/home/2016/6/21/a-night-of-horseshoe-crab-tagging</a> . . . . .	110
13.3. Función de masa de probabilidad para el número de satélites por hembra. . . . .	111

13.4. Función de distribución acumulada para el número de satélites por hembra. . . . .	112
13.5. Función de distribución acumulada para el número de satélites por hembra diferenciando por grupo. . . . .	114
14.1. Función de densidad para una $Beta(2, 5)$ . . . . .	116
14.2. Área sombreada para los ejemplos. . . . .	119
14.3. Área sombreada para el ejemplo de los tornillos. . . . .	122
14.4. Función de densidad $f(x)$ para el peso de los cangrejos. . . . .	123
14.5. Función acumulada $F(x)$ para el peso de los cangrejos. . . . .	124
14.6. Función de densidad $f(x)$ para el peso del cangrejo diferenciando por el color. . . . .	125
14.7. Función de densidad $f(x)$ para el peso del cangrejo diferenciando por el color y usando ggplot2. . . . .	126
15.1. Función de log-verosimilitud para el ejemplo sobre binomial. . . . .	130
15.2. Gráfico de niveles para la función de log-verosimilitud para el ejemplo sobre normal. . . . .	132
15.3. Gráfico cuantil cuantil normal y gamma para la muestra simulada. . . . .	136
15.4. Histograma para la muestra simulada con la densidad de una $\text{Gamma}(\mu=4.308, \sigma=0.6682)$ . . . . .	139
16.1. Densidad para 4 muestras de una $N(0, 1)$ con diferente tamaño de muestra. . . . .	145
16.2. Densidad para el peso corporal de hombres y mujeres. . . . .	147
16.3. Ejemplo de un QQplot. . . . .	148
16.4. QQplot para 4 muestras de una $N(0, 1)$ con diferente tamaño de muestra. . . . .	149
16.5. QQplot para muestras generadas de poblaciones Poisson, Binomial Negativa, Gamma y Weibull. . . . .	151
16.6. QQplot para el peso corporal de hombres y mujeres. . . . .	153
16.7. QQplot con bandas de confianza para el peso corporal de hombres y mujeres. . . . .	154
17.1. Perfil de la función $f(x)$ . . . . .	160

---

## ***Prefacio***

---

Este libro fue creado con la intención de apoyar el aprendizaje del lenguaje de programación 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 las principales funciones para realizar análisis estadísticos.

El presente material está en proceso de elaboración, si el lector desea tener la última versión del libro recomendamos consultar la versión alojada en el repositorio de GitHub disponible en el siguiente enlace: [https://github.com/fhernanb/Manual-de-R/blob/master/\\_book/Manual\\_de\\_R.pdf](https://github.com/fhernanb/Manual-de-R/blob/master/_book/Manual_de_R.pdf)

---

## **Estructura del libro**

El libro está estructurado de la siguiente manera.

En el capítulo 1 se presenta una breve introducción sobre el lenguaje de programación R; en el capítulo 2 se explican los tipos de objetos más comunes en R; en el capítulo 3 se muestran las normas de estilo sugeridas para escribir código en R; el capítulo 4 presenta las funciones básicas que todo usuario debe conocer para usar con éxito R; el capítulo 6 trata sobre cómo crear funciones; el capítulo 7 muestra cómo leer bases de datos desde R; en el capítulo 8 se ilustra la forma para construir tablas de frecuencia; en el capítulo 9 se muestra cómo obtener las diversas medidas de tendencia central para variables cuantitativas, el capítulo 10 muestra cómo calcular las medidas de variabilidad, en el capítulo 11 se ilustra cómo usar las funciones para obtener medidas de posición; en el capítulo 12 se muestra cómo obtener medidas de correlación entre pares de variables; en los capítulos 13 y 14 se tratan los temas de distribuciones discretas y continuas; en el capítulo 15 se aborda el tema de verosimilitud; en el capítulo 17 se muestra el tema de aproximación de integrales.

---

## Información del software y convenciones

Para realizar este libro se usaron los paquetes de **R knitr** (Xie, 2015) y **bookdown** (Xie, 2016), estos paquetes permiten construir todo el libro desde R y sirven para incluir código que se ejecute de forma automática incluyendo las salidas y gráficos.

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 el usuario **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.



Nota aclaratoria.



Sugerencia.



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:

- Rbloggers<sup>1</sup>.
- Comunidad hispana de R<sup>2</sup>.
- Nabble<sup>3</sup>.
- Foro en portugués<sup>4</sup>.
- Stackoverflow<sup>5</sup>.
- Cross Validated<sup>6</sup>.
- R-Help Mailing List<sup>7</sup>.
- Revolutions<sup>8</sup>.
- R-statistics blog<sup>9</sup>.
- RDataMining<sup>10</sup>.

---

<sup>1</sup><https://www.r-bloggers.com/>

<sup>2</sup><http://r-es.org/>

<sup>3</sup><http://r.789695.n4.nabble.com/>

<sup>4</sup><http://r-br.2285057.n4.nabble.com/>

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

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

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

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

<sup>9</sup><https://www.r-statistics.com/>

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



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

---

## 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 enlace<sup>11</sup>. 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 hiperenlace **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 hiperenlace **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 actualizada. Una vez allí usted debe dar clic sobre **Download R 3.2.5 for Windows**

---

<sup>11</sup><https://cran.r-project.org/>

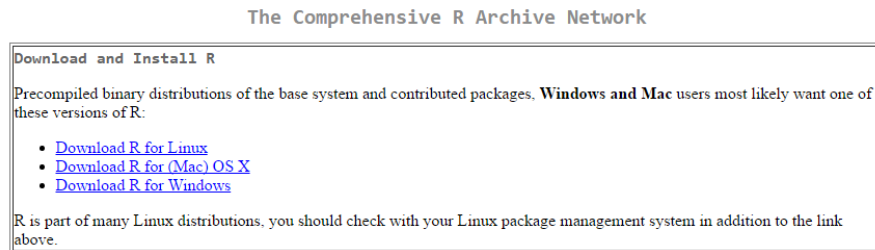


Figura 1.2: Página del Cran.

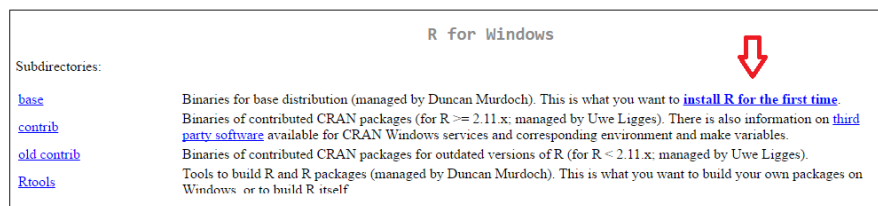


Figura 1.3: Página de instalación para la primera ocasión.

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 enlace<sup>12</sup> para facilitar la tarea de instalación.

<sup>12</sup><http://tinyurl.com/jd7b9ks>

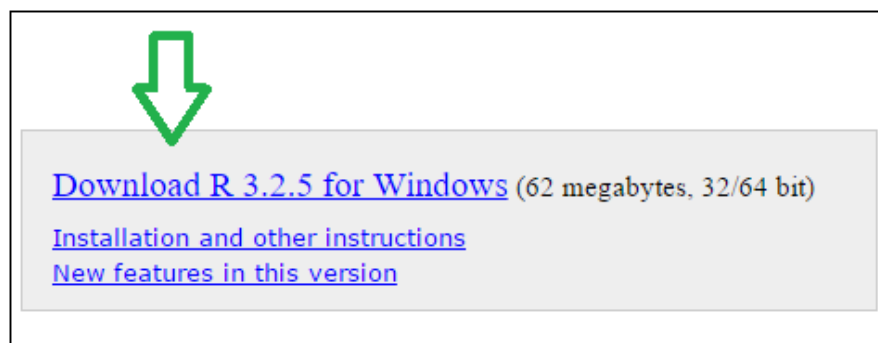


Figura 1.4: Página de descarga.



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

---

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

**Figura 1.6:** Apariencia de R.



## 2

### *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.

#### 2.1. 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"
```

### 2.1.1. ¿Cómo extraer elementos de un vector?

Para extraer un elemento almacenado dentro un vector se usan los corchetes [] y dentro de ellos la posición o posiciones que interesan.

#### Ejemplo

Si queremos extraer la edad de la tercera persona escribimos el nombre del vector y luego [3] para indicar la tercera posición de `edad`, a continuación el código.

```
edad[3]
```

```
## [1] 13
```



Si queremos conocer el cómic favorito de la segunda y quinta persona, escribimos el nombre del vector y luego, dentro de los corchetes, escribimos otro vector con las posiciones 2 y 5 que nos interesan así `c(2, 5)`, a continuación el código.

```
comic.fav[c(2, 5)]
```

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

Si nos interesan las respuestas de la práctica de deporte, excepto la de la persona 3, usamos `[-3]` luego del nombre del vector para obtener todo, excepto la tercera posición.

```
deporte[-3]
```

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



Si desea extraer varias posiciones de un vector NUNCA escriba esto: `mivector[2, 5, 7]`. Tiene que crear un vector con las posiciones y luego colocarlo dentro de los corchetes así: `mivector[c(2, 5, 7)]`

## 2.2. 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  $4 \times 5$ ) 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
```

### 2.2.1. ¿Cómo extraer elementos de una matriz?

Al igual que en el caso de los vectores, para extraer elementos almacenados dentro de una matriz se usan los corchetes [ , ] y dentro, separado por una coma, el número de fila(s) y el número de columna(s) que nos interesan.

#### Ejemplo

Si queremos extraer el valor almacenado en la fila 3 y columna 4 usamos el siguiente código.

```
mimatriz[3, 4]
```

```
## [1] 15
```

Si queremos recuperar **toda** la fila 2 usamos el siguiente código.

```
mimatriz[2, ] # No se escribe nada luego de la coma
```

```
## [1]  2  6 10 14 18
```

Si queremos recuperar **toda** la columna 5 usamos el siguiente código.

```
mimatriz[, 5] # No se escribe nada antes de la coma
```

```
## [1] 17 18 19 20
```

Si queremos recuperar la matriz original sin las columnas 2 y 4 usamos el siguiente código.

```
mimatriz[, -c(2, 4)] # Las columnas como vector
```

```
##      [,1] [,2] [,3]
```

```
## [1,]    1    9   17
## [2,]    2   10   18
## [3,]    3   11   19
## [4,]    4   12   20
```

Si queremos recuperar la matriz original sin la fila 1 ni columna 3 usamos el siguiente código.

```
mimatriz[-1, -3] # Signo de menos para eliminar
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    6   14   18
## [2,]    3    7   15   19
## [3,]    4    8   16   20
```

---

## 2.3. Arreglos

Un arreglo es una matriz de varias dimensiones con información numérica, alfanumérica o lógica. Para construir una 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"
```

### 2.3.1. ¿Cómo extraer elementos de un arreglo?

Para recuperar elementos almacenados en un arreglo se usan también corchetes, y dentro de los corchetes, las coordenadas del objeto de interés.

#### Ejemplo

Si queremos extraer la letra almacenada en la fila 1 y columna 3 de la segunda capa de `miarray` usamos el siguiente código.

```
miarray[1, 3, 2] # El orden es importante
```

```
## [1] "s"
```

Si queremos extraer la segunda capa completa usamos el siguiente código.

```
miarray[, 2] # No se coloca nada en las primeras posiciones
```

```
##      [,1] [,2] [,3] [,4]
## [1,] "m"  "p"  "s"  "v"
## [2,] "n"  "q"  "t"  "w"
## [3,] "o"  "r"  "u"  "x"
```

Si queremos extraer la tercera columna de todas las capas usamos el siguiente código.

```
miarray[, 3,] # No se coloca nada en las primeras posiciones
```

```
##      [,1] [,2]
## [1,] "g"  "s"
## [2,] "h"  "t"
## [3,] "i"  "u"
```

---

## 2.4. Marco de datos

El marco de datos 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.

### 2.4.1. ¿Cómo extraer elementos de un marco de datos?

Para recuperar las variables (columnas) almacenadas en un marco de datos se puede usar el operador `$` o también corchetes `[]`. A continuación algunos ejemplos para entender el uso del operador `$`.

#### Ejemplo

Si queremos extraer la variable `deporte` del marco de datos `mimarco` usamos el siguiente código.

```
mimarco$deporte # Se recomienda si el nombre es corto
```

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

Otra forma de recuperar la variable `deporte` es indicando la columna donde se encuentra la variable

```
mimarco[, 2] # Se recomienda si recordamos su ubicacion
```

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

Si queremos la `edad` de las personas que están en las posiciones 2 hasta 4 usamos el siguiente código.

```
mimarco[2:4, 1]
```

```
## [1] 19 13 NA
```

#### 2.4.2. ¿Cómo extraer subconjuntos de un marco de datos?

Para extraer partes de un marco de datos se puede utilizar la función `subset(x, subset, select)`. El parámetro `x` sirve para indicar el marco de datos original, el parámetro `subset` sirve para colocar la condición y el parámetro `select` sirve para quedarnos sólo con algunas de las variables del marco de datos. A continuación varios ejemplos de la función `subset` para ver su utilidad.

#### Ejemplos

Si queremos el marco de datos `mimarco` sólo con las personas que SI practican deporte usamos el siguiente código.

```
subset(mimarco, subset=deporte == TRUE)
```

```
##   edad deporte comic.fav
## 1   15     TRUE      <NA>
## 2   19     TRUE Superman
## 5   20     TRUE   Batman
```

Si queremos el marco de datos `mimarco` sólo con las personas mayores o iguales a 17 años usamos el siguiente código.

```
subset(mimarco, subset=edad >= TRUE)
```

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

Si queremos el submarco con deporte y comic de las personas menores de 20 años usamos el siguiente código.

```
subset(mimarco, subset=edad < 20, select=c('deporte', 'comic.fav'))
```

```
##  deporte comic.fav
## 1     TRUE    <NA>
## 2     TRUE Superman
## 3      NA   Batman
```

Si queremos el marco de datos `mimarco` sólo con las personas menores de 20 años y que SI practican deporte usamos el siguiente código.

```
subset(mimarco, subset=edad < 20 & deporte == TRUE)
```

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

## Ejemplo

Leer la base de datos medidas del cuerpo disponible en este enlace [https://raw.githubusercontent.com/fhernanb/datos/master/medidas\\_cuerpo](https://raw.githubusercontent.com/fhernanb/datos/master/medidas_cuerpo).

Extraer de esta base de datos una sub-base o subconjunto que contenga sólo la edad, peso, altura y sexo de aquellos que miden más de 185 cm y pesan más de 80 kg.

```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/medidas_cuerpo'
dt1 <- read.table(url, header=T)
dim(dt1) # Para conocer la dimensión de la base original
```

```
## [1] 36 6
```

```
dt2 <- subset(x=dt1, subset=altura > 185 & peso > 80,
              select=c('sexo', 'edad', 'peso', 'altura'))
dt2 # Para mostrar la base de datos final
```

```
##      sexo edad peso altura
## 1  Hombre  43  87.3  188.0
## 6  Hombre  33  85.9  188.0
## 15 Hombre  30  98.2  190.5
```

Al almacenar la nueva base de datos en el objeto `dt2` se puede manipular este nuevo objeto para realizar los análisis de interés.

## 2.5. 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  $6 \times 2$  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
```

### 2.5.1. ¿Cómo extraer elementos de una lista?

Para recuperar los elementos almacenados en una lista se usa el operador `$`, corchetes dobles `[[ ]]` o corchetes sencillos `[ ]`. A continuación unos ejemplos para entender cómo extraer elementos de una lista.

#### Ejemplos

Si queremos la matriz almacenada con el nombre de `E2` dentro del objeto `milista` se puede usar el siguiente código.

```
milista$E2
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    3    5    7    9   11
## [2,]    2    4    6    8   10   12
```

Es posible indicar la posición del objeto en lugar del nombre, para eso se usan los corchetes dobles.

```
milista[[2]]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    3    5    7    9   11
## [2,]    2    4    6    8   10   12
```

El resultado obtenido con `milista$E2` y `milista[[2]]` es **exactamente** el mismo. Vamos ahora a solicitar la posición 2 pero usando corchetes sencillos.

```
milista[2]
```

```
## $E2
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    3    5    7    9   11
## [2,]    2    4    6    8   10   12
```

La apariencia de este último resultado es similar, no igual, al encontrado al usar `$` y `[]`. Para ver la diferencia vamos a pedir la clase a la que pertenecen los tres últimos objetos usando la función `class`. A continuación el código usado.

```
class(milista$E2)
```

```
## [1] "matrix"
```

```
class(milista[[2]])
```

```
## [1] "matrix"
```

```
class(milista[2])
```

```
## [1] "list"
```

De lo anterior se observa claramente que cuando usamos `$` o `[]` el resultado es el objeto almacenado, una matriz. Cuando usamos `[]` el resultado es una **lista** cuyo contenido es el objeto almacenado.



Al manipular listas con `$` y `[]` se obtienen los objetos ahí almacenados, al manipular listas con `[]` se obtiene una lista.

---

## EJERCICIOS

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

1. Construya un vector con las primeras 20 letras MAYÚSCULAS usando la función `LETTERS`.

2. Construya una matriz de  $10 \times 10$  con los primeros 100 números positivos pares.
3. Construya una matriz identidad de dimension  $3 \times 3$ . Recuerde que una matriz identidad tiene sólo unos en la diagonal principal y los demás elementos son cero.
4. Construya una lista con los anteriores tres objetos creados.
5. Construya un marco de datos o data frame con las respuestas de 3 personas a las preguntas: (a) ¿Cuál es su edad en años? (b) ¿Tipo de música que más le gusta? (c) ¿Tiene usted pareja sentimental estable?
6. ¿Cuál es el error al correr el siguiente código? ¿A qué se debe?

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



## 3

---

### *Guía de estilo*

---

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*<sup>1</sup>.

---

#### 3.1. 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.

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

---

#### 3.2. 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:

---

<sup>1</sup><https://google.github.io/styleguide/Rguide.xml>

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

Para funciones:

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

### 3.3. 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.

### 3.4. 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, ptiles[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 {  
  one or more lines  
} # Correcto  
  
if (condition) {  
  one or more lines  
}  
else {  
  one or more lines  
} # Incorrecto  
  
if (condition)  
  one line  
else  
  one line # Incorrecto
```

---

### 3.5. 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 enlace<sup>2</sup>.

---

### 3.6. Punto y coma

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

```
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.

---

<sup>2</sup><http://www.win-vector.com/blog/2016/12/the-case-for-using-in-r/>



# 4

## *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.

### 4.1. ¿Qué es una función de R?

En la Figura 4.1 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 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.



**Figura 4.1:** Ilustración de una función, tomada de [www.mathinsight.org](http://www.mathinsight.org).

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

## [1] 2.775
```

---

## 4.2. 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 # Recommended
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 `<-`.

---

## 4.3. 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
```

---

## 4.4. 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 crear 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
```

## Ejemplo

Retome el marco de datos `mimarco` construido en la sección 2.4 y use una prueba lógica para extraer la información de las personas que tienen una edad superior o igual a 15 años.

Inicialmente vamos a construir nuevamente el objeto `mimarco` de la sección 2.4 usando el siguiente código.

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

```
mimarco # Para ver el contenido de 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
```

Para extraer de `mimarco` la información de las personas que tienen una edad superior o igual a 15 años se coloca dentro de corchetes la condición `mimarco$edad >= 15`, esto servirá para chequear cuáles de las edades del vector `mimarco$edad` cumplen la condición. El resultado de evaluar `mimarco$edad >= 15` será un vector lógico (TRUE o FALSE), que al ser colocado dentro de `mimarco[, ]`, entregará la información de las personas que cumplen la condición. A continuación el código para extraer la información solicitada.



```
mimarco[mimarco$edad >= 15, ]
```

```
##      edad deporte comic.fav
## 1      15      TRUE      <NA>
## 2      19      TRUE  Superman
## NA     NA       NA      <NA>
## 5      20      TRUE    Batman
```

De la salida anterior se observa que 4 personas de las 5 cumplen la condición.



Note que la condición `mimarco$edad >= 15` se debe ubicar **antes** de la coma para obtener todos individuos que cumplen con la condición.

## 4.5. 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 `&`.

## Ejemplo

Retome el marco de datos `mimarco` construido en la sección 2.4 y use una prueba lógica para extraer la información de las personas que tienen una edad superior o igual a 15 años y que practican deporte.

Aquí interesa extraer la información de los individuos que cumplen dos condiciones simultáneamente, aquellos con edad  $\geq 15$  y que SI practiquen deporte. El código necesario para obtener la información solicitada es el siguiente.

```
mimarco[mimarco$edad >= 15 & mimarco$deporte == TRUE, ]
```

```
##   edad deporte comic.fav
## 1   15     TRUE      <NA>
## 2   19     TRUE   Superman
## 5   20     TRUE     Batman
```

De la anterior salida se observa que sólo 3 de las 5 personas cumplen ambas condiciones.



La función `with` es útil porque nos permite realizar algún procedimiento **CON** un objeto, escribiendo menos y de una forma más natural.

Una forma alternativa para escribir lo anterior usando la función `with` es la siguiente.

```
with(mimarco, mimarco[edad >= 15 & deporte == TRUE, ])
```

```
##   edad deporte comic.fav
## 1   15     TRUE    <NA>
## 2   19     TRUE  Superman
## 5   20     TRUE   Batman
```

Al usar `with` sólo se tuvo que escribir el objeto `mimarco` dos veces. Cuando hay muchas condiciones o cuando el objeto tiene un nombre largo es aconsejable usar `with`.

---

## 4.6. 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.

---

## 4.7. 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
```

---

## 4.8. 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  $\pi$  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
```

---

## 4.9. 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
```

---

#### 4.10. 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
```

---

### 4.11. 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”.

En R hay unas bases de datos incluídas, una de ellas es la base de datos llamada `mtcars`. Para conocer las variables que están en `mtcars` usted puede escribir en la consola `?mtcars` o también `help(mtcars)`. De la base `mtcars` obtenga bases de datos que cumplan las siguientes condiciones.

22. Autos que tengan un rendimiento menor a 18 millas por galón de combustible.
23. Autos que tengan 4 cilindros.
24. Autos que pesen más de 2500 libras y tengan transmisión manual.



# 5

## *Instrucciones de control*

En R se disponen de varias instrucciones de control para facilitar los procedimientos que un usuario debe realizar. A continuación se explican esas instrucciones de control.

### 5.1. Instrucción `if`

Esta instrucción sirve para realizar un conjunto de operaciones **si** se cumple cierta condición. A continuación se muestra la estructura básica de uso.

```
if (condicion) {  
  operación 1  
  operación 2  
  ...  
  operación final  
}
```

### Ejemplo

Una secretaria recibe la información del salario básico semanal de un empleado y las horas trabajadas durante la semana por ese empleado. El salario básico es la remuneración por 40 horas de labor por semana, las horas extra son pagadas a cincuenta mil pesos. Escriba el procedimiento en R que debe usar la secretaria para calcular el salario semanal de un empleado que trabajó 45 horas y tiene salario básico de un millón de pesos.

El código para calcular el salario final del empleado es el siguiente:

```
sal <- 1 # Salario básico por semana  
hlab <- 45 # Horas laboradas por semana
```

```
if(hlab > 40) {  
  hext <- hlab - 40  
  salext <- hext * 0.05  
  sal <- sal + salext  
}
```

```
sal # Salarío semanal
```

```
## [1] 1.25
```

---

## 5.2. Instrucción if else

Esta instrucción sirve para realizar un conjunto de operaciones cuando **NO** se cumple cierta condición evaluada por un **if**. A continuación se muestra la estructura básica de uso.

```
if (condicion) {  
  operación 1  
  operación 2  
  ...  
  operación final  
}  
else {  
  operación 1  
  operación 2  
  ...  
  operación final  
}
```

---

## 5.3. Instrucción ifelse

Se recomienda usar la instrucción **ifelse** cuando hay una sola instrucción para el caso **if** y para el caso **else**. A continuación se muestra la estructura básica de uso.



```
ifelse(condición, operación SI cumple, operación NO cumple)
```

### Ejemplo

Suponga que usted recibe un vector de números enteros, escriba un procedimiento que diga si cada elemento del vector es par o impar.

```
x <- c(5, 3, 2, 8, -4, 1)

ifelse(x %% 2 == 0, 'Es par', 'Es impar')

## [1] "Es impar" "Es impar" "Es par"   "Es par"
## [5] "Es par"   "Es impar"
```

---

## 5.4. Instrucción *for*

La instrucción *for* es muy útil para repetir un procedimiento cierta cantidad de veces. A continuación se muestra la estructura básica de uso.

```
for (i in secuencia) {
  operación 1
  operación 2
  ...
  operación final
}
```

### Ejemplo

Escriba un procedimiento para crear 10 muestras de tamaño 100 de una distribución uniforme entre uno y tres. Para cada una de las muestra, se debe contar el número de elementos de la muestra que fueron mayores o iguales a 2.5.

```
nrep <- 10 # Número de repeticiones
n <- 100  # Tamaño de la muestra
```

```
conteo <- numeric(nrep) # Vector para almacenar el conteo

for (i in 1:nrep) {
  x <- runif(n=n, min=1, max=3)
  conteo[i] <- sum(x >= 2.5)
}

conteo # Para obtener el conteo

## [1] 24 37 28 26 30 18 29 23 19 19
```

---

### 5.5. Instrucción while

La instrucción `while` es muy útil para repetir un procedimiento siempre que se cumple una condición. A continuación se muestra la estructura básica de uso.

```
while (condición) {
  operación 1
  operación 2
  ...
  operación final
}
```

#### Ejemplo

Suponga que se lanza una moneda en la cual el resultado es cara o sello. Escribir un procedimiento que simule lanzamientos hasta que el número de caras obtenidas sea 5. El procedimiento debe entregar el historial de lanzamientos.

Para simular el lanzamiento de una moneda se puede usar la función `sample` y definiendo el vector `resultados` con `size=1` para simular un lanzamiento, a continuación el código y tres pruebas ilustrativas.

```
resultados <- c('Cara', 'Sello')
sample(x=resultados, size=1) # Prueba 1

## [1] "Cara"
```

Una vez seamos capaces de simular un lanzamiento podemos escribir el procedimiento para generar tantos lanzamientos hasta que se cumpla la condición. El código mostrado abajo permite hacer lo solicitado.

```
num.lanza <- 0      # Contador de lanzamientos
num.caras <- 0      # Contados de caras obtenidas
historial <- NULL    # Vector vacío para almacenar

while (num.caras < 5) {
  res <- sample(x=resultados, size=1)
  num.lanza <- num.lanza + 1
  historial[num.lanza] <- res
  if (res == 'Cara') {
    num.caras <- num.caras + 1
  }
}

historial

## [1] "Cara" "Sello" "Sello" "Sello" "Cara" "Sello"
## [7] "Sello" "Cara" "Sello" "Sello" "Cara" "Sello"
## [13] "Cara"

num.lanza

## [1] 13
```



La instrucción **for** se usa cuando sabemos el número de veces que se debe repetir el procedimiento, mientras que la instrucción **while** se usa cuando debemos repetir un procedimiento cuando se cumpla una condición.

## 5.6. Instrucción *repeat*

La instrucción **while** es muy útil para repetir un procedimiento siempre que se cumple una condición. A continuación se muestra la estructura básica de uso.

```
repeat {
  operación 1
```

```
operación 2
...
operación final
if (condición) break
}
```

### Ejemplo

Escribir un procedimiento para ir aumentando de uno en uno el valor de  $x$  hasta que  $x$  sea igual a siete. El procedimiento debe imprimir por pantalla la secuencia de valores de  $x$ .

```
x <- 3 # Valor de inicio

repeat {
  print(x)
  x <- x + 1
  if (x == 8) {
    break
  }
}
```

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



La instrucción **break** sirve para salir de un procedimiento iterativo.

# 6

---

## *Creación de funciones en R*

---

En este capítulo se explica cómo crear funciones en R para realizar tareas específicas.

---

### 6.1. Función en R

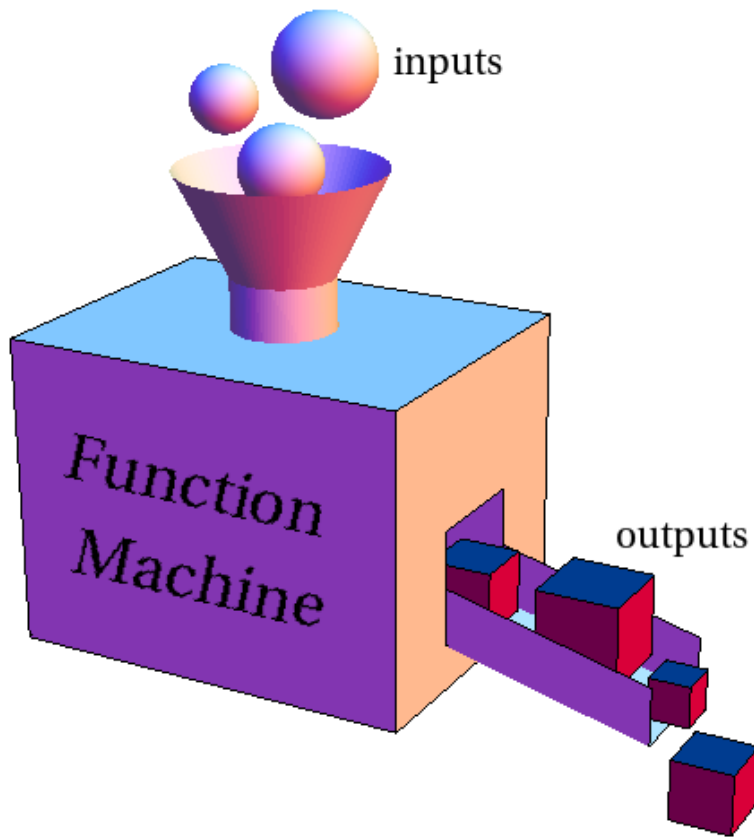
Una función es un conjunto de instrucciones que convierten las entradas (inputs) en resultados (outputs) deseados. En la Figura 6.1 se muestra una ilustración de lo que es una función o máquina general.

---

### 6.2. Partes de una función en R

Las partes de una función son:

- Entradas: o llamadas también **argumentos**, sirven para ingresar información necesaria para realizar el procedimiento de la función. Los argumentos pueden estar vacíos y a la espera de que el usuario ingrese valores, o pueden tener valores por defecto, esto significa que si el usuario no ingresa un valor a la función usará el valor por defecto. Una función puede tener o no argumentos de entrada, en los ejemplos se mostrarán estos casos.
- Cuerpo: el cuerpo de la función está formado por un conjunto de instrucciones que transforman las entradas en las salidas deseadas. Si el cuerpo de la función está formado por varias instrucciones éstas deben ir entre llaves.
- Salidas: son los resultados de la función. Toda función debe tener al menos un resultado, si una función no genera un resultado entonces no sirve para nada. Si una función entrega varios tipos de objetos se acostumbra a organizarlos en una lista que puede manejar los diferentes tipos de objetos.



**Figura 6.1:** Ilustración de una función, tomada de [www.mathinsight.org](http://www.mathinsight.org).

```
nombre_de_funcion <- function(par1, par2, ...) {  
  cuerpo  
  cuerpo  
  cuerpo  
  cuerpo  
  return(resultado)  
}
```

A continuación se mostrarán varios ejemplos **sencillos** para que el lector aprenda a construir funciones.

### Ejemplo

Construir una función que reciba dos números y que entregue la suma de estos números.

Lo primero es elegir un nombre apropiado para la función, aquí se usó el nombre `suma` porque así se tiene una idea clara de lo que hace la función. La función `suma` recibe dos parámetros, `x` representa el primer valor ingresado mientras que `y` representa el segundo. El cuerpo de la función está formado por dos líneas, en la primera se crea el objeto `resultado` en el cual se almanacena el valor de la suma, en la segunda línea se le indica a R que queremos que retorne el valor de la suma almacenada en el objeto `resultado`. A continuación se muestra el código para crear la función solicitada.

```
suma <- function(x, y) {  
  resultado <- x + y  
  return(resultado)  
}
```

Para usar la función creada sólo se debe ejecutar, vamos a obtener la suma de los valores 4 y 6 usando la función `suma`, a continuación el código necesario.

```
suma(x=4, y=6)
```

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

Para funciones simples como la anterior es posible escribirlas en forma más compacta. Es posible reducir el cuerpo de la función de 2 líneas a sólo una línea solicitándole a R que retorne directamente la suma sin almacenarla en ningún objeto. A continuación la función `suma` modificada.

```
suma <- function(x, y) {  
  return(x + y)  
}  
  
suma(x=4, y=6)  # Probando la función
```

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

Debido a que la función `suma` tiene un cuerpo muy reducido es posible escribirla en forma más compacta, en una sola línea. A continuación se muestra el código para reescribir la función.

```
suma <- function(x, y) x + y

suma(x=4, y=6) # Probando la función

## [1] 10
```

## Ejemplo

Construir una función que genere números aleatorios entre cero y uno hasta que la suma de éstos números supere por primera vez el valor de 3. La función debe entregar la cantidad de números aleatorios generados para que se cumpla la condición.

Vamos a llamar la función solicitada con el nombre **fun1**, esta función **NO** necesita ningún parámetro de entrada. El valor de 3 que está en la condición puede ir dentro del cuerpo y por eso no se necesitan parámetros para esta función. En el cuerpo de la función se genera un vector con un número aleatorio y luego se chequea si la suma de sus elementos es menor de 3, si se cumple que la suma es menor que 3 se siguen generando números que se almacenan en el vector **num**. Una vez que la suma exceda el valor de 3 **NO** se ingresa al **while** y se pide la longitud del vector o el valor de **veces** solicitado. A continuación el código de la función.

```
fun1 <- function() {
  num <- runif(1)
  veces <- 1
  while (sum(num) < 3) {
    veces <- veces + 1
    num[veces] <- runif(1)
  }
  return(veces)
}

fun1() # primera prueba

## [1] 8
```

## Ejemplo

Construir una función que, dado un número entero positivo (cota) ingresado por el usuario, genere números aleatorios entre cero y uno hasta que la suma



de los números generados exceda por primera vez la cota. La función debe entregar un vector con los números aleatorios, la suma y la cantidad de números aleatorios. Si el usuario no ingresa el valor de la cota, se debe asumir igual a 1.

La función aquí solicitada es similar a la construída en el ejemplo anterior. La función `fun2` tiene un sólo parámetro con el valor por defecto, si el usuario no ingresa valor a este parámetro, se asumirá el valor de uno. El cuerpo de la función es similar al anterior. Como la función debe entregar un vector y dos números, se construye la lista `resultado` que almacena los tres objetos solicitados. A continuación el código para función solicitada.

```
fun2 <- function(cota=1) {  
  num <- runif(1)  
  while (sum(num) < cota) {  
    num <- c(num, runif(1))  
  }  
  resultado <- list(vector=num,  
                    suma=sum(num),  
                    cantidad=length(num))  
  return(resultado)  
}
```

Probando la función con cota de uno.

```
fun2()  
  
## $vector  
## [1] 0.7704 0.6568  
##  
## $suma  
## [1] 1.427  
##  
## $cantidad  
## [1] 2
```

Probando la función con cota de tres.

```
fun2(cota=3)  
  
## $vector  
## [1] 0.5174 0.7786 0.6926 0.8071 0.2143  
##  
## $suma  
## [1] 3.01
```

```
##  
## $cantidad  
## [1] 5
```

## Ejemplo

Construya una función que reciba dos números de la recta real y que entregue el punto medio de estos números. El resultado debe ser un mensaje por pantalla.

El punto medio entre dos valores es la suma de los números dividido entre dos. La función `cat` sirve para concatenar objetos y presentarlos por pantalla. A continuación el código para la función requerida.

```
medio <- function(a, b) {  
  medio <- (a + b) / 2  
  cat("El punto medio de los valores",  
      a, "y", b,  
      "ingresados es", medio)  
}  
  
medio(a=-3, b=-1) # Probando la función
```

```
## El punto medio de los valores -3 y -1 ingresados es -2
```



La función `cat` es muy útil para presentar resultados por pantalla. Consulte la ayuda de la función para ver otros ejemplos.

---

## EJERCICIOS

Construir funciones en R que realicen lo solicitado.

1. Construya una función que reciba dos números reales `a` y `b`, la función debe decir cuál es el mayor de ellos.
2. Escriba una función llamada `media` que calcule la media muestral de un vector numérico `x` ingresado a la función. A continuación la fórmula para calcular la media muestral.

$$\bar{x} = \frac{\sum_{i=1}^n x_i}{n}$$

Nota: no puede usar la función `mean()`.

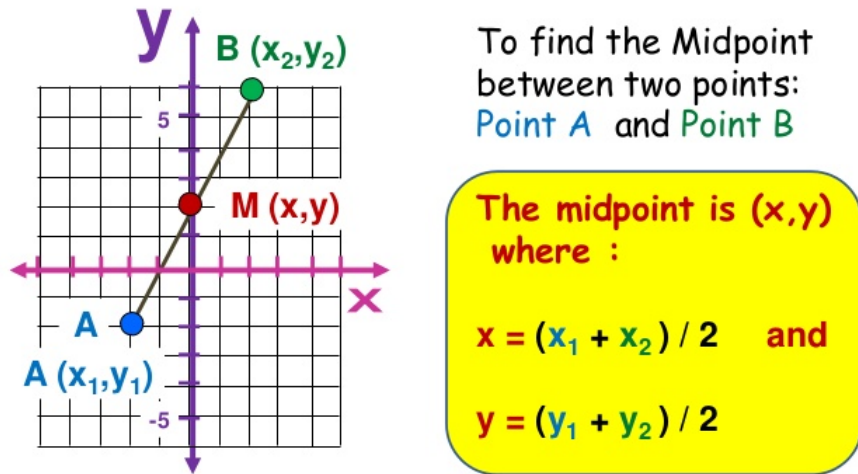
3. Construya una función que encuentre las raíces de una ecuación de segundo grado. El usuario debe suministrar los coeficientes **a**, **b** y **c** de la ecuación  $ax^2 + bx + c = 0$  y la función debe entregar las raíces.
4. Escribir una función que calcule la velocidad de un proyectil dado que el usuario ingresa la distancia recorrida en Km y el tiempo necesario en minutos. Expresar el resultado se debe entregar en metros/segundo, recuerde que

$$velocidad = \frac{espacio}{tiempo}$$

5. Construya una función que calcule las coordenadas del punto medio *M* entre dos puntos *A* y *B*. Vea la Figura 6.2 para una ilustración. ¿Cuáles cree usted que deben ser los parámetros de entrada de la función?
6. Escribir una función que reciba dos valores *a* y *b* y que los intercambie. Es decir, si ingresa  $a = 4$  y  $b = 9$  que la función entregue  $a = 9$  y  $b = 4$ .
7. Construya una función a la cual le ingrese el salario por hora y el número de horas trabajadas durante una semana por un trabajador. La función debe calcular el salario neto.
8. Construya una función llamada **precio** que calcule el precio total de sacar *A* fotocopias y *B* impresiones, sabiendo que los precios son 50 y 100 pesos para *A* y *B* respectivamente si el cliente es un estudiante, y de 75 y 150 para *A* y *B* si el cliente es un profesor. La función debe tener dos argumentos cuantitativos (*A* y *B*) y el argumento lógico **estudiante** que por defecto tenga el valor de **TRUE**. Use la estructura mostrada abajo.

```
precio <- function(A, B, estudiante=TRUE) {
  ...
  ...
  ...
  return(precio.total)
}
```

# Midpoint Formula



**Figura 6.2:** Ilustración del punto medio entre dos puntos, tomada de <https://www.slideshare.net/bigpassy/midpoint-between-two-points>

9. Construya una función llamada `salario` que le ingrese el salario por hora y el número de horas trabajadas durante una semana por un trabajador. La función debe calcular el salario neto semanal, teniendo en cuenta que si el número de horas trabajadas durante la semana es mayor de 48, esas horas de demás se consideran horas extras y tienen un 35 % de recargo. Imprima el salario neto. Use la estructura mostrada abajo.

```
salario <- function(num.horas, valor.hora) {
  ...
  ...
  ...
  return(salario.neto)
}
```

10. Construya una función llamada `nota` que calcule la nota obtenida por un alumno en una evaluación de tres puntos cuya ponderación o importancia son 20 %, 30 % y 50 % para los puntos I, II y III res-

pectivamente. Adicionalmente la función debe generar un mensaje sobre si el estudiante aprobó la evaluación o no. El usuario debe ingresar las notas individuales de los tres puntos y la función debe entregar la nota final de la evaluación. Use la estructura mostrada abajo.

```
nota <- function(p1, p2, p3) {  
  ...  
  ...  
  ...  
}
```

11. Escriba una función llamada `minimo` que permita obtener el valor mínimo de un vector numérico. No puede usar ninguna de las funciones básicas de R como `which.min()`, `which.max()`, `order()`, `min()`, `max()`, `sort()` u `order()`. Use la estructura mostrada abajo.

```
minimo <- function(x) {  
  ...  
  ...  
  return(minimo)  
}
```



# 7

## *Lectura de bases de datos*

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

### 7.1. ¿En qué formato almacenar una base de datos?

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

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

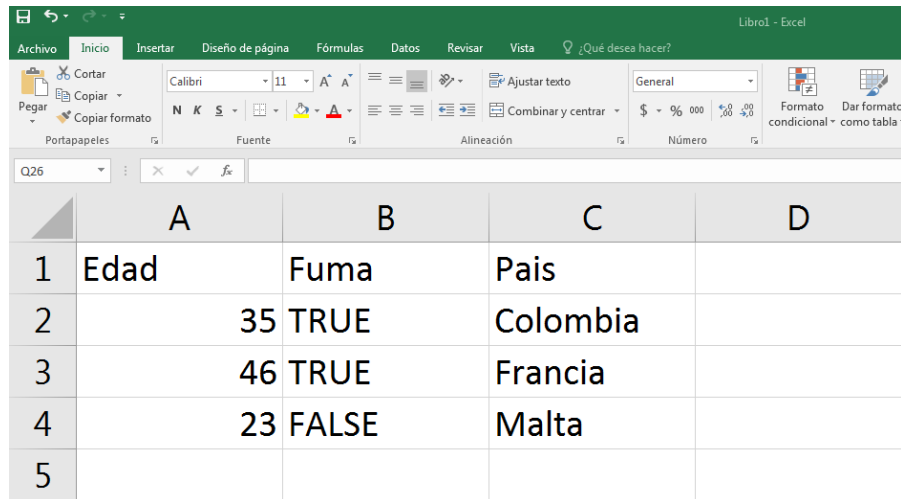
En las secciones siguientes se mostrará cómo almacenar datos en los dos formatos para ser leídos en R. En el Cuadro 7.1 se presenta una base de datos pequeña, tres observaciones y tres variables, que nos servirá como ejemplo para mostrar cómo se debe almacenar la información.

#### 7.1.1. Almacenamiento de información en Excel

Para almacenar la información del Cuadro 7.1 en Excel, abrimos un archivo nuevo de Excel y copiamos la información tal como se muestra en la Figura 7.1. 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 7.1:** 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 7.1:** 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**.

### 7.1.2. Almacenamiento de información en bloc de notas

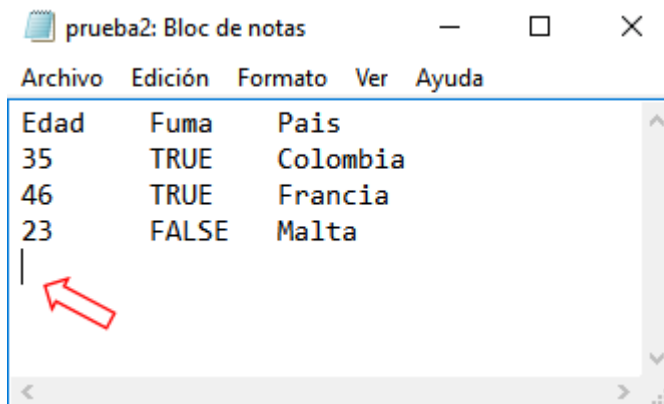
Para almacenar la información del Cuadro 7.1 en bloc de notas, abrimos un archivo nuevo de bloc de notas y copiamos la información tal como se muestra en la Figura 7.2. Se copian los nombres de las variables o los datos separados por un espacio obtenido con la tecla tabuladora, cada línea se finaliza con un enter. Se recomienda al guardar el archivo que el cursor al inicio de una línea vacía, en la Figura 7.2 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 7.3 se muestra la información para el ejemplo, claramente se nota la organización de la información.





**Figura 7.2:** Almacenamiento de los datos en bloc de notas usando la barra espaciadora



**Figura 7.3:** Almacenamiento de los datos en bloc de notas usando la barra tabuladora



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

## 7.2. 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 una 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 7.1 en Excel y bloc de notas para practicar la lectura de base de datos desde R.

Lo primero que se debe hacer para realizar lo solicitado es construir tres archivos (uno de Excel y dos bloc de notas) igual a los mostrados en las figuras 7.1, 7.2 y 7.3, 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 en qué lugar del computador debe buscar el archivo, note que se debe usar el símbolo `/` para que sea una dirección válida. Substituya la dirección del código anterior con la dirección donde se encuentra su archivo para que pueda leer la base de datos.

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 `/`.

### Ejemplo

Leer la base de datos sobre apartamentos usados en la ciudad de Medellín que está disponible en la página web cuya url es: <https://raw.githubusercontent.com/fhernanb/datos/master/aptos2015>

Para leer la base de datos desde una url usamos el siguiente código.

```
enlace <- 'https://raw.githubusercontent.com/fhernanb/datos/master/aptos2015'  
datos <- read.table(file=enlace, header=TRUE)
```

La base de datos ingresada queda en el marco de datos llamado `datos` y ya está disponible para usarla.

---

## 7.3. Lectura de bases de datos en Excel

Algunas veces los datos están disponibles en un archivo estándar de Excel, y dentro de cada archivo hojas con la información a utilizar. En estos casos se recomienda usar el paquete `readxl` (Wickham, 2016) y en particular la función `readxl`. A continuación un ejemplo de cómo proceder en estos casos.

### Ejemplo

En este enlace<sup>1</sup> está disponible un archivo de Excel llamado `BD_Excel.xlsx`, una vez se ha abierto la página donde está alojado el archivo, se debe descargar y guardar en alguna carpeta. El archivo contiene dos bases de datos muy pequeñas, en la primera hoja llamada **Hijos** está la información de un grupo de niños y en la segunda hoja llamada **Padres** está la información de los padres. ¿Cómo se pueden leer las dos bases de datos?

Lo primero que se debe hacer es instalar el paquete `readxl`, la instalación de cualquier paquete en un computador se hace una sola vez y éste quedará instalado para ser usado las veces que se requiera. La función para instalar un paquete cualquiera es `install.packages`, a continuación se muestra el código necesario para instalar el paquete `readxl`.

```
install.packages("readxl")
```

---

<sup>1</sup>[https://github.com/fhernanb/datos/blob/master/BD\\_Excel.xlsx](https://github.com/fhernanb/datos/blob/master/BD_Excel.xlsx)

Una vez instalado el paquete es necesario cargarlo, la función para cargar el paquete en la sesión actual de R es `library`. La instrucción para cargar el paquete es la siguiente:

```
library(readxl)
```



La instalación de un paquete con `install.packages` se hace sólo una vez y no más. Cargar el paquete con `library` en la sesión actual se debe hacer siempre que se vaya a usar el paquete.

Luego de haber cargado el paquete `readxl` se puede usar la función `readxl` para leer la información contenida en las hojas. A continuación el código para crear la base de datos `hijos` contenida en el archivo `BD_Excel.xlsx`.

```
hijos <- read_excel(file.choose(), sheet='Hijos')
hijos # Para ver el contenido
```

```
##  Edad Grado    ComicFav
## 1     8     2    Superman
## 2     6     1     Batman
## 3     9     3     Batman
## 4    10     5 Bob Esponja
## 5     8     4     Batman
## 6     9     4 Bob Esponja
```

A continuación el código para crear la base de datos `padres` contenida en el archivo `BD_Excel.xlsx`.

```
padres <- read_excel('BD_Excel.xlsx', sheet='Padres')
padres # Para ver el contenido
```

```
##  Edad  EstCivil NumHijos
## 1   45   Soltero      1
## 2   50   Casado      0
## 3   35   Casado      3
## 4   65 Divorciado     1
```

La función `readxl` tiene otros parámetros adicionales útiles para leer bases de datos, se recomienda consultar la ayuda de la función escribiendo en la consola `help(read.xl)`.

**Cuadro 7.2:** Base de datos para practicar lectura.

Fuma	Pasatiempo	Num_hermanos	Mesada
Si	Lectura	0	4500
Si	NA	2	2600
No	Correr	4	1000
No	Correr	NA	3990
Si	TV	3	2570
No	TV	1	2371
Si	Correr	1	1389
NA	Correr	0	4589
Si	Lectura	2	NA

---

## EJERCICIOS

Realice los siguiente ejercicios propuestos.

1. En el Cuadro 7.2 se presenta una base de datos sencilla. Almacene la información del cuadro en dos archivos diferentes, en Excel y en bloc de notas. Lea los dos archivos con la función `read.table` y compare los resultados obtenidos con la del Cuadro 7.2 fuente.
2. En la url [https://raw.githubusercontent.com/fhernanb/datos/master/medidas\\_cuerpo](https://raw.githubusercontent.com/fhernanb/datos/master/medidas_cuerpo) están disponibles los datos sobre medidas corporales para un grupo de estudiante de la universidad, use la función `read.table` para leer la base de datos.

# 8

## *Tablas de frecuencia*

Las tablas de frecuencia son muy utilizadas en estadística y R permite crear tablas de una forma sencilla. En este capítulo se explican las principales funciones para la elaboración de tablas.

### 8.1. Tabla de contingencia con `table`

La función `table` sirve para construir tablas de frecuencia de una vía, a continuación la estructura de la función.

```
table(..., exclude, useNA)
```

Los parámetros de la función son:

- `...` espacio para ubicar los nombres de los objetos (variables o vectores) para los cuales se quiere construir la tabla.
- `exclude`: vector con los niveles a remover de la tabla. Si `exclude=NULL` implica que se desean ver los `NA`, lo que equivale a `useNA = 'always'`.
- `useNA`: instrucción de lo que se desea con los `NA`. Hay tres posibles valores para este parámetro: `'no'` si no se desean usar, `'ifany'` y `'always'` si se desean incluir.

#### Ejemplo: tabla de frecuencia de una vía

Considere el vector `fuma` mostrado a continuación y construya una tabla de frecuencias absolutas para los niveles de la variable frecuencia de fumar.

```
fuma <- c('Frecuente', 'Nunca', 'A veces', 'A veces', 'A veces',  
          'Nunca', 'Frecuente', NA, 'Frecuente', NA, 'hola',  
          'Nunca', 'Hola', 'Frecuente', 'Nunca')
```

A continuación se muestra el código para crear la tabla de frecuencias para la variable `fuma`.

```
table(fuma)
```

```
## fuma
##   A veces Frecuente      hola      Hola      Nunca
##         3         4         1         1         4
```

De la tabla anterior vemos que NO aparece el conteo de los NA, para obtenerlo usamos lo siguiente.

```
table(fuma, useNA='always')
```

```
## fuma
##   A veces Frecuente      hola      Hola      Nunca
##         3         4         1         1         4
##      <NA>
##         2
```

Vemos que hay dos niveles errados en la tabla anterior, `Hola` y `hola`. Para construir la tabla sin esos niveles errados usamos lo siguiente.

```
table(fuma, exclude=c('Hola', 'hola'))
```

```
## fuma
##   A veces Frecuente      Nunca      <NA>
##         3         4         4         2
```

Por último construyamos la tabla sin los niveles errados y los NA, a esta última tabla la llamaremos `tabla1` para luego poder usarla. Las instrucciones para hacer esto son las siguientes.

```
tabla1 <- table(fuma, exclude=c('Hola', 'hola', NA))
tabla1
```

```
## fuma
##   A veces Frecuente      Nunca
##         3         4         4
```



Al crear una tabla con la instrucción `table(var1, var2)`, la variable 1 quedará por filas mientras que la variable 2 estará en las columnas.



**Ejemplo: tabla de frecuencia de dos vías**

Considere otro vector `sexo` mostrado a continuación y construya una tabla de frecuencias absolutas para ver cómo se relaciona el sexo con fumar del ejemplo anterior.

```
sexo <- c('Hombre', 'Hombre', 'Hombre', NA, 'Mujer',
          'Casa', 'Mujer', 'Mujer', 'Mujer', 'Hombre', 'Mujer',
          'Hombre', NA, 'Mujer', 'Mujer')
```

Para construir la tabla solicitada usamos el siguiente código.

```
table(sexo, fuma)
```

```
##          fuma
## sexo      A veces Frecuente hola Hola Nunca
## Casa           0           0   0   0   1
## Hombre          1           1   0   0   2
## Mujer           1           3   1   0   1
```

De la tabla anterior vemos que aparecen niveles errados en `fuma` y en `sexo`, para retirarlos usamos el siguiente código incluyendo en el parámetro `exclude` un vector con los niveles que **NO** deseamos en la tabla.

```
tabla2 <- table(sexo, fuma, exclude=c('Hola', 'hola', 'Casa', NA))
tabla2
```

```
##          fuma
## sexo      A veces Frecuente Nunca
## Hombre          1           1   2
## Mujer           1           3   1
```

**8.2. Función *prop.table***

La función `prop.table` se utiliza para crear tablas de frecuencia relativa a partir de tablas de frecuencia absoluta, la estructura de la función se muestra a continuación.

```
prop.table(x, margin=NULL)
```

- **x**: tabla de frecuencia.
- **margin**: valor de 1 si se desean proporciones por filas, 2 si se desean por columnas, **NULL** si se desean frecuencias globales.

### Ejemplo: tabla de frecuencia relativa de una vía

Obtener la tabla de frecuencia relativa para la **tabla1**.

Para obtener la tabla solicitada se usa el siguiente código.

```
prop.table(x=tabla1)
```

```
## fuma
##   A veces Frecuente   Nunca
##   0.2727    0.3636    0.3636
```

### Ejemplo: tabla de frecuencia relativa de dos vías

Obtener la tabla de frecuencia relativa para la **tabla2**.

Si se desea la tabla de frecuencias relativas global se usa el siguiente código. El resultado se almacena en el objeto **tabla3** para ser usado luego.

```
tabla3 <- prop.table(x=tabla2)
tabla3
```

```
##           fuma
## sexo      A veces Frecuente  Nunca
##  Hombre  0.1111    0.1111 0.2222
##   Mujer  0.1111    0.3333 0.1111
```

Si se desea la tabla de frecuencias relativas marginal por **columnas** se usa el siguiente código.

```
tabla4 <- prop.table(x=tabla2, margin=2)
tabla4
```

```
##           fuma
## sexo      A veces Frecuente  Nunca
##  Hombre  0.5000    0.2500 0.6667
##   Mujer  0.5000    0.7500 0.3333
```

### 8.3. Función *addmargins*

Esta función se puede utilizar para agregar los totales por filas o por columnas a una tabla de frecuencia absoluta o relativa. La estructura de la función es la siguiente.

```
addmargins(A, margin)
```

- A: tabla de frecuencia.
- margin: valor de 1 si se desean proporciones por columnas, 2 si se desean por filas, NULL si se desean frecuencias globales.

#### Ejemplo

Obtener las tablas `tabla3` y `tabla4` con los totales margenes global y por columnas respectivamente.

Para hacer lo solicitado usamos las siguientes instrucciones.

```
addmargins(tabla3)
```

```
##          fuma
## sexo      A veces Frecuente  Nunca    Sum
##  Hombre  0.1111    0.1111 0.2222 0.4444
##   Mujer  0.1111    0.3333 0.1111 0.5556
##    Sum   0.2222    0.4444 0.3333 1.0000
```

```
addmargins(tabla4, margin=1)
```

```
##          fuma
## sexo      A veces Frecuente  Nunca
##  Hombre  0.5000    0.2500 0.6667
##   Mujer  0.5000    0.7500 0.3333
##    Sum   1.0000    1.0000 1.0000
```



Note que los valores de 1 y 2 en el parámetro `margin` de las funciones `prop.table` y `addmargins` significan lo contrario.

### 8.4. Función `hist`

Construir tablas de frecuencias para variables cuantitativas es necesario en muchos procedimientos estadísticos, la función `hist` sirve para obtener este tipo de tablas. La estructura de la función es la siguiente.

```
hist(x, breaks='Sturges', include.lowest=TRUE, right=TRUE,
      plot=FALSE)
```

Los parámetros de la función son:

- `x`: vector numérico.
- `breaks`: vector con los límites de los intervalos. Si no se especifica se usar la regla de Sturges para definir el número de intervalos y el ancho.
- `include.lowest`: valor lógico, si `TRUE` una observación  $x_i$  que coincida con un límite de intervalo será ubicada en el intervalo izquierdo, si `FALSE` será incluida en el intervalo a la derecha.
- `right`: valor lógico, si `TRUE` los intervalos serán cerrados a derecha de la forma  $(lim_{inf}, lim_{sup}]$ , si es `FALSE` serán abiertos a derecha.
- `plot`: valor lógico, si `FALSE` sólo se obtiene la tabla de frecuencias mientras que con `TRUE` se obtiene la representación gráfica llamada histograma.

### Ejemplo

Genere 200 observaciones aleatorias de una distribución normal con media  $\mu = 170$  y desviación  $\sigma = 5$ , luego construya una tabla de frecuencias para la muestra obtenida usando (a) la regla de Sturges y (b) tres intervalos con límites 150, 170, 180 y 190.

Primero se construye el vector `x` con las observaciones de la distribución normal por medio de la función `rnorm` y se especifica la media y desviación solicitada. Luego se aplica la función `hist` con el parámetro `breaks='Sturges'`, a continuación el código utilizado.

```
x <- rnorm(n=200, mean=170, sd=5)

res1 <- hist(x=x, breaks='Sturges', plot=FALSE)
res1

## $breaks
## [1] 155 160 165 170 175 180 185
##
```

```
## $counts
## [1]  4 31 61 70 26  8
##
## $density
## [1] 0.004 0.031 0.061 0.070 0.026 0.008
##
## $mids
## [1] 157.5 162.5 167.5 172.5 177.5 182.5
##
## $xname
## [1] "x"
##
## $equidist
## [1] TRUE
##
## attr("class")
## [1] "histogram"
```

El objeto `res1` es una lista donde se encuentra la información de la tabla de frecuencias para `x`. Esa lista tiene en el elemento `breaks` los límites inferior y superior de los intervalos y en el elemento `counts` están las frecuencias de cada uno de los intervalos.

Para obtener las frecuencias de tres intervalos con límites 150, 170, 180 y 190 se especifica en el parámetros `breaks` los límites. El código para obtener la segunda tabla de frecuencias se muestra a continuación.

```
res2 <- hist(x=x, plot=FALSE,
             breaks=c(150, 170, 180, 190))
res2
```

```
## $breaks
## [1] 150 170 180 190
##
## $counts
## [1] 96 96  8
##
## $density
## [1] 0.024 0.048 0.004
##
## $mids
## [1] 160 175 185
##
## $xname
## [1] "x"
##
```

```
## $equidist  
## [1] FALSE  
##  
## attr("class")  
## [1] "histogram"
```

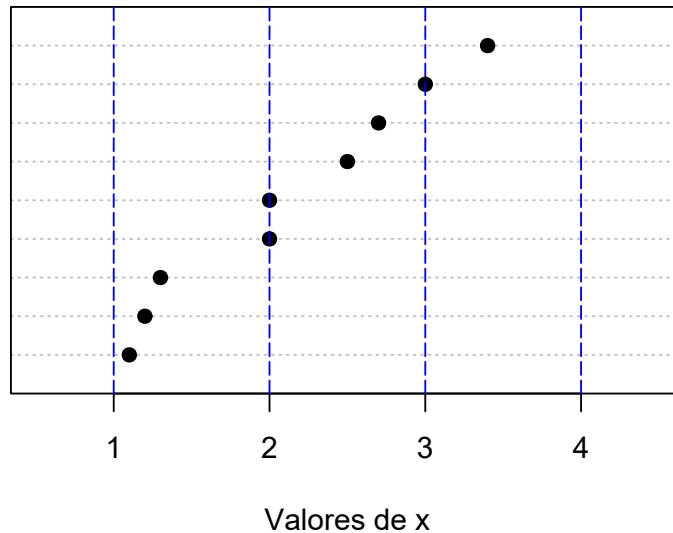
### Ejemplo

Construya el vector **x** con los siguientes elementos: 1.0, 1.2, 1.3, 2.0, 2.5, 2.7, 3.0 y 3.4. Obtenga varias tablas de frecuencia con la función **hist** variando los parámetros **include.lowest** y **right**. Use como límite de los intervalos los valores 1, 2, 3 y 4.

Lo primero que debemos hacer es crear el vector **x** solicitado así:

```
x <- c(1.1, 1.2, 1.3, 2.0, 2.0, 2.5, 2.7, 3.0, 3.4)
```

En la Figura 8.1 se muestran los 9 puntos y con color azul se representan los límites de los intervalos.



**Figura 8.1:** Ubicación de los puntos del ejemplo con límites en color azul.

A continuación se presenta el código para obtener la tabla de frecuencia usando `right=TRUE`, los resultados se almacenan en el objeto `res3` y se solicitan sólo los dos primeros elementos que corresponden a los límites y frecuencias.

```
res3 <- hist(x, breaks=c(1, 2, 3, 4), right=TRUE, plot=FALSE)
res3[1:2]
```

```
## $breaks
## [1] 1 2 3 4
##
## $counts
## [1] 5 3 1
```

Ahora vamos a repetir la tabla pero usando `right=FALSE` para ver la diferencia, en `res4` están los resultados.

```
res4 <- hist(x, breaks=c(1, 2, 3, 4), right=FALSE, plot=FALSE)
res4[1:2]
```

```
## $breaks
## [1] 1 2 3 4
##
## $counts
## [1] 3 4 2
```

Al comparar los últimos dos resultados vemos que la primera frecuencia es 5 cuando `right=TRUE` porque los intervalos se consideran cerrados a la derecha.

Ahora vamos a construir una tabla de frecuencia usando `FALSE` para los parámetros `include.lowest` y `right`.

```
res5 <- hist(x, breaks=c(1, 2, 3, 4),
             include.lowest=FALSE, right=FALSE,
             plot=FALSE)
res5[1:2]
```

```
## $breaks
## [1] 1 2 3 4
##
## $counts
## [1] 3 4 2
```

De este último resultado se ve claramente el efecto de los parámetros `include.lowest` y `right` en la construcción de tablas de frecuencia.

---

## EJERCICIOS

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

En el Cuadro 7.2 se presenta una base de datos sencilla. Lea la base de datos usando la función `read.table` y construya lo que se solicita a continuación.

1. Construya una tabla de frecuencia absoluta para la variable pasatiempo.
2. Construya una tabla de frecuencia relativa para la variable fuma.
3. Construya una tabla de frecuencia relativa para las variables pasatiempo y fuma.
4. ¿Qué porcentaje de los que no fuman tienen como pasatiempo la lectura.
5. ¿Qué porcentaje de los que corren no fuman?



## 9

### *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
```

---

### 9.1. 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.

---

## 9.2. 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
```

---

### 9.3. 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.



# 10

## 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
```

### 10.1. 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.

---

## 10.2. 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** ( $\sigma$ ) para el siguiente conjunto de 5 observaciones: 12, 25, 32, 15, 26.

Recordemos que las expresiones matemáticas para obtener  $S$  y  $\sigma$  son muy similares, la diferencia está en el denominador, para  $S$  el denominador es  $n - 1$  mientras que para  $\sigma$  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
```

### 10.3. 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.

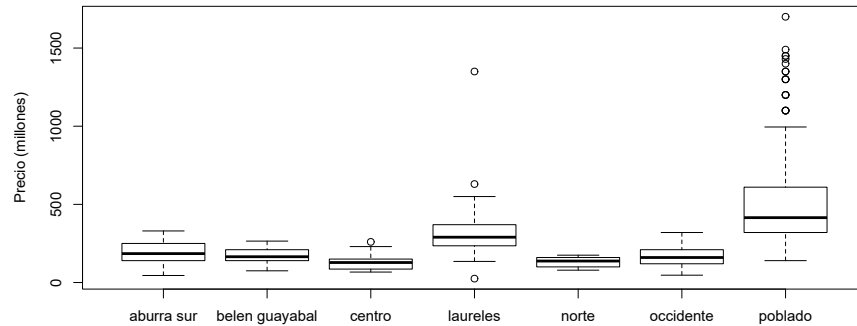
Para realizar esto debemos usar en conjunto la función `split`, `sapply` y `var` ya que se quiere la varianza de una variable (`precio`) dado los valores de otra variable (`ubicacion`). El código para obtener las varianzas es el siguiente.

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

```
##      aburra sur belen guayabal      centro
##      4169              2528      2588
##      laureles          norte  occidente
##      25351            1009      3596
##      poblado
##      84497
```

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 10.1 se muestra el boxplot y se ve claramente la dispersión de los precios en el Poblado. El código usado para generar la Figura 10.1 se presenta a continuación.

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



**Figura 10.1:** 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  $3 \times 3$ , 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  $3 \times 3$ .

---

## 10.4. 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
```



# 11

## *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
```

---

### 11.1. 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
```



# 12

## *Medidas de correlación*

En este capítulo se mostrará cómo obtener el coeficiente de correlación lineal para variables cuantitativas.

### 12.1. Función `cor`

La función `cor` permite calcular el coeficiente de correlación de Pearson, Kendall o Spearman para dos variables cuantitativas. La estructura de la función es la siguiente.

```
cor(x, y, use="everything",  
    method=c("pearson", "kendall", "spearman"))
```

Los parámetros de la función son:

- `x`, `y`: vectores cuantitativos.
- `use`: parámetro que indica lo que se debe hacer cuando se presenten registros NA en alguno de los vectores. Las diferentes posibilidades son: `everything`, `all.obs`, `complete.obs`, `na.or.complete` y `pairwise.complete.obs`, el valor por defecto es `everything`.
- `method`: tipo de coeficiente de correlación a calcular, por defecto es `pearson`, otros valores posibles son `kendall` y `spearman`.

### Ejemplo

Calcular el coeficiente de correlación de Pearson para las variables área y precio de la base de datos sobre apartamentos usados.

Lo primero que se debe hacer es cargar la base de datos usando la url apropiada. Luego de esto se usa la función `cor` sobre las variables de interés. A continuación se muestra el código necesario.

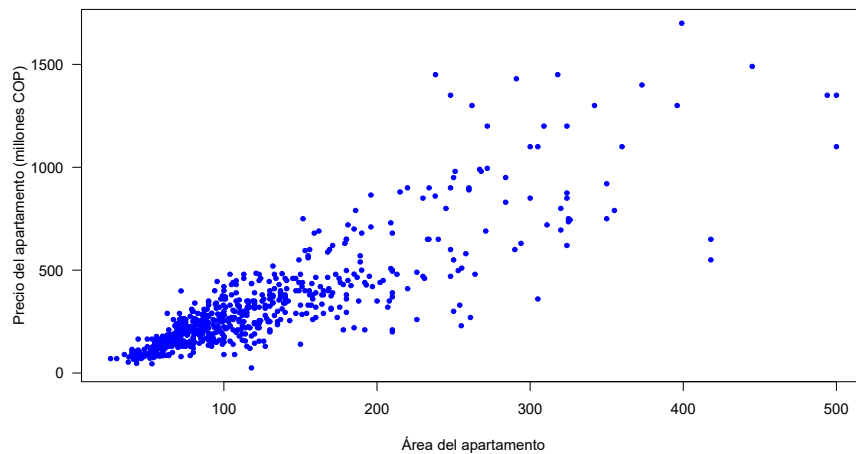
```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/aptos2015'
datos <- read.table(file=url, header=T)
cor(x=datos$mt2, y=datos$precio)
```

```
## [1] 0.8583
```

Del resultado anterior vemos que existe una correlación de 0.8583 entre las dos variables, eso significa que apartamentos de mayor área tienden a tener precios de venta más alto. Este resultado se ilustra en la Figura 12.1, se nota claramente que la nube de puntos tiene un pendiente positiva y por eso el signo del coeficiente de correlación.

A continuación el código para generar la Figura 12.1.

```
with(datos, plot(x=mt2, y=precio, pch=20, col='blue',
                 xlab='Área del apartamento', las=1,
                 ylab='Precio del apartamento (millones COP)'))
```



**Figura 12.1:** Diagrama de dispersión para precio versus área de los apartamentos usados.

## Ejemplo

Para las mismas variables del ejemplo anterior calcular los coeficientes de correlación Kendall y Spearman.

A continuación el código para obtener lo solicitado.

```
cor(x=datos$mt2, y=datos$precio, method='pearson')
```

```
## [1] 0.8583
```

```
cor(x=datos$mt2, y=datos$precio, method='kendall')
```

```
## [1] 0.6911
```

```
cor(x=datos$mt2, y=datos$precio, method='spearman')
```

```
## [1] 0.8603
```

## Ejemplo

Para la base de datos de apartamentos usados, ¿cuáles de las variables cuantitativas tienen mayor correlación?

Lo primero que debemos hacer es determinar cuáles son las cuantitativas de la base de datos. Para obtener información de las variables que están almacenadas en el marco de datos llamado `datos` usamos la función `str` que muestra la estructura interna de objeto.

```
str(datos)
```

```
## 'data.frame':    694 obs. of  11 variables:
## $ precio       : num  79 93 100 123 135 140 145 160 160 175 ...
## $ mt2          : num  43.2 56.9 66.4 61.9 89.8 ...
## $ ubicacion    : Factor w/ 7 levels "aburra sur","belen guayabal",...: 5 5 5 5 5 5 5 5
## $ estrato      : int   3 2 3 2 4 3 3 3 4 4 ...
## $ alcobas      : int   3 2 2 3 3 3 2 3 4 3 ...
## $ banos        : int   1 1 2 2 2 2 2 2 2 2 ...
## $ balcon       : Factor w/ 2 levels "no","si": 2 2 1 2 2 1 2 2 2 2 ...
## $ parqueadero  : Factor w/ 2 levels "no","si": 2 2 1 2 1 2 2 2 1 2 ...
## $ administracion: num   0.05 0.069 0 0.13 0 0.12 0.14 0.127 0 0.123 ...
## $ avaluo       : num   14.9 27 15.7 27 39.6 ...
## $ terminado    : Factor w/ 2 levels "no","si": 1 2 1 1 2 2 2 2 2 2 ...
```

Del anterior resultado vemos que las variables `precio`, `mt2`, `alcobas`, `banos`, `administracion` y `avaluo` son las variables cuantitativas, las restantes son cualitativas (nominal u ordinal). Las posiciones de las variables cuantitativas en

el objeto `datos` son 1, 2, 5, 6, 9, 10, así podemos construir un marco de datos sólo con la información cuantitativa, a continuación el código usado.

```
datos.cuanti <- datos[, c(1, 2, 5, 6, 9, 10)]
# La siguiente instrucción para editar los nombres de las variables
colnames(datos.cuanti) <- c('Precio', 'Área', 'Alcobas',
                             'Baños', 'Admon', 'Avaluo')
M <- round(cor(datos.cuanti), digits=2)
M
```

```
##          Precio Área Alcobas Baños Admon Avaluo
## Precio    1.00 0.86   0.19  0.63  0.75  0.79
## Área      0.86 1.00   0.31  0.67  0.77  0.75
## Alcobas   0.19 0.31   1.00  0.35  0.16  0.15
## Baños     0.63 0.67   0.35  1.00  0.55  0.53
## Admon     0.75 0.77   0.16  0.55  1.00  0.70
## Avaluo    0.79 0.75   0.15  0.53  0.70  1.00
```

El anterior resultado representa la matriz de correlaciones entre las variables cuantitativas, se observa que la mayor correlación es entre las variables precio y área del apartamento.

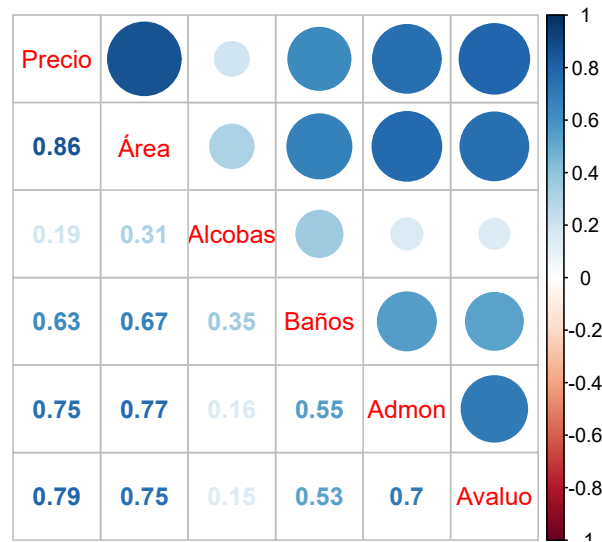
Es posible representar gráficamente la matriz de correlaciones `M` por medio de la función `corrplot` del paquete **corrplot** (Wei and Simko, 2016), a continuación el código para obtener su representación gráfica.

```
library('corrplot') # Para cargar el paquete corrplot
corrplot.mixed(M)
```

En la Figura 12.2 se muestra la matriz con los coeficientes de correlación. En la diagonal de la Figura 12.2 están las variables, por encima están unos círculos de colores, entre más intensidad del color, ya sea azul o rojo, mayor es la correlación, colores ténues significan correlación baja; el tamaño de los círculos está asociado al valor absoluto de correlación. Por debajo de la diagonal se observan los valores exactos de correlación en colores.



La función `corrplot` es muy versátil, se pueden obtener diferentes representaciones gráficas de la matriz de correlaciones, para conocer las diferentes posibilidades recomendamos consultar este enlace: <https://cran.r-project.org/web/packages/corrplot/vignettes/corrplot-intro.html>.



**Figura 12.2:** Matriz de coeficientes de correlación.

### Ejemplo

Construya dos vectores hipotéticos con el gasto y ahorro de un grupo de 7 familias, incluya dos NA. Calcule el coeficiente de correlación entre **ahorro** y **gasto**, use el parámetro **use** para manejar los NA.

A continuación se presenta el código para crear los objetos **ahorro** y **gasto** con datos ficticios. Observe que en el primer caso donde se calcula la correlación no es posible obtener un resultado debido a que por defecto **use='everything'** y por lo tanto usa todas las observaciones incluyendo los NA. En el segundo caso si se obtiene un valor para la correlación debido a que se usó **use='complete.obs'**.

```
gasto <- c(170, 230, 120, 156, 256, NA, 352)
ahorro <- c(45, 30, NA, 35, 15, 65, 15)

cor(gasto, ahorro)
```

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

```
cor(gasto, ahorro, use='complete.obs')
```

```
## [1] -0.8465
```

---

## EJERCICIOS

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

1. Para cada uno de los estratos socioeconómicos, calcular el coeficiente de correlación lineal de Pearson para las variables precio y área de la base de datos de los apartamentos usados.
2. Calcular los coeficientes de correlación Pearson, Kendall y Spearman para las variables cuantitativas de la base de datos sobre medidas del cuerpo explicada en el Capítulo 9. La url con la información es la siguiente: [https://raw.githubusercontent.com/fhernanb/datos/master/medidas\\_cuerpo](https://raw.githubusercontent.com/fhernanb/datos/master/medidas_cuerpo)
3. Represente gráficamente las matrices de correlación obtenidas en el ejercicio anterior.

# 13

## *Distribuciones discretas*

En este capítulo se mostrarán las funciones de R para distribuciones discretas.

### 13.1. Funciones disponibles para distribuciones discretas

Para cada distribución discreta se tienen 4 funciones, a continuación el listado de funciones y su utilidad.

```
dxxx(x, ...) # Función de masa de probabilidad,  $f(x)$ 
pxxx(q, ...) # Función de distribución acumulada hasta  $q$ ,  $F(x)$ 
qxxx(p, ...) # Cuantil para el cual  $P(X \leq q) = p$ 
rxxx(n, ...) # Generador de números aleatorios.
```

En el lugar de las letras xxx se debe colocar el nombre de la distribución en R, a continuación el listado de nombres disponibles para las 6 distribuciones discretas básicas.

```
binom      # Binomial
geo        # Geométrica
nbinom     # Binomial negativa
hyper      # Hipergeométrica
pois       # Poisson
multinom   # Multinomial
```

Combinando las funciones y los nombres se tiene un total de 24 funciones, por ejemplo, para obtener la función de masa de probabilidad  $f(x)$  de una binomial se usa la función `dbinom( )` y para obtener la función acumulada  $F(x)$  de una Poisson se usa la función `ppois( )`.

### Ejemplo binomial

Suponga que un grupo de agentes de tránsito sale a una vía principal para revisar el estado de los buses de transporte intermunicipal. De datos históricos se sabe que un 10% de los buses generan una mayor cantidad de humo de la permitida. En cada jornada los agentes revisan siempre 18 buses, asuma que el estado de un bus es independiente del estado de los otros buses.

- 1) Calcular la probabilidad de que se encuentren exactamente 2 buses que generan una mayor cantidad de humo de la permitida.

Aquí se tiene una distribución  $Binomial(n = 18, p = 0.1)$  y se desea calcular  $P(X = 2)$ . Para obtener esta probabilidad se usa la siguiente instrucción.

```
dbinom(x=2, size=18, prob=0.10)
```

```
## [1] 0.2835
```

Así  $P(X = 2) = 0.2835$ .

- 2) Calcular la probabilidad de que el número de buses que sobrepasan el límite de generación de gases sea al menos 4.

En este caso interesa calcular  $P(X \geq 4)$ , para obtener esta probabilidad se usa la siguiente instrucción.

```
sum(dbinom(x=4:18, size=18, prob=0.10))
```

```
## [1] 0.0982
```

Así  $P(X \geq 4) = 0.0982$

- 3) Calcular la probabilidad de que tres o menos buses emitan gases por encima de lo permitido en la norma.

En este caso interesa  $P(X \leq 3)$  lo cual es  $F(x = 3)$ , por lo tanto, la instrucción para obtener esta probabilidad es

```
pbinom(q=3, size=18, prob=0.10)
```

```
## [1] 0.9018
```

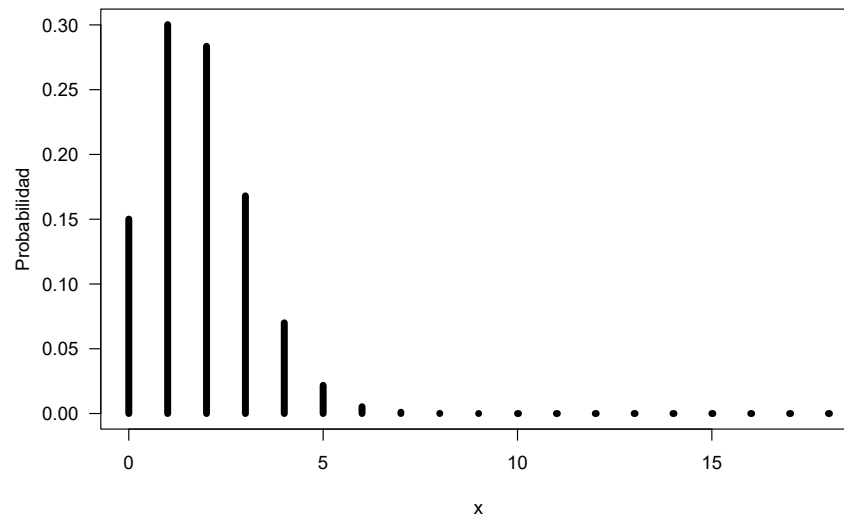
Así  $P(X \leq 3) = F(x = 3) = 0.9018$

- 4) Dibujar la función de masa de probabilidad.



Para dibujar la función de masa de probabilidad para una  $\text{Binomial}(n = 18, p = 0.1)$  se usa el siguiente código.

```
x <- 0:18 # Soporte (dominio) de la variable
Probabilidad <- dbinom(x=x, size=18, prob=0.1)
plot(x=x, y=Probabilidad,
      type='h', las=1, lwd=6)
```



**Figura 13.1:** Función de masa de probabilidad para una  $\text{Binomial}(n = 18, p = 0.1)$ .

En la Figura 13.1 se muestra la función de masa de probabilidad para la  $\text{Binomial}(n = 18, p = 0.1)$ , de esta figura se observa claramente que la mayor parte de la probabilidad está concentrada para valores pequeños de  $X$  debido a que la probabilidad de éxito individual es  $p = 0.10$ . Valores de  $X \geq 7$  tienen una probabilidad muy pequeña y es por eso que las longitudes de sus barras son muy cortas.

- 5) Generar con 100 de una distribución  $\text{Binomial}(n = 18, p = 0.1)$  y luego calcular las frecuencias muestrales y compararlas con las probabilidades teóricas.

La muestra aleatoria se obtiene con la función `rbinom` y los resultados se

almacenan en el objeto `m`, por último se construye la tabla de frecuencias relativas, a continuación el código usado.

```
m <- rbinom(n=100, size=18, prob=0.1)
m # Para ver lo que hay dentro de m

## [1] 4 1 3 4 2 3 1 0 3 2 1 4 1 1 1 3 0 5 0 3 5 3 2 3 1
## [26] 1 2 4 1 3 2 2 3 2 2 3 2 3 0 2 2 2 1 1 3 1 1 1 1 1
## [51] 1 1 1 3 1 2 4 4 7 2 4 0 1 2 1 1 2 3 2 1 3 1 1 2 3
## [76] 5 1 5 0 1 2 1 4 1 1 2 1 2 0 2 2 3 3 0 1 1 3 2 3 0
```

```
prop.table(table(m)) # Tabla de frecuencia relativa
```

```
## m
## 0 1 2 3 4 5 7
## 0.09 0.34 0.24 0.20 0.08 0.04 0.01
```

A pesar de ser una muestra aleatoria de sólo 100 observaciones, se observa que las frecuencias relativas obtenidas son muy cercanas a las mostradas en la Figura 13.1.

### Ejemplo geométrica

En una línea de producción de bombillos se sabe que sólo el 1 % de los bombillos son defectuosos. Una máquina automática toma un bombillo y lo prueba, si el bombillo enciende, se siguen probando los bombillos hasta que se encuentre **un** bombillo defectuoso, ahí se para la línea de producción y se toman los correctivos necesarios para mejorar el proceso.

- 1) Calcular la probabilidad de que se necesiten probar 125 bombillos para encontrar el primer bombillo defectuoso.

En la distribución geométrica, la variable  $X$  representa el número de fracasos antes de encontrar el único éxito, por lo tanto, en este caso el interés es calcular  $P(X = 124)$ . La instrucción para obtener esta probabilidad es la siguiente.

```
dgeom(x=124, prob=0.01)
```

```
## [1] 0.002876
```

- 2) Calcular  $P(X \leq 8)$ .

En este caso interesa  $P(X \leq 50)$  lo que equivale a  $F(8)$ , la instrucción para obtener la probabilidad es la siguiente.

```
pgeom(q=50, prob=0.01)
```

```
## [1] 0.401
```

3) Encontrar el cuantil  $q$  tal que  $P(X \leq q) = 0.40$ .

En este caso interesa encontrar el cuantil  $q$  que cumpla la condición de que hasta  $q$  esté el 40 % de las observaciones, por esa razón se usa la función `qgeom` como se muestra a continuación.

```
qgeom(p=0.4, prob=0.01)
```

```
## [1] 50
```



Note que las funciones `pxxx` y `qxxx` están relacionadas, `pxxx` entrega la probabilidad hasta el cuantil  $q$  mientras `qxxx` entrega el cuantil en el que se acumula  $p$  probabilidad.

### Ejemplo binomial negativa

Una familia desea tener hijos hasta conseguir **2 niñas**, la probabilidad individual de obtener una niña es 0.5 y se supone que todos los nacimientos son individuales, es decir, un sólo bebé.

- 1) Calcular la probabilidad de que se necesiten 4 hijos, es decir, 4 nacimientos para conseguir las dos niñas.

En este problema se tiene una distribución binomial negativa con  $r = 2$  niñas, los éxitos deseados por la familia. La variable  $X$  representa los fracasos, es decir los niños, hasta que se obtienen los éxitos  $r = 2$  deseados.

En este caso lo que interesa es  $P(\text{familia tenga } 4)$ , en otras palabras interesa  $P(X = 2)$ , la instrucción para calcular la probabilidad es la siguiente.

```
dnbinom(x=2, size=2, prob=0.5)
```

```
## [1] 0.1875
```

- 2) Calcular  $P(\text{familia tenga al menos 4 hijos})$ .

Aquí interesa calcular  $P(X \geq 2) = P(X = 2) + P(X = 3) + \dots$ , como esta probabilidad va hasta infinito, se debe usar el complemento así:

$$P(X \geq 2) = 1 - [P(X = 0) + P(X = 1)]$$

y para obtener la probabilidad solicitada se puede usar la función `dnbinom` de la siguiente manera.

```
1 - sum(dnbinom(x=0:1, size=2, prob=0.5))
```

```
## [1] 0.5
```

Otra forma para obtener la probabilidad solicitada es por medio de la función `pnbinom` de la siguiente manera.

```
1 - pnbinom(q=1, size=2, prob=0.5)
```

```
## [1] 0.5
```

### Ejemplo hipergeométrica

Un lote de partes para ensamblar en una empresa está formado por 100 elementos del proveedor A y 200 elementos del proveedor B. Se selecciona una muestra de 4 partes al azar sin reemplazo de las 300 para una revisión de calidad.

- 1) Calcular la probabilidad de que todas las 4 partes de la muestra sean del proveedor A.

Aquí se tiene una situación que se puede modelar por medio de una distribución hipergeométrica con  $m = 100$  éxitos en la población,  $n = 200$  fracasos en la población y  $k = 4$  el tamaño de la muestra. El objetivo es calcular  $P(X = 4)$ , para obtener esta probabilidad se usa la siguiente instrucción.

```
dhyper(x=4, m=100, k=4, n=200)
```

```
## [1] 0.01185
```

- 2) Calcular la probabilidad de que dos o más de las partes sean del proveedor A.

Aquí interesa  $P(X \geq 2)$ , la instrucción para obtener esta probabilidad es.

```
sum(dhyper(x=2:4, m=100, k=4, n=200))
```

```
## [1] 0.4074
```

### Ejemplo Poisson

En una editorial se asume que todo libro de 250 páginas tiene en promedio 50 errores.

- 1) Encuentre la probabilidad de que en una página cualquiera no se encuentren errores.

Este es un problema de distribución Poisson con tasa promedio de éxitos dada por:

$$\lambda = \frac{50 \text{ errores}}{\text{libro}} = \frac{0.2 \text{ errores}}{\text{pagina}}$$

El objetivo es calcular  $P(X = 0)$ , para obtener esta probabilidad de usa la siguiente instrucción.

```
dpois(x=0, lambda=0.2)
```

```
## [1] 0.8187
```

Así  $P(X = 0) = 0.8187$ .

---

## 13.2. Distribuciones discretas generales

En la práctica nos podemos encontrar con variables aleatorias discretas que no se ajustan a una de las distribuciones mostradas anteriormente, en esos casos, es posible manejar ese tipo de variables por medio de unas funciones básicas de R como se muestra en el siguiente ejemplo.

### Ejemplo

El cangrejo de herradura hembra se caracteriza porque a su caparazón se adhieren los machos de la misma especie, en la Figura 13.2 se muestra una fotografía de este cangrejo. Los investigadores están interesados en determinar



**Figura 13.2:** Fotografía del cangrejo de herradura, tomada de <http://sccoastalresources.com/home/2016/6/21/a-night-of-horseshoe-crab-tagging>.

cual es el patrón de variación del número de machos sobre cada hembra, para esto, se recolectó una muestra de hembras a las cuales se les observó el color, la condición de la espina, el peso en kilogramos, el ancho del caparazón en centímetros y el número de satélites o machos sobre el caparazón, la base de datos está disponible en el siguiente enlace<sup>1</sup>.

- 1) Encontrar la distribución de probabilidad para la variable **Sa** que corresponde al número de machos sobre el caparazón de cada hembra.

Primero se debe leer la base de datos usando la url suministrada y luego se construye la tabla de frecuencia relativa y se almacena en el objeto **t1**.

```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/crab'
crab <- read.table(file=url, header=T)

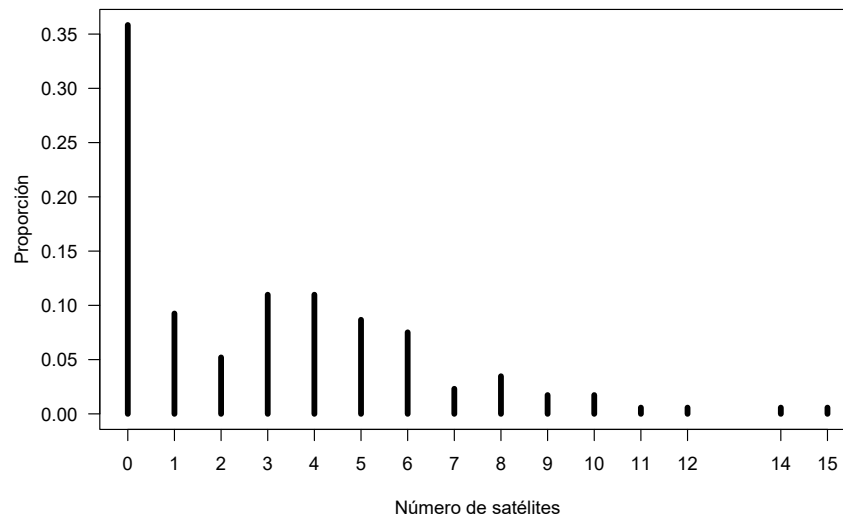
t1 <- prop.table(table(crab$Sa))
t1
```

<sup>1</sup><https://raw.githubusercontent.com/fhernanb/datos/master/crab>

```
##
##      0      1      2      3      4      5
## 0.35838 0.09249 0.05202 0.10983 0.10983 0.08671
##      6      7      8      9     10     11
## 0.07514 0.02312 0.03468 0.01734 0.01734 0.00578
##     12     14     15
## 0.00578 0.00578 0.00578
```

La anterior tabla de frecuencias relativas se puede representar gráficamente usando el siguiente código.

```
plot(t1, las=1, lwd=5, xlab='Número de satélites',
     ylab='Proporción')
```



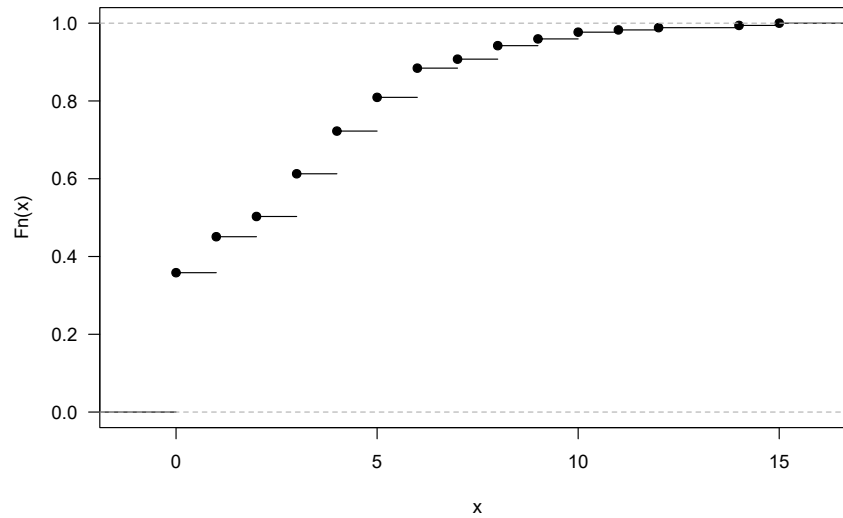
**Figura 13.3:** Función de masa de probabilidad para el número de satélites por hembra.

- 2) Sea  $X$  la variable número de satélites por hembra, construir la función  $F(x)$ .

Para construir  $F(x)$  se utiliza la función `ecdf` o empirical cumulative density function, a esta función le debe ingresar el vector con la información de la variable cuantitativa, a continuación del código usado. En la Figura 13.4 se

muestra la función de distribución acumulada para el número de satélites por hembra.

```
F <- ecdf(crab$Sa)
plot(F, las=1, main='')
```



**Figura 13.4:** Función de distribución acumulada para el número de satélites por hembra.

3) Calcular  $P(X \leq 9)$ .

Para obtener esta probabilidad se usa el objeto F que es en realidad una función, a continuación la instrucción usada.

```
F(9)
```

```
## [1] 0.9595
```

Así  $P(X \leq 9) = 0.9595$ .

4) Calcular  $P(X > 4)$ .

Para obtener esta probabilidad se usa el hecho de que  $P(X > 4) = 1 - P(X \leq 4)$ , así la instrucción a usar es.



```
1 - F(4)
```

```
## [1] 0.2775
```

Por lo tanto  $P(X > 4) = 0.2775$ .

- 5) Suponga que el grupo 1 está formado por las hembras cuyo ancho de caparazón es menor o igual al ancho mediano, el grupo 2 está formado por las demás hembras. ¿Será  $F(x)$  diferente para los dos grupos?

Para realizar esto vamos a particionar el vector `Sa` en los dos grupos de acuerdo a la nueva variable `grupo` creada como se muestra a continuación.

```
grupo <- ifelse(crab$Wt <= median(crab$Wt), 'Grupo 1', 'Grupo 2')
x <- split(x=crab$Sa, f=grupo)
```

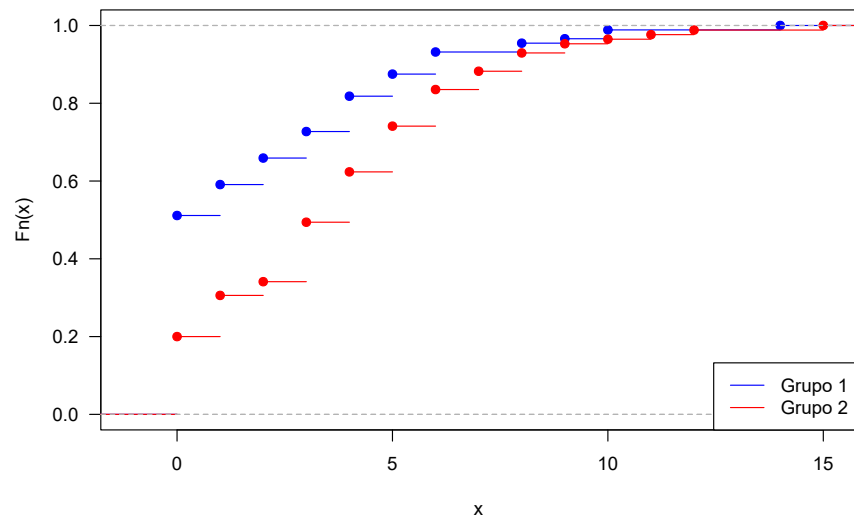
El objeto `x` es una lista y para acceder a los vectores allí almacenados usamos los corchetes `[[ ]]`, uno dentro del otro. Luego para calcular  $F(x)$  para los dos grupos se procede así:

```
F1 <- ecdf(x[[1]])
F2 <- ecdf(x[[2]])
```

Para obtener las dos  $F(x)$  en la misma figura se usa el código siguiente.

```
plot(F1, col='blue', main='', las=1)
plot(F2, col='red', add=T)
legend('bottomright', legend=c('Grupo 1', 'Grupo 2'),
      col=c('blue', 'red'), lwd=1)
```

En la Figura 13.5 se muestran las dos  $F(x)$ , en color azul para el grupo 1 y en color rojo para el grupo 2. Se observa claramente que las curvas son diferentes antes de  $x = 9$ . El hecho de que la curva azul esté por encima de la roja para valores menores de 9, es decir,  $F_1(x) \geq F_2(x)$ , indica que las hembras del grupo 1 tienden a tener menos satélites que las del grupo 2, esto es coherente ya que las del grupo 2 son más grandes en su caparazón.



**Figura 13.5:** Función de distribución acumulada para el número de satélites por hembra diferenciando por grupo.

# 14

## *Distribuciones continuas*

En este capítulo se mostrarán las funciones de R para distribuciones continuas.

### 14.1. Funciones disponibles para distribuciones continuas

Para cada distribución continua se tienen 4 funciones, a continuación el listado de las funciones y su utilidad.

```
dxxx(x, ...) # Función de densidad de probabilidad,  $f(x)$ 
pxxx(q, ...) # Función de distribución acumulada hasta  $q$ ,  $F(x)$ 
qxxx(p, ...) # Cuantil para el cual  $P(X \leq q) = p$ 
rxxx(n, ...) # Generador de números aleatorios.
```

En el lugar de las letras xxx se debe colocar el nombre de la distribución en R, a continuación el listado de nombres disponibles para las 11 distribuciones continuas básicas.

```
beta      # Beta
cauchy    # Cauchy
chisq     # Chi-cuadrada
exp       # Exponencial
f         # F
gamma     # Gama
lnorm     # log-normal
norm      # normal
t         # t-student
unif      # Uniforme
weibull   # Weibull
```

Combinando las funciones y los nombres se tiene un total de 44 funciones, por ejemplo, para obtener la función de densidad de probabilidad  $f(x)$  de una

normal se usa la función `dnorm()` y para obtener la función acumulada  $F(x)$  de una Beta se usa la función `pbeta()`.

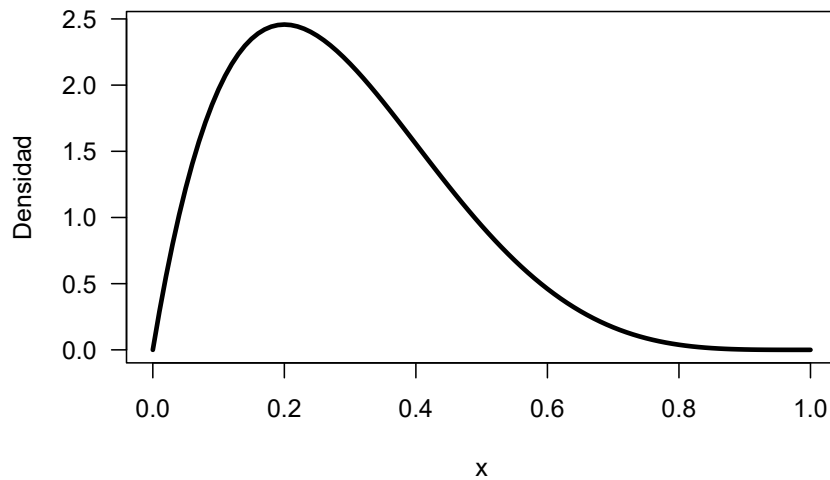
### Ejemplo beta

Considere que una variable aleatoria  $X$  se distribuye beta con parámetros  $a = 2$  y  $b = 5$ .

- 1) Dibuje la densidad de la distribución.

La función `dbeta` sirve para obtener la altura de la curva de una distribución beta y combinándola con la función `curve` se puede dibujar la densidad solicitada. En la Figura 14.1 se presenta la densidad, observe que para la combinación de parámetros  $a = 2$  y  $b = 5$  la distribución es sesgada a la derecha.

```
curve(dbeta(x, shape1=2, shape2=5), lwd=3, las=1,  
      ylab='Densidad')
```



**Figura 14.1:** Función de densidad para una  $Beta(2, 5)$ .

- 2) Calcular  $P(0.3 \leq X \leq 0.7)$ .

Para obtener la probabilidad o área bajo la densidad se puede usar la función `integrate`, los límites de la integral se ingresan por medio de los parámetros `lower` y `upper`. Si la función a integrar tiene parámetros adicionales como en este caso, éstos parámetros se ingresan luego de los límites de la integral. A continuación el código necesario para obtener la probabilidad solicitada.

```
integrate(f=dbeta, lower=0.3, upper=0.7,
          shape1=2, shape2=5)
```

```
## 0.4092 with absolute error < 4.5e-15
```

Otra forma de obtener la probabilidad solicitada es restando de  $F(x_{max})$  la probabilidad  $F(x_{min})$ . Las probabilidades acumuladas hasta un valor dado se obtienen con la función `pbeta`, a continuación el código necesario.

```
pbeta(q=0.7, shape1=2, shape2=5) - pbeta(q=0.3, shape1=2, shape2=5)
```

```
## [1] 0.4092
```

De ambas formas se obtiene que  $P(0.3 \leq X \leq 0.7) = 0.4092$ .



Recuerde que para distribuciones continuas

$$P(a < X < b) = P(a \leq X < b) = P(a < X \leq b) = P(a \leq X \leq b)$$

### Ejemplo normal estándar

Suponga que la variable aleatoria  $Z$  se distribuye normal estándar, es decir,  $Z \sim N(0, 1)$ .

- 1) Calcular  $P(Z < 1.45)$ .

Para calcular la probabilidad acumulada hasta un punto dado se usa la función `pnorm` y se evalúa en el cuantil indicado, a continuación el código usado.

```
pnorm(q=1.45)
```

```
## [1] 0.9265
```

En la Figura 14.2 se muestra el área sombreada correspondiente a  $P(Z < 1.45)$ .

- 2) Calcular  $P(Z > -0.37)$ .

Para calcular la probabilidad solicitada se usa nuevamente la función `pnorm` evaluada en el cuantil dado. Como el evento de interés es  $Z > -0.37$ , la probabilidad solicitada se obtiene como  $1 - \text{pnorm}(q=-0.37)$ , esto debido a que por defecto las probabilidades entregadas por la función `pxxx` son siempre a izquierda. A continuación el código usado.

```
1 - pnorm(q=-0.37)
```

```
## [1] 0.6443
```

En la Figura 14.2 se muestra el área sombreada correspondiente a  $P(Z > -0.37)$ .

Otra forma para obtener la probabilidad solicitada sin hacer la resta es usar el parámetro `lower.tail` para indicar que interesa la probabilidad a la derecha del cuantil dado, a continuación un código alternativo para obtener la misma probabilidad.

```
pnorm(q=-0.37, lower.tail=FALSE)
```

```
## [1] 0.6443
```

3) Calcular  $P(-1.56 < Z < 2.58)$ .

Para calcular la probabilidad solicitada se obtiene la probabilidad acumulada hasta 2.58 y de ella se resta lo acumulado hasta -1.56, a continuación el código usado.

```
pnorm(q=2.58) - pnorm(-1.56)
```

```
## [1] 0.9357
```

En la Figura 14.2 se muestra el área sombreada correspondiente a  $P(-1.56 < Z < 2.58)$ .

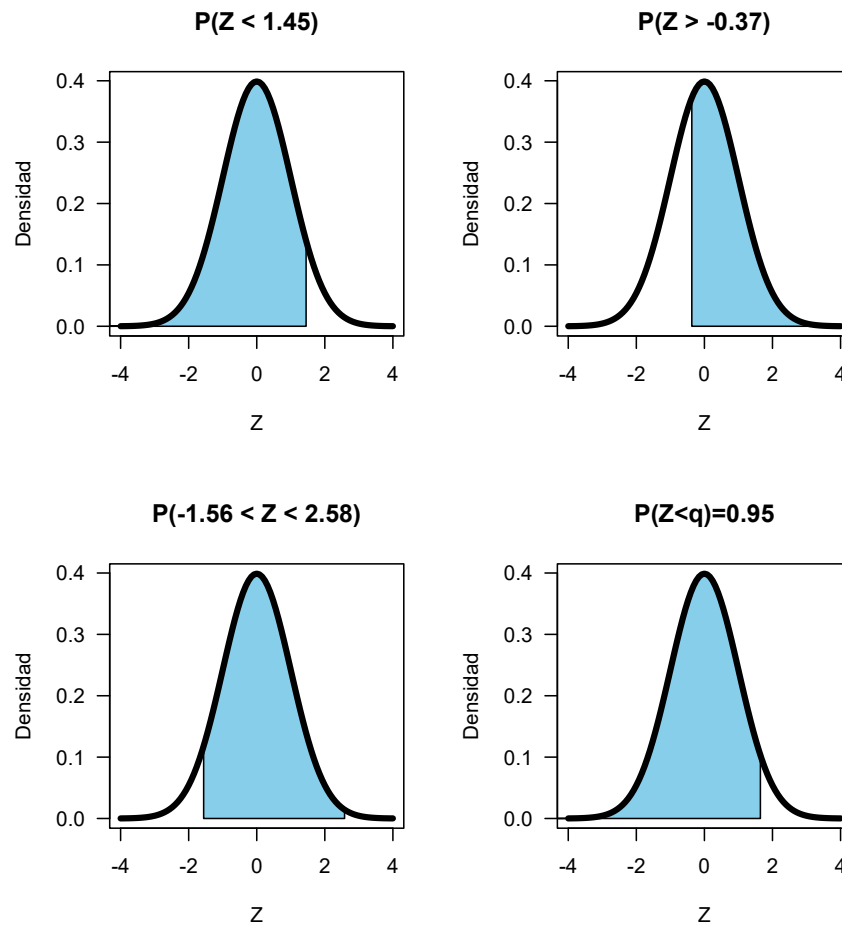
4) Calcular el cuantil  $q$  para el cual se cumple que  $P(Z < q) = 0.95$ .

Para calcular el cuantil en el cual se cumple que  $P(Z < q) = 0.95$  se usa la función `qnorm`, a continuación el código usado.

```
qnorm(p=0.95)
```

```
## [1] 1.645
```

En la Figura 14.2 se muestra el área sombreada correspondiente a  $P(Z < q) = 0.95$ .



**Figura 14.2:** Área sombreada para los ejemplos.



El parámetro `lower.tail` es muy útil para indicar si estamos trabajando una cola a izquierda o una cola a derecha.

### Ejemplo normal general

Considere un proceso de elaboración de tornillos en una empresa y suponga que el diámetro de los tornillos sigue una distribución normal con media de 10 mm y varianza de 4 mm<sup>2</sup>.

- 1) Un tornillo se considera que cumple las especificaciones si su diámetro está entre 9 y 11 mm. ¿Qué porcentaje de los tornillos cumplen las especificaciones?

Como se solicita probabilidad se debe usar `pnorm` indicando que la media es  $\mu = 10$  y la desviación de la distribución es  $\sigma = 2$ . A continuación el código usado.

```
pnorm(q=11, mean=10, sd=2) - pnorm(q=9, mean=10, sd=2)
```

```
## [1] 0.3829
```

- 2) Un tornillo con un diámetro mayor a 11 mm se puede reprocesar y recuperar. ¿Cuál es el porcentaje de reprocesos en la empresa?

Como se solicita una probabilidad a derecha se usa `lower.tail=FALSE` dentro de la función `pnorm`. A continuación el código usado.

```
pnorm(q=11, mean=10, sd=2, lower.tail=FALSE)
```

```
## [1] 0.3085
```

- 3) El 5 % de los tornillos más delgados no se pueden reprocesar y por lo tanto son desperdicio. ¿Qué diámetro debe tener un tornillo para ser clasificado como desperdicio?

Aquí interesa encontrar el cuantil tal que  $P(Diámetro < q) = 0.05$ , por lo tanto se usa la función `qnorm`. A continuación el código usado.

```
qnorm(p=0.05, mean=10, sd=2)
```

```
## [1] 6.71
```

- 4) El 10 % de los tornillos más gruesos son considerados como sobredimensionados. ¿cuál es el diámetro mínimo de un tornillo para que sea considerado como sobredimensionado?



Aquí interesa encontrar el cuantil tal que  $P(\text{Diametro} > q) = 0.10$ , por lo tanto se usa la función `qnorm` pero incluyendo `lower.tail=FALSE` por ser una cola a derecha. A continuación el código usado.

```
qnorm(p=0.10, mean=10, sd=2, lower.tail=FALSE)
```

```
## [1] 12.56
```

En la Figura 14.3 se muestran las áreas sombreadas para cada de las anteriores preguntas.

```
## Warning in library(package, lib.loc = lib.loc,
## character.only = TRUE, logical.return = TRUE, : there
## is no package called 'mytools'
```

---

## 14.2. Distribuciones continuas generales

En la práctica nos podemos encontrar con variables aleatorias continuas que no se ajustan a una de las distribuciones mostradas anteriormente, en esos casos, es posible manejar ese tipo de variables por medio de unas funciones básicas de R como se muestra en el siguiente ejemplo.

### Ejemplo

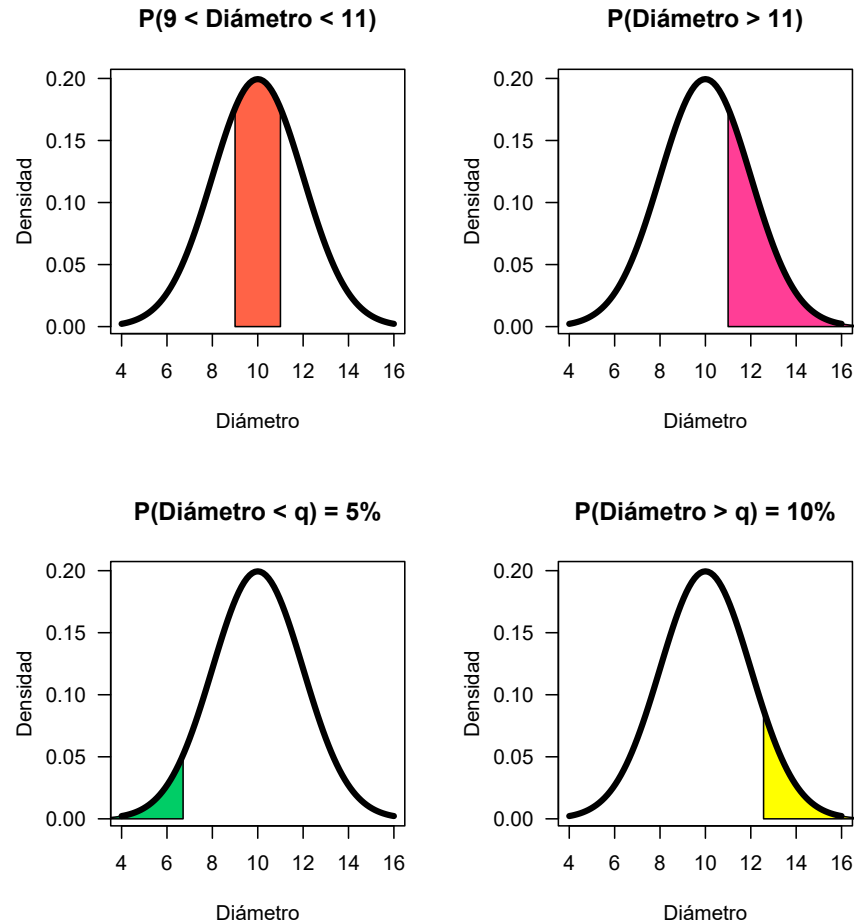
En este ejemplo se retomará la base de datos `crab` sobre el cangrejo de herradura hembra presentado en el capítulo anterior. La base de datos `crab` contiene las siguientes variables: el color del caparazón, la condición de la espina, el peso en kilogramos, el ancho del caparazón en centímetros y el número de satélites o machos sobre el caparazón, la base de datos está disponible en el siguiente enlace<sup>1</sup>.

- 1) Sea  $X$  la variable peso del cangrejo, dibuje la densidad para  $X$ .

Para obtener la densidad muestral de un vector cuantitativo se usa la función `density`, y para dibujar la densidad se usa la función `plot` aplicada a un objeto obtenido con `density`, a continuación el código necesario para dibujar la densidad.

---

<sup>1</sup><https://raw.githubusercontent.com/fhernanb/datos/master/crab>

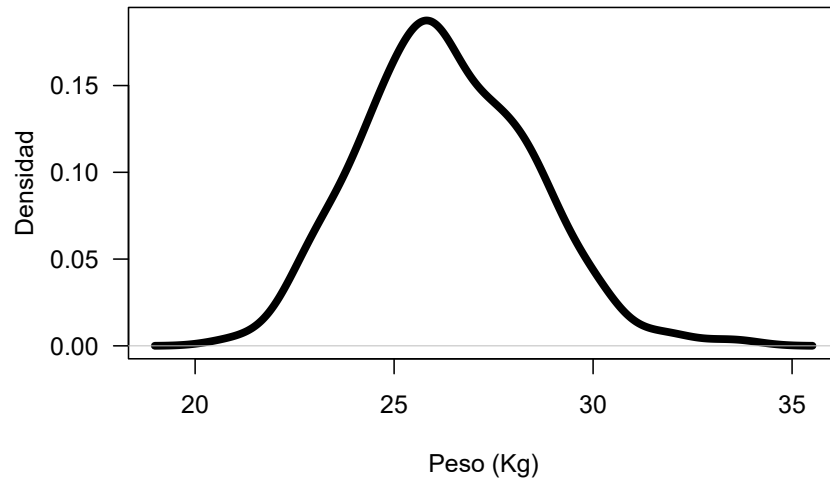


**Figura 14.3:** Área sombreada para el ejemplo de los tornillos.

```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/crab'
crab <- read.table(file=url, header=T)

plot(density(crab$W), main='', lwd=5, las=1,
      xlab='Peso (Kg)', ylab='Densidad')
```

En la Figura 14.4 se muestra la densidad para la variable peso de los cangrejos, esta densidad es bastante simétrica y el intervalo de mayor densidad está entre 22 y 30 kilogramos.



**Figura 14.4:** Función de densidad  $f(x)$  para el peso de los cangrejos.

- 2) Dibujar  $F(x)$  para el peso del cangrejo.

Para dibujar la función  $F(x)$  se usa la función `ecdf` y se almacena el resultado en el objeto `F`, luego se dibuja la función deseada usando `plot`. A continuación el código utilizado. En la Figura 14.5 se presenta el dibujo para  $F(x)$ .

```
F <- ecdf(crab$W)
plot(F, main='', xlab='Peso (Kg)', ylab='F(x)', cex=0.5, las=1)
```

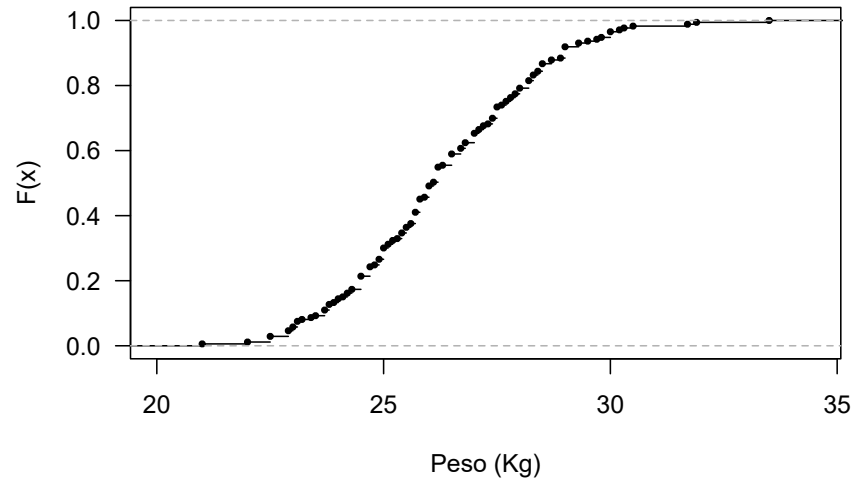
- 3) Calcular la probabilidad de que un cangrejo hembra tenga un peso inferior o igual a 28 kilogramos.

Para obtener  $P(X \leq 28)$  se evalúa en la función  $F(x)$  el cuantil 28 así.

```
F(28)
```

```
## [1] 0.7919
```

Por lo tanto  $P(X \leq 28) = 0.7919$ .



**Figura 14.5:** Función acumulada  $F(x)$  para el peso de los cangrejos.

- 4) Dibujar la función de densidad para el peso de los cangrejos hembra diferenciando por el color del caparazón.

Como son 4 los colores de los caparazones se deben construir 4 funciones de densidad. Usando la función `split` se puede partir el vector de peso de los cangrejos según su color. Luego se construyen las cuatro densidades usando la función `density` aplicada a cada uno de los pesos, a continuación el código.

```
pesos <- split(x=crab$W, f=crab$C)
f1 <- density(pesos[[1]])
f2 <- density(pesos[[2]])
f3 <- density(pesos[[3]])
f4 <- density(pesos[[4]])
```

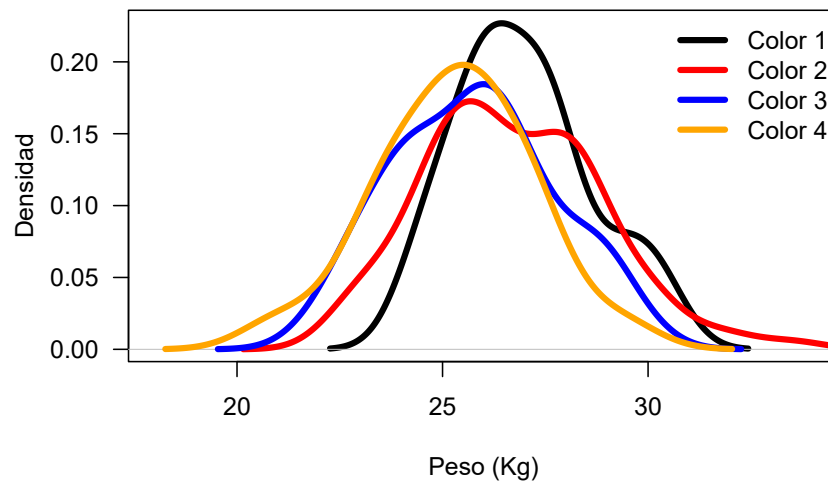
Luego de tener las densidades muestrales se procede a dibujar la primera densidad con `plot`, luego se usa la función `lines` para agregar a la densidad inicial las restantes densidades. En la Figura 14.6 se muestran las 4 densidades, una por cada color de caparazón.

```
plot(f1, main='', las=1, lwd=4,
     xlim=c(18, 34),
```

```

xlab='Peso (Kg)', ylab='Densidad')
lines(f2, lwd=4, col='red')
lines(f3, lwd=4, col='blue')
lines(f4, lwd=4, col='orange')
legend('topright', lwd=4, bty='n',
      col=c('black', 'red', 'blue', 'orange'),
      legend=c('Color 1', 'Color 2', 'Color 3', 'Color 4'))

```



**Figura 14.6:** Función de densidad  $f(x)$  para el peso del cangrejo diferenciando por el color.

Otra forma para dibujar las densidades es usar el paquete **ggplot2** (Wickham and Chang, 2016). En la Figura 14.7 se muestra el resultado obtenido de correr el siguiente código.

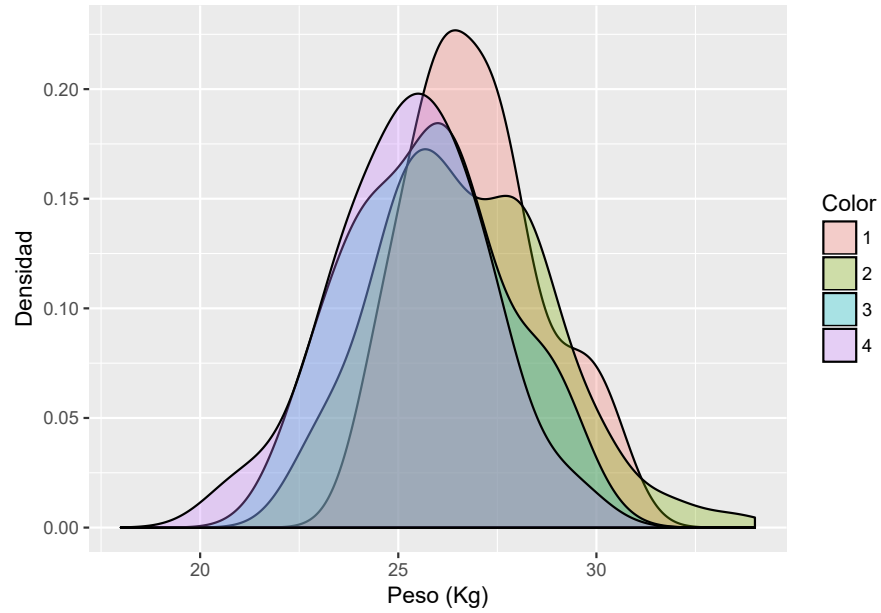
```

require(ggplot2) # Recuerde que primero debe instalarlo

crab$Color <- as.factor(crab$C) # Para convertir en factor

ggplot(crab, aes(x=W)) +
  geom_density(aes(group=Color, fill=Color), alpha=0.3) +
  xlim(18, 34) + xlab("Peso (Kg)") + ylab("Densidad")

```



**Figura 14.7:** Función de densidad  $f(x)$  para el peso del cangrejo diferenciando por el color y usando ggplot2.

Para aprender más sobre el paquete **ggplot2** se recomienda consultar este enlace<sup>2</sup>.

---

<sup>2</sup><http://tutorials.iq.harvard.edu/R/Rgraphics/Rgraphics.html>

# 15

## *Verosimilitud*

En este capítulo se mostrará como usar R para obtener la función de log-verosimilitud y estimadores por el método de máxima verosimilitud.

### 15.1. Función de verosimilitud

La función de verosimilitud para un vector de parámetros  $\Theta$  dada una muestra aleatoria  $\mathbf{x}$  con una distribución asumida se define como:

$$L(\Theta|\mathbf{x}) = \prod_{i=1}^n f(x_i|\Theta), \quad (15.1)$$

donde  $x_i$  representa uno de los elementos de la muestra aleatoria y  $f$  es la función de masa/densidad de la distribución de la cual se obtuvo  $\mathbf{x}$ .

### 15.2. Función de log-verosimilitud

La función de log-verosimilitud  $l$  se define como el logaritmo de la función de verosimilitud  $L$ , es decir

$$l(\Theta|\mathbf{x}) = \log L(\Theta|\mathbf{x}) = \sum_{i=1}^n \log f(x_i|\Theta) \quad (15.2)$$

### 15.3. Método de máxima verosimilitud

El método de máxima verosimilitud se usa para estimar los parámetros de una distribución. El objetivo de este método es encontrar los valores de  $\Theta$  que maximizan  $L$  o  $l$  y valores encontrados se representan por  $\hat{\Theta}$ .

#### Ejemplo

Suponga que se tiene el vector `rta` que corresponde a una muestra aleatoria de una distribución binomial con parámetro `size=5` conocido.

```
rta <- c(2, 2, 1, 1, 1, 1, 0, 2, 1, 2, 1, 0, 1, 2, 1, 0, 0, 2, 2, 1)
```

- 1) Calcular el valor de log-verosimilitud  $l$  si asumimos que  $p = 0.30$  en la distribución binomial.

Para obtener el valor de  $l$  en el punto  $p = 0.30$  se aplica la definición dada en la expresión (15.2). Como el problema trata de una binomial se usa entonces la función de masa `dbinom` evaluada en la muestra `rta`, el parámetro `size` como es conocido se reemplaza por el valor de cinco y en el parámetro `prob` se cambia por 0.3. Como interesa la función de log-verosimilitud se debe incluir `log=TRUE`. A continuación el código necesario.

```
sum(dbinom(x=rta, size=5, prob=0.3, log=TRUE))
```

```
## [1] -24.55
```

Por lo tanto  $l(\theta) = -24.55$

- 2) Construir una función llamada `l1` a la cual le ingrese uno o varios valores del parámetro  $p$  de la binomial, la función debe entregar el valor de log-verosimilitud en cada uno de los  $p$  ingresados.

La función solicitada tiene un cuerpo igual al usado en el numeral anterior. Para asegurar que la función construida pueda recibir un vector de valores de  $p$  se vectoriza la función con `Vectorize`, abajo el código usado.

```
l1 <- function(prob) sum(dbinom(x=rta, size=5, prob=prob, log=T))
l1 <- Vectorize(l1) # Para vectorizar la función
```



Vamos a probar la función en  $p = 0.15$  y  $p = 0.80$  para saber si la función `ll` está vectorizada.

```
ll(prob=0.15) # Individual para p=0.15
```

```
## [1] -25.54
```

```
ll(prob=0.80) # Individual para p=0.80
```

```
## [1] -98.46
```

```
ll(prob=c(0.15, 0.80)) # Evaluada en un vector
```

```
## [1] -25.54 -98.46
```



La mayoría de las funciones en R creadas por los usuarios quedan vectorizadas otras no. La función `ll` no estaba vectorizada y por esa razón se necesitó usar `Vectorize`. No abuse de la función `Vectorize`, sólo úsela cuando sea necesario.

- 3) Dibujar la curva log-verosimilitud  $l$ .

En la Figura 15.1 se presenta la curva solicitada, para su construcción se usa el siguiente código.

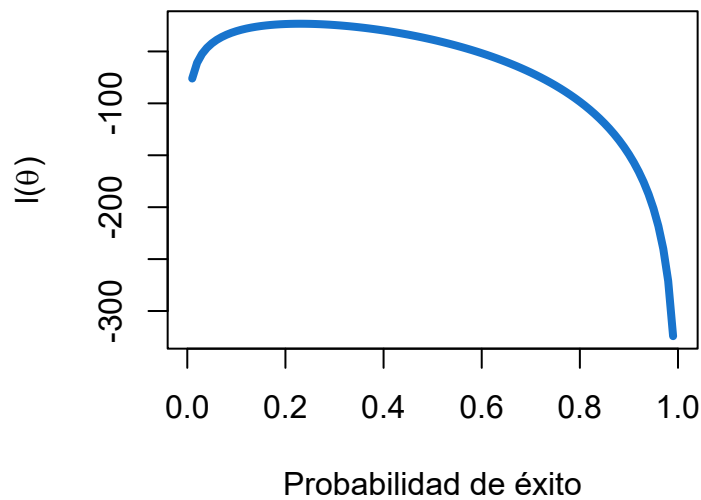
```
curve(ll, lwd=4, col='dodgerblue3',
      xlab='Probabilidad de éxito',
      ylab=expression(paste("l(", theta, ")")))
```

- 4) Observando la Figura 15.1, ¿cuál es el valor de  $p$  que maximiza la función de log-verosimilitud?

Al observar la Figura 15.1 se nota que el valor de  $p$  que maximiza la función log-verosimilitud está muy cerca de 0.2.

- 5) ¿Cuál es el valor exacto de  $p$  que maximiza la función log-verosimilitud?

En R existe la función `optimize` que sirve para encontrar el valor que **minimiza** una función uniparamétrica en un intervalo dado, sin embargo, aquí



**Figura 15.1:** Función de log-verosimilitud para el ejemplo sobre binomial.

interesa es maximizar la función de log-verosimilitud, por esa razón se construye la función `minusll` que es el negativo de la función `ll` para así poder usar `optimize`. A continuación el código usado.

```
minusll <- function(x) -ll(x)

optimize(f=minusll, interval=c(0, 1))

## $minimum
## [1] 0.23
##
## $objective
## [1] 23.32
```

Del resultado anterior se observa que cuando  $p = 0.23$  el valor máximo de log-verosimilitud es 23.32.

### Ejemplo

Suponga que la estatura de una población se puede asumir como una normal  $N(170, 25)$ . Suponga también que se genera una muestra aleatoria de 50 observaciones de la población con el objetivo de recuperar los valores de la media y varianza poblacionales a partir de la muestra aleatoria.

La muestra se va a generar con la función `rnorm` pero antes se fijará una semilla con la intención de que el lector pueda replicar el ejemplo y obtener la misma muestra aleatoria aquí generada, el código para hacerlo es el siguiente.

```
set.seed(1235) # La semilla es 1235
y <- rnorm(n=50, mean=170, sd=5)
```

- 1) construya la función de log-verosimilitud para los parámetros de la normal dada la muestra aleatoria `y`.

```
ll <- function(param) {
  media <- param[1] # param es el vector de parámetros
  desvi <- param[2]
  sum(dnorm(x=y, mean=media, sd=desvi, log=TRUE))
}
```



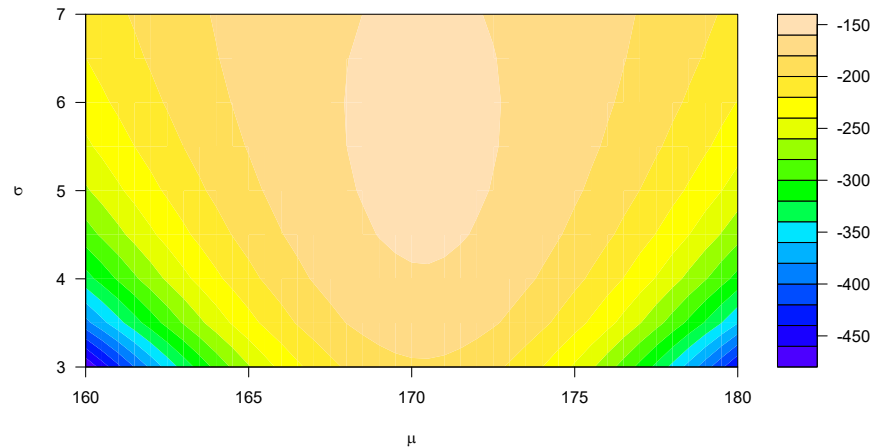
Siempre que el interés sea encontrar los valores que maximizan una función de log-verosimilitud, los parámetros de la distribución **deben** ingresar a la función `ll` como un vector. Esto se debe hacer para poder usar las funciones de búsqueda `optim` y `nlminb`.

- 2) Dibuje la función de log-verosimilitud.

En la Figura 15.2 se muestra el gráfico de niveles para la superficie de log-verosimilitud. De esta figura se nota claramente que los valores que maximizan la superficie están alrededor de  $\mu = 170$  y  $\sigma = 5$ . Para ver el código usado en la creación de la Figura 15.2 se recomienda consultar la sección sobre `contour` en [Hernández \(2018\)](#).

- 3) Obtenga los valores de  $\mu$  y  $\sigma$  que maximizan la función de log-verosimilitud.

Para obtener los valores solicitados vamos a usar la función `nlminb` que es un optimizador. A la función `nlminb` se le debe indicar por medio del parámetro `objective` la función que queremos optimizar (minimizar); el parámetro



**Figura 15.2:** Gráfico de niveles para la función de log-verosimilitud para el ejemplo sobre normal.

`start` es un vector con los valores iniciales para comenzar la búsqueda de  $\mu$  y  $\sigma$ ; los parámetros `lower` y `upper` sirven para delimitar el espacio de búsqueda. A continuación se muestra el código usado para obtener los valores que minimizan a `minusll`, es decir, los valores que maximizan la función de log-verosimilitud.

```
minusll <- function(x) -ll(x)
nlminb(objective=minusll, start=c(163, 3.4),
        lower=c(160, 3), upper=c(180, 7))
```

```
## $par
## [1] 170.338  5.424
##
## $objective
## [1] 155.5
##
## $convergence
## [1] 0
##
## $iterations
## [1] 13
##
## $evaluations
```

```
## function gradient
##      16      35
##
## $message
## [1] "relative convergence (4)"
```

De la salida anterior podemos observar que los valores óptimos de  $\mu$  y  $\sigma$  son 170.338 y 5.424 respectivamente, esto coincide con lo observado en la 15.2 y con los valores reales de simulación de la muestra. Esto indica que el procedimiento de estimación de parámetros por máxima verosimilitud entrega valores insesgados de los parámetros a estimar.

Un resultado interesante de la salida anterior es que se reporta el valor mínimo que alcanza la función `minusll`, este valor fue de 155.5, por lo tanto, se puede afirmar que el valor máximo de log-verosimilitud es -155.5.

Otros resultados importantes de la salida anterior son el valor de `convergence=0` que indica que la búsqueda fue exitosa; `iterations=13` indica que se realizaron 13 pasos desde el punto inicial `start` hasta las coordenadas de optimización.



En R se tienen dos funciones básicas para optimizar funciones, es decir, para encontrar los valores que minimizan una función dada. Esas dos funciones son `nlminb` y `optim`. Para optimizar en una sola dimensión se usa la función `optimize`.

- 4) ¿Hay alguna función para obtener directamente el valor que maximiza la función log-verosimilitud?

La respuesta es si. Si la distribución estudiada es una de las distribuciones básicas se puede usar la función `fitdistr` del paquete básico **MASS**. La función `fitdistr` requiere de los datos que se ingresan por medio del parámetro `x`, y de la distribución de los datos que se ingresa por medio del parámetro `densfun`. La función `fitdistr` admite 15 distribuciones diferentes para hacer la búsqueda de los parámetros que caracterizan una distribución, se sugiere consultar la ayuda de la función `fitdistr` escribiendo en la consola `help(fitdistr)`. A continuación el código usado.

```
require(MASS) # El paquete ya está instalado, solo se debe cargar
res <- fitdistr(x=y, densfun='normal')
res

##      mean      sd
## 170.3384    5.4235
## ( 0.7670) ( 0.5424)
```

El objeto `res` contiene los resultados de usar `fitdistr`. En la primer línea están los valores de los parámetros que maximizan la función de log-verosimilitud, en la parte de abajo, dentro de paréntesis, están los errores estándar o desviaciones de éstos estimadores.

Al objeto `res` es de la clase `fitdistr` y por lo tanto se le puede aplicar la función genérica `logLik` para obtener el valor de la log-verosimilitud. Se sugiere consultar la ayuda de la función `logLik` escribiendo en la consola `help(logLik)`. A continuación el código para usar `logLik` sobre el objeto `res`.

```
logLik(res)
```

```
## 'log Lik.' -155.5 (df=2)
```

De esta última salida se observa que el valor coincide con el obtenido cuando se usó `nlminb`.

## Ejemplo

Generar  $n = 100$  observaciones de una gamma con parámetro de forma igual a 2 y parámetro de tasa igual a 0.5 y luego responder las preguntas.

Para generar la muestra aleatoria `ma` solicitada se fijó la semilla con el objetivo de que el lector pueda obtener los mismos resultados de este ejemplo.

```
n <- 100
set.seed(12345)
ma <- rgamma(n=n, shape=2, rate=0.5)
```

- 1) Asumiendo que la muestra aleatoria proviene de una normal (lo cual es incorrecto) estime los parámetros de la distribución

```
fit1 <- fitdistr(x=ma, densfun='normal')
fit1
```

```
##      mean      sd
##  4.3083  2.8085
## (0.2808) (0.1986)
```

- 2) Asumiendo que la muestra aleatoria proviene de una gamma estime los parámetros de la distribución.

```
fit2 <- fitdistr(x=ma, densfun='gamma')
fit2
```

```
##      shape      rate
##  2.23978  0.51988
## (0.29620) (0.07703)
```

En la salida anterior están los valores estimados de los parámetros de la distribución por el método de máxima verosimilitud, observe la cercanía de éstos con los verdaderos valores de 2 y 0.5 para forma y tasa respectivamente.

- 3) Dibuje dos qqplot, uno asumiendo distribución gamma y el otro distribución normal. ¿Cuál distribución se ajusta mejor a los datos simulados?

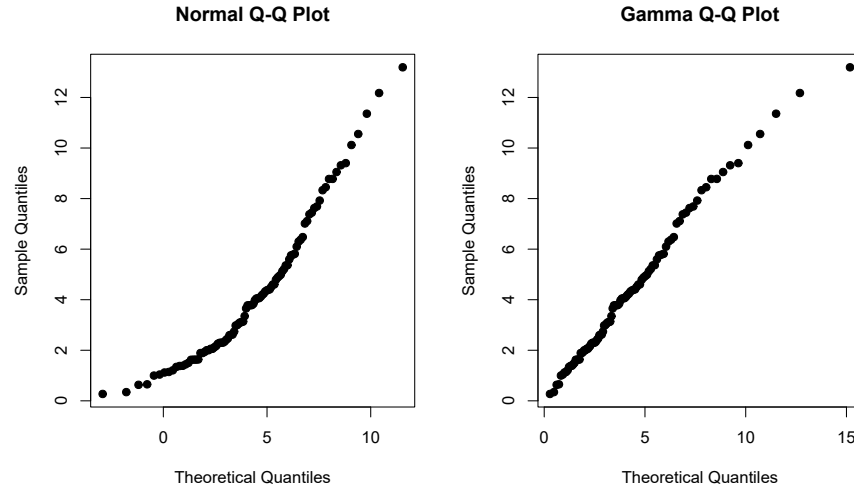
Para dibujar el qqplot se usa la función genérica `qqplot`, recomendamos consultar [Hernández \(2018\)](#) para los detalles de cómo usar esta función. Al usar `qqplot` para obtener el qqplot normal y gamma es necesario indicar los valores  $\hat{\Theta}$  obtenidos en el numeral anterior, por eso es que en el código mostrado a continuación aparece `mean=4.3083`, `sd=2.8085` en el qqplot normal y `shape=2.23978`, `rate=0.51988` en el qqplot gamma.

```
par(mfrow=c(1, 2))

qqplot(y=ma, pch=19,
       x=qnorm(ppoints(n), mean=4.3083, sd=2.8085),
       main='Normal Q-Q Plot',
       xlab='Theoretical Quantiles',
       ylab='Sample Quantiles')

qqplot(y=ma, pch=19,
       x=qgamma(ppoints(n), shape=2.23978, rate=0.51988),
       main='Gamma Q-Q Plot',
       xlab='Theoretical Quantiles',
       ylab='Sample Quantiles')
```

En la Figura 15.3 se muestran los qqplot solicitados. Se observa claramente que al asumir normalidad (lo cual es incorrecto), los puntos del qqplot no están alineados, mientras que al asumir distribución gamma (lo cual es correcto), los puntos si están alineados. De esta figura se concluye que la muestra `ma` puede provenir de una  $Gamma(2.23978, 0.51988)$ .



**Figura 15.3:** Gráfico cuantil cuantil normal y gamma para la muestra simulada.



Para obtener el gráfico cuantil cuantil bajo normalidad se puede usar directamente la función `qqnorm`, consultar [Hernández \(2018\)](#) para mayores detalles.



En este ejemplo se eligió la mejor distribución entre dos candidatas usando una herramienta gráfica, lo que se recomienda usar algún método menos subjetivo (cuantitativo) para tomar decisiones.

- 4) Para comparar modelos se puede utilizar el Akaike information criterion (*AIC*) propuesto por [Akaike \(1974\)](#) que sirve para medir la calidad relativa de los modelos estadísticos, la expresión para calcular el indicador es  $AIC = -2\hat{l} + 2df$ , donde  $\hat{l}$  corresponde al valor de log-verosimilitud y  $df$  corresponde al número de parámetros estimados del modelo. Siempre el modelo elegido es aquel modelo con el menor valor de *AIC*. Calcular el *AIC* para los modelos asumidos normal y gamma.

```
-2 * logLik(fit1) + 2 * 2 # AIC para modelo normal
```

```
## 'log Lik.' 494.3 (df=2)
```



```
-2 * logLik(fit2) + 2 * 2 # AIC para modelo gamma
```

```
## 'log Lik.' 466 (df=2)
```

De los resultados anteriores se concluye que entre los dos modelos, el mejor es el gamma porque su  $AIC = 466$  es el menor de todos los  $AIC$ .



Modelos anidados pueden ser comparados por medio del global deviance ( $GD$ ) dado por  $GD = -2\hat{l}$  y modelos no anidados por medio del Generalized Akaike information criterion ( $GAIC$ ) propuesto por Akaike (1983) y dado por  $GAIC = -2\hat{l} + \#df$  siendo  $\#$  el valor de penalidad por cada parámetro adicional en el modelo; cuando  $\# = 2$ , el  $GAIC$  coincide con el  $AIC$  y el Schwarz Bayesian criterion ( $SBC$ ) propuesto por Schwarz (1978) se da cuando el valor de penalidad es  $\# = \log(n)$  donde  $n$  es el número de observaciones del modelo; siempre el modelo elegido es aquel modelo con el menor valor de cualquiera de los criterios de información anteriores.

## Ejemplo

¿Será posible explorar entre muchas distribuciones estadísticas aquella que mejor explica una variable de interés?

La respuesta es si, se puede usar la función `fitDist` del paquete **gamlss**. Esta función tiene la siguiente estructura:

```
fitDist(y, k = 2,
        type = c("realAll", "realline", "realplus",
                  "real0to1", "counts", "binom"))
```

El parámetro `y` sirve para ingresar el vector con la información; `k=2` es la penalización por cada parámetro estimado para calcular el  $GAIC$ , por defecto es 2; y el parámetro `type` sirve para indicar el tipo de distribución, los posibles valores son:

- **realAll**: para hacer la búsqueda en todas las distribuciones disponibles en **gamlss**.
- **realline**: para variables en  $\mathcal{R}$ .
- **realplus**: para variables en  $\mathcal{R}^+$ .
- **real0to1**: para variables en el intervalo  $(0, 1)$ .
- **counts**: para variables de conteo.
- **binom**: para variables de tipo binomial.

- 1) Usar la función `fitDist` para elegir la distribución que mejor se

ajusta a los datos simulados en el ejemplo anterior, usar una penalización de  $k = 2$  para calcular el *AIC*.

Se usa la función `fitDist` con `type='realplus'` porque se sabe que la muestra aleatoria tiene valores sólo en  $\mathbb{R}^+$ , los resultados de almacenan en el objeto `modelos`. Para obtener la lista de los mejores modelos con su respectivo *BAIC* se escribe en la consola `modelos$fits`. Abajo el código usado.

```
require(gamlss)
modelos <- fitDist(y=ma, type='realplus', k=2)
modelos$fits
```

##	GA	WEI3	GG	GIG	BCCGo	BCPEo
##	466.0	466.8	468.0	468.0	468.4	469.4
##	BCTo	exGAUS	LOGNO	IG	EXP	PARETO2
##	470.4	471.2	474.6	483.9	494.1	496.1
##	IGAMMA					
##	504.2					

De la lista anterior se observa que la función gamma está en el primer lugar con un *BAIC* = 466, valor que coincide con el *AIC* obtenido en el ejercicio anterior. Esto significa que como la gamma tiene el menor *BAIC*, es la distribución que mejor se ajusta a los datos de la muestra, esto coincide con la realidad, ya que la muestra fue generada de una distribución gamma.

¿Por qué *AIC* y *BAIC* coinciden en este ejemplo?

- 2) ¿Cuáles son los valores estimados de los parámetros para la distribución gamma identificada en el paso anterior?

En `gamlss` los parámetros de las distribuciones se nombran de una forma especial, el parámetro de posición se denomina  $\mu$ , el de dispersión se denomina  $\sigma$  y los de simetría y curtosis con  $\nu$  y  $\tau$ . La distribución gamma sólo tiene 2 parámetros y para obtener sus valores estimados se usa el siguiente código.

```
modelos$mu
```

```
## [1] 4.308
```

```
modelos$sigma
```

```
## [1] 0.6682
```

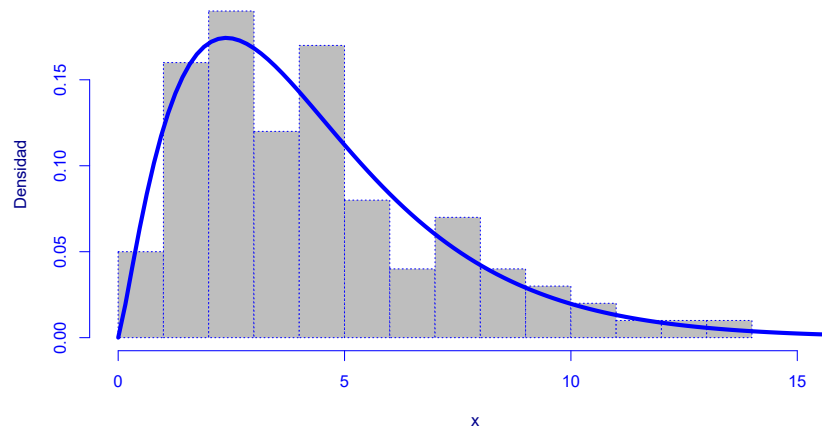
Si comparamos los anteriores resultados,  $\hat{\mu} = 4.308$  y  $\hat{\sigma} = 0.6682$ , con los encontrados al usar la función `fitdist`,  $\widehat{shape} = 2.23978$  y  $\widehat{rate} = 0.51988$ ,

vemos que no coinciden, esto se debe a que la parametrización de la gamma usada por **gamlss** es diferente a la parametrización usada por la base de R. Lo anterior no es motivo de preocupación, lo que se recomienda es que el usuario elija una de las parametrizaciones y trabaje con ella, no debe mezclarlas.

- 3) Dibujar el histograma para la muestra aleatoria y agregar la densidad de la distribución gamma identificada como la distribución que mejor explica el comportamiento de la variable.

Para hacer lo solicitado se usa la función **histDist** del paquete **gamlss**, sólo es necesario ingresar los datos y la familia o distribución de la densidad requerida. Abajo el código usado.

```
histDist(y=ma, family=GA, main='', xlab='x', ylab='Densidad',
         line.col='blue', line.wd=4)
```



**Figura 15.4:** Histograma para la muestra simulada con la densidad de una  $\text{Gamma}(\mu=4.308, \sigma=0.6682)$ .

```
##
## Family:  c("GA", "Gamma")
## Fitting method: "nlminb"
##
## Call:   gamlssML(y = y, family = "GA", formula = ma)
##
## Mu Coefficients:
## [1] 1.46
```

```
## Sigma Coefficients:
## [1] -0.403
##
## Degrees of Freedom for the fit: 2 Residual Deg. of Freedom 98
## Global Deviance:      462
##           AIC:        466
##           SBC:        471.3
```

En la Figura 15.4 se presenta el histograma para muestra aleatoria y la densidad de la gamma que mejor explica estos datos. Se observa claramente que la curva de densidad azul acompaña el patrón de los datos.

---

## EJERCICIOS

- 1) Al inicio del Capítulo 9 se presentó la base de datos sobre medidas del cuerpo, consulte la explicación sobre la base de datos y responda lo siguiente.
  - Si se asume que la `edad` tiene distribución normal, ¿cuáles son los estimadores de máxima verosimilitud para  $\mu$  y  $\sigma$ ?
  - Como el histograma para la edad muestra un sesgo a la derecha se podría pensar que la distribución gamma sería una buena candidata para explicar las edades observadas. Asumiendo una distribución gamma, ¿cuáles son los estimadores de máxima verosimilitud para los parámetros?
  - ¿Cuál de los dos modelos es más apropiado para explicar la variable de interés? Calcule el *AIC* para decidir.
- 2) En la sección 13.2 se presentó la base de datos sobre cangrejos hembra, consulte la explicación sobre la base de datos y responda lo siguiente.
  - Suponga que el número de satélites sobre cada hembra es una variable que se distribuye Poisson. Construya en R la función de log-verosimilitud  $l$ , dibuje la función  $l$  y encuentre el estimador de máxima verosimilitud de  $\lambda$ .
  - Repita el ejercicio anterior asumiendo que el número de satélites se distribuye binomial negativo.
  - ¿Cuál de los dos modelos es más apropiado para explicar la variable de interés? Calcule el *AIC* para decidir.
- 3) Al inicio del Capítulo 10 se presentó la base de datos sobre apartamentos usados en Medellín, consulte la explicación sobre la base de datos y responda lo siguiente.

- Dibuje una densidad para la variable área del apartamento.
- Describa lo encontrado en esa densidad.
- ¿Qué distribuciones de 2 parámetros podrían explicar el comportamiento del área de los apartamentos? Mencione al menos 3.
- Para cada una de las distribuciones anteriores dibuje un gráfico de contornos o calor para la función de log-verosimilitud y estime los parámetros de la distribución elegida.
- ¿Cuál de los dos modelos es más apropiado para explicar la variable de interés? Calcule el  $AIC$  para decidir.



# 16

## *Estudiando la normalidad*

En este capítulo se mostrará cómo utilizar las herramientas de R para estudiar la normalidad univariada de un conjunto de datos.

### 16.1. Consideraciones iniciales

En estadística hay una gran cantidad de modelos, pruebas y procedimientos que tienen como supuesto la normalidad, por lo tanto, se hace necesario contar con herramientas que nos guíen para responder si se cumple o no el supuesto de normalidad.

Para estudiar si una muestra aleatoria proviene de una población con distribución normal se disponen de tres herramientas que se listan a continuación.

1. Histograma y/o densidad.
2. Gráficos cuantil cuantil (QQplot).
3. Pruebas de hipótesis.

Al construir un histograma y/o densidad para la variable de interés se puede evaluar visualmente la simetría de la distribución de los datos. Si se observa una violación clara de la simetría (sesgo a uno de los lados) o si se observa una distribución con más de una moda, eso sería indicio de que la muestra no proviene de una población normal. Por otra parte, si se observa simetría en los datos, esto NO garantiza que la muestra aleatoria proviene de una población normal y se hace necesario recurrir a otras herramientas específicas para estudiar normalidad como lo son los gráficos QQplot y pruebas de hipótesis.

A continuación se presentan varias secciones donde se profundiza sobre el uso de cada de las tres herramientas anteriormente listadas para estudiar la normalidad.

## 16.2. Histograma y densidad

El histograma y el gráfico de densidad son herramientas muy útiles porque sirven para mostrar la distribución, la simetría, el sesgo, variabilidad, moda, mediana y observaciones atípicas de un conjunto de datos. Para explorar la normalidad de un conjunto de datos lo que se busca es que el histograma o gráfico de densidad presenten un patrón más o menos simétrico. Para obtener detalles de la construcción de histogramas y gráficos de densidad se recomienda consultar [Hernández \(2018\)](#).

A continuación se presentan dos ejemplos, uno con datos simulados y otro con datos reales, con los cuales se muestra la utilidad del histograma y gráfico de densidad al explorar la normalidad.

### Ejemplo densidad con datos simulados

Simular 4 muestra aleatorias de una  $N(0, 1)$  con tamaños de muestra  $n=10$ , 100, 1000 y 10000; para cada una de las muestras construir los gráficos de densidad y analizar si son simétricos.

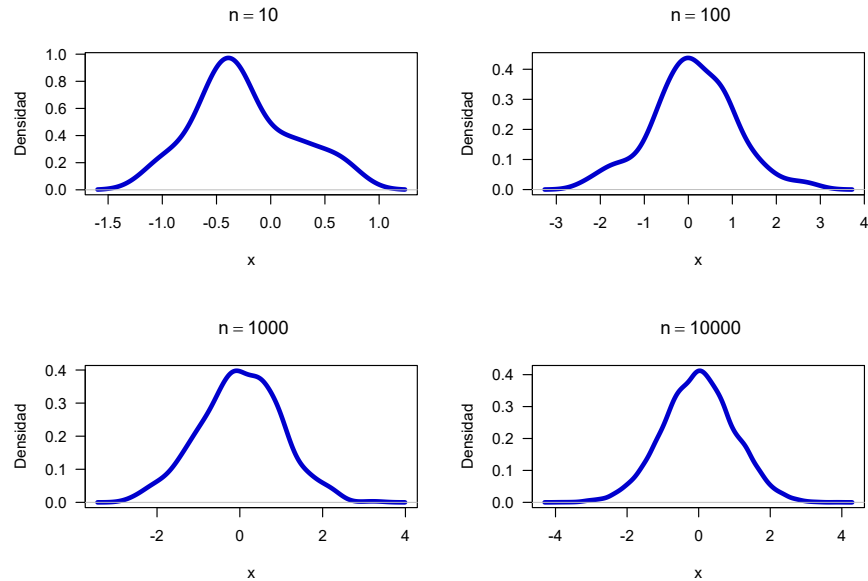
Lo primero que se debe hacer es definir el vector `n` con los valores de los tamaños de muestra, luego dentro de una sentencia `for` se simula cada muestra y se dibuja su densidad.

```
par(mfrow=c(2, 2))
n <- c(10, 100, 1000, 10000)
for (i in n) {
  x <- rnorm(i)
  plot(density(x), main=bquote(~ n == .(i)),
       ylab='Densidad', col='blue3', xlab='x', las=1, lwd=4)
}
```

En la Figura 16.1 se muestran las cuatro densidades, de esta figura se observa que, a pesar de haber generado las muestras de una normal estándar, las densidades no son perfectamente simétricas, sólo para el caso de tamaño de muestra  $n = 10000$  la densidad muestral fue bastante simétrica. Esto significa que el gráfico de densidad se debe usar con precaución para decidir sobre la normalidad de un conjunto de datos, como regla práctica se aconseja lo siguiente:

- Si la densidad muestral es muy pero muy asimétrica se debe desconfiar de la normalidad de los datos.





**Figura 16.1:** Densidad para 4 muestras de una  $N(0, 1)$  con diferente tamaño de muestra.

- Si la densidad muestral es más o menos simétrica, se deben usar otras herramientas como QQplot o pruebas de hipótesis para obtener una mejor conclusión.



Si un histograma o densidad es muy asimétrico o sesgado, es evidencia en contra de la normalidad de los datos. Si el histograma o densidad presentan simetría, esto no es garantía de la normalidad de los datos.

### Ejemplo densidad del peso corporal

Considerando la base de datos **medidas del cuerpo** presentada en el Capítulo 9, se desea saber si el peso corporal, tanto de hombres y mujeres, tiene una distribución normal.

Para hacer la exploración lo primero es cargar la base de datos usando el siguiente código.

```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/medidas_cuerpo'
datos <- read.table(file=url, header=T)
```

La variable `peso` del objeto `datos` contiene la información sobre el peso corporal de ambos sexos, debemos entonces partir o dividir esta información diferenciando entre hombres y mujeres, para esto usamos la función `split` de la siguiente forma.

```
pesos <- split(datos$peso, datos$sexo)
pesos  # Para ver los elementos de la lista pesos
```

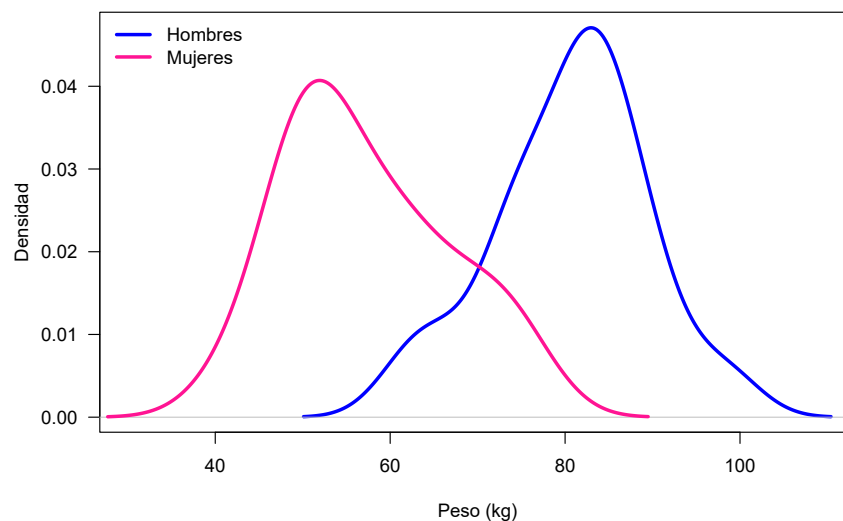
```
## $Hombre
## [1] 87.3 80.0 82.3 73.6 74.1 85.9 73.2 76.3 65.9 90.9
## [11] 89.1 62.3 82.7 79.1 98.2 84.1 83.2 83.2
##
## $Mujer
## [1] 51.6 59.0 49.2 63.0 53.6 59.0 47.6 69.8 66.8 75.2
## [11] 55.2 54.2 62.5 42.0 50.0 49.8 49.2 73.2
```

El objeto `pesos` es una lista con dos elementos, el primero contiene los pesos de los hombres mientras que el segundo contiene los pesos de las mujeres. Note que `pesos` es un objeto mientras que `peso` es el nombre usado para la variable peso corporal en la base de datos.

Para explorar la normalidad de los pesos se dibujan dos densidades, una para el peso de hombres y otra para el peso de las mujeres, a continuación el código utilizado.

```
plot(density(pesos$Hombre), lwd=3, col='blue',
     xlim=c(30, 110), main='', las=1,
     xlab='Peso (kg)', ylab='Densidad')
lines(density(pesos$Mujer), lwd=3, col='deeppink')
legend('topleft', legend=c('Hombres', 'Mujeres'),
      lwd=3, col=c('blue', 'deeppink'), bty='n')
```

En la Figura 16.2 se muestran las dos densidades, en la figura no se observa una evidencia clara de sesgo, lo que se observa es que la densidad para los hombres es un poco más simétrica que la densidad para las mujeres. De estos resultados no se puede rechazar la premisa de que pesos corporales provienen de una distribución normal, lo recomendable es construir QQplot y aplicar prueba de hipótesis.



**Figura 16.2:** Densidad para el peso corporal de hombres y mujeres.

---

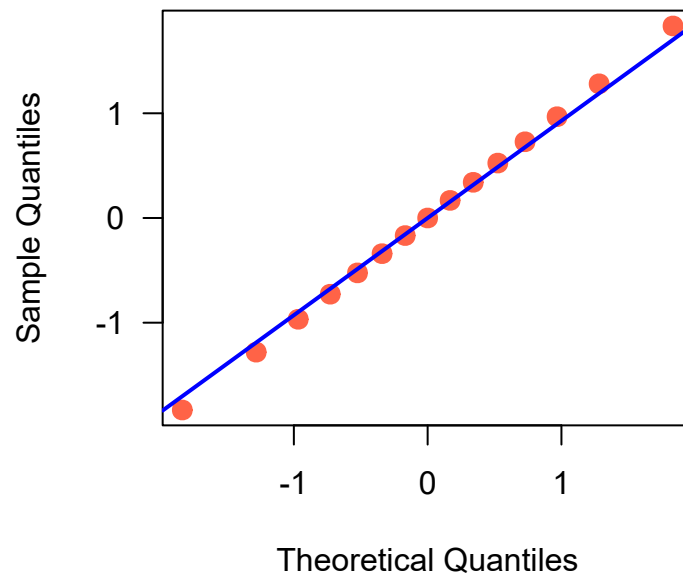
### 16.3. Gráficos cuantil cuantil

Los gráficos cuantil cuantil (QQplot) son una herramienta gráfica para explorar si un conjunto de datos o muestra proviene de una población con cierta distribución, en particular aquí nos interesan para estudiar la normalidad de un conjunto de datos. La función `qqnorm` sirve para construir el QQplot y la función `qqline` agrega una línea de referencia que ayuda a interpretar el gráfico QQplot, para obtener una explicación de cómo construir este gráfico se recomienda ver el video disponible en este enlace<sup>1</sup>.

En la Figura 16.3 se muestra un ejemplo de un QQplot y de sus partes, los puntos y la línea de referencia. Si se tuviese una muestra distribuida perfectamente normal, se esperaría que los puntos estuviesen perfectamente alineados con la línea de referencia, sin embargo, las muestras con las que se trabajan en la práctica casi nunca presentan este comportamiento aún si fueron obtenidas de una población normal. En la práctica se aceptan alejamientos del patrón lineal para aceptar que los datos si provienen de una población normal.

---

<sup>1</sup>[https://www.youtube.com/watch?v=kx\\_o9rnI4DE](https://www.youtube.com/watch?v=kx_o9rnI4DE)



**Figura 16.3:** Ejemplo de un QQplot.

A continuación se presentan cuatro ejemplos, dos con datos simulados y otro con datos reales para mostrar la utilidad del qqplot al explorar la normalidad.

### Ejemplo 1 QQplot con datos simulados

Simular 4 muestra aleatorias de una  $N(0,1)$  con tamaños de muestra  $n=10$ , 30, 50 y 100, para cada una de ellas construir el QQplot.

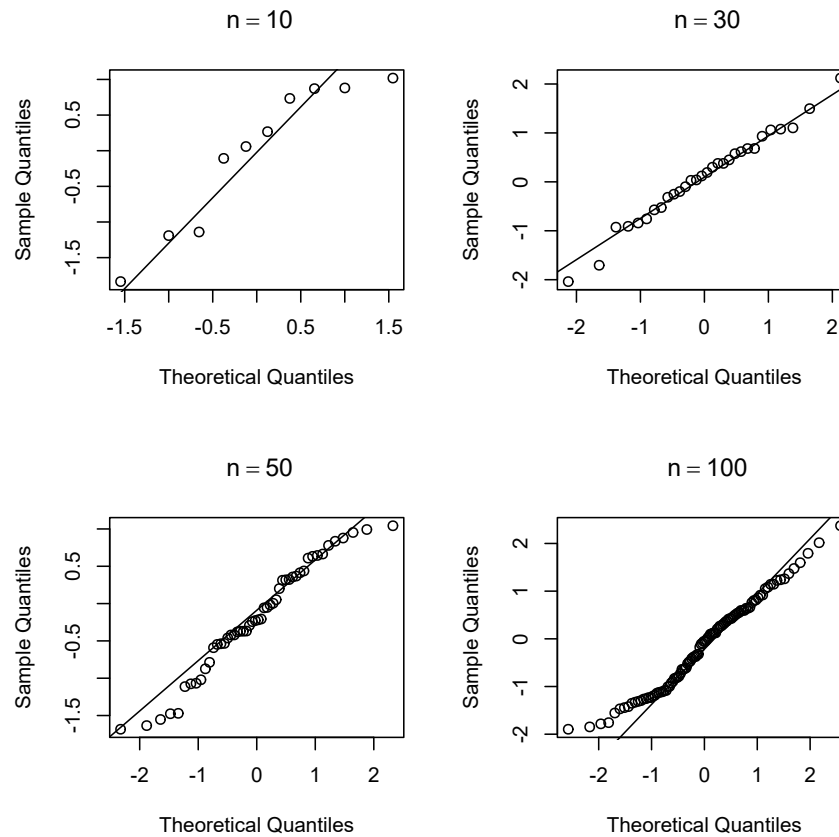
Lo primero que se debe hacer es definir el vector **n** con los valores del tamaño de muestra, luego dentro de una sentencia **for** se simula cada muestra **x** y por último se dibuja el QQplot para cada muestra, a continuación el código utilizado.

```
par(mfrow=c(2, 2))
n <- c(10, 30, 50, 100)
for (i in n) {
```

```

x <- rnorm(i)
qqnorm(x, main=bquote(~ n == .(i)))
qqline(x) # Para agregar la linea de referencia
}

```



**Figura 16.4:** QQplot para 4 muestras de una  $N(0, 1)$  con diferente tamaño de muestra.

La Figura 16.4 muestra que, a pesar de haber simulado cada muestra  $x$  de una  $N(0, 1)$ , los puntos no se alinean de forma perfecta, esto significa que en la práctica se debe ser prudente con la interpretación de un QQplot, un alejamiento del patrón lineal NO significa que la muestra no provenga de una población normal.

### Ejemplo 2 QQplot con datos simulados

Simular 1 muestra aleatoria con  $n = 50$  de cada una de las siguientes poblaciones Poisson(5), NBinom(5, 0.5), Gamma(2, 3) y Weibull(1, 3), para cada una de las muestras construir el QQplot para explorar la normalidad de las muestras.

Las muestras de cada una de las poblaciones se generan con las funciones `rpois`, `rnbinom`, `rgamma` y `rweibull` especificando los parámetros. A continuación el código necesario para obtener los QQplot solicitados.

```
m1 <- rpois(n=50, lambda=5)
m2 <- rnbinom(n=50, size=5, prob=0.35)
m3 <- rgamma(n=50, shape=2, scale=3)
m4 <- rweibull(n=50, shape=1, scale=3)
par(mfrow=c(2, 2))
qqnorm(m1, main='Poisson(5)')
qqline(m1)
qqnorm(m2, main='NBinom(5, 0.35)')
qqline(m1)
qqnorm(m3, main='Gamma(2, 3)')
qqline(m1)
qqnorm(m4, main='Weibull(1, 3)')
qqline(m1)
```

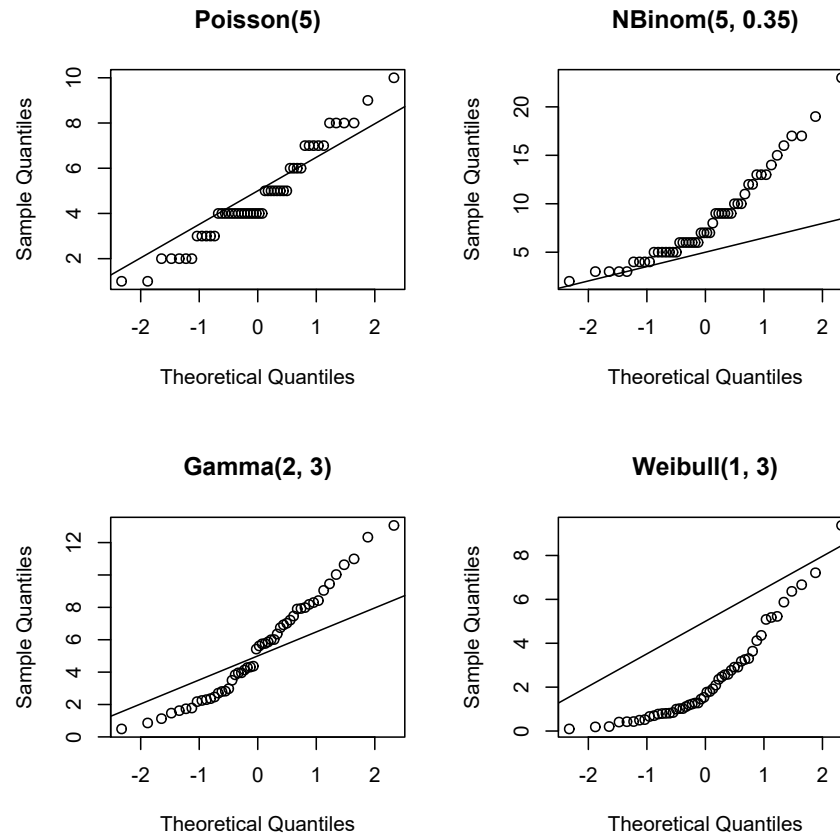
En la Figura 16.5 se muestran los cuatro QQplot para cada una de las muestras generadas de las distribuciones indicadas. Se observa claramente que los puntos del QQplot no están alineados, esto es una clara evidencia de que las muestras NO provienen de poblaciones normales. Otro aspecto interesante a resaltar es el patrón de escalera que se observa para las muestras generadas de poblaciones discretas (Poisson y Binomial Negativa).



Se debe tener cuidado al concluir con un QQplot, lo que para una persona puede estar alineado, para otra puede no estarlo. El QQplot es un gráfico exploratorio de normalidad.

### Ejemplo QQplot para peso corporal

Retomando la base de datos **medidas del cuerpo** presentada en el Capítulo 9, se desea saber si el peso corporal, tanto de hombres y mujeres, tiene una distribución normal usando QQplots.



**Figura 16.5:** QQplot para muestras generadas de poblaciones Poisson, Binomial Negativa, Gamma y Weibull.

Para hacer la exploración lo primero es cargar la base de datos si aún no se ha cargado.

```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/medidas_cuerpo'
datos <- read.table(file=url, header=T)
```

La variable `peso` del objeto `datos` contiene la información sobre el peso corporal de ambos sexos, debemos entonces partir o dividir esta información diferenciando entre hombres y mujeres, para esto usamos la función `split` de la siguiente forma.

```
pesos <- split(datos$peso, datos$sexo)
pesos

## $Hombre
## [1] 87.3 80.0 82.3 73.6 74.1 85.9 73.2 76.3 65.9 90.9
## [11] 89.1 62.3 82.7 79.1 98.2 84.1 83.2 83.2
##
## $Mujer
## [1] 51.6 59.0 49.2 63.0 53.6 59.0 47.6 69.8 66.8 75.2
## [11] 55.2 54.2 62.5 42.0 50.0 49.8 49.2 73.2
```

El objeto `pesos` es una lista con dos elementos, el primero contiene los pesos de los hombres mientras que el segundo contiene los pesos de las mujeres. Note que `pesos` es un objeto mientras que `peso` es el nombre usado para la variable peso corporal en la base de datos.

Para explorar la normalidad de los pesos se dibujan dos densidades, una para el peso de hombres y otra para el peso de las mujeres, a continuación el código utilizado.

```
par(mfrow=c(1, 2))
qqnorm(pesos$Hombre, pch=20,
       main='QQplot para peso de hombres')
qqline(pesos$Hombre)

qqnorm(pesos$Mujer, pch=20,
       main='QQplot para peso de mujeres')
qqline(pesos$Mujer)
```

La Figura 16.6 muestra el QQplot para el peso corporal de hombres y mujeres, de la figura se observa que los puntos no están tan desalineados, lo cual nos lleva a no rechazar la hipótesis de normalidad.

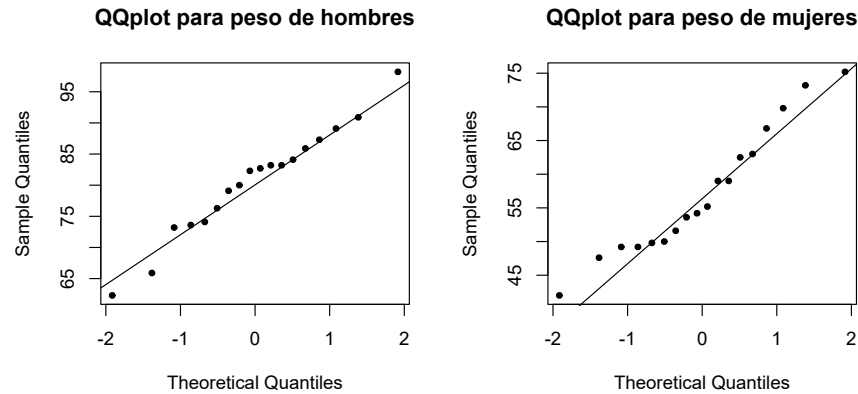


Se debe interpretar con precaución el QQplot, del ejemplo con datos simulados se vió que a pesar de haber generado las muestras de una  $N(0, 1)$  los QQplot no siempre están perfectamente alineados.

### Ejemplo QQplot con bandas

Construir QQplot con bandas de confianza para el peso corporal de hombres y mujeres con los datos del ejemplo anterior. ¿Se puede afirmar que los pesos corporales provienen de una distribución normal?





**Figura 16.6:** QQplot para el peso corporal de hombres y mujeres.

Para construir este tipo de QQplot se usa la función `qqplot` del paquete `car`. A continuación el código para construir el gráfico solicitado.

```
require(car)
```

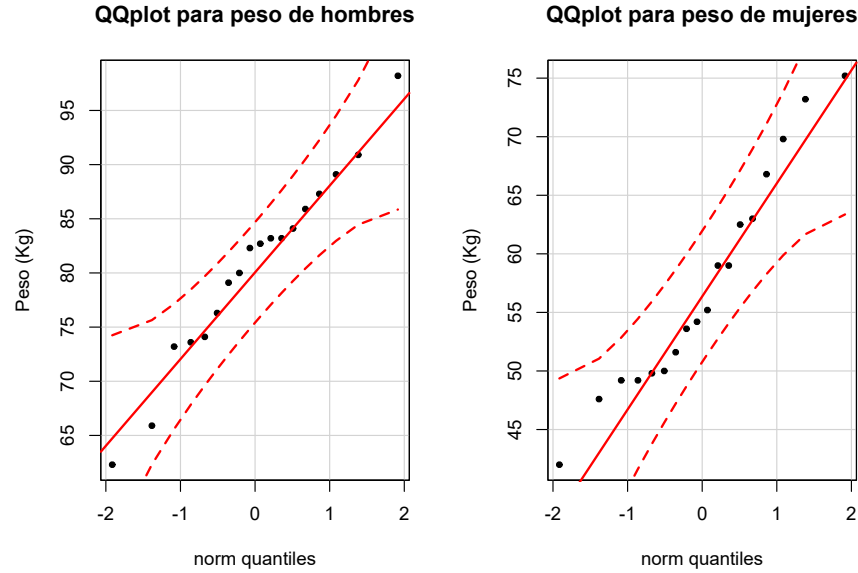
```
## Warning: package 'car' was built under R version 3.3.3
```

```
par(mfrow=c(1, 2))
qqPlot(pesos$Hombre, pch=20, ylab='Peso (Kg)',
       main='QQplot para peso de hombres')

qqPlot(pesos$Mujer, pch=20, ylab='Peso (Kg)',
       main='QQplot para peso de mujeres')
```

En la Figura 16.7 se muestra el QQplot solicitado, como los puntos del QQplot están dentro de las bandas se puede aceptar que los pesos corporales provienen de una población normal.

La función `qqPlot` tiene varios parámetros adicionales que recomendamos consultar en la ayuda de la función `help(qqPlot)`.



**Figura 16.7:** QQplot con bandas de confianza para el peso corporal de hombres y mujeres.

## 16.4. Pruebas de normalidad

Una forma menos subjetiva de explorar la normalidad de un conjunto de datos es por medio de las pruebas de normalidad. Las hipótesis para este tipo de pruebas son:

$$\begin{aligned} H_0 &: \text{la muestra proviene de una población normal.} \\ H_A &: \text{la muestra NO proviene de una población normal.} \end{aligned} \quad (16.1)$$

En la literatura estadística se reportan varias pruebas, algunas de ellas se listan a continuación.

1. Prueba Shapiro-Wilk con la función `shapiro.test`.
2. Prueba Anderson-Darling con la función `ad.test` del paquete **nor-test**.
3. Prueba Cramer-von Mises con la función `cvm.test` del paquete **nor-test**.

4. Prueba Lilliefors (Kolmogorov-Smirnov) con la función `lillie.test` del paquete **nortest**.
5. Prueba Pearson chi-square con la función `pearson.test` del paquete **nortest**.
6. Prueba Shapiro-Francia con la función `sf.test` del paquete **nortest**).

### Ejemplo con datos simulados

Generar una muestra aleatoria con  $n = 100$  de una  $N(150, 25)$  y aplicar las pruebas de normalidad Shapiro-Wilk y Anderson-Darling con un nivel de significancia del 3 %.

Lo primero es generar la muestra aleatoria **x** así:

```
x <- rnorm(n=100, mean=150, sd=5)
```

Para aplicar la prueba Shapiro-Wilk se usa la función `shapiro.test` al vector **x** así:

```
shapiro.test(x)
```

```
##  
## Shapiro-Wilk normality test  
##  
## data:  x  
## W = 0.98, p-value = 0.2
```

De la salida anterior se tiene que el valor-P para la prueba fue de 0.2 y que es mayor al nivel de significancia 3 %, lo cual indica que no hay evidencias para rechazar la hipótesis nula de normalidad.

Para aplicar la prueba Anderson-Darling se usa la función `ad.test` al vector **x** así:

```
require(nortest) # Se debe haber instalado nortest  
ad.test(x)
```

```
##  
## Anderson-Darling normality test  
##  
## data:  x  
## A = 0.6, p-value = 0.1
```

De la salida anterior se tiene que el valor-P para la prueba fue de 0.1 y que es mayor al nivel de significancia 3 %, esto indica que no hay evidencias para rechazar la hipótesis nula de normalidad.

### Ejemplo normalidad para peso corporal

Retomando la base de datos **medidas del cuerpo** presentada en el Capítulo 9, se desea saber si el peso corporal, tanto de hombres y mujeres, tiene una distribución normal usando las pruebas normalidad Shapiro-Wilks y Anderson-Darling con un nivel de significancia del 5 %.

Lo primero es cargar la base de datos si aún no se ha cargado.

```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/medidas_cuerpo'
datos <- read.table(file=url, header=T)
```

La variable **peso** del objeto **datos** contiene la información sobre el peso corporal de ambos sexos, debemos entonces partir o dividir esta información diferenciando entre hombres y mujeres, para esto usamos la función **split** de la siguiente forma.

```
pesos <- split(datos$peso, datos$sexo)
pesos
```

```
## $Hombre
## [1] 87.3 80.0 82.3 73.6 74.1 85.9 73.2 76.3 65.9 90.9
## [11] 89.1 62.3 82.7 79.1 98.2 84.1 83.2 83.2
##
## $Mujer
## [1] 51.6 59.0 49.2 63.0 53.6 59.0 47.6 69.8 66.8 75.2
## [11] 55.2 54.2 62.5 42.0 50.0 49.8 49.2 73.2
```

Para aplicar la prueba Shapiro-Wilk se usa la función **shapiro.test**. Como el objeto **pesos** es una lista se debe usar la función **lapply** para aplicar **shapiro.test** a la lista, a continuación el código usado.

```
lapply(pesos, shapiro.test)
```

```
## $Hombre
##
## Shapiro-Wilk normality test
##
## data:  X[[i]]
```

```
## W = 0.98, p-value = 0.9
##
##
## $Mujer
##
## Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.95, p-value = 0.4
```

De la salida anterior se observa que ambos valores-P fueron mayores al nivel de significancia 5%, por lo tanto, se puede concluir que ambas muestras provienen de poblaciones con distribución normal.

Para aplicar la prueba Anderson-Darling se usa la función `ad.test` del paquete `nortest`. Como el objeto `pesos` es una lista se debe usar la función `lapply` para aplicar `ad.test` a la lista, a continuación el código usado.

```
require(nortest) # Se debe haber instalado nortest
lapply(pesos, shapiro.test)
```

```
## $Hombre
##
## Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.98, p-value = 0.9
##
##
## $Mujer
##
## Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.95, p-value = 0.4
```

De la salida anterior se observa que ambos valores-P fueron mayores al nivel de significancia 5%, por lo tanto, se puede concluir que ambas muestras provienen de poblaciones con distribución normal.

Al usar las pruebas Shapiro-Wilks y Anderson-Darling se concluye que no hay evidencias para pensar que los pesos corporales de hombres y mujeres no provienen de una población normal.

---

## EJERCICIOS

1. Para la base de datos **medidas del cuerpo** presentada en el Capítulo 9, explorar si la variable estatura, diferenciada por hombres y mujeres, tiene una distribución normal.

# 17

## Aproximación de integrales

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

### 17.1. 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  $\Re$ , 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

$$f(x) \approx f(x_0) \exp\left(-\frac{c}{2}(x - x_0)^2\right) \quad (17.1)$$

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

La expresión 17.1 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:

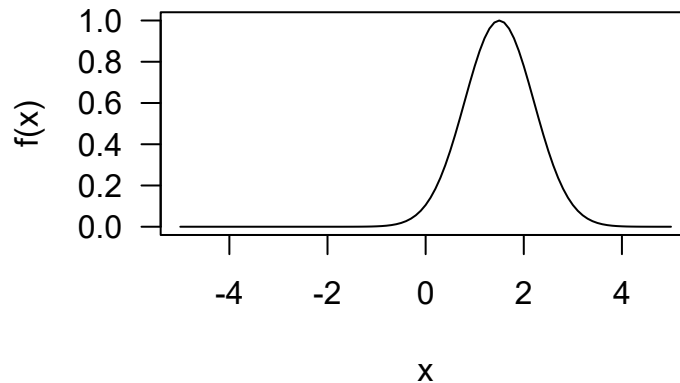
$$I = \int_{-\infty}^{\infty} f(x) d(x) = f(x_0) \sqrt{2\pi/c} \quad (17.2)$$

### 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 17.1:** 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 17.2 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 17.2 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*

---

- Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723.
- Akaike, H. (1983). Information measures and model selection. *Bulletin of the International Statistical Institute*, 50:277–290.
- Hernández, F. & Correa, J. (2018). *Gráficos con R*. Universidad Nacional de Colombia, Medellín, Colombia, primera edition. ISBN xxx-xxxxxxx.
- Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464.
- Wei, T. and Simko, V. (2016). *corrplot: Visualization of a Correlation Matrix*. R package version 0.77.
- Wickham, H. (2015). *R Packages*. O’Reilly Media, Inc.
- Wickham, H. (2016). *readxl: Read Excel Files*. R package version 0.1.1.
- Wickham, H. and Chang, W. (2016). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 2.2.1.
- 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 0.3.9.



---

## Índice alfabético

---

.csv, 63  
.txt, 63  
  
addmargins, 75  
array, 11  
arreglo, 11  
asignación, 29  
  
bloc de notas, 64  
  
ceiling, 42  
coeficiente de variación, 93  
cor, 97  
correlación, 97  
corrplot, 100  
cuantiles, 96  
cuartiles, 96  
  
data.frame, 13  
deciles, 96  
desviación, 89  
distribuciones discretas, 103  
  
else, 48  
estilo, 21  
Excel, 63  
  
fitdistr, 133  
floor, 42  
for, 49  
función, 27, 53  
function, 27  
  
gamlss, 137  
guía de estilo, 21  
  
hist, 76  
  
if, 47  
  
ifelse, 48  
  
lectura de bases de datos, 63  
length, 35  
list, 16  
lista, 16  
  
marco de datos, 13  
matrices, 9  
max, 35  
mean, 82  
media, 82  
median, 83  
mediana, 83  
min, 35  
moda, 84  
  
nlminb, 132  
  
objetos, 7  
operaciones básicas, 29  
operadores lógicos, 33  
optimize, 130  
ordenar, 43  
  
partes de función, 53  
percentiles, 96  
posición, 43  
prod, 35  
prop.table, 73  
pruebas lógicas, 31  
  
qqline, 147  
qqnorm, 147  
QQplot, 147  
quantile, 96  
  
range, 35, 88

rango, 88  
rank, 43  
read.table, 66  
readxl, 68  
rep, 40  
repeat, 51  
repeticiones, 40  
round, 42  
  
sd, 89  
secuencias, 38  
seq, 38  
sort, 43  
subset, 14  
sum, 35  
  
tablas de frecuencia, 71  
table, 71  
trunc, 42  
  
var, 91  
varianza, 91  
vector, 7  
  
which.max, 35  
which.min, 35  
while, 50  
with, 34