Probabilidad y Estadística: Laboratorio 01

Felipe Ossa y Mauricio Toro

Segundo Semestre 2023

Equipo

Profesor	Correo	Día	Ayudantes
Felipe Ossa	foossa@uc.cl	Lunes (5,6)	Por definir
Mauricio Toro	matoro4@mat.uc.cl	Martes (7,8)	Por definir
Felipe Ossa	foossa@uc.cl	Miércoles (1,2,9)	Por definir
Mauricio Toro	matoro4@mat.uc.cl	Jueves (3,4,10)	Por definir

NOTA: Revisa tu sección en Mi Portal UC, NO EN CANVAS

Evaluaciones

- Lunes (secciones 5 y 6):
  - Evaluación 1: lunes 25 de septiembre
  - Evaluación 2: lunes 27 de noviembre
- Martes (secciones 7 y 8):
  - Evaluación 1: martes 26 de septiembre
  - Evaluación 2: martes 28 de noviembre
- Miércoles (secciones 1, 2 y 9):
  - Evaluación 1: miércoles 27 de septiembre
  - Evaluación 2: miércoles 29 de noviembre
- Jueves (secciones 3, 4 y 10):
  - Evaluación 1: jueves 28 de septiembre
  - Evaluación 2: jueves 30 de noviembre

NOTA: Revisa tu sección en Mi Portal UC, NO EN CANVAS

Introducción a R

¿Qué es R?

R es un conjunto integrado de programas para manipulación de datos, cálculo y gráficos.

Introducción a R

R es un software estadístico de libre acceso el cual puede ser utilizado en diferentes sistemas operativos como Windows, MacOS y Linux.

R es un lenguaje de programación en el que se introducen códigos para posteriormente ser ejecutados.

Una de las grandes ventajas de R es que es un programa de código abierto en el que miles de personas de todo el mundo colaboran en el desarrollo de nuevas metodologías, de manera que se pueden acceder a los paquetes descargándolos como también compartir los propios con otros.

Introducción a R

Instalación

La descarga del archivo de instalación de R se realiza desde uno de los links de abajo dependiendo del sistema operativo:

- Microsoft Windows: http://cran.r-project.org/bin/windows/base/
- OSX: http://cran.r-project.org/bin/macosx/
- Linux: http://cran.r-project.org/bin/linux/

Una vez instalado R, se puede instalar RStudio desde:

- https://www.rstudio.com/products/rstudio/download/

Como alternativa también existe RStudio Cloud (ahora llamado Posit Cloud):

- https://rstudio.cloud/

Introducción a R

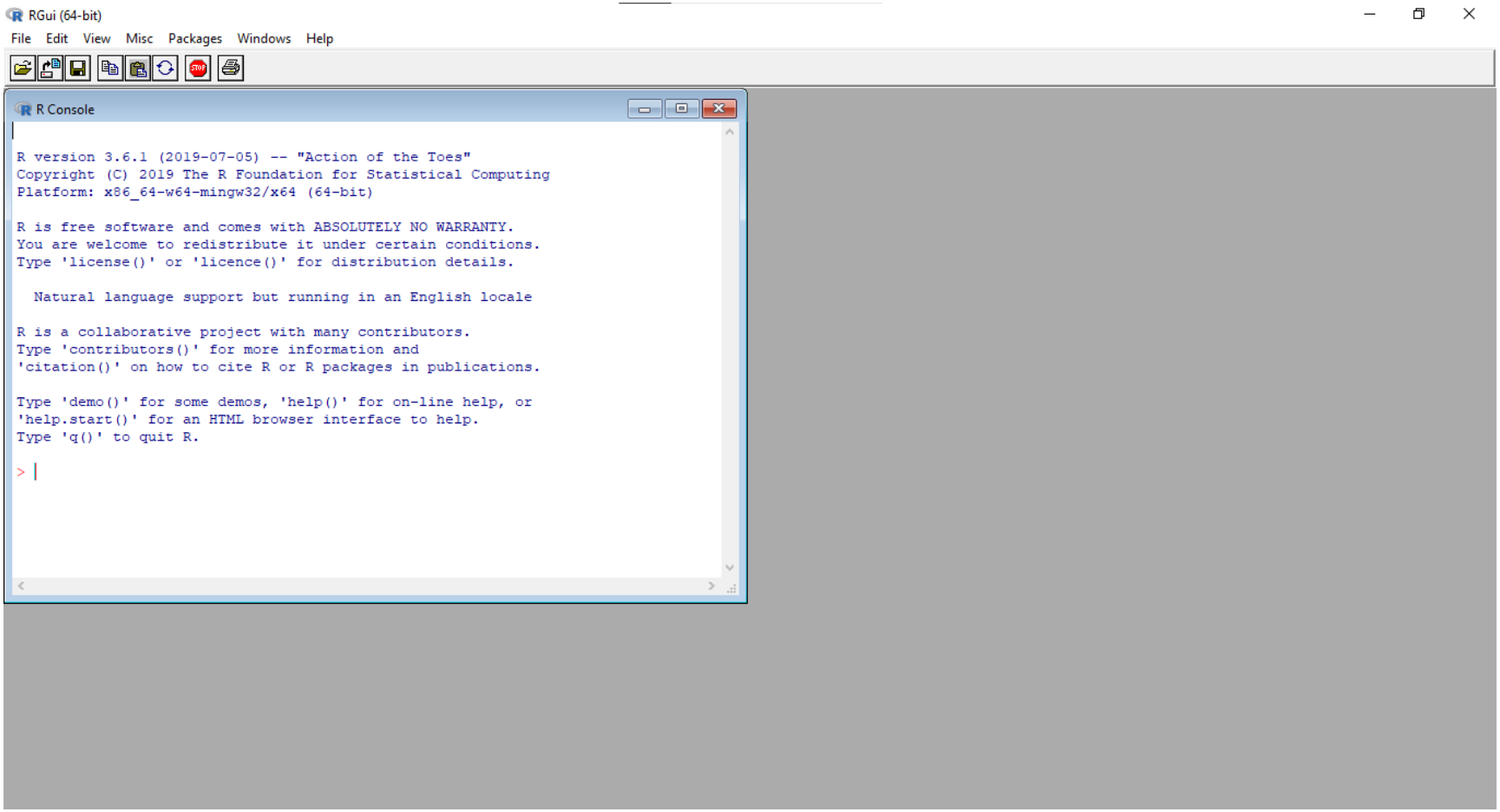
¿R? ¿RStudio? ¿RStudio Cloud? ¿Y por qué?

Al instalarse R se cuenta por defecto con una interfaz básica llamada *RGui*. También es llamada *R x86*. Tiene todas las facultades que R puede generar, pero el manejo de ventanas y organización puede ser un poco desordenado.

No obstante, RGui ocupa pocos recursos adicionales y funciona muy establemente en cualquier plataforma.

Introducción a R

¿R? ¿RStudio? ¿RStudio Cloud? ¿Y por qué?



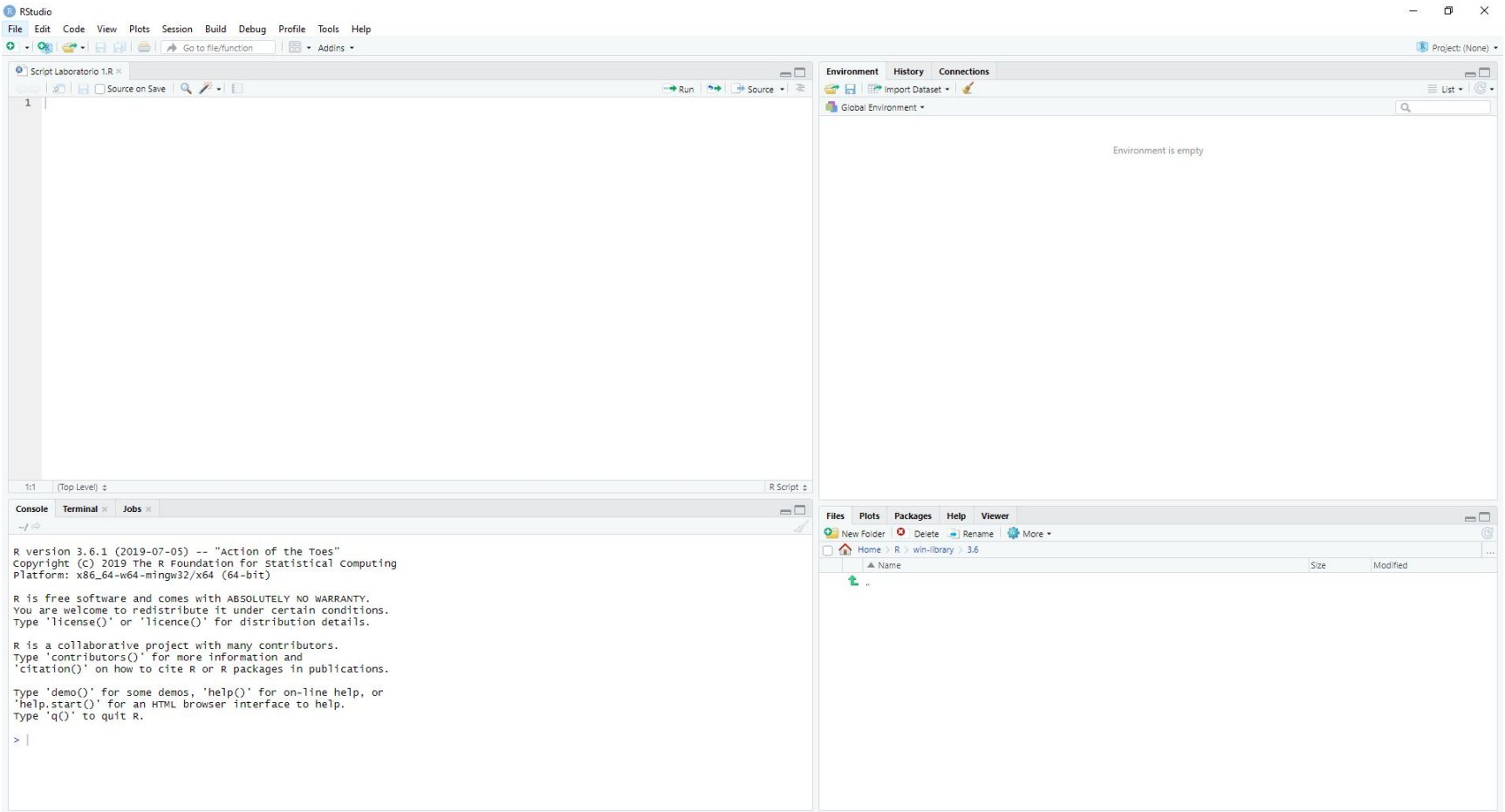
Introducción a R

¿R? ¿RStudio? ¿RStudio Cloud? ¿Y por qué?

RStudio, en cambio es una interfaz programada de modo de utilizar R en una plataforma mucho más comprensiva e interactiva.

- Los gráficos se muestran en una pestaña especial
- Se puede monitorear las variables y recursos utilizados y cargados en todo momento.
- Autocompletar incluido
- Uso de pestañas y paneles en lugar de ventanas.
- Compatibilidad con otros lenguajes.

Introducción a R



Introducción a R

¿R? ¿RStudio? ¿RStudio Cloud? ¿Y por qué?

RStudio creció en vías de compatibilidad con otros software de motor además de R, con lo que hoy se conoce como Posit.

El equipo de RStudio también creó una versión que se accede por internet y se procesa en la nube. Puedes crear una cuenta gratis (sin spam) y puedes guardar proyectos y utilizarlos de manera análoga.

- Su gran ventaja es la portabilidad cuando utilizas diferentes computadores, y no requiere de mayor instalación.
- Como todo ambiente tipo *Cloud*, hay que realizar mayor tráfico de archivos y datos desde y hacia nuestros computadores de forma manual.
- Y por supuesto, utiliza internet. Los servidores son robustos pero una conexión débil puede ocasionar cortes en la sesión.

Introducción a R

En la consola de R, se pueden escribir directamente comandos, y pulsar **enter** para ver el resultado, salida o retorno. Sin embargo, esta no es la manera más eficiente de trabajar en R.

Si se quiere guardar el trabajo, corregirlo, repetirlo, etc., es más conveniente usar el editor de R. Se debe seleccionar archivo, nuevo Script (documento en blanco del editor) en el cual se puede escribir los programas y guardar.

La ejecución del código desde el script se hace desde cualquier posición del cursor en la línea o iluminando parte de ella o más líneas, y luego presionar:

Windows: Tecla F5 o Control + Enter, MacOS: Cmd + Enter.

Se puede incluir comentarios que R no leerá si utilizamos el símbolo # al comienzo de la línea.

Introducción a R

Operaciones y comandos

En R es posible llevar a cabo distintas operaciones matemáticas y aritméticas usando operadores básicos tales como +, -, /, *,** .

Calcule en la consola de R las siguientes operaciones y presione ENTER. Observe el resultado.

```
1+2+3
```

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

```
100+200
```

```
## [1] 300
```

```
3.14+2.17
```

```
## [1] 5.31
```

Introducción a R

```
2-1
```

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

```
3/4
```

```
## [1] 0.75
```

```
1/3
```

```
## [1] 0.3333333
```

```
2*33
```

```
## [1] 66
```

```
2**3
```

```
## [1] 8
```

```
2*5+200/2**3
```

```
## [1] 35
```

Introducción a R

Funciones usadas en R

Los resultados obtenidos de cualquier operación, o de aplicar una función, van apareciendo antecedidos por el símbolo $[n]$, mientras que cualquier código o sentencia que escribamos aparecerá se destacará con $>$.

A continuación se presentan distintos comandos de utilidad usados comúnmente en R.

Funciones matemáticas:

Nombre de la función	Descripción
`sqrt`	Raíz cuadrada
`log, log2, log10`	Logaritmos
`exp`	Función exponencial
`abs`	Valor absoluto
`sign`	Signo
`cos, sin, tan`	Funciones trigonométricas
`acos, asin, atan`	Funciones trigonométricas inversas
`%%`	Resto de una división
`factorial, lfactorial`	Factorial y su logaritmo

Introducción a R

Algunos ejemplos:

```
sqrt(3)
```

```
## [1] 1.732051
```

```
log(10); log2(10); log10(10)
```

```
## [1] 2.302585
```

```
## [1] 3.321928
```

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

```
exp(2)
```

```
## [1] 7.389056
```

```
abs(-3.5)
```

```
## [1] 3.5
```

Introducción a R

```
sign(-7)
```

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

```
sign(7)
```

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

```
cos(2*pi); sin(pi); tan(pi)
```

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

```
## [1] 1.224606e-16
```

```
## [1] -1.224647e-16
```

```
acos(0); asin(1); atan(-1)
```

```
## [1] 1.570796
```

```
## [1] 1.570796
```

```
## [1] -0.7853982
```

Introducción a R

```
10%%3
```

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

```
factorial(4)
```

```
## [1] 24
```

```
lfactorial(5)
```

```
## [1] 4.787492
```

Objetos

Definición de objetos

R es un software que permite la creación de objetos de varios tipos: alfanuméricos, escalares, vectores, matrices, etc. Los valores que se asignen a los objetos quedan guardados en la memoria de R mientras dure la sesión de trabajo. Los objetos pueden definirse de la siguiente forma:

```
objeto <- expresión
```

donde `objeto` es el nombre que se le asigna al objeto, y `expresión` puede ser una fórmula, un vector, una palabra, etc.

Objetos

En R podemos definir distintos tipos de variables, los que conoceremos primero son los siguientes:

- Variables numéricas: se definen los objetos como el número u operación numérica a definir.

```
a <- 3*6+9/7
a
```

```
## [1] 19.28571
```

```
b <- sqrt(9)/log(10)
b
```

```
## [1] 1.302883
```

Objetos

- Variables booleanas: en R los valores booleanos se definen como `TRUE` o `T` y `FALSE` o `F`.

```
d <- T
d
```

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

```
e <- FALSE
e
```

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

Objetos

- Variables de texto: los strings o char se escriben siempre entre comillas " ".

```
f <- "Laboratorio EYP1113"
f
```

```
## [1] "Laboratorio EYP1113"
```

```
g <- "Primer Semestre 2022"
g
```

```
## [1] "Primer Semestre 2022"
```

Vectores

Para crear vectores debemos utilizar el comando `c()`.

```
numeros.ejemplo <- c(1,2,3,4,5)
textos_ej <- c("a","b","c","d","e")
booleanosEJEMPLO <- c(T,F,TRUE,FALSE)
```

Para nombrar ciertos elementos de un vector, se utiliza el comando `names`.

```
notas  <- c(4.5,5.1,5.5,4.7,5.5,6.7)
nombres <- c("I1","I2","I3","I4","Lab 1","Lab 2")
names(notas) <- nombres
notas
```

```
##    I1    I2    I3    I4 Lab 1 Lab 2
##   4.5   5.1   5.5   4.7   5.5   6.7
```

Vectores

Operaciones con vectores

En R podemos realizar operaciones con vectores, por ejemplo:

```
v1 <- c(1,1,1,3)
v2 <- c(2,0,2,3)
v1+v2
```

```
## [1] 3 1 3 6
```

```
v1-v2
```

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

```
v1*v2
```

```
## [1] 2 0 2 9
```

Vectores

```
v1/v2
```

```
## [1] 0.5 Inf 0.5 1.0
```

```
v1*2
```

```
## [1] 2 2 2 6
```

```
v1/2
```

```
## [1] 0.5 0.5 0.5 1.5
```

Vectores

También podremos crear funciones, pero R, viene con funciones o comandos ya creados para operar con vectores o para crearlos tales como:

```
sum(v1)
```

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

```
prod(v1)
```

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

```
mean(v1)
```

```
## [1] 1.5
```

Vectores

```
sd(v1)
```

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

```
min(v1)
```

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

```
max(v1)
```

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

Pueden encontrar una lista más completa de funciones de R en: https://cran.r-project.org/doc/contrib/Short-refcard.pdf

Vectores

Expresiones lógicas

Podemos realizar comparaciones lógicas, por ejemplo:

```
1>2
```

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

```
1<2
```

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

```
1>=2
```

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

```
1<=2
```

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

```
1==2
```

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

```
1!=2
```

Vectores

Comparaciones con los vectores creados anteriormente:

```
v1<2
```

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

```
v1==1
```

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

```
v1<v2
```

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

```
v1>v2
```

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

Vectores

Para unir condiciones lógicas el "y" (and) se representa con carácter & y el "o" (or) con el carácter "|".

```
v2<3 & v2>0
```

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

```
v2<3 | v2>0
```

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

Vectores: funciones which e %in%

Dos funciones lógicas muy útiles en R son `which()` y el operador `%in%`.

La función `which()` sirve para obtener los índices de las filas de una base de datos que cumplan alguna condición dada.

```
which(v1==1)
```

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

```
which(v2==1)
```

```
## integer(0)
```

```
c(2,4,6) %in% v1
```

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

Vectores

El operador %in% indica si un valor o los componentes de un vector se encuentran dentro de los valores de otro vector. Esto nos retorna un valor booleano.

```
1 %in% v1
```

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

```
1 %in% v2
```

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

Manipulación de vectores

Trabajaremos con los vectores `numeros` y `textos`.
Para conocer el largo de un vector está el comando `length()`.

```
length(nombres)
```

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

```
length(textos_ej)
```

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

Manipulación de vectores

Para poder acceder al `i`-ésimo componente del vector `v` debemos ejecutar `v[i]`.

```
nombres[1]
```

```
## [1] "I1"
```

```
notas[2]
```

```
##  I2
## 5.1
```

Manipulación de vectores

Para acceder a más de un índice, entonces debemos entregar un vector dentro de los corchetes de la forma `v[c(i,j,k,...)]`

```
nombres[c(1,2,5)]
```

```
## [1] "I1"    "I2"    "Lab 1"
```

```
notas[c(2,3,6)]
```

```
##    I2    I3 Lab 2
##   5.1   5.5   6.7
```

Manipulación de vectores

Un modo de crear una secuencia desde un índice `i` hasta un índice `j` es con el comando `i:j`.

```
1:10
```

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

```
5:10
```

```
## [1]  5  6  7  8  9 10
```

Manipulación de vectores

Con lo visto anteriormente podemos acceder a varios términos continuos dentro de un vector de mayor longitud. Utilizaremos el vector `letters` para dar un ejemplo.

```
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

```
letters[4:14]
```

```
##  [1] "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
```

Manipulación de vectores

Retomando el uso de `names()` y la manipulación de vectores podemos utilizar:

```
notas[2]
```

```
##  I2
## 5.1
```

```
notas["Lab 2"]
```

```
## Lab 2
##   6.7
```

```
notas[c("Lab 3","Lab 1","Lab 2")]
```

```
##  <NA> Lab 1 Lab 2
##    NA   5.5   6.7
```

Manipulación de vectores

```
notas[notas>5]
```

```
##    I2    I3 Lab 1 Lab 2
##   5.1   5.5   5.5   6.7
```

Podemos guardar distintos filtros lógicos en variables auxiliares tales como:

```
notas>5
```

```
##    I1    I2    I3    I4 Lab 1 Lab 2
## FALSE  TRUE  TRUE FALSE  TRUE  TRUE
```

```
notas<6
```

```
##    I1    I2    I3    I4 Lab 1 Lab 2
##  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
```

Manipulación de vectores

```
notas==max(notas)
```

```
##    I1    I2    I3    I4 Lab 1 Lab 2
## FALSE FALSE FALSE FALSE FALSE  TRUE
```

```
notas==min(notas)
```

```
##    I1    I2    I3    I4 Lab 1 Lab 2
##  TRUE FALSE FALSE FALSE FALSE FALSE
```

```
notas>=mean(notas)
```

```
##    I1    I2    I3    I4 Lab 1 Lab 2
## FALSE FALSE  TRUE FALSE  TRUE  TRUE
```

```
notas<=mean(notas)
```

```
##    I1    I2    I3    I4 Lab 1 Lab 2
##  TRUE  TRUE FALSE  TRUE FALSE FALSE
```

Manipulación de vectores

Para manipular vectores con variables categóricas es conveniente convertir nuestra variable a `factor`. En el caso de ser nominal, no importa el orden.

```
textos2 <- factor(textos_ej)
```

En el caso en que nuestra variable sea ordinal, con el comando `factor()` podemos indicar el orden de los niveles de la variable con el argumento `levels=`.

```
opiniones <- c("Bueno","Malo","Neutro","Bueno","Malo")
opiniones <- factor(opiniones)
opiniones <- factor(opiniones,
                    levels=c("Malo","Neutro","Bueno"))
levels(opiniones)
```

```
## [1] "Malo"   "Neutro" "Bueno"
```

Ayuda

En R podemos buscar ayuda para funciones con los `help` o ?

```
help(class)
?class
```

Siempre se puede buscar ayuda en `Google` para problemas que tengan con sus códigos. `Stack Overflow` es un sitio recomendable para realizar consultas.

Matrices

En R podemos definir una matriz con el comando `matrix()`. Si al comando matrix le entregamos la secuencia guardada en `v3`.

```
v3 <- 1:15
m <- matrix(v3)
```

Por defecto, nos crea una matriz de tamaño 15×1. Nosotros podemos entregarle la cantidad de filas y/o columnas que queremos que tenga la matriz con los argumentos `nrow=` y `ncol=`. También podemos indicarle si queremos o no que la matriz se rellene por filas, con el argumento `byrow=` que puede tomar los valores `TRUE` o `FALSE`. Podemos ver un ejemplo creando la matriz de 3 filas y 5 columnas. Rellenando la matriz por filas.

Matrices

```
m1 <- matrix(v3, nrow=3, ncol=5, byrow=TRUE)
m1
```

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

Matrices

Podemos llamar a un elemento o a un conjunto de datos mediante de una matriz `m` llamando a `m[i,j]`, donde `i` y `j` puede ser un elemento o un vector de posiciones de fila o columna. Si se deja el espacio de `i` o `j` en blanco, R entiende que se está llamando a todas las filas o columnas. Podemos ver ejemplos ejecutando lo siguiente:

```
m1[1,]
```

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

```
m1[,1]
```

```
## [1]  1  6 11
```

```
m1[1,1]
```

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

Matrices

```
m1[c(1,2),c(3,4)]
```

```
##      [,1] [,2]
## [1,]    3    4
## [2,]    8    9
```

También podemos seleccionar la matriz sin las filas o columnas indicadas anteponiendo un signo "-" al número o vector indicado:

```
m1[-1,]
```

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

Matrices

```
m1[,-1]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    3    4    5
## [2,]    7    8    9   10
## [3,]   12   13   14   15
```

```
m1[-1,-1]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    7    8    9   10
## [2,]   12   13   14   15
```

```
m1[-c(1,2),-c(3,4)]
```

```
## [1] 11 12 15
```

Matrices

Tal como en los vectores podemos asignar nombres a las filas y columnas, esta vez con los comandos `rownames()` y `colnames()`.

```
nombresfilas <- c("f1","f2","f3")
nombrescolumnas <- c("c1","c2","c3","c4","c5")
rownames(m1) <- nombresfilas
colnames(m1) <- nombrescolumnas
m1
```

```
##    c1 c2 c3 c4 c5
## f1  1  2  3  4  5
## f2  6  7  8  9 10
## f3 11 12 13 14 15
```

Matrices

Operaciones con matrices

Para realizar operaciones simples con matrices tales como:

```
m1*3
```

```
##    c1 c2 c3 c4 c5
## f1  3  6  9 12 15
## f2 18 21 24 27 30
## f3 33 36 39 42 45
```

```
m1/3
```

```
##           c1        c2       c3       c4       c5
## f1 0.3333333 0.6666667 1.000000 1.333333 1.666667
## f2 2.0000000 2.3333333 2.666667 3.000000 3.333333
## f3 3.6666667 4.0000000 4.333333 4.666667 5.000000
```

Matrices

```
m1+3
```

```
##    c1 c2 c3 c4 c5
## f1  4  5  6  7  8
## f2  9 10 11 12 13
## f3 14 15 16 17 18
```

```
m1**3
```

```
##      c1   c2   c3   c4   c5
## f1    1    8   27   64  125
## f2  216  343  512  729 1000
## f3 1331 1728 2197 2744 3375
```

Matrices

```
m1+m1
```

```
##    c1 c2 c3 c4 c5
## f1  2  4  6  8 10
## f2 12 14 16 18 20
## f3 22 24 26 28 30
```

```
m1-2*m1
```

```
##    c1  c2  c3  c4  c5
## f1 -1  -2  -3  -4  -5
## f2 -6  -7  -8  -9 -10
## f3 -11 -12 -13 -14 -15
```

Matrices

Comparaciones lógicas

```
m1>5
```

```
##       c1    c2    c3    c4    c5
## f1 FALSE FALSE FALSE FALSE FALSE
## f2  TRUE  TRUE  TRUE  TRUE  TRUE
## f3  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
m1<=5
```

```
##       c1    c2    c3    c4    c5
## f1  TRUE  TRUE  TRUE  TRUE  TRUE
## f2 FALSE FALSE FALSE FALSE FALSE
## f3 FALSE FALSE FALSE FALSE FALSE
```

Matrices

```
m1>5 & m1<12
```

```
##       c1    c2    c3    c4    c5
## f1 FALSE FALSE FALSE FALSE FALSE
## f2  TRUE  TRUE  TRUE  TRUE  TRUE
## f3  TRUE FALSE FALSE FALSE FALSE
```

```
m1>12 | m1<5
```

```
##       c1    c2    c3    c4    c5
## f1  TRUE  TRUE  TRUE  TRUE FALSE
## f2 FALSE FALSE FALSE FALSE FALSE
## f3 FALSE FALSE  TRUE  TRUE  TRUE
```

Matrices

Con esto podemos filtrar la matriz:

```
m1[m1>5]
```

```
##  [1]  6 11  7 12  8 13  9 14 10 15
```

```
m1[m1<=5]
```

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

```
m1[m1>5 & m1<12]
```

```
## [1]  6 11  7  8  9 10
```

```
m1[m1>12 | m1<5]
```

```
## [1]  1  2  3 13  4 14 15
```

Matrices

Funciones asociadas a matrices

Algunas funciones asociadas a matrices en R:

Función	Descripción
`diag()`	Diagonal
`*`	Producto elemento a elemento
`%*%`	Producto matricial
`dim()`	Dimensiones
`ncol()`	Número de columnas
`nrow()`	Número de filas
`t()`	Transpuesta
`det()`	Determinante
`solve()`	Inversa
`rowSums()`	Suma de filas
`colSums()`	Suma de columnas
`rowMeans()`	Promedio simple de filas
`colMeans()`	Promedio simple de columnas

Matrices

Podemos unir filas o columnas a una matriz con los comandos `rbind()` y `cbind()`

```
nuevafila <- c(5,10,15,20,25)
m2 <- rbind(m1,nuevafila)
m2
```

```
##             c1 c2 c3 c4 c5
## f1           1  2  3  4  5
## f2           6  7  8  9 10
## f3          11 12 13 14 15
## nuevafila    5 10 15 20 25
```

Matrices

```
nuevacolumna <- c(3,6,9,12)
m3 <- cbind(m2,nuevacolumna)
m3
```

```
##             c1 c2 c3 c4 c5 nuevacolumna
## f1           1  2  3  4  5            3
## f2           6  7  8  9 10            6
## f3          11 12 13 14 15            9
## nuevafila    5 10 15 20 25           12
```

Bases de datos

En R se pueden abrir distintos tipos de bases de datos. También hay varias bases que vienen integradas como `mtcars`. Pueden revisar un listado más completo con `data()`. Antes de aprender a abrir una base de datos vamos a aprender algunas funciones utiles. Las funciones `head()` y `tail()` nos permiten ver las primeras y últimas 6 observaciones de una base respectivamente.

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Bases de datos

```
tail(mtcars)
```

```
##                  mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Porsche 914-2   26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa    30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L  15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino    19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora   15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E      21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

Bases de datos

El comando `str()` nos entrega una breve descripción de la base de datos y el tipo de variables que contiene.

```
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Bases de datos

La función `summary()` nos entrega estadísticas descriptivas de las variables de la base. En el caso de las variables numéricas nos entrega: mínimo, primer cuartil, mediana, media, tercer cuartil y máximo. En el caso de variables categóricas realiza un conteo de éstas.

```
summary(mtcars)
```

```
##       mpg             cyl             disp             hp       
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0  
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
##       drat             wt             qsec             vs        
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
##        am              gear            carb      
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000  
```

Bases de datos

Para crear nuestra propia base de datos podemos utilizar la función `data.frame()`. Con el ejemplo anterior de las notas vamos a crear una nueva base.

```
Tipo <- c("I1","I2","I3","I4","L1","L2")
Nota <- c(4.5,5.3,3.9,6.0,6.5,7.0)
Azul <- c(T,T,F,T,T,T)
Libreta <- data.frame(Tipo,Nota,Azul)
Libreta
```

```
##   Tipo Nota  Azul
## 1   I1  4.5  TRUE
## 2   I2  5.3  TRUE
## 3   I3  3.9 FALSE
## 4   I4  6.0  TRUE
## 5   L1  6.5  TRUE
## 6   L2  7.0  TRUE
```

Bases de datos

```
str(Libreta)
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ Tipo: Factor w/ 6 levels "I1","I2","I3",..: 1 2 3 4 5 6
##  $ Nota: num  4.5 5.3 3.9 6 6.5 7
##  $ Azul: logi  TRUE TRUE FALSE TRUE TRUE TRUE
```

```
summary(Libreta)
```

```
##  Tipo        Nota          Azul
##  I1:1   Min.   :3.900   Mode :logical
##  I2:1   1st Qu.:4.700   FALSE:1
##  I3:1   Median :5.650   TRUE :5
##  I4:1   Mean   :5.533
##  L1:1   3rd Qu.:6.375
##  L2:1   Max.   :7.000
```

Bases de datos

Esta función reconoce inmediatamente los nombres de los vectores como nombres de las columnas y las filas las indexa con el número de la observación. Los nombres de las columnas también se pueden definir al momento de crear la base de datos de la siguiente forma:

```
Libreta2 <- data.frame(Tipo2=Tipo,Nota2=Nota,Azul2=Azul)
Libreta2
```

```
##   Tipo2 Nota2 Azul2
## 1    I1   4.5  TRUE
## 2    I2   5.3  TRUE
## 3    I3   3.9 FALSE
## 4    I4   6.0  TRUE
## 5    L1   6.5  TRUE
## 6    L2   7.0  TRUE
```

Bases de datos

Podemos acceder a los subconjuntos de esta base de datos tal como se hacía con la matrices. También podemos llamar a la columna que requiramos por su nombre.

```
Libreta[c(2,4),c("Azul","Nota")]
```

```
##    Azul Nota
## 2 TRUE  5.3
## 4 TRUE  6.0
```

Bases de datos

Si requerimos acceder a la columna `v1` de una base de datos `df`, también se pueden usar las siguientes notaciones:

```
df$v1
df["v1"]
```

Un ejemplo en este caso sería:

```
Libreta$Nota
Libreta["Nota"]
```

Bases de datos

Para acceder a un subconjunto de datos con alguna restricción en específico existe la función `subset()` a la cual entregamos la base de datos como primer argumento y luego como segundo argumento `subset=` se entregan las condiciones requeridas.

```
subset(Libreta,subset= Azul==TRUE)
```

```
##   Tipo Nota Azul
## 1   I1  4.5 TRUE
## 2   I2  5.3 TRUE
## 4   I4  6.0 TRUE
## 5   L1  6.5 TRUE
## 6   L2  7.0 TRUE
```

Bases de datos

```
subset(Libreta,subset= Nota>6)
```

```
##   Tipo Nota Azul
## 5   L1  6.5 TRUE
## 6   L2  7.0 TRUE
```

Otra forma de filtrar que es frecuentemente usada es mediante los corchetes, indicando condiciones para filas y también seleccionando columnas de interés.

```
Libreta[Libreta$Azul==TRUE,]
```

```
##   Tipo Nota Azul
## 1   I1  4.5 TRUE
## 2   I2  5.3 TRUE
## 4   I4  6.0 TRUE
## 5   L1  6.5 TRUE
## 6   L2  7.0 TRUE
```

```
Libreta[Libreta$Azul==TRUE & Libreta$Nota>6,]
```

```
##   Tipo Nota Azul
## 5   L1  6.5 TRUE
## 6   L2  7.0 TRUE
```

Bases de datos

```
Libreta[Libreta$Azul==TRUE & Libreta$Nota>6,]
```

```
##   Tipo Nota Azul
## 5   L1  6.5 TRUE
## 6   L2  7.0 TRUE
```

Como se puede ver, la diferencia radica en que con la función `subset` se llama directamente a las variables por su nombre, porque se entrega la base completa en la función, en cambio al realizar el filtro "manual" se debe llamar a la base y con el signo $ indicar recién el nombre de la variable.

Bases de datos

Para ordenar una base de datos en función de una variable existe la función `order()`. Por ejemplo para ordenar la base de datos en orden creciente de `Nota`. Si se antepone el signo "-" entonces se ordena de manera decreciente.

```
Libreta[order(Libreta$Nota),]
```

```
##   Tipo Nota  Azul
## 3   I3  3.9 FALSE
## 1   I1  4.5  TRUE
## 2   I2  5.3  TRUE
## 4   I4  6.0  TRUE
## 5   L1  6.5  TRUE
## 6   L2  7.0  TRUE
```

Bases de datos

```
Libreta[order(-Libreta$Nota),]
```

```
##   Tipo Nota  Azul
## 6   L2  7.0  TRUE
## 5   L1  6.5  TRUE
## 4   I4  6.0  TRUE
## 2   I2  5.3  TRUE
## 1   I1  4.5  TRUE
## 3   I3  3.9 FALSE
```

Bases de datos

Para agregar una nueva fila a una base de datos del tipo `data.frame` se puede utilizar el comando `rbind()` conocido anteriormente, siempre preocupándonos que la o las nuevas filas tengan los mismos nombres de columnas de la base de datos.

```
nuevafila <- data.frame(Tipo="I5", Nota=4.5, Azul=TRUE)
nuevaLibreta <- rbind(Libreta, nuevafila)
nuevaLibreta
```

```
##   Tipo Nota  Azul
## 1   I1  4.5  TRUE
## 2   I2  5.3  TRUE
## 3   I3  3.9 FALSE
## 4   I4  6.0  TRUE
## 5   L1  6.5  TRUE
## 6   L2  7.0  TRUE
## 7   I5  4.5  TRUE
```

Bases de datos

Para agregar una nueva columna se puede escribir dentro de la misma base.

```
nuevaLibreta$nuevacolumna <- nuevaLibreta$Nota+1
nuevaLibreta
```

```
##   Tipo Nota  Azul nuevacolumna
## 1   I1  4.5  TRUE          5.5
## 2   I2  5.3  TRUE          6.3
## 3   I3  3.9 FALSE          4.9
## 4   I4  6.0  TRUE          7.0
## 5   L1  6.5  TRUE          7.5
## 6   L2  7.0  TRUE          8.0
## 7   I5  4.5  TRUE          5.5
```

Bases de datos

Los nombres de filas y columnas se pueden renombrar con los comandos `rownames()` y `colnames()`. También se puede renombrar una o más filas o columnas en específico indicando la posición de la fila o columna dentro de los corchetes que hemos ocupado anteriormente.

```
colnames(nuevaLibreta) <- c("Tipo2","Nota2","Azul2","Col2")
nuevaLibreta
```

```
##   Tipo2 Nota2 Azul2 Col2
## 1    I1   4.5  TRUE  5.5
## 2    I2   5.3  TRUE  6.3
## 3    I3   3.9 FALSE  4.9
## 4    I4   6.0  TRUE  7.0
## 5    L1   6.5  TRUE  7.5
## 6    L2   7.0  TRUE  8.0
## 7    I5   4.5  TRUE  5.5
```

Bases de datos

```
colnames(nuevaLibreta)[4] <- "NuevaColumna2"
nuevaLibreta
```

```
##   Tipo2 Nota2 Azul2 NuevaColumna2
## 1    I1   4.5  TRUE           5.5
## 2    I2   5.3  TRUE           6.3
## 3    I3   3.9 FALSE           4.9
## 4    I4   6.0  TRUE           7.0
## 5    L1   6.5  TRUE           7.5
## 6    L2   7.0  TRUE           8.0
## 7    I5   4.5  TRUE           5.5
```

Listas

Hasta ahora ya hemos visto que podemos crear vectores, matrices y bases de datos (`data.frame`). Para juntar todos estos tipos de variables y guardarlos en un objeto existen las listas. con el comando `list()` se pueden crear listas con distintos tipos de objetos en su interior. Sólo le debemos entregar los objetos que se quieren almacenar en ella.

```
lista <- list(opiniones,m1,Libreta)
```

Listas

```
lista
```

```
## [[1]]
## [1] Bueno  Malo   Neutro Bueno  Malo
## Levels: Malo Neutro Bueno
##
## [[2]]
##    c1 c2 c3 c4 c5
## f1  1  2  3  4  5
## f2  6  7  8  9 10
## f3 11 12 13 14 15
##
## [[3]]
##   Tipo Nota  Azul
## 1   I1  4.5  TRUE
## 2   I2  5.3  TRUE
## 3   I3  3.9 FALSE
## 4   I4  6.0  TRUE
## 5   L1  6.5  TRUE
## 6   L2  7.0  TRUE
```

Listas

Tal como en los `data.frame`, podemos asignarle nombre a los componentes de la lista.

```
lista2 <- list(v=opiniones, m=m1, bd=Libreta)
```

Luego podemos acceder a estos objetos llamándolos por su nombre establecido o por su coordenada dentro de la lista:

```
lista[1]
```

```
## [[1]]
## [1] Bueno  Malo   Neutro Bueno  Malo
## Levels: Malo Neutro Bueno
```

```
lista2$v
```

```
## [1] Bueno  Malo   Neutro Bueno  Malo
## Levels: Malo Neutro Bueno
```