

Manual de R

Freddy Hernández Barajas

Olga Cecilia Usuga Manco

2019-05-02

Índice general

Índice de cuadros	5
Índice de figuras	7
Prefacio	7
Estructura del libro	7
Información del software y convenciones	8
Bloques informativos	8
Agradecimientos	8
Sobre los autores	8
0.1. Introducción	1
0.2. Orígenes	1
0.3. Descarga e instalación	1
0.4. Apariencia del programa	3
0.5. Tipos de objetos	4
0.5.1. Vectores	4
0.5.2. Matrices	6
0.5.3. Arreglos	7
0.5.4. Marco de datos	8
0.5.5. Listas	11
EJERCICIOS	13
0.6. Guía de estilo	13
0.6.1. Nombres de los archivos	14
0.6.2. Nombres de los objetos	14
0.6.3. Longitud de una línea de código	14
0.6.4. Espacios	14
0.6.5. Asignación	16
0.6.6. Punto y coma	16
0.7. Creación de funciones en R	16
0.7.1. Función en R	17
0.7.2. Partes de una función en R	17
EJERCICIOS	21
0.8. Lectura de bases de datos	24
0.8.1. ¿En qué formato almacenar una base de datos?	24
0.8.2. Función <code>read.table</code>	26
0.8.3. Lectura de bases de datos en Excel	28
EJERCICIOS	29
0.9. Tablas de frecuencia	29
0.9.1. Tabla de contingencia con <code>table</code>	29
0.9.2. Función <code>prop.table</code>	31
0.9.3. Función <code>addmargins</code>	32
0.9.4. Función <code>hist</code>	33
EJERCICIOS	36
0.10. Medidas de tendencia central	36
0.10.1. Media	37
0.10.2. Mediana	38

0.10.3. Moda	38
0.11. Medidas de variabilidad	39
0.11.1. Rango	40
0.11.2. Desviación estándar muestral (S)	41
0.11.3. Varianza muestral (S^2)	42
0.11.4. Coeficiente de variación (CV)	43
0.12. Medidas de posición	44
0.12.1. Cuantiles	44
0.13. Curiosidades	45
0.13.1. ¿Cómo verificar si un paquete no está instalado para instalarlo de forma automática?	45



Índice de cuadros

0.1. Ejemplo de una base de datos simple.	24
0.2. Base de datos para practicar lectura.	29

Índice de figuras

1.	Robert Gentleman (izquierda) y Ross Ihaka (derecha) creadores de R.	2
2.	Página del Cran.	2
3.	Página de instalación para la primera ocasión.	3
4.	Página de descarga.	3
5.	Apariencia del acceso directo para ingresar a R.	3
6.	Apariencia de R.	4
7.	Ilustración de una función, tomada de www.mathinsight.org	18
8.	Ilustración del punto medio entre dos puntos, tomada de https://www.slideshare.net/bigpassy/midpoint-between-two-points	23
9.	Forma de almacenar los datos en Excel.	25
10.	Almacenamiento de los datos en bloc de notas usando la barra espaciadora	25
11.	Almacenamiento de los datos en bloc de notas usando la barra tabuladora	26
12.	Ubicación de los puntos del ejemplo con límites en color azul.	35
13.	Boxplot para el precio de los apartamentos dada la ubicación.	43

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 estadístico.

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

Estructura del libro

El libro está estructurado de la siguiente manera.

En el capítulo 0.1 se presenta una breve introducción sobre el lenguaje de programación R; en el capítulo 0.5 se explican los tipos de objetos más comunes en R; en el capítulo 0.6 se muestran las normas de estilo sugeridas para escribir código en R; el capítulo ?? presenta las funciones básicas que todo usuario debe conocer para usar con éxito R; el capítulo 0.7 trata sobre cómo crear funciones; el capítulo 0.8 muestra cómo leer bases de datos desde R; en el capítulo 0.9 se ilustra la forma para construir tablas de frecuencia; en el capítulo 0.10 se muestra cómo obtener las diversas medidas de tendencia central para variables cuantitativas, el capítulo 0.11 muestra cómo calcular las medidas de variabilidad, en el capítulo 0.12 se ilustra cómo usar las funciones para obtener medidas de posición; en el capítulo ?? se muestra cómo obtener medidas de correlación entre pares de variables; en los capítulos ?? y ?? se tratan los temas de distribuciones discretas y continuas; en el capítulo ?? se aborda el tema de verosimilitud; en el capítulo ?? 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, 2018), 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.

0.1. Introducción

0.2. Orígenes

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

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

0.3. Descarga e instalación

Para realizar la instalación de R usted debe visitar la página del CRAN (*Comprehensive R Archive Network*) disponible en este enlace¹¹. Una vez ingrese a la página encontrará un cuadro similar al mostrado en la Figura 2 donde aparecen los enlaces de la instalación para los sistemas operativos Linux, Mac y Windows.

Supongamos que se desea instalar R en Windows, para esto se debe dar clic sobre el hipervínculo **Download R for Windows** de la Figura 2. Una vez hecho esto se abrirá una página con el contenido mostrado en la Figura 3. Una vez ingrese a esa nueva página usted debe dar clic sobre el hipervínculo **install R for the first time** como es señalado por la flecha roja en la Figura 3.

Luego de esto se abrirá otra página con un encabezado similar al mostrado en la Figura 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** como es señalado

¹<https://www.r-bloggers.com/>

²<http://r-es.org/>

³<http://r.789695.n4.nabble.com/>

⁴<http://r-br.2285057.n4.nabble.com/>

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

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

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

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

⁹<https://www.r-statistics.com/>

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

¹¹<https://cran.r-project.org/>



Figura 1 Robert Gentleman (izquierda) y Ross Ihaka (derecha) creadores de R.

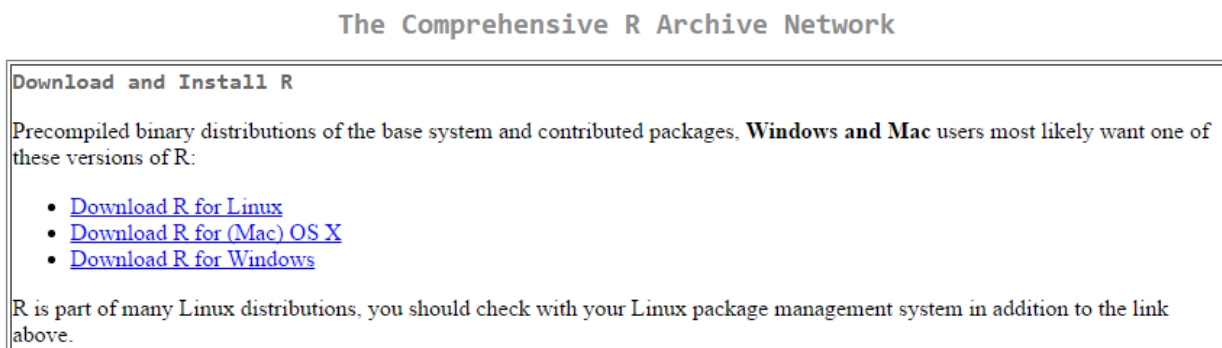


Figura 2 Página del Cran.

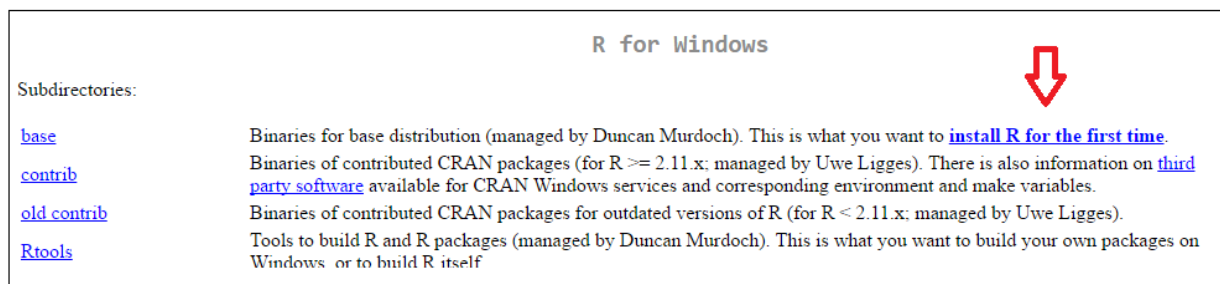


Figura 3 Página de instalación para la primera ocasión.

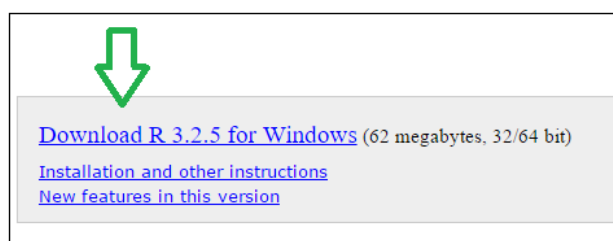


Figura 4 Página de descarga.

por la flecha verde. Luego de esto se descargará el instalador R en el computador el cual deberá ser instalado con las opciones que vienen por defecto.

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

0.4. 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 5 se muestra la apariencia del acceso directo para ingresar a R.

¹²<http://tinyurl.com/jd7b9ks>



Figura 5 Apariencia del acceso directo para ingresar a R.

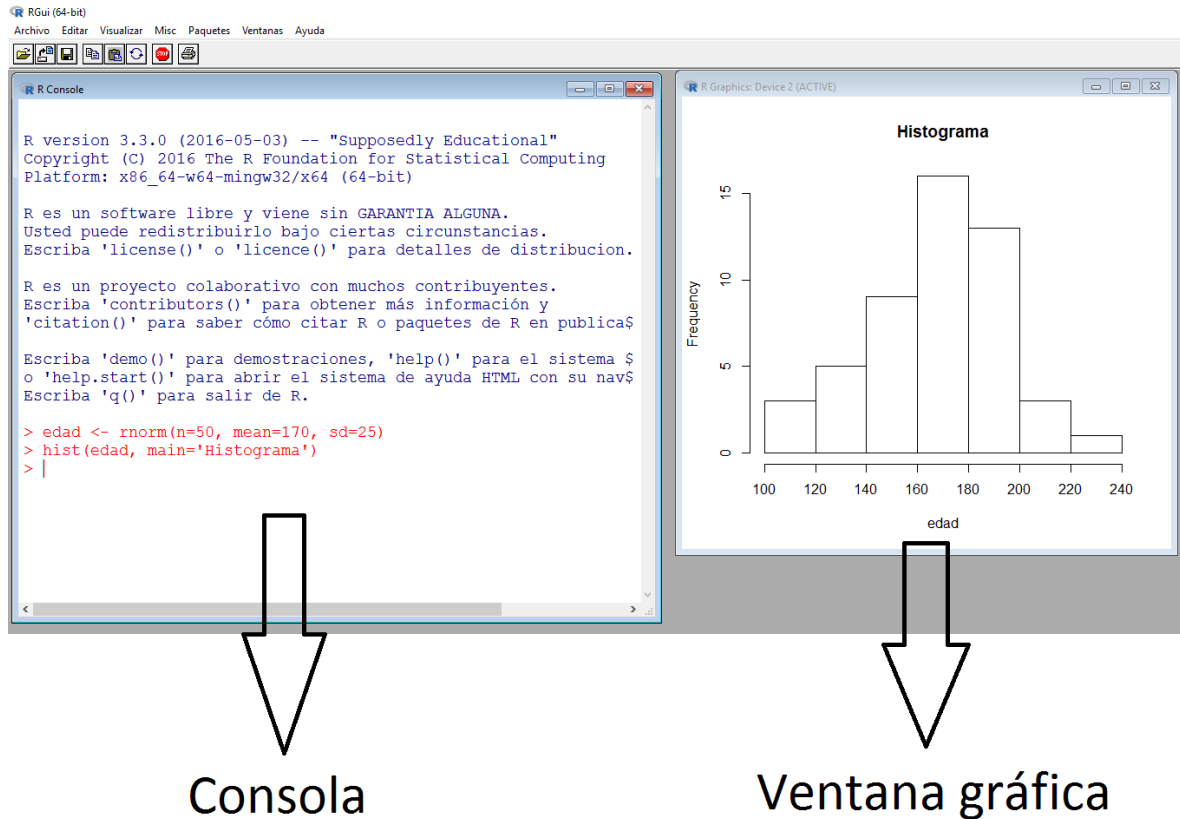


Figura 6 Apariencia de R.

Al abrir R aparecerá en la pantalla de su computador algo similar a lo que está en la Figura 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.

0.5. 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.

0.5.1. 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"
```

0.5.1.1. ¿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)]`

0.5.2. Matrices

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

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

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

```
mimatriz
```

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

0.5.2.1. ¿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
```

0.5.3. 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"
```

0.5.3.1. ¿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"
```

0.5.4. Marco de datos

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

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


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

```
mimarco
```

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

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

0.5.4.1. ¿Cómo extraer elementos de un marco de datos?

Para recuperar las variables (columnas) almacenadas en un marco de datos se puede usar el operador `$`, corchetes simples `[]` o corchetes dobles `[][]`. A continuación algunos ejemplos para entender las diferencias entre estas opciones.

Ejemplo

Si queremos extraer la variable `deporte` del marco de datos `mimarco` como un vector 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` como vector es indicando el número de la columna donde se encuentra la variable.

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

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

Otra forma de extraer la variable `deporte` como vector es usando `[][]` y dentro el nombre de la variable.

```
mimarco[["deporte"]]
```

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

Si usamos `mimarco["deporte"]` el resultado es la variable `deporte` pero en forma de marco de datos, no en forma vectorial.

```
mimarco["deporte"]
```

```
##   deporte
## 1    TRUE
## 2    TRUE
## 3     NA
## 4   FALSE
## 5    TRUE
```

Si queremos extraer un marco de datos sólo con las variables deporte y edad podemos usar el siguiente código.

```
mimarco[c("deporte", "edad")]
```

```
##   deporte edad
## 1    TRUE   15
## 2    TRUE   19
## 3     NA   13
## 4   FALSE   NA
## 5    TRUE   20
```

Por otra, 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
```

0.5.4.2. ¿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 >= 17)
```

```
##   edad deporte comic.fav
## 2   19    TRUE  Superman
## 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. 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.

0.5.5. Listas

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

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

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

```
milista
```

```
## $E1
## [1] 0.7209 0.8758 0.7610 0.8861 0.4565
##
## $E2
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    3    5    7    9   11
## [2,]    2    4    6    8   10   12
##
## $E3
##   edad deporte comic.fav
## 1   15     TRUE      <NA>
## 2   19     TRUE   Superman
## 3   13      NA     Batman
## 4   NA    FALSE      <NA>
## 5   20     TRUE     Batman
```

0.5.5.1. ¿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 la primeras 20 letras MAYÚSCULAS usando la función `LETTERS`.
2. Construya una matriz de 10×10 con los primeros 100 números positivos pares.
3. Construya una matriz identidad de dimension 3×3 . 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)
```

0.6. 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*¹³.

0.6.1. 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`

0.6.2. 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:

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

Para funciones:

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

0.6.3. 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.

0.6.4. 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
```

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

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 {
  # Correcto
```

```

    one or more lines
}

if (condition) {
    one or more lines
}
else {
    one or more lines
}

if (condition)
    one line
else
    one line

```

0.6.5. Asignación

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

```

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

```

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

0.6.6. 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
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.

0.7. Creación de funciones en R

En este capítulo se explica cómo crear funciones en R.

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

0.7.1. Función en R

Una función es un conjunto de instrucciones que convierten las entradas (*inputs*) en resultados (*outputs*) deseados. En la Figura 7 se muestra una ilustración de lo que es una función o máquina general.

0.7.2. 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 una valor al 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.

```
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

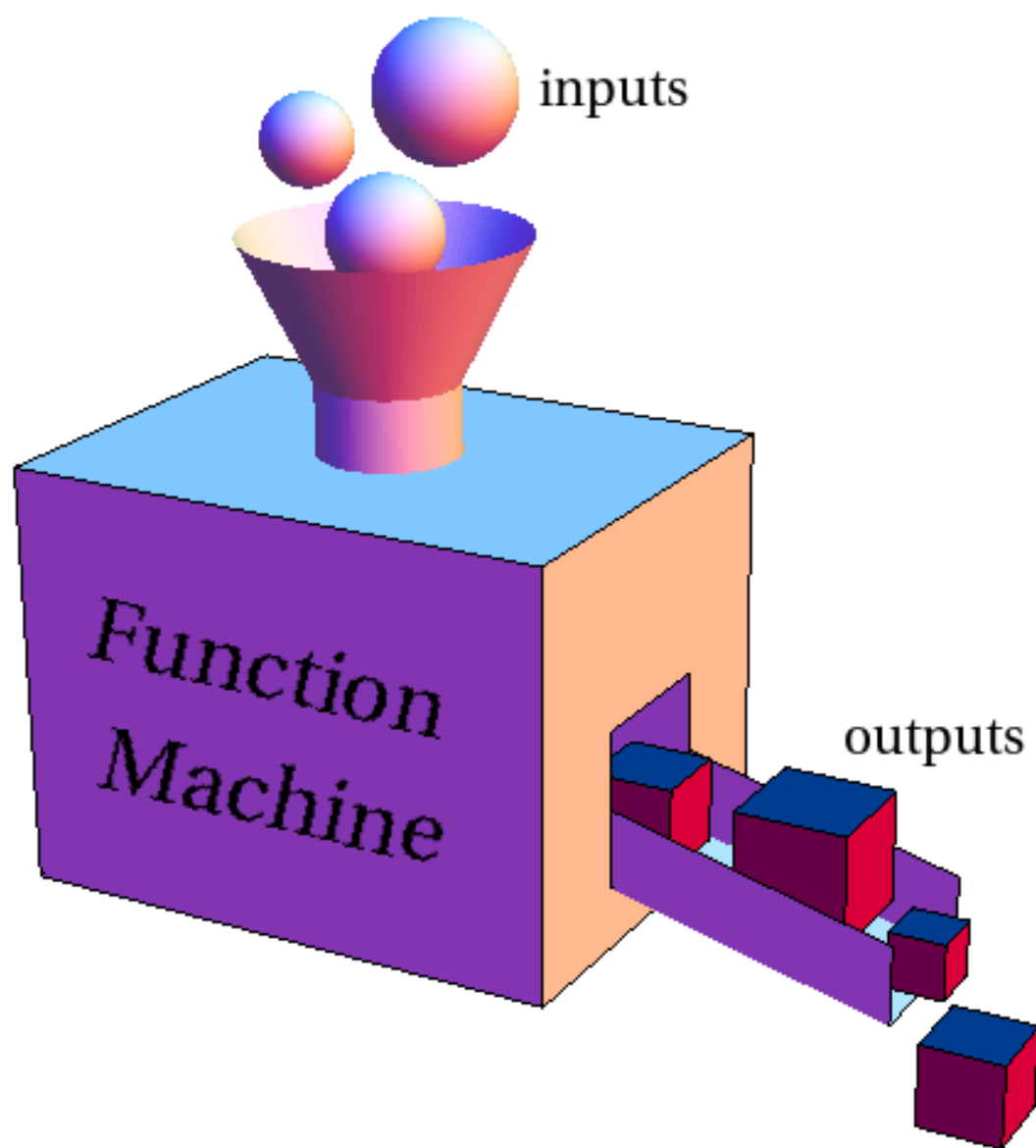


Figura 7 Ilustración de una función, tomada de www.mathinsight.org

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] 0.001137 0.391203 0.462495 0.388144
##
## $suma
## [1] 1.243
##
## $cantidad
## [1] 4
```

Probando la función con cota de tres.

```
fun2(cota=3)
```

```
## $vector
## [1] 0.4025 0.1790 0.9517 0.4537 0.3268 0.9654
##
## $suma
## [1] 3.279
##
## $cantidad
## [1] 6
```

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. 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 $b = 4$.

6. 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.
7. 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)  
}
```

8. 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)  
}
```

9. 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 respectivamente. 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.

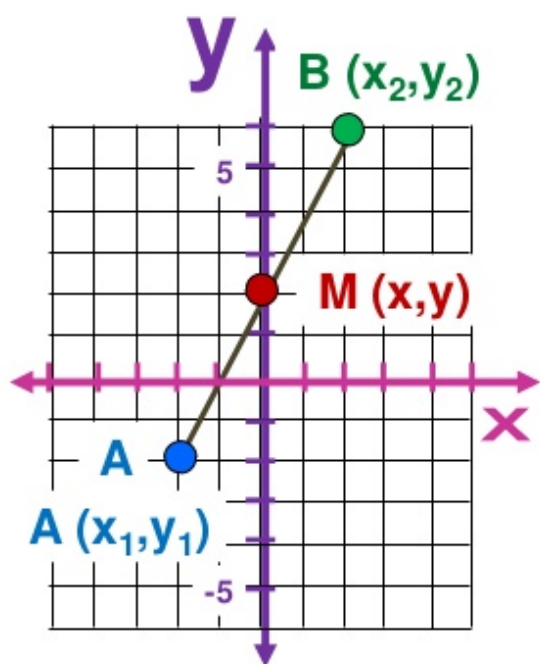
```
nota <- function(p1, p2, p3) {  
  ...  
  ...  
  ...  
}
```

10. 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)  
}
```

11. Construya una función que calcule las coordenadas del punto medio M entre dos puntos A y B . Vea la Figura 8 para una ilustración. ¿Cuáles cree usted que deben ser los parámetros de entrada de la función?

Midpoint Formula



To find the Midpoint
between two points:
Point A and Point B

The midpoint is (x, y)
where :

$$x = (x_1 + x_2) / 2 \quad \text{and}$$

$$y = (y_1 + y_2) / 2$$

Figura 8 Ilustración del punto medio entre dos puntos, tomada de <https://www.slideshare.net/bigpassy/midpoint-between-two-points>

Cuadro 0.1 Ejemplo de una base de datos simple.

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

0.8. Lectura de bases de datos

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

0.8.1. ¿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 0.1 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.

0.8.1.1. Almacenamiento de información en Excel

Para almacenar la información del Cuadro 0.1 en Excel, abrimos un nuevo archivo de Excel y copiamos la información tal como se muestra en la Figura 9. 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 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**.

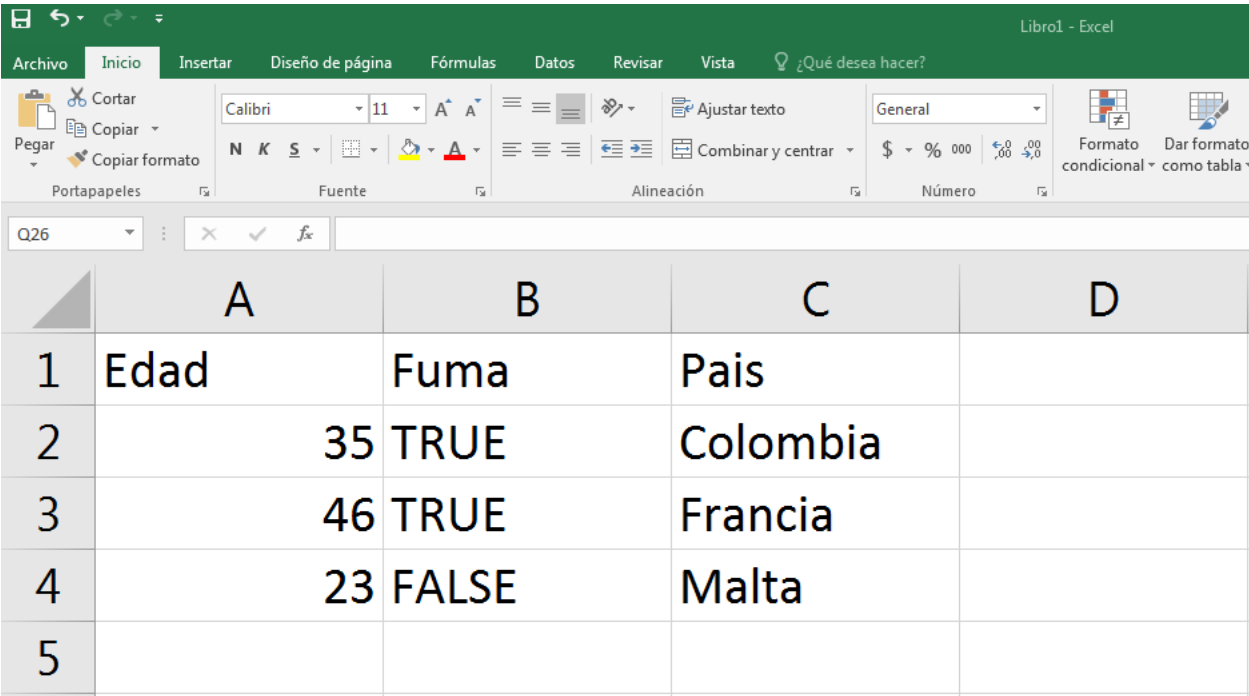
0.8.1.2. Almacenamiento de información en bloc de notas

Para almacenar la información del Cuadro 0.1 en bloc de notas, abrimos un nuevo archivo de bloc de notas y copiamos la información tal como se muestra en la Figura 10. 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 10 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 11 se muestra la información para el ejemplo, claramente se nota la organización de la información.

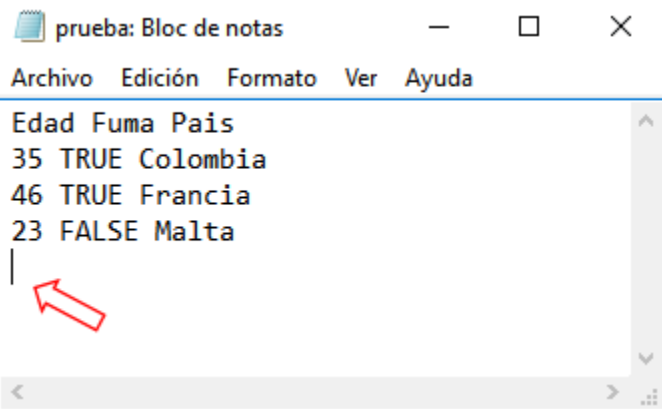


Una buena práctica es usar la barra tabuladora para separar, eso permite que la información se vea ordenada.



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

Figura 9 Forma de almacenar los datos en Excel.



```
prueba: Bloc de notas
Archivo Edición Formato Ver Ayuda
Edad Fuma Pais
35 TRUE Colombia
46 TRUE Francia
23 FALSE Malta
|
```

Figura 10 Almacenamiento de los datos en bloc de notas usando la barra espaciadora



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.

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

- ### Ejemplo

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 9, 10 y 11, vamos a suponer que los nombres para cada uno de ellos son `base1.csv`, `base2.txt` y `base3.txt` respectivamente.

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 un 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
```

0.8.2.0.2. 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
```

0.8.2.0.3. 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.

0.8.3. 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 and Bryan, 2019) y en particular la función **readxl**. A continuación un ejemplo de cómo proceder en estos casos.

Ejemplo

En este enlace¹⁵ 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")
```

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 **read_xl** 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')
as.data.frame(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')
as.data.frame(padres) # Para ver el contenido
```

```
##  Edad  EstCivil NumHijos
## 1   45    Soltero        1
```

¹⁵https://github.com/fhernanb/datos/blob/master/BD_Excel.xlsx

Cuadro 0.2 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

```
## 2  50    Casado      0
## 3  35    Casado      3
## 4  65 Divorciado    1
```

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

EJERCICIOS

Realice los siguiente ejercicios propuestos.

1. En el Cuadro 0.2 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 0.2 fuente.
2. En la url 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.

0.9. 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.

0.9.1. 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
```

0.9.2. 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
```

0.9.3. 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
```


0.9 Tablas de frecuencia

```
## 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.

0.9.4. 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] 6 17 67 83 26 1
##
## $density
## [1] 0.006 0.017 0.067 0.083 0.026 0.001
##
## $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] 90 109 1
##
## $density
## [1] 0.0225 0.0545 0.0005
##
## $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 12 se muestran los 9 puntos y con color azul se representan los límites de los intervalos.

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.

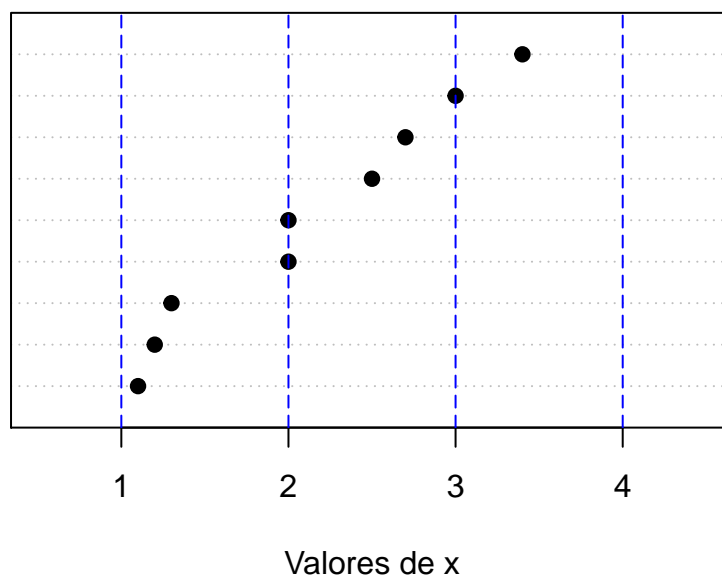


Figura 12 Ubicación de los puntos del ejemplo con límites en color azul.

```
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 0.2 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?

0.10. 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
```

0.10.1. 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.

0.10.2. Mediana

Para calcular la mediana de una variable cantitativa 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
```

0.10.3. 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.

0.11 Medidas de variabilidad

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.

0.11. 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
```

0.11.1. 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 si.

0.11.2. Desviación estándar muestral (S)

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

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

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

Ejemplo

Suponga que queremos obtener la desviación estándar muestral para la variable precio de los apartamentos. Para obtener la desviación solicitada usamos el siguiente código:

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

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

Ejemplo

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

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

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

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

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

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

0.11.3. 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 13 se muestra el boxplot y se ve claramente la dispersión de los precios en el Poblado. El código usado para generar la Figura 13 se presenta a continuación.

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

Ejemplo

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

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

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

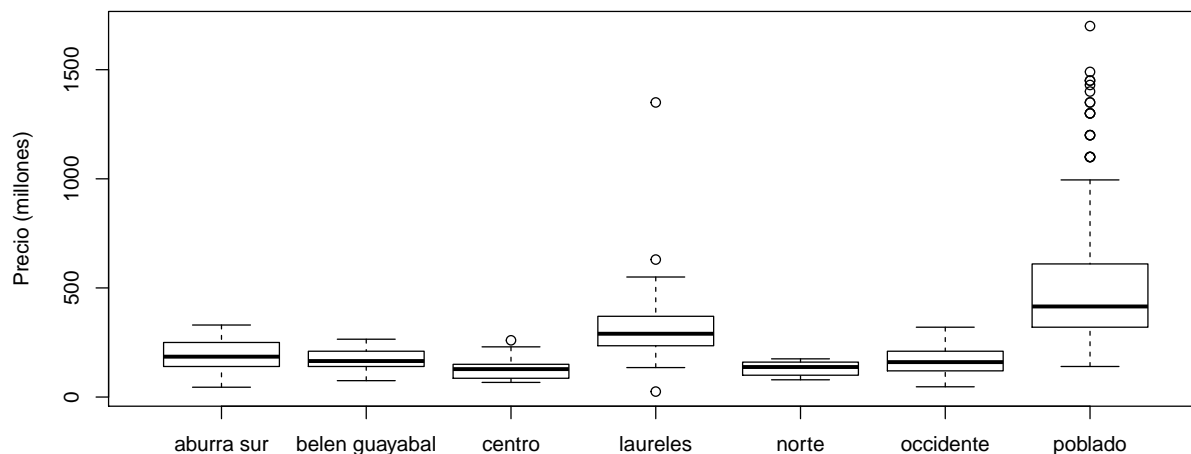


Figura 13 Boxplot para el precio de los apartamentos dada la ubicación.

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

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

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

0.11.4. 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
```

0.12. 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
```

0.12.1. 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
```

0.13. Curiosidades

En este capítulo se mostrarán algunos procedimientos de R para solucionar problemas frecuentes.

0.13.1. ¿Cómo verificar si un paquete no está instalado para instalarlo de forma automática?

Muchas veces compartimos código de R con otros colegas y si ellos no tienen instalados ciertos paquetes el código no funcionará. Para evitar ese problema podemos colocar al inicio del código unas líneas que chequeen si ciertos paquetes están instalados o no, si están instalados, se cargan esos paquetes y caso contrario, el código instala los paquetes y luego los carga, todo de forma automática sin que el usuario tenga que identificar los paquetes que le faltan.

Ejemplo

El código mostrado abajo revisa si los paquetes `knitr`, `png` y `markdown` están instalados e instala los ausentes y luego carga todos los paquetes que estén en el vector `packages`.

```
packages <- c("knitr", "png", "markdown")  
package.check <- lapply(packages, FUN = function(x) {  
  if (!require(x, character.only = TRUE)) {  
    install.packages(x, dependencies = TRUE)  
    library(x, character.only = TRUE)  
  }  
})
```

Bibliografía

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

Wickham, H. and Bryan, J. (2019). *readxl: Read Excel Files*. R package version 1.3.0.

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

Xie, Y. (2018). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.9.