Primer Laboratorio - Introducción al R

Bioestadística 2020

R es un es un entorno y lenguaje de programación libre especialmente adaptado al cálculo estadístico. Además permite hacer gráficos fácilmente. R es gratis y compila y funciona en una gran variedad de plataformas de UNIX, Windows y MacOS. Se puede bajar el código fuente completo, la distribución binaria compilada, las extensiones y documentación de R de "Comprehensive R Archive Network" (CRAN): http://www.cran.r-project.org.

Se recomienda utilizar RStudio https://www.rstudio.com/ en cualquiera de los sistemas operativos, aunque se puede usar R desde una terminal. Se abre una terminal y al escribir el comando R en la misma se abre la consola de R.

Para mayor comodidad, siempre trabajaremos usando RStudio. Al comienzo de la sesión, podrán observar el script (ventana superior izquierda) que contiene un archivo con varias líneas de comando, y la terminal (ventana inferior izquierda) que muestra los resultados de las operaciones realizadas. Para ejecutar una sección del código, seleccionar dicha sección, y presionar 'ctrl+enter'. El resultado de la operación se mostrará en la terminal. Para ejecutar una linea de código, se puede colocar el cursor sobre la linea, y luego presionar 'ctrl+enter'.

Comenzando con lo más sencillo, podemos realizar operaciones aritméticas escribiendo el siguiente comando en la consola, y luego apretando 'enter'. Lo mismo se puede lograr escribiendo el comando en script y luego presionando 'ctrl+enter'.

```
1 + 2
```

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

Observamos que **R** nos retorna el resultado de la operación, pero no lo guarda. Si queremos indicar que el resultado de una operación debe ser guardado, debemos guardarlo en una variable. Para asignar un valor a una variable se puede usar una flecha "<-" o el igual "=".

```
a <- 4
b = 3
a # muestra el valor de la variable 'a'
```

El símbolo # significa que todo lo que le sigue es solo comentario.

El software **R** contiene varias funciones básicas, las cuales están a nuestra disposición. Por ejemplo, para calcular la raiz cuadrada de un número, podemos usar la función *sqrt()*

```
sqrt(a)
```

Para obtener información sobre una función específica, el comando es

```
help(sqrt)
```

El nombre de la función se pasa como argumento de la función help. R es sensible a mayúsculas y minúsculas, por lo tanto A y a son símbolos diferentes y por lo tanto refieren a variables distintas. Si no se recuerda el nombre exacto de una función, se puede buscar en los archivos de ayuda de R tipeando

```
help.search("square root")
```

el término buscado (en inglés) debe estar dentro del paréntesis y con comillas. Al utilizar help.search aparecen todos los comandos que contengan dicha palabra, con la sintaxis paquete:comando. Varios paquetes ya vienen instalados con el R básico. Aquellos que no lo están se pueden instalar y luego cargarlos con el comando *library(paquete)*.

Ejercicio 1. Mirar la ayuda de la función mean(). Determinar si existe una función que calcule el máximo de un grupo de números.

```
help(mean)
help.search('maxima') #Maxima es el plural de maximum.
```

Manipulaciones simples: números y vectores

R opera en datos con estructuras predeterminadas. La estructura más simple es el vector numérico. Como ya vimos, para asignar un valor a una variable se puede usar una flecha "<-" o el igual "=".

```
 a = 5
```

Para determinar un vector se deben concatenar varios números, utilizando la función c().

```
x <- c(3.5, 4, 0.6, 8, 12)
```

La flecha también funciona en la otra dirección "->"

```
c(3.5, 4, 0.6, 8, 12) -> z
```

Para ver el contenido de una variable en la consola podemos ver su contenido. Tipeando x y z verificamos que son el mismo vector.

```
x
```

```
## [1]  3.5  4.0  0.6  8.0 12.0
```

```
z
```

```
## [1]  3.5  4.0  0.6  8.0 12.0
```

También se pueden crear vectores concatenando vectores ya existentes. **R** contiene todos los operadores aritméticos estándar (ver tabla). Estas operaciones se pueden usar también con vectores. Además de las expresiones artiméticas comunes también se pueden usar: log(), sin(), cos(), exp(), tan().

```
u<- 4*x + z
```

Ejercicio 2

- Crear un vector y que contenga al vector x, al número 3 y al vector z multiplicado por 2.
- Cuál es el resultado de $x+y$ (dos vectores de diferente largo). Y de $y+x$?

Ejercicio 3

Explorar qué hacen las funciones range(), mean(), length(), sum() aplicadas a un vector.

Ejercicio 4 Explorar qué hace la función sort(). Es posible ordenar los elementos de manera decreciente?

Los vectores se pueden crear "a mano", escribiendo todos los elementos

diez <-c(1,2,3,4,5,6,7,8,9,10)

o usando las facilidades de **R**. Para crear vectores que contienen secuencias de números se puede usar "a:b" que crea una secuencia que empieza en a y va sumando 1 hasta llegar a b, o la función seq(a,b,c), que hace lo mismo, pero suma c en vez de uno (c puede ser cualquier real positivo). La función ":" tiene alta prioridad ante otras operaciones.

Ejercicio 5

- Comparar los resultados de la expresión *1:n-1* y *1:(n-1)* para algún número natural n.
- Generar una secuencia de números del 20 al 1.
- Generar otra secuencia del 20 al 2, sólo de números pares.

Para elegir subconjuntos de un vector alcanza con poner el nombre del vector y el índice o índices de los elementos que se quieren elegir con paréntesis rectos []. Los índices deben de variar entre 1 y *n=length(vector)* largo del vector. Por ejemplo, para obtener la primera componente de x

x[1]

Ejercicio 6 Usar la función ":" para elegir los elementos del 2 al 5 del vector y.

Operadores

Aritméticos	**Comparación**	**Lógicos**
+ adición	< menor	! x NO lógico
- substracción	> mayor	x & y Y lógico
* multiplicación	<= menor o igual	x | y O inclusivo
/ división	<= menor o igual	xor(x,y) O exclusivo
^ exponente	>= mayor o igual	
%% módulo (resto)	== igual	
%/% división entera	!= distinto	

Los vectores lógicos pueden ser utilizados con operadores aritméticos. En ese caso el FALSE se convierte en 0 y el TRUE se convierte en 1.

```
a <- TRUE + TRUE
b <- TRUE * FALSE
```

También se pueden usar para indicar qué entradas mostrar en un vector.

```
x[x >= 4]
```

```
## [1]  4  8 12
```

Observamos que el resultado es las entradas de x mayores o iguales a 4. Comparar con el siguiente vector,

```
x >= 4
```

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

que devuelve un vector de TRUE y FALSE según si cada valor de x es mayor o igual que 4 o no.

Ejercicio 7 Crear un vector m que contenga los elementos de y mayores que 2 y menores que 5. Explorar la diferencia entre las multiplicaciones

m * m
m % * % m

Condiciones

Las condiciones son una herramienta muy usada en cualquier lenguaje de programación. Muchas veces, al escribir código, necesitamos hacer preguntas y tomar decisiones en función de ellas. Pongamos el ejemplo de calcular el valor absoluto de un número. Intentemos escribir un código que, al dar un número, devuelva su valor absoluto. La manera más intuitiva de hacerlo es preguntando si dicho numero es negativo o no. Si es negativo, se le cambia el signo. De lo contrario, se deja intacto. La manera de realizar esa pregunta es usando el enunciado *if(){}*. Este enunciado evalúa una variable lógica, que está entre paréntesis curvos. Si esa variable es 'TRUE', entonces se ejecutan las órdenes que estén dentro de los corchetes. Si la variable es *FALSE*, las órdenes entre corchetes se ignoran.

```
x = -1
if(x < 0){
```

```
  # Observar que la variable lógica aquí es 'x < 0'. Será 'TRUE' si 'x es negativo.
  print("x es negativo")
}
```

```
## [1] "x es negativo"
```

```
x = 3
if(x < 0){
  print("x es negativo")
} # No imprime nada.
```

Si queremos ejecutar otras órdenes cuando la variable lógica sea *FALSE*, tenemos el enunciado *if(){}else{}*. Este enunciado funciona igual que el anterior, salvo que si la variable lógica es *FALSE*, entonces se ejecutan las órdenes en los corchetes luego del *else*.

```
x = -1
if(x < 0){
  print("x es negativo")
} else{
  print("x es positivo")
}
```

```
## [1] "x es negativo"
```

```
x = 3
if(x < 0){
  print("x es negativo")
} else{
  print("x es positivo")
}
```

```
## [1] "x es positivo"
```

También es posible que necesitmos discutir varios casos. Supongamos que el caso $x = 0$ es complicado para nosotros (no sabemos si el cero es positivo o negativo). Podemos hacer una discusión más completa encadenando enunciados *if* y *else*.

```
x = 0
if(x < 0){
  print("x es negativo")
} else if(x>0){
  # Podemos pensar que iniciamos una nueva discusión
  # para los casos que rechazamos en el primer 'if'.
  print("x es positivo")
} else{
  print("x es cero")
}
```

```
## [1] "x es cero"
```

```
x = 1
if(x < 0){
  print("x es negativo")
} else if(x > 0){
  print("x es positivo")
} else{
  print(" es cero")
}
```

```
## [1] "x es positivo"
```

Ejercicio 8

Escribir un código que, dado un x, retorne 0 si $x<0$, y $x\hat{}2$ en caso contrario.

Ejercico 9

Escribir un código que, dado un par de números x,y, retorne el producto si ambos son positivos, su suma si ambos son negativos y 0 en caso contrario.